INTERMEDIATE T-SQL

Welcome

Ginger Grant
Instructor

Course overview

- Chapter 1: Summarizing data
- Chapter 2: Date and math functions
- Chapter 3: Processing data with T-SQL
- Chapter 4: Window functions

Exploring Data with Aggregation

- Reviewing summarized values for each column is a common first step in analyzing data
- If the data exists in a database, fastest way to aggregate is to use SQL

Data Exploration with EconomicIndicators

```
SELECT Country, Year, InternetUse, GDP,
       ExportGoodsPercent, CellPhonesper100
FROM EconomicIndicators

+-----------+-----+------------+------------+------------------+---------------
|Country    |Year |InternetUse | GDP        |ExportGoodsPercent|CellPhonesper10(
+-----------+-----+------------+------------+------------------+---------------
|Swaziland  |2011 |20.43165813 |7335004354  |56.30476059       |63.7015615
|Sweden     |2011 |90.88204559 |394271163688|49.93022195       |118.5711258
|Switzerland|2011 |82.98773087 |395111518596|51.20242546       |130.0623629
...
+-----------+-----+------------+------------+------------------+---------------
```

Common summary statistics

- `MIN()` for the minimum value of a column
- `MAX()` for the maximum value of a column
- `AVG()` for the mean or average value of a column

Common summary statistics in T-SQL

```
/*This T-SQL query returns the aggregated values
of column InternetUse*/

SELECT AVG(InternetUse) AS MeanInternetUse,
MIN(InternetUse) AS MINInternet,
MAX(InternetUse) AS MAXInternet
FROM EconomicIndicators
```

```
+------------------+------------+-------------+
|MeanInternetUse    |MINInternet |  MAXInternet|
|------------------+------------+-------------|
|   18.9854496196171|          0 |  375.5970064|
+------------------+------------+-------------+
```

Filtering Summary Data with WHERE

```
/*This T-SQL query filters the aggregated values using a WHERE clause
Notice the text value is in */

SELECT AVG(InternetUse) AS MeanInternetUse,
MIN(InternetUse) AS MINInternet,
MAX(InternetUse) AS MAXInternet
FROM EconomicIndicators
WHERE Country = 'Solomon Islands'
```

```
+------------------+------------+-------------+
|MeanInternetUse    |MINInternet |  MAXInternet|
|------------------+------------+-------------|
|            1.79621|          0 |         6.00|
+------------------+------------+-------------+
```

Subtotaling Aggregations into Groups with GROUP BY

```
SELECT Country, AVG(InternetUse) AS MeanInternetUse,
MIN(InternetUse) AS MINInternet,
MAX(InternetUse) AS MAXInternet
FROM EconomicIndicators
GROUP BY Country
```

```
+--------------------+------------------+------------+-------------+
|  Country           |MeanInternetUse   |MINInternet |  MAXInternet|
|--------------------+------------------+------------+-------------|
|Solomon Islands     |           1.79621|           0|         6.00|
|Hong Kong           |          245.1067|           0|       375.00|
|Liechtenstein       |           63.8821|     36.5152|        85.00|
...
+--------------------+------------------+------------+-------------+
```

HAVING is the WHERE for Aggregations

Cannot use `WHERE` with `GROUP BY` as it will give you an error

```
-- This throws an error
...
GROUP BY
WHERE Max(InternetUse) > 100
```

Instead, use `HAVING`

```
-- This is how you filter with a GROUP BY
...
GROUP BY
HAVING Max(InternetUse) > 100
```

HAVING is the WHERE for Aggregations

```
SELECT Country, AVG(InternetUse) AS MeanInternetUse,
MIN(GDP) AS SmallestGDP,
MAX(InternetUse) AS MAXInternetUse
FROM EconomicIndicators
GROUP BY Country
HAVING MAX(InternetUse) > 100
```

```
+--------------+------------------+-------------+---------------+
|Country       |MeanInternetUse    |SmallestGDP  | MAXInternetUse|
|--------------+------------------+-------------+---------------|
|Macedonia     |   71.3060150792857| -0.465059948|    110.5679538|
|Hong Kong     |   245.106718614286|            0|    375.5970064|
|Congo         |   60.8972476010714| -9.492757847|    104.6455529|
...
+--------------+------------------+-------------+---------------+
```

Examining UFO Data in the Incidents Table

- The exercise will explore data gathered from Mutual UFO Network
- UFO spotted all over the world are contained in the Incidents Table

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Finding and Resolving Missing Data

Ginger Grant
Instructor

Detecting missing values

- When you have no data, the empty database field contains the word `NULL`
- Because `NULL` is not a number, it is not possible to use `=`, `<`, or `>` to find or compare missing values
- To determine if a column contains a `NULL` value, use `IS NULL` and `IS NOT NULL`

Returning No NULL Values in T-SQL

```
SELECT Country, InternetUse, Year
FROM EconomicIndicators
WHERE InternetUse IS NOT NULL
```

```
+------------------+------------------+------------+
|Country           |InternetUse       |Year        |
|------------------+------------------+------------+
|Afghanistan       |4.58066992        |2011        |
|Albania           |49                |2011        |
|Algeria           |14                |2011        |
....
+------------------+------------------+------------+
```

Detecting NULLs in T-SQL

```
SELECT  Country, InternetUse, Year
FROM EconomicIndicators
WHERE InternetUse IS NULL
```

```
+-------------------+-------------------+------------+
|Country            |InternetUse        |Year        |
|-------------------+-------------------+------------+
|Angola             |NULL               |2013        |
|Argentina          |NULL               |2013        |
|Armenia            |NULL               |2013        |
....
+-------------------+-------------------+------------+
```

Blank is not NULL

- A blank is not the same as a NULL value
- May show up in columns containing text
- An empty string `''` can be used to find blank values
- The best way is to look for a column where the Length or LEN > 0

Blank is not NULL

```
SELECT Country, GDP, Year
FROM EconomicIndicators
WHERE LEN(GDP) > 0
```

```
+-------------------+-------------------+------------+
|Country            |GDP                |Year        |
|-------------------+-------------------+------------+
|Afghanistan        |54852215624        |2011        |
|Albania            |29334492905        |2011        |
|Algeria            |453558093404       |2011        |
....
+-------------------+-------------------+------------+
```

Substituting missing data with a specific value using ISNULL

```
SELECT GDP, Country,
ISNULL(Country, 'Unknown') AS NewCountry
FROM EconomicIndicators
```

```
+------------------+----------------+----------------+
|GDP               |Country         |NewCountry      |
|------------------+----------------+----------------+
|5867920022        |NULL            |Unknown         |
|597873038497      |South Africa    |South Africa    |
|1474091271101     |NULL            |Unknown         |
...
+------------------+----------------+----------------+
```

Substituting missing data with a column using ISNULL

```
/*Substituting values from one column for another with ISNULL*/
SELECT TradeGDPPercent, ImportGoodPercent,
ISNULL(TradeGDPPercent, ImportGoodPercent) AS NewPercent
FROM EconomicIndicators
```

```
+------------------+------------------+----------------+
|TradeGDPPercent   |ImportGoodPercent |NewPercent      |
|------------------+------------------+----------------+
|NULL              |56.7              |56.7            |
|52.18720739       |51.75273421       |52.18720739     |
|NULL              |NULL              |NULL            |
...
+------------------+------------------+----------------+
```

Substituting NULL values using COALESCE

`COALESCE` returns the first non-missing value

```
COALESCE( value_1, value_2, value_3, ... value_n )
```

- If `value_1` is `NULL` and `value_2` is not `NULL`, return `value_2`
- If `value_1` and `value_2` are `NULL` and `value_3` is not `NULL`, return `value_3`
- ...

SQL Statement using COALESCE

```
SELECT TradeGDPPercent, ImportGoodPercent,
COALESCE(TradeGDPPercent, ImportGoodPercent, 'N/A') AS NewPercent
FROM EconomicIndicators
```

```
+------------------+--------------------+---------------+
|TradeGDPPercent   |ImportGoodPercent   |NewPercent     |
|------------------+--------------------+---------------+
|NULL              |56.7                |56.7           |
|NULL              |NULL                |N/A            |
|52.18720739       |51.75273421         |52.18720739    |
+------------------+--------------------+---------------+
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Binning Data with Case

Ginger Grant
Instructor

Changing column values with CASE

```
CASE
     WHEN Boolean_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

Changing column values with CASE in T-SQL

```
SELECT Continent,
CASE WHEN Continent = 'Europe' or Continent = 'Asia' THEN 'Eurasia'
     ELSE 'Other'
     END AS NewContinent
FROM EconomicIndicators
```

```
+-----------+--------------+
|Continent  |NewContinent  |
+-----------+--------------+
|Europe     |Eurasia       |
|Asia       |Eurasia       |
|Americas   |Other         |
...
+-----------+--------------+
```

Changing column values with CASE in T-SQL

```
SELECT Continent,
CASE WHEN Continent = 'Europe' or Continent = 'Asia' THEN 'Eurasia'
     ELSE Continent
     END AS NewContinent
FROM EconomicIndicators
```

```
+-----------+--------------+
|Continent  |NewContinent  |
+-----------+--------------+
|Europe     |Eurasia       |
|Asia       |Eurasia       |
|Americas   |Americas      |
...
+-----------+--------------+
```

Using CASE statements to create value groups

```
-- We are binning the data here into discrete groups
SELECT Country, LifeExp,
CASE WHEN LifeExp < 30 THEN 1
     WHEN LifeExp > 29 AND LifeExp < 40 THEN 2
     WHEN LifeExp > 39 AND LifeExp < 50 THEN 3
     WHEN LifeExp > 49 AND LifeExp < 60 THEN 4
     ELSE 5
     END AS LifeExpGroup
FROM EconomicIndicators
WHERE Year = 2007
```

```
+-----------+--------------+
|LifeExp    |LifeExpGroup  |
+-----------+--------------+
|25         |1             |
|30         |2             |
|65         |5             |
...
+-----------+--------------+
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Counts and Totals

Ginger Grant
Instructor

Examining Totals with Counts

```
SELECT COUNT(*) FROM Incidents
```

```
+-----------------+
|(No column name) |
+-----------------+
|6452             |
+-----------------+
```

COUNT with DISTINCT

```
COUNT(DISTINCT COLUMN_NAME)
```

COUNT with DISTINCT in T-SQL (I)

```
SELECT COUNT(DISTINCT Country) AS Countries
FROM Incidents
```

```
+----------------+
|Countries       |
------------------+
|3               |
------------------+
```

COUNT with DISTINCT in T-SQL (II)

```
SELECT COUNT(DISTINCT Country) AS Countries,
COUNT(DISTINCT City) AS Cities
FROM Incidents
```

```
+----------------+-------------+
|Countries       | Cities      |
-----------------+-------------+
|3               | 3566        |
-----------------+-------------+
```

COUNT AGGREGATION

- `GROUP BY` can be used with `COUNT()` in the same way as the other aggregation functions such as `AVG()`, `MIN()`, `MAX()`
- Use the `ORDER BY` command to sort the values
  - `ASC` will return the smallest values first (default)
  - `DESC` will return the largest values first

COUNT with GROUP BY in T-SQL

```
-- Count the rows, subtotaled by Country
SELECT COUNT(*) AS TotalRowsbyCountry, Country
FROM Incidents
GROUP BY Country
```

```
----------------------+-----------------+
|TotalRowsbyCountry   | Country         |
----------------------+-----------------+
|5452                 |us               |
|750                  |NULL             |
|249                  |ca               |
|1                    |gb               |
----------------------+-----------------+
```

COUNT with GROUP BY and ORDER BY in T-SQL (I)

```
-- Count the rows, subtotaled by Country
SELECT COUNT(*) AS TotalRowsbyCountry, Country
FROM Incidents
GROUP BY Country
ORDER BY Country ASC
```

```
----------------------+-----------------+
|TotalRowsbyCountry   | Country         |
----------------------+-----------------+
|750                  |NULL             |
|249                  |ca               |
|1                    |gb               |
|5452                 |us               |
----------------------+-----------------+
```

COUNT with GROUP BY and ORDER BY in T-SQL (II)

```
-- Count the rows, subtotaled by Country
SELECT COUNT(*) AS TotalRowsbyCountry, Country
FROM Incidents
GROUP BY Country
ORDER BY Country DESC
```

```
----------------------+-----------------+
|TotalRowsbyCountry   | Country         |
----------------------+-----------------+
|5452                 |us               |
|1                    |gb               |
|249                  |ca               |
|750                  |NULL             |
----------------------+-----------------+
```

Column totals with SUM

- `SUM()` provides a numeric total of the values in a column
- It follows the same pattern as other aggregations
- Combine it with GROUP BY to get subtotals based on columns specified

Adding column values in T-SQL

```
-- Calculate the values subtotaled by Country
SELECT SUM(DurationSeconds) AS TotalDuration, Country
FROM Incidents
GROUP BY Country
```

```
+----------+--------------------+
|Country   |TotalDuration       |
+----------+--------------------+
|us        |17024946.750001565  |
|null      |18859192.800000012  |
|ca        |200975              |
|gb        |120                 |
+----------+--------------------+
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Math with Dates

Ginger Grant
Instructor

DATEPART

`DATEPART` is used to determine what part of the date you want to calculate. Some of the common abbreviations are:

- `DD` for Day
- `MM` for Month
- `YY` for Year
- `HH` for Hour

Common date functions in T-SQL

- `DATEADD()`: Add or subtract datetime values
  - Always returns a date
- `DATEDIFF()`: Obtain the difference between two datetime values
  - Always returns a number

DATEADD

To Add or subtract a value to get a new date use `DATEADD()`

```
DATEADD (DATEPART, number, date)
```

- `DATEPART`: Unit of measurement (DD, MM etc.)
- `number`: An integer value to add
- `date`: A datetime value

Date math with DATEADD (I)

What date is 30 days from June 21, 2020?

```
SELECT DATEADD(DD, 30, '2020-06-21')
+----------------------------+
|(No Column Name)            |
|----------------------------+
|2020-07-21 00:00            |
+----------------------------+
```

Date math with DATEADD (II)

What date is 30 days before June 21, 2020?

```
SELECT DATEADD(MM, -30, '2020-06-21')
+----------------------------+
|(No Column Name)            |
|----------------------------+
|2020-05-22 00:00            |
+----------------------------+
```

DATEDIFF

Returns a date after a number has been added or subtracted to a date

```
DATEDIFF (datepart, startdate, enddate)
```

- `datepart`: Unit of measurement (DD, MM etc.)
- `startdate`: An integer value to add
- `enddate`: A datetime value

Date math with DATEDIFF

```
SELECT DATEDIFF(DD, '2020-05-22', '2020-06-21') AS Difference1,
       DATEDIFF(DD, '2020-07-21', '2020-06-21') AS Difference2

+--------------------+--------------------+
|Difference1         |Difference2         |
+--------------------+--------------------+
|30                  |-30                 |
+--------------------+--------------------+
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Rounding and Truncating numbers

Ginger Grant
Instructor

Rounding numbers in T-SQL

```
ROUND(number, length [,function])
```

Rounding numbers in T-SQL

```
SELECT DurationSeconds,
ROUND(DurationSeconds, 0) AS RoundToZero,
ROUND(DurationSeconds, 1) AS RoundToOne
FROM Incidents
```

```
+--------------------+--------------------+--------------------+
|DurationSeconds     |RoundToZero         |RoundToOne          |
+--------------------+--------------------+--------------------+
|121.6480            |122.0000            |121.6000            |
|170.3976            |170.0000            |170.4000            |
|336.0652            |336.0000            |336.1000            |
...
+--------------------+--------------------+--------------------+
```

Rounding on the left side of the decimal

```
SELECT DurationSeconds,
ROUND(DurationSeconds, -1) AS RoundToTen,
ROUND(DurationSeconds, -2) AS RoundToHundred
FROM Incidents
```

```
+--------------------+-------------------+-------------------+
|DurationSeconds     |RoundToTen         |RoundToHundred     |
+--------------------+-------------------+-------------------+
|121.6480            |120.0000           |100.0000           |
|170.3976            |170.0000           |200.0000           |
|336.0652            |340.0000           |300.0000           |
...
+--------------------+-------------------+-------------------+
```

Truncating numbers

TRUNCATE

$17.85 \rightarrow 17$

ROUND

$17.85 \rightarrow 18$

Truncating with ROUND()

The `ROUND()` function can be used to truncate values when you specify the third argument

```
ROUND(number, length [,function])
```

- Set the third value to a non-zero number

Truncating in T-SQL

```
SELECT Profit,
ROUND(DurationSeconds, 0) AS RoundingtoWhole,
ROUND(DurationSeconds, 0, 1) AS Truncating
FROM Incidents
```

```
+--------------------+--------------------+--------------------+
|Profit              |RoundingtoWhole     |Truncating          |
+--------------------+--------------------+--------------------+
|15.6100             |16.0000             |15.0000             |
|13.2444             |13.0000             |13.0000             |
|17.9260             |18.0000             |17.0000             |
...
+--------------------+--------------------+--------------------+
```

Truncating just cuts all numbers off after the specified digit

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

More math functions

Ginger Grant
Instructor

Absolute value

Use ABS() to return non-negative values

```
ABS(number)
```

Using ABS in T-SQL (I)

```
SELECT ABS(-2.77), ABS(3), ABS(-2)
```

```
+--------------------+--------------------+--------------------+
|(No column name)    |(No column name)    |(No column name)    |
+--------------------+--------------------+--------------------+
|2.77                |3                   |2                   |
+--------------------+--------------------+--------------------+
```

Using ABS in T-SQL (II)

```
SELECT DurationSeconds, ABS(DurationSeconds) AS AbsSeconds
FROM Incidents
```

```
+-------------------+-------------------+
|DurationSeconds    |AbsSeconds         |
+-------------------+-------------------+
|-25.36             |25.36              |
|-258482.44         |258482.44          |
|45.66              |45.66              |
+-------------------+-------------------+
```

Squares and square roots in T-SQL

```
SELECT SQRT(9) AS Sqrt,
       SQUARE(9) AS Square

+--------+-----------+
|Sqrt    |Square     |
+--------------------+
|3       |81         |
+--------+-----------+
```

Logs

- `LOG()` returns the natural logarithm
- Optionally, you can set the base, which if not set is 2.718281828

```
LOG(number [,Base])
```

Calculating logs in T-SQL

```
SELECT  DurationSeconds, LOG(DurationSeconds, 10) LogSeconds
FROM Incidents

+--------------------+--------------------+
|DurationSeconds     |LogSeconds          |
+--------------------+--------------------+
|37800               |4.577491799837225   |
|5                   |0.6989700043360187  |
|20                  |1.301029995663981   |
...
+--------------------+--------------------+
```

Log of 0

You cannot take the log of 0 as it will give you an error

```
SELECT LOG(0, 10)

An invalid floating point operation occurred.
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

WHILE loops

Ginger Grant
Instructor

Using variables in T-SQL

- Variables are needed to set values

```
DECLARE @variablename data_type
```

  - Must start with the character @

Variable data types in T-SQL

- `VARCHAR(n)`: variable length text field
- `INT`: integer values from -2,147,483,647 to +2,147,483,647
- `DECIMAL(p ,s)` or `NUMERIC(p ,s)`:
  - `p`: total number of decimal digits that will be stored, both to the left and to the right of the decimal point
  - `s`: number of decimal digits that will be stored to the right of the decimal point

Declaring variables in T-SQL

```
-- Declare Snack as a VARCHAR with length 10
DECLARE @Snack VARCHAR(10)
```

Assigning values to variables

```
-- Declare the variable
DECLARE @Snack VARCHAR(10)
-- Use SET a value to the variable
SET @Snack = 'Cookies'
-- Show the value
SELECT @Snack
```

```
+--------------------+
|(No column name)    |
+--------------------+
|Cookies             |
+--------------------+
```

```
-- Declare the variable
DECLARE @Snack VARCHAR(10)
-- Use SELECT assign a value
SELECT @Snack = 'Candy'
-- Show the value
SELECT @Snack
```

```
+--------------------+
|(No column name)    |
+--------------------+
|Candy               |
+--------------------+
```

WHILE loops

- WHILE evaluates a true or false condition
- After the WHILE, there should be a line with the keyword BEGIN
- Next include code to run until the condition in the WHILE loop is true
- After the code add the keyword END
- BREAK will cause an exit out of the loop
- CONTINUE will cause the loop to continue

WHILE loop in T-SQL (I)

```
-- Declare ctr as an integer
DECLARE @ctr INT
-- Assign 1 to ctr
SET @ctr = 1

-- Specify the condition of the WHILE loop
WHILE @ctr < 10

    -- Begin the code to execute inside WHILE loop
    BEGIN
       -- Keep incrementing the value of @ctr
       SET @ctr = @ctr + 1
       -- End WHILE loop
    END
-- View the value after the loop
SELECT @ctr
```

```
+--------------------+
|(No column name)    |
+--------------------+
|10                  |
+--------------------+
```

WHILE loop in T-SQL (II)

```
-- Declare ctr as an integer
DECLARE @ctr INT
-- Assign 1 to ctr
SET @ctr = 1
-- Specify the condition of the WHILE loop
WHILE @ctr < 10
    -- Begin the code to execute inside WHILE loop
    BEGIN
       -- Keep incrementing the value of @ctr
       SET @ctr = @ctr + 1

       -- Check if ctr is equal to 4
       IF @ctr = 4
           -- When ctr is equal to 4, the loop will break
           BREAK
       -- End WHILE loop
    END
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Derived tables

Ginger Grant
Instructor

What are Derived tables?

- Query which is treated like a temporary table
- Always contained within the main query
- They are specified in the `FROM` clause
- Can contain intermediate calculations to be used the main query or different joins than in the main query

Derived tables in T-SQL

```
SELECT a.* FROM Kidney a
-- This derived table computes the Average age joined to the actual table
JOIN (SELECT AVG(Age) AS AverageAge
      FROM Kidney) b
ON a.Age = b.AverageAge
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Common Table Expressions

Ginger Grant
Instructor

CTE syntax

```
-- CTE definitions start with the keyword WITH
-- Followed by the CTE names and the columns it contains
WITH CTEName (Col1, Col2)
AS
-- Define the CTE query
(
-- The two columns from the definition above
    SELECT Col1, Col2
    FROM TableName
)
```

CTEs in T-SQL

```
-- Create a CTE to get the Maximum BloodPressure by Age
WITH BloodPressureAge(Age, MaxBloodPressure)
AS
(SELECT Age, MAX(BloodPressure) AS MaxBloodPressure
 FROM Kidney
 GROUP BY Age)

-- Create a query to use the CTE as a table
SELECT a.Age, MIN(a.BloodPressure), b.MaxBloodPressure
FROM Kidney a
-- Join the CTE with the table
JOIN BloodpressureAge b
     ON a.Age = b.Age
GROUP BY a.Age, b.MaxBloodPressure
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Window functions

Ginger Grant
Instructor

	SalesPerson	SalesYear	CurrentQuota	ModifiedDate
1	Bob	2011	28000.00	2011-04-16
2	Bob	2011	7000.00	2011-07-17
3	Bob	2011	91000.00	2011-10-17
4	Mary	2011	367000.00	2011-04-16
5	Mary	2011	556000.00	2011-07-17
6	Mary	2011	502000.00	2011-10-17
7	Bob	2012	140000.00	2012-01-15
8	Bob	2012	70000.00	2012-04-15

Grouping data in T-SQL

```
SELECT SalesPerson, SalesYear,
       CurrentQuota, ModifiedDate
FROM SaleGoal
WHERE SalesYear = 2011
```

```
+------------+----------+---------------+----------------+
|SalesPerson |SalesYear | CurrentQuota  | ModifiedDate   |
+------------+----------+---------------+----------------+
| Bob        | 2011     | 28000.00      | 2011-04-16     |
| Bob        | 2011     |  7000.00      | 2011-07-16     |
| Bob        | 2011     | 91000.00      | 2011-10-16     |
| Mary       | 2011     |367000.00      | 2011-04-16     |
| Mary       | 2011     |556000.00      | 2011-07-16     |
| Mary       | 2011     |502000.00      | 2011-10-16     |
+------------+----------+---------------+----------------+
```

Window syntax in T-SQL

- Create the window with `OVER` clause
- `PARTITION BY` creates the frame
- If you do not include `PARTITION BY` the frame is the entire table
- To arrange the results, use `ORDER BY`
- Allows aggregations to be created at the same time as the window

```
. . .
-- Create a Window data grouping
OVER (PARTITION BY SalesYear ORDER BY SalesYear)
```

Window functions (SUM)

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       SUM(CurrentQuota)
       OVER (PARTITION BY SalesYear) AS YearlyTotal,
       ModifiedDate AS ModDate
FROM SaleGoal
```

```
+------------+----------+------------+------------+----------+
|SalesPerson |SalesYear |CurrentQuota| YearTotal  | ModDate  |
+------------+----------+------------+------------+----------+
|Bob         |2011      |28000.00    |1551000.00  |2011-04-16|
|Bob         |2011      |7000.00     |1551000.00  |2011-07-17|
|Mary        |2011      |367000.00   |1551000.00  |2011-04-16|
|Mary        |2011      |556000.00   |1551000.00  |2011-07-15|
|Bob         |2012      |70000.00    |1551000.00  |2012-01-15|
|Bob         |2012      |154000.00   |1551000.00  |2012-04-16|
|Bob         |2012      |107000.00   |1859000.00  |2012-07-16|
...
+------------+----------+------------+------------+----------+
```

Window functions (COUNT)

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       COUNT(CurrentQuota)
       OVER (PARTITION BY SalesYear) AS QuotaPerYear,
       ModifiedDate AS ModDate
FROM SaleGoal
```

```
+------------+----------+------------+--------------+----------+
|SalesPerson |SalesYear |CurrentQuota|QuotaPerYear  | ModDate  |
+------------+----------+------------+--------------+----------+
|Bob         |2011      |28000.00    |4             |2011-04-16|
|Bob         |2011      |7000.00     |4             |2011-07-17|
|Mary        |2011      |367000.00   |4             |2011-04-16|
|Mary        |2011      |556000.00   |4             |2011-07-15|
|Bob         |2012      |70000.00    |8             |2012-01-15|
|Bob         |2012      |154000.00   |8             |2012-04-15|
|Bob         |2012      |107000.00   |8             |2012-10-16|
...
+------------+----------+------------+------------+----------+
```

- Notice the count starts over for each window in column QuotaPerYear

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Common window functions

Ginger Grant
Instructor

FIRST_VALUE() and LAST_VALUE()

- `FIRST_VALUE()` returns the first value in the window
- `LAST_VALUE()` returns the last value in the window

	SalesPerson	SalesYear	CurrentQuota	ModifiedDate
1	Bob	2011	28000.00	2011-04-16 00:00:00.000
2	Bob	2011	7000.00	2011-07-17 00:00:00.000
3	Bob	2011	91000.00	2011-10-17 00:00:00.000
4	Bob	2012	140000.00	2012-01-15 00:00:00.000
5	Bob	2012	70000.00	2012-04-15 00:00:00.000
6	Bob	2012	154000.00	2012-07-16 00:00:00.000
7	Bob	2012	107000.00	2012-10-16 00:00:00.000
8	Mary	2011	367000.00	2011-04-16 00:00:00.000
9	Mary	2011	556000.00	2011-07-17 00:00:00.000
10	Mary	2011	502000.00	2011-10-17 00:00:00.000

FIRST_VALUE() and LAST_VALUE() in T-SQL

- Note that for FIRST_VALUE and LAST_VALUE the ORDER BY command is required

```
-- Select the columns
SELECT SalesPerson, SalesYear, CurrentQuota,
  -- First value from every window
     FIRST_VALUE(CurrentQuota)
     OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS StartQuota,
  -- Last value from every window
     LAST_VALUE(CurrentQuota)
     OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS EndQuota,
     ModifiedDate as ModDate
FROM SaleGoal
```

Results

```
+------------+----------+------------+----------+----------+----------+
|SalesPerson |SalesYear |CurrentQuota|StartQuota| EndQuota |ModDate   |
+------------+----------+------------+----------+----------+----------+
|Bob         |2011      |28000.00    |28000.00  |91000.00  |2011-04-16|
|Bob         |2011      |7000.00     |28000.00  |91000.00  |2011-07-17|
|Bob         |2011      |91000.00    |28000.00  |91000.00  |2011-10-17|
|Bob         |2012      |140000.00   |140000.00 |107000.00 |2012-01-15|
|Bob         |2012      |70000.00    |140000.00 |107000.00 |2012-04-15|
|Bob         |2012      |154000.00   |140000.00 |107000.00 |2012-07-16|
|Bob         |2012      |107000.00   |140000.00 |107000.00 |2012-10-16|
...
+------------+----------+------------+----------+----------+----------+
```

Getting the next value with LEAD()

- Provides the ability to query the value from the next row
- NextQuota column is created by using `LEAD()`
- Requires the use of `ORDER BY` to order the rows

	SalesPerson	SalesYear	CurrentQuota	NextQuota	ModDate
1	Bob	2011	28000.00	367000.00	2011-04-15
2	Mary	2011	367000.00	556000.00	2011-04-16
3	Mary	2011	556000.00	7000.00	2011-07-15
4	Bob	2011	7000.00	NULL	2011-07-17
5	Bob	2012	70000.00	502000.00	2012-01-15

LEAD() in T-SQL

```
SELECT SalesPerson, SalesYear, CurrentQuota,
-- Create a window function to get the values from the next row
       LEAD(CurrentQuota)
       OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS NextQuota,
       ModifiedDate AS ModDate
FROM SaleGoal
```

```
+------------+----------+------------+------------+----------+
|SalesPerson |SalesYear |CurrentQuota|NextQuota   | ModDate  |
+------------+----------+------------+------------+----------+
|Bob         |2011      |28000.00    |367000.00   |2011-04-15|
|Mary        |2011      |367000.00   |556000.00   |2011-04-16|
|Mary        |2011      |556000.00   |7000.00     |2011-07-15|
|Bob         |2011      |7000.00     |NULL        |2011-07-17|
|Bob         |2012      |70000.00    |502000.00   |2012-01-15|
|Mary        |2012      |502000.00   |154000.00   |2012-01-16|
...
+------------+----------+------------+------------+----------+
```

Getting the previous value with LAG()

- Provides the ability to query the value from the previous row
- PreviousQuota column is created by using `LAG()`
- Requires the use of `ORDER BY` to order the rows

	SalesPerson	SalesYear	CurrentQuota	PreviousQuota	ModDate
1	Bob	2011	28000.00	NULL	2011-04-15
2	Mary	2011	367000.00	28000.00	2011-04-16
3	Mary	2011	556000.00	367000.00	2011-07-15
4	Bob	2011	7000.00	556000.00	2011-07-17
5	Bob	2012	70000.00	NULL	2012-01-15
6	Mary	2012	502000.00	70000.00	2012-01-15

LAG() in T-SQL

```
SELECT SalesPerson, SalesYear, CurrentQuota,
-- Create a window function to get the values from the previous row
       LAG(CurrentQuota)
       OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) AS PreviousQuota,
       ModifiedDate AS ModDate
FROM SaleGoal
```

```
+------------+----------+------------+-------------+----------+
|SalesPerson |SalesYear |CurrentQuota|PreviousQuota |ModDate   |
+------------+----------+------------+-------------+----------+
|Bob         |2011      |28000.00    |NULL         |2011-04-15|
|Mary        |2011      |367000.00   |28000.00     |2011-04-16|
|Mary        |2011      |556000.00   |367000.00    |2011-07-15|
|Bob         |2011      |7000.00.00  |556000.00    |2011-07-17|
|Bob         |2012      |7000.00     |NULL         |2012-01-15|
|Mary        |2012      |502000.00   |7000.00      |2012-01-16|
...
+------------+----------+------------+-------------+----------+
```

INTERMEDIATE T-SQL

Let's practice !

INTERMEDIATE T-SQL

Increasing window complexity

Ginger Grant
Instructor

Reviewing aggregations

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       SUM(CurrentQuota)
       OVER (PARTITION BY SalesYear) AS YearlyTotal,
       ModifiedDate as ModDate
FROM SaleGoal
```

```
+------------+----------+------------+------------+----------+
|SalesPerson |SalesYear |CurrentQuota|YearTotal   | ModDate  |
+------------+----------+------------+------------+----------+
|Bob         |2011      |28000.00    |1551000.00  |2011-04-16|
|Bob         |2011      |7000.00     |1551000.00  |2011-07-17|
|Bob         |2011      |91000.00    |1551000.00  |2011-10-17|
|Mary        |2011      |140000.00   |1551000.00  |2012-04-15|
|Mary        |2011      |70000.00    |1551000.00  |2012-07-15|
|Mary        |2011      |154000.00   |1551000.00  |2012-01-15|
|Mary        |2012      |107000.00   |1859000.00  |2012-01-16|
...
+------------+----------+------------+------------+----------+
```

Adding ORDER BY to an aggregation

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       SUM(CurrentQuota)
       OVER (PARTITION BY SalesYear ORDER BY SalesPerson) AS YearlyTotal,
       ModifiedDate as ModDate
FROM SaleGoal
```

```
+------------+----------+------------+------------+----------+
|SalesPerson |SalesYear |CurrentQuota|YearTotal   | ModDate  |
+------------+----------+------------+------------+----------+
|Bob         |2011      |28000.00    |35000.00    |2011-04-16|
|Bob         |2011      |7000.00     |35000.00    |2011-07-17|
|Mary        |2011      |367000.00   |958000.00   |2011-10-17|
|Mary        |2011      |556000.00   |958000.00   |2012-04-15|
|Bob         |2012      |70000.00    |401000.00   |2012-07-15|
|Bob         |2012      |154000.00   |401000.00   |2012-10-16|
...
+------------+----------+------------+------------+----------+
```

Creating a running total with ORDER BY

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       SUM(CurrentQuota)
       OVER (PARTITION BY SalesYear ORDER BY ModifiedDate) as RunningTotal,
       ModifiedDate as ModDate
FROM SaleGoal
```

```
+------------+----------+------------+------------+----------+
|SalesPerson |SalesYear |CurrentQuota|YearTotal   | ModDate  |
+------------+----------+------------+------------+----------+
|Bob         |2011      |28000.00    |28000.00    |2011-04-16|
|Mary        |2011      |367000.00   |395000.00   |2011-07-17|
|Mary        |2011      |556000.00   |951000.00   |2011-10-17|
|Bob         |2011      |7000.00     |958000.00   |2012-04-15|
|Bob         |2012      |70000.00    |70000.00    |2012-01-15|
|Mary        |2012      |502000.00   |572000.00   |2012-01-16|
...
+------------+----------+------------+------------+----------+
```

Adding row numbers

- `ROW_NUMBER()` sequentially numbers the rows in the window
- `ORDER BY` is required when using `ROW_NUMBER()`

	SalesPerson	SalesYear	CurrentQuota	QuotabySalesPerson
1	Bob	2011	28000.00	1
2	Bob	2011	7000.00	2
3	Bob	2012	70000.00	3
4	Bob	2012	154000.00	4
5	Bob	2012	70000.00	5
6	Bob	2012	107000.00	6
7	Bob	2013	91000.00	7
8	Mary	2011	367000.00	1
9	Mary	2011	556000.00	2

Adding row numbers in T-SQL

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       ROW_NUMBER()
       OVER (PARTITION BY SalesPerson ORDER BY SalesYear) AS QuotabySalesPerson
FROM SaleGoal
```

```
+------------+----------+------------+------------------+
|SalesPerson |SalesYear |CurrentQuota|QuotabySalesPerson|
+------------+----------+------------+------------------+
|Bob         |2011      |28000.00    |1                 |
|Bob         |2011      |7000.00     |2                 |
|Bob         |2011      |70000.00    |3                 |
|Bob         |2011      |154000.00   |4                 |
|Bob         |2012      |70000.00    |5                 |
|Bob         |2012      |107000.00   |6                 |
|Bob         |2012      |91000.00    |7                 |
|Mary        |2011      |367000.00   |1                 |
...
+------------+----------+------------+------------------+
```

INTERMEDIATE T-SQL

Let's practice!

INTERMEDIATE T-SQL

Using windows for calculating statistics

Ginger Grant
Instructor

Calculating the standard deviation

- Calculate standard deviation either for the entire table or for each window
- `STDEV()` calculates the standard deviation

Calculating the standard deviation for the entire table

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       STDEV(CurrentQuota)
       OVER () AS StandardDev,
       ModifiedDate  AS ModDate
FROM SaleGoal
```

```
+------------+----------+------------+-----------------+----------+
|SalesPerson |SalesYear |CurrentQuota|StandardDev      | ModDate  |
+------------+----------+------------+-----------------+----------+
|Bob         |2011      |28000.00    |267841.370964233 |2011-04-16|
|Bob         |2011      |7000.00     |267841.370964233 |2011-07-17|
|Bob         |2011      |91000.00    |267841.370964233 |2011-10-17|
|Bob         |2012      |140000.00   |267841.370964233 |2012-01-15|
|Bob         |2012      |70000.00    |267841.370964233 |2012-04-15|
...
+------------+----------+------------+-----------------+----------+
```

Calculating the standard deviation for each partition

```
SELECT SalesPerson, SalesYear, CurrentQuota,
       STDEV(CurrentQuota)
       OVER (PARTITION BY SalesYear ORDER BY SalesYear) AS StDev,
       ModifiedDate AS ModDate
FROM SaleGoal
```

```
+------------+----------+------------+-------------+----------+
|SalesPerson |SalesYear |CurrentQuota|StDev        | ModDate  |
+------------+----------+------------+-------------+----------+
|Bob         |2011      |28000.00    |267841.54080 |2011-04-16|
|Bob         |2011      |7000.00     |267841.54080 |2011-07-17|
|Mary        |2011      |91000.00    |267841.54080 |2011-04-16|
|Mary        |2011      |140000.00   |267841.54080 |2011-07-15|
|Bob         |2012      |70000.00    |246538.86248 |2012-01-15|
|Bob         |2012      |154000.00   |246538.86248 |2012-04-15|
|Bob         |2012      |107000.00   |246538.86248 |2012-07-16|
...
+------------+----------+------------+-------------+----------+
```

Calculating the mode

- Mode is the value which appears the most often in your data
- To calculate mode:
  - Create a CTE containing an ordered count of values using ROW_NUMBER
  - Write a query using the CTE to pick the value with the highest row number

Calculating the mode in T-SQL (I)

```
WITH QuotaCount AS (
SELECT SalesPerson, SalesYear, CurrentQuota,
       ROW_NUMBER()
       OVER (PARTITION BY CurrentQuota ORDER BY CurrentQuota) AS QuotaList
FROM SaleGoal
)
SELECT * FROM QuotaCount
```

```
+------------+----------+------------+-------------+
|SalesPerson |SalesYear |CurrentQuota|QuotaList    |
+------------+----------+------------+-------------+
|Bob         |2011      |7000.00     |1            |
|Bob         |2011      |28000.00    |1            |
|Bob         |2011      |700000.00   |1            |
|Bob         |2012      |700000.00   |2            |
|Mary        |2012      |73000.00    |1            |
...
+------------+----------+------------+--------------+
```

- Notice there are two values for 70,000.00

Calculating the mode in T-SQL (II)

```
WITH QuotaCount AS (
SELECT SalesPerson, SalesYear, CurrentQuota,
       ROW_NUMBER()
       OVER (PARTITION BY CurrentQuota ORDER BY CurrentQuota) AS QuotaList
FROM SaleGoal
)

SELECT CurrentQuota, QuotaList AS Mode
FROM QuotaCount
WHERE QuotaList IN (SELECT  MAX(QuotaList) FROM QuotaCount)

+------------+----------+
|CurrentQuota|Mode      |
+------------+----------+
|70000.00    |2         |
+------------+----------+
```

INTERMEDIATE T-SQL

Let's practice!