Summarizing Data

- Examining data through aggregations in SQL Server, a common first step in data exploration.
- Clean missing data and categorize data into bins with CASE statements.

Creating aggregations

Incidents table:

```
 IncidentDateTime    City    IncidentState    Country    Shape    DurationSeconds    Comments
2005-10-31 18:00:00    poughkeepsie    ny    us    light    37800    Several bright lights moving erratically for extended periods of time&#44 including growing in size and brightness&#44 and disappearing.
2005-10-31 18:30:00    linwood    nj    us    light    5    VERY bright apparent meteor over Southern New Jersey on 10/31 and 11/1 2005
2005-10-31 19:00:00    clarksville    md    us    other    5    White ball shaped bright object whizzing across the holloween sky
2005-10-31 19:00:00    newark    de    us    light    45    Very fast&#44 brillant bluish/white light travelling horizontally without dimming or falling
2005-10-31 19:00:00    scottsdale    az    us    triangle    600    Gilbert
2005-10-31 19:15:00    chambersburg    pa    us    fireball    20    Fireball
2005-10-31 19:30:00    scottsdale    az    us    triangle    120    On Halloween Night 2005&#44 3 Lights in shape of a triangle moving over Scottsdale&#44 AZ.
2005-10-31 19:40:00    bristol    tn    us    changing    90    TENNESSEE MUFON REPORT:  Slowly falling&#44 plasma-like object.
2005-10-31 20:00:00    fairfield    ca    us    triangle    20    Triangular craft hovering over highway
2005-10-31 20:30:00    greensboro    md    us    fireball    60    At about 8:30 EST  on 10/31/05 Something lit up the sky and fell towards Earth as a fireball with a long firery trail behind it
2005-10-31 21:00:00    augusta springs    va    us    unknown    60    We saw an unusual bright blue flash in the sky and then noticed a trail in the sky that quickly faded away.
```

```
SELECT AVG(DurationSeconds) AS Average,
MIN(DurationSeconds) AS Minimum,
```

```
MAX(DurationSeconds) AS Maximum
FROM Incidents
```

```
 Average     Minimum     Maximum
5592.875782703282     0.07999999821186066 10526400
```

Creating grouped aggregations

```
-- Calculate the aggregations by Shape
SELECT Shape,
AVG(DurationSeconds) AS Average,
MIN(DurationSeconds) AS Minimum,
MAX(DurationSeconds) AS Maximum
FROM Incidents
GROUP BY Shape
```

```
 Shape     Average     Minimum     Maximum
unknown     2053.769722814499     1     259200
disk     1445.3135593220338     1     253800
sphere     903.3034398034398     1     46800
circle     29999.86076672103     0.07999999821186066     10526400
cone     1558.9565217391305     1     18000
formation     459.11649483749545     1     5400
chevron     1100.59375     2     21600
rectangle     969.6132075471698     4     28800
other     2620.983905579399     1     345600
light     2050.7791762037373     1     604800
changing     3191.6744186046512     2     172800
null     18368.96402877698     1     2419200
egg     558.9561403508771     1.5     7200
triangle     886.8864696734059     1     172800
NULL     1500.6666666666667     2     3600
oval     23440.539792387543     1     6312000
teardrop     3501.685185185185     2     172800
```

```
diamond    1265.6971153846155    1    37800
fireball    464.2863805970149    0.5    14400
crescent    10    10    10
cylinder    795.2417582417582    3    37800
flash    1068.8019607844305    0.30000001192092896    25200
cross    848.1333333333333    2    7200
cigar    1736.4055555555556    1    172800
```

Removing missing values

-- Return the specified columns
SELECT IncidentDateTime, IncidentState
From Incidents
-- Exclude all the missing values from IncidentState
WHERE IncidentState IS NOT NULL

part of results

```
 IncidentDateTime    IncidentState
2005-10-31 18:00:00    ny
2005-10-31 18:30:00    nj
2005-10-31 19:00:00    md
```

Imputing missing values (I)

Replace the missing values with another value instead of omitting them.
Here we replace all the missing values in the Shape column using the word 'Saucer':

SELECT Shape, ISNULL(Shape, 'Saucer') AS Shape2
FROM Incidents

You can also use ISNULL() to replace values from a different column instead of a specified word.

Write a T-SQL query which only returns rows where IncidentState is missing.
Replace all the missing values in the IncidentState column with the values in the City column and name this new column Location.

```
-- Check the IncidentState column for missing values and replace them with the City column
SELECT IncidentState, ISNULL(IncidentState, City) AS Location
FROM Incidents
-- Filter to only return missing values from IncidentState
WHERE IncidentState IS NULL
```

```
 IncidentState    Location
null    aust
null    ivan
null    tuni
```

Imputing missing values (II)

Replace missing values in one column with another and check the replacement column to make sure it doesn't have any missing values. To do that you need to use the COALESCE statement.

```
SELECT Shape, City, COALESCE(Shape, City, 'Unknown') as NewShape
FROM Incidents
```

```
 +----------------+-----------+-------------+
| Shape          |  City     |  NewShape   |
+----------------+-----------+-------------+
| NULL           | Orb       | Orb         |
| Triangle       | Toledo    | Triangle    |
| NULL           | NULL      | Unknown     |
+----------------+-----------+-------------+
```

Replace missing values in Country with the first non-missing value from IncidentState or City, in that order. Name the new column Location.

```
-- Replace missing values
SELECT Country, COALESCE(Country, IncidentState, City) AS Location
FROM Incidents
WHERE Country IS NULL
```

Country Location

```
 null    australia
null    ivanka pri dunaji (slovakia)
null    tunisia (in-flight; over ocean)
null    dehradun (uttar pradesh) (india)
null    erode (india)
null    new delhi (india)
null    chawton&#44 hampshire (uk/england)
null    setif (algeria)
```

Using CASE statements

SELECT Country,
CASE WHEN Country = 'us' THEN 'USA'
ELSE 'International'
END AS SourceCountry
FROM Incidents

```
 Country    SourceCountry
us    USA
us    USA
us    USA
```

Creating several groups with CASE

SELECT DurationSeconds,

```
        CASE WHEN (DurationSeconds <= 120) THEN 1

        WHEN (DurationSeconds > 120 AND DurationSeconds <= 600) THEN 2

        WHEN (DurationSeconds > 601 AND DurationSeconds <= 1200) THEN 3

        WHEN (DurationSeconds > 1201 AND DurationSeconds <= 5000) THEN 4

        ELSE 5
        END AS SecondGroup
```

FROM Incidents

```
 DurationSeconds    SecondGroup
37800    5
5    1
5    1
45    1
600    2
```

Math Functions

Calculating the total

Shipments table:

```
 MixID    MixDesc    Plant    ShipDate    DeliveryWeight    Cost    Quantity    OrderDate    WeightValue
100900    ABC SLURRY    1    2017-09-28 08:50:26    3848.800048828125    11.2480    27.01099967956543    2017-
09-27 06:50:26    2.1969099044799805
100900    ABC SLURRY    2    2016-06-24 10:48:19    3848.800048828125    10.3976    27.01099967956543    2016-
06-23 08:48:19    2.1969099044799805
105900    1/2 SACK ABC SLURRY    1    2016-06-16 15:05:09    3855.800048828125    13.2444    27.008899688720703
2016-06-16 09:05:09    2.198899984359741
165899    6.0 SACK GROUT-NO ADMIX    2    2016-06-24 10:58:49    3835.905029296875    36.0652
27.014999389648438    2016-06-23 13:58:49    2.1932199001312256
```

```
SELECT MixDesc, SUM(Quantity) AS Total
FROM Shipments
GROUP BY MixDesc
```

```
 MixDesc    Total
".40W/C 24"" SPD 5000PSI W/FIBER"    26.533899307250977
"1.0 SACK CLSM 3/8"""    114.50430229306221
"2500 PSI 1/2"" NATURAL"    27.127500534057617
```

Counting the number of rows

Create a query that returns the number of rows for each type of MixDesc.

```
-- Count the number of rows by MixDesc
SELECT MixDesc, COUNT(*) FROM Shipments GROUP BY MixDesc
```

```
 MixDesc
".40W/C 24"" SPD 5000PSI W/FIBER"    1
"1.0 SACK CLSM 3/8"""    6
"2500 PSI 1/2"" NATURAL"    1
```

Which date function should you use?

Suppose you want to calculate the number of years between two different dates, DateOne and DateTwo. Which SQL statement would you use to perform that calculation?

SELECT DATEADD(YYY, DateOne, DateTwo)

SELECT DATEDIFF(DateOne, MM, DateTwo)

SELECT DATEDIFF(YYYY, DateOne, DateTwo)

SELECT DATEDIFF(DateOne, DateTwo, YYYY)

Answer 3

Counting the number of days between dates

```
-- Return the difference in OrderDate and ShipDate
SELECT OrderDate, ShipDate,
DATEDIFF(DD, OrderDate, ShipDate) AS Duration
FROM Shipments
```

```
 OrderDate    ShipDate    Duration
2017-09-27 06:50:26    2017-09-28 08:50:26    1
2016-06-23 08:48:19    2016-06-24 10:48:19    1
2016-06-16 09:05:09    2016-06-16 15:05:09    0
```

Adding days to a date

Calculate the approximate delivery date of an order based on ShipDate.

Write a query that returns the approximate delivery date as five days after the ShipDate.

```
-- Return the DeliveryDate as 5 days after the ShipDate
SELECT OrderDate,
DATEADD(DD, 5, ShipDate) AS DeliveryDate
FROM Shipments
```

```
 OrderDate    DeliveryDate
2017-09-27 06:50:26    2017-10-03 08:50:26
2016-06-23 08:48:19    2016-06-29 10:48:19
2016-06-16 09:05:09    2016-06-21 15:05:09
```

Rounding numbers

Round the cost to the nearest dollar.

```
-- Round Cost to the nearest dollar
SELECT Cost,
ROUND(Cost, 0) AS RoundedCost
FROM Shipments
```

```
 Cost    RoundedCost
11.2480    11.0000
10.3976    10.0000
13.2444    13.0000
```

Truncating numbers

Since rounding can sometimes be misleading, i.e., 16.8 becomes 17 while 16.4 remains 16, you may want to truncate the values after the decimal instead of rounding them. When you truncate the numbers, both 16.8 and 16.4 remain 16.

Write a SQL query to truncate the values in the Cost column to the nearest whole number.

```
-- Truncate cost to whole number
SELECT Cost,
ROUND(Cost, 0, 1) AS TruncateCost
FROM Shipments
```

```
 Cost    TruncateCost
11.2480    11.0000
10.3976    10.0000
13.2444    13.0000
```

Calculating the absolute value

```
-- Return the absolute value of DeliveryWeight
SELECT DeliveryWeight,
ABS(DeliveryWeight) AS AbsoluteValue
FROM Shipments
```

```
 DeliveryWeight    AbsoluteValue
3848.800048828125    3848.800048828125
3848.800048828125    3848.800048828125
3855.800048828125    3855.800048828125
```

Calculating squares and square roots

SELECT WeightValue,
SQUARE(WeightValue) AS WeightSquare,
SQRT(WeightValue) AS WeightSqrt
FROM Shipments

```
 WeightValue    WeightSquare    WeightSqrt
2.1969099044799805    4.826413128402237    1.4821976603948546
2.1969099044799805    4.826413128402237    1.4821976603948546
2.198899984359741    4.83516114121727    1.482868835858297
```

Processing Data in SQL Server

- Create variables and write while loops to process data.
- Write complex queries by using derived tables and common table expressions.

Creating and using variables

In T-SQL, to create a variable you use the DECLARE statement. The variables must have an at sign (@) as their first character. Like most things in T-SQL, variables are not case sensitive. To assign a value to a variable, you can either use the keyword SET or a SELECT statement followed by an equal sign and a value.

artery table:

```
 RecordID    CoronaryArteryDisease
1    No
```

```
2     No
3     No
4     No
```

kidney table:

```
 Age     BloodPressure     SpecificGravity     Albumin     Sugar     RedBloodCells     PusCell     PusCellClumps
Bacteria     BloodGlucoseRandom     BloodUrea     SerumCreatinine     Sodium     Potassium     Hemoglobin
PackedCellVolume     WhiteBloodCellCount     RedBloodCellCount     Hypertension     DiabetesMellitus
CoronaryArteryDisease     Appetite     PedalEdema     Anemia     Class     RecordID
48     80     1.0199999809265137     1     0     null     normal     notpresent     notpresent     121     36
1.2000000476837158     null     null     15.399999618530273     44     7800     5.199999809265137     yes     yes
null     good     no     no     ckd     1
7     50     1.0199999809265137     4     0     null     normal     notpresent     notpresent     0     18
0.800000011920929     null     null     11.300000190734863     38     6000     null     null     null     null     good
no     no     ckd     1
62     80     1.0099999904632568     2     3     normal     normal     notpresent     notpresent     423     53
1.7999999523162842     null     null     9.600000381469727     31     7500     null     null     yes     null     poor
no     yes     ckd     2
48     70     1.0049999952316284     4     0     normal     abnormal     present     notpresent     117     56
3.799999952316284     111     2.5     11.199999809265137     32     6700     3.9000000953674316     yes     null
null     poor     yes     yes     ckd     3
51     80     1.0099999904632568     2     0     normal     normal     notpresent     notpresent     106     26
1.399999976158142     null     null     11.600000381469727     35     7300     4.599999904632568     null     null
null     good     no     no     ckd     4
```

```
-- Create the variable
DECLARE @counter INT

-- Assign a value to the variable
SET @counter = 20

-- Print the variable
SELECT @counter
```

Creating a WHILE loop

Write a WHILE loop that increments counter by 1 until counter is less than 30.

```
DECLARE @counter INT SET @counter = 20

WHILE @counter < 30

BEGIN SELECT @counter = @counter + 1 END

SELECT @counter
```

30

Queries with derived tables (I)

```
SELECT a.RecordId, a.Age, a.BloodGlucoseRandom,
-- Maximum Glucose value from the derived table
b.MaxGlucose
FROM Kidney a
-- Derived table
JOIN (SELECT Age, MAX(BloodGlucoseRandom) AS MaxGlucose FROM Kidney GROUP BY Age) b
-- Join on Age
ON a.Age = b.Age
```

```
 RecordId    Age    BloodGlucoseRandom    MaxGlucose
29    0    93    220
71    0    129    220
109    0    0    220
```

Queries with derived tables (II)

```
SELECT *
```

```
FROM Kidney a
-- JOIN and create the derived table
JOIN (SELECT Age, MAX(BloodPressure) AS MaxValue FROM Kidney GROUP BY Age) b
-- JOIN on BloodPressure and MaxValue
ON a.BloodPressure = b.MaxValue
-- Join on Age
AND a.Age = b.Age
```

```
 Age    BloodPressure    SpecificGravity    Albumin    Sugar    RedBloodCells    PusCell    PusCellClumps
Bacteria    BloodGlucoseRandom    BloodUrea    SerumCreatinine    Sodium    Potassium    Hemoglobin
PackedCellVolume    WhiteBloodCellCount    RedBloodCellCount    Hypertension    DiabetesMellitus
CoronaryArteryDisease    Appetite    PedalEdema    Anemia    Class    RecordID    Age    MaxValue
90    90    1.024999976158142    1    0    null    normal    notpresent    notpresent    139    89    3    140
4.099999904632568    12    37    7900    3.9000000953674316    yes    yes    null    good    no    no    ckd
187    90    90
83    70    1.0199999809265137    3    0    normal    normal    notpresent    notpresent    102    60
2.5999999046325684    115    5.699999809265137    8.699999809265137    26    12800    3.0999999046325684    yes
null    null    poor    no    yes    ckd    159    83    70
82    80    1.0099999904632568    2    2    normal    null    notpresent    notpresent    140    70
3.4000000953674316    136    4.199999809265137    13    40    9800    4.199999809265137    yes    yes    null
good    no    no    ckd    38    82    80
81    60    null    0    0    null    null    notpresent    notpresent    148    39    2.0999999046325684
147    4.199999809265137    10.899999618530273    35    9400    2.4000000953674316    yes    yes    yes    poor
yes    no    ckd    150    81    60
80    80    1.024999976158142    0    0    normal    normal    notpresent    notpresent    119    46
0.699999988079071    141    4.900000095367432    13.899999618530273    49    5100    5    null    null    null
good    no    no    notckd    362    80    80
```

CTE syntax

Select all the T-SQL keywords used to create a Common table expression.

1. DEALLOCATE
2. OPEN
3. AS

4. WITH
5. CTE

Answer 3 and 4.

Creating CTEs (I)

A Common table expression or CTE is used to create a table that can later be used with a query. To create a CTE, you will always use the WITH keyword followed by the CTE name and the name of the columns the CTE contains. The CTE will also include the definition of the table enclosed within the AS().

Use a CTE to return all the ages with the maximum BloodGlucoseRandom in the table.

Create a CTE BloodGlucoseRandom that returns one column (MaxGlucose) which contains the maximum BloodGlucoseRandom in the table. Join the CTE to the main table (Kidney) on BloodGlucoseRandom and MaxGlucose.

Common Table Expressions or CTE's for short are used within SQL Server to simplify complex joins and subqueries, and to provide a means to query hierarchical data such as an organizational chart. **What the purpose of creating CTE if we can use temporary tables etc.?**

```
-- Create the CTE
WITH BloodGlucoseRandom (MaxGlucose)
AS (SELECT MAX(BloodGlucoseRandom) AS MaxGlucose FROM Kidney)

SELECT a.Age, b.MaxGlucose
FROM Kidney a
-- Join the CTE
JOIN BloodGlucoseRandom b
ON a.BloodGlucoseRandom = b.MaxGlucose
```

```
 Age    MaxGlucose
50    490
60    490
```

Creating CTEs (II)

Use a CTE to return all the information regarding the patient(s) with the maximum BloodPressure. Create a CTE BloodPressure that returns one column (MaxBloodPressure) which contains the maximum BloodPressure in the table. Join this CTE (using an alias b) to the main table (Kidney) to return information about patients with the maximum BloodPressure.

```
-- Create the CTE
WITH BloodPressure (MaxBloodPressure)
AS (SELECT MAX(BloodPressure) AS MaxBloodPressure FROM Kidney)

SELECT *
FROM Kidney a
-- Join the CTE
JOIN BloodPressure b
ON a.BloodPressure = b.MaxBloodPressure
```

```
 Age    BloodPressure    SpecificGravity    Albumin    Sugar    RedBloodCells    PusCell    PusCellClumps
Bacteria    BloodGlucoseRandom    BloodUrea    SerumCreatinine    Sodium    Potassium    Hemoglobin
PackedCellVolume    WhiteBloodCellCount    RedBloodCellCount    Hypertension    DiabetesMellitus
CoronaryArteryDisease    Appetite    PedalEdema    Anemia    Class    RecordID    MaxBloodPressure
56    180    null    0    4    null    abnormal    notpresent    notpresent    298    24    1.2000000476837158
139    3.9000000953674316    11.199999809265137    32    10400    4.199999809265137    yes    yes    null
poor    yes    no    ckd    96    180
```

Window Functions

Partitions of data and window functions to calculate several summary stats and see how easy it is to create running totals and compute the mode of numeric columns.

Window functions are similar to aggregate functions, but there is **one important difference**. When we use aggregate functions with the GROUP BY clause, we 'lose' the individual rows. We cannot mix attributes from an individual row with the results of an aggregate function; the function is performed on the rows as an entire group. This is not the case when we use window functions: we can generate a result set with some attributes of an individual row together with the results of the window function.

Window functions with aggregations (I)

- Using OVER() to create a window for the entire table.

- To create partitions using a specific column, use OVER() along with PARTITION BY.

Write a T-SQL query that returns the sum of OrderPrice by creating partitions for each TerritoryName.

```
 order table:
OrderID    OrderDate    TerritoryName    YearOrdered    ExpectedDeliveryDate    CustomerPurchaseOrderNumber
PickingCompletedWhen    OrderPrice
40646    2015-01-01 01:00:00    Germany    2015    2015-01-02 00:00:00    15990    2015-01-01 11:00:00    13
40648    2015-01-01 02:00:00    Southeast    2015    2015-01-02 00:00:00    11073    2015-01-01 11:00:00    45
40662    2015-01-01 03:00:00    Southeast    2015    2015-01-02 00:00:00    12521    2015-01-01 12:00:00    20
40664    2015-01-01 04:00:00    Southeast    2015    2015-01-02 00:00:00    12192    2015-01-01 11:00:00    230
40671    2015-01-01 05:00:00    Central    2015    2015-01-02 00:00:00    13141    2015-01-01 11:00:00    13
```

```
SELECT OrderID, TerritoryName,
-- Total price using the partition
SUM(OrderPrice)
-- Create the window and partitions
OVER(PARTITION BY TerritoryName) AS TotalPrice
FROM Orders
```

```
 OrderID    TerritoryName    TotalPrice
43706    Australia    1469
43722    Australia    1469
43729    Australia    1469
```

Window functions with aggregations (II)

```
SELECT OrderID, TerritoryName,
-- Number of rows per partition
COUNT(*)
-- Create the window and partitions
OVER(PARTITION BY TerritoryName) AS TotalOrders
FROM Orders
```

```
 OrderID    TerritoryName    TotalOrders
43706    Australia    13
43722    Australia    13
43729    Australia    13
```

Do you know window functions?

Which of the following statements is **incorrect** regarding window queries?

Answer the question

- The window functions LEAD(), LAG(), FIRST_VALUE(), and LAST_VALUE() require ORDER BY in the OVER() clause.
- The standard aggregations like SUM(), AVG(), and COUNT() require ORDER BY in the OVER() clause.
- If the query contains OVER() and PARTITION BY the table is partitioned.
- **The first row in a window where the LAG() function is used is NULL.**
  Answer 2

First value in a window

Figure out the first or last OrderDate in each territory with the window functions FIRST_VALUE() and LAST_VALUE(), respectively.

```
SELECT TerritoryName, OrderDate,
-- Select the first value in each partition
FIRST_VALUE(OrderDate)
-- Create the partitions and arrange the rows
OVER(PARTITION BY TerritoryName ORDER BY OrderDate) AS FirstOrder
FROM Orders
```

```
 TerritoryName    OrderDate    FirstOrder
Australia    2015-02-23 09:00:00    2015-02-23 09:00:00
Australia    2015-02-23 11:00:00    2015-02-23 09:00:00
Australia    2015-02-23 12:00:00    2015-02-23 09:00:00
```

Previous and next values

Shift the values in a column by one row up or down? Use the exact same steps as in the previous exercise but with two new functions, LEAD(), for the next value, and LAG(), for the previous value.

```
SELECT TerritoryName, OrderDate,
-- Previous OrderDate in the window
LAG(OrderDate)
-- Create the partitions and arrange the rows
OVER(PARTITION BY TerritoryName ORDER BY OrderDate) AS PreviousOrder,
-- Next OrderDate in the window
LEAD(OrderDate)
-- Create the partitions and arrange the rows
OVER(PARTITION BY TerritoryName ORDER BY OrderDate) AS NextOrder
FROM Orders
```

```
 TerritoryName    OrderDate    PreviousOrder    NextOrder
Australia    2015-02-23 09:00:00    null    2015-02-23 11:00:00
Australia    2015-02-23 11:00:00    2015-02-23 09:00:00    2015-02-23 12:00:00
Australia    2015-02-23 12:00:00    2015-02-23 11:00:00    2015-04-23 02:00:00
```

Creating running totals

You usually don't have to use ORDER BY when using aggregations, but if you want to create running totals, you should arrange your rows! In this exercise, you will create a running total of OrderPrice. **What is the underlying reason?**

Create the window, partition by TerritoryName and order by OrderDate to calculate a running total of OrderPrice.

```
SELECT TerritoryName, OrderDate,
-- Create a running total
SUM(OrderPrice)
-- Create the partitions and arrange the rows
OVER(PARTITION BY TerritoryName ORDER BY OrderDate) AS TerritoryTotal
FROM Orders
```

```
 TerritoryName    OrderDate    TerritoryTotal
Australia    2015-02-23 09:00:00    48
```

```
Australia    2015-02-23 11:00:00    83
Australia    2015-02-23 12:00:00    313
```

Assigning row numbers

Records in T-SQL are inherently unordered. Although in certain situations, you may want to assign row numbers for reference. In this exercise, you will do just that.

Write a T-SQL query that assigns row numbers to all records partitioned by TerritoryName and ordered by OrderDate.

```
SELECT TerritoryName, OrderDate,
-- Assign a row number
ROW_NUMBER()
-- Create the partitions and arrange the rows
OVER(PARTITION BY TerritoryName ORDER BY OrderDate) AS OrderCount
FROM Orders
```

```
 TerritoryName    OrderDate    OrderCount
Australia    2015-02-23 09:00:00    1
Australia    2015-02-23 11:00:00    2
Australia    2015-02-23 12:00:00    3
```

Calculating standard deviation

Calculate the running standard deviation, similar to the running total you calculated in the previous lesson.
Again, why we need ORDER BY in the code belwo?

```
SELECT OrderDate, TerritoryName,
-- Calculate the standard deviation
STDEV(OrderPrice)
OVER(PARTITION BY TerritoryName ORDER BY OrderDate) AS StdDevPrice
FROM Orders
```

```
 OrderDate    TerritoryName    StdDevPrice
```

```
2015-02-23 09:00:00    Australia    null
2015-02-23 11:00:00    Australia    9.192388155425117
2015-02-23 12:00:00    Australia    109.02446208687908
```

Calculating mode (I)

Unfortunately, there is no function to calculate the mode, the most recurring value in a column. To calculate the mode:

First, 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
In this exercise, you will write the CTE needed to calculate the mode of OrderPrice.

```
-- Create a CTE Called ModePrice which contains two columns
WITH ModePrice (OrderPrice, UnitPriceFrequency)
AS
(
SELECT OrderPrice,
ROW_NUMBER()
OVER(PARTITION BY OrderPrice ORDER BY OrderPrice) AS UnitPriceFrequency
FROM Orders
)

-- Return all of the rows in the CTE
SELECT *
FROM ModePrice
```

```
 OrderPrice    UnitPriceFrequency
3.5    1
3.5    2
3.700000047683716    1
```

Calculating mode (II)

In the last exercise, you created a CTE which assigned row numbers to each unique value in OrderPrice. All you need to do now is to find the OrderPrice with the highest row number.

```
-- CTE from the previous exercise
WITH ModePrice (OrderPrice, UnitPriceFrequency)
AS
(
SELECT OrderPrice,
ROW_NUMBER()
OVER (PARTITION BY OrderPrice ORDER BY OrderPrice) AS UnitPriceFrequency
FROM Orders
)

-- Calculate the mode
SELECT OrderPrice AS Mode
FROM ModePrice
-- The WHERE clause should only return the maximum value of UnitPriceFrequency
WHERE UnitPriceFrequency IN (SELECT MAX(UnitPriceFrequency) From ModePrice)
```

Mode 32