Instructor Notes

Let's look at this again. We have broken down the syntax to explain LAG and LEAD functions separately.

LAG function

Purpose

It returns the value from a previous row to the current row in the table.

Step 1:

Let's first look at the **inner query** and see what this creates.

```
SELECT     account_id, SUM(standard_qty) AS standard_sum
FROM       orders
GROUP BY   1
```

What you see after running this SQL code:

1. The query sums the standard_qty amounts for each account_id to give the standard paper each account has purchased over all time. E.g., account_id 2951 has purchased 8181 units of standard paper.
2. Notice that the results are not ordered by account_id or standard_qty.

account_id	standard_sum
2951	8181
2651	4231
3401	116
2941	790
1501	7941
1351	3174
1721	1176
1191	6304

Step 2:

We start building the **outer query**, and name the inner query as `sub`.

```
SELECT account_id, standard_sum
FROM   (
        SELECT   account_id, SUM(standard_qty) AS standard_sum
        FROM     orders
        GROUP BY 1
       ) sub
```

This still returns the same table you see above, which is also shown below.

account_id	standard_sum
2951	8181
2651	4231
3401	116
2941	790
1501	7941

Step 3 (Part A):

We add the Window Function `OVER (ORDER BY standard_sum)` in the outer query that will create a result set in ascending order based on the *standard_sum* column.

```
SELECT account_id,
       standard_sum,
       LAG(standard_sum) OVER (ORDER BY standard_sum) AS lag
FROM   (
        SELECT   account_id, SUM(standard_qty) AS standard_sum
        FROM     orders
        GROUP BY 1
       ) sub
```

This ordered column will set us up for the other part of the Window Function (see below).

Step 3 (Part B):

The `LAG` function creates a new column called *lag* as part of the **outer query**: `LAG(standard_sum) OVER (ORDER BY standard_sum) AS lag`. This new column named *lag* uses the values from the ordered *standard_sum* (Part A within Step 3).

```
SELECT account_id,
       standard_sum,
       LAG(standard_sum) OVER (ORDER BY standard_sum) AS lag
FROM   (
        SELECT   account_id,
                 SUM(standard_qty) AS standard_sum
        FROM     demo.orders
        GROUP BY 1
       ) sub
```

Each row's value in *lag* is pulled from the previous row. E.g., for account_id 1901, the value in *lag* will come from the previous row. However, since there is no previous row to pull from, the value in *lag* for account_id 1901 will be NULL. For account_id 3371, the value in *lag* will be pulled from the previous row (i.e., account_id 1901), which will be 0. This goes on for each row in the table.

What you see after running this SQL code:

account_id	standard_sum	lag
1901	0	
3371	79	0
1961	102	79
3401	116	102
3741	117	116
4321	123	117
3941	148	123
1671	149	148

Step 4:

To compare the values between the rows, we need to use both columns (*standard_sum* and *lag*). We add a new column named `lag_difference`, which subtracts the *lag* value from the value in *standard_sum* for each row in the table:

`standard_sum - LAG(standard_sum) OVER (ORDER BY standard_sum) AS lag_difference`

```
SELECT account_id,
       standard_sum,
       LAG(standard_sum) OVER (ORDER BY standard_sum) AS lag,
       standard_sum - LAG(standard_sum) OVER (ORDER BY standard_sum) AS lag_difference
FROM (
       SELECT account_id,
       SUM(standard_qty) AS standard_sum
       FROM orders
       GROUP BY 1
      ) sub
```

Each value in *lag_difference* is comparing the row values between the 2 columns (*standard_sum* and *lag*). E.g., since the value for *lag* in the case of account_id 1901 is NULL, the value in *lag_difference* for account_id 1901 will be NULL. However, for account_id 3371, the value in *lag_difference* will compare the value 79 (*standard_sum* for account_id 3371) with 0 (*lag* for account_id 3371) resulting in 79. This goes on for each row in the table.

What you see after running this SQL code:

account_id	standard_sum	lag	lag_difference
1901	0		
3371	79	0	79
1961	102	79	23
3401	116	102	14
3741	117	116	1
4321	123	117	6
3941	148	123	25
1671	149	148	1

Now let's look at the LEAD function.

LEAD function

Purpose:

Return the value from the row following the current row in the table.

Step 1:

Let's first look at the **inner query** and see what this creates.

```
SELECT     account_id,
           SUM(standard_qty) AS standard_sum
FROM       demo.orders
GROUP BY   1
```

What you see after running this SQL code:

1. The query sums the standard_qty amounts for each account_id to give the standard paper each account has purchased over all time. E.g., account_id 2951 has purchased 8181 units of standard paper.
2. Notice that the results are not ordered by account_id or standard_qty.

account_id	standard_sum
2951	8181
2651	4231
3401	116
2941	790
1501	7941
1351	3174
1721	1176
1191	6304

Step 2:

We start building the **outer query**, and name the inner query as `sub`.

```
SELECT account_id,
       standard_sum
FROM   (
        SELECT   account_id,
                 SUM(standard_qty) AS standard_sum
        FROM     demo.orders
        GROUP BY 1
       ) sub
```

This will produce the same table as above, but sets us up for the next part.

account_id	standard_sum
2951	8181
2651	4231
3401	116
2941	790
1501	7941
1351	3174
1721	1176
1191	6304

Step 3 (Part A):

We add the Window Function `(OVER BY standard_sum)` in the outer query that will create a result set ordered in ascending order of the *standard_sum* column.

```
SELECT account_id,
       standard_sum,
       LEAD(standard_sum) OVER (ORDER BY standard_sum) AS lead
FROM   (
        SELECT   account_id,
                 SUM(standard_qty) AS standard_sum
        FROM     demo.orders
        GROUP BY 1
       ) sub
```

This ordered column will set us up for the other part of the Window Function (see below).

Step 3 (Part B):

The `LEAD` function in the Window Function statement creates a new column called *lead* as part of the outer query: `LEAD(standard_sum) OVER (ORDER BY standard_sum) AS lead`

This new column named *lead* uses the values from *standard_sum* (in the ordered table from Step 3 (Part A)). Each row's value in *lead* is pulled from the row after it. E.g., for account_id 1901, the value in *lead* will come from the row following it (i.e., for account_id 3371). Since the value is 79, the value in *lead* for account_id 1901 will be 79. For account_id 3371, the value in *lead* will be pulled from the following row (i.e., account_id 1961), which will be 102. This goes on for each row in the table.

```
SELECT account_id,
       standard_sum,
       LEAD(standard_sum) OVER (ORDER BY standard_sum) AS lead
FROM   (
        SELECT   account_id,
                 SUM(standard_qty) AS standard_sum
        FROM     demo.orders
        GROUP BY 1
       ) sub
```

What you see after running this SQL code:

account_id	standard_sum	lead
1901	0	79
3371	79	102
1961	102	116
3401	116	117
3741	117	123
4321	123	148
3941	148	149
1671	149	183

Step 4: To compare the values between the rows, we need to use both columns (*standard_sum* and *lag*). We add a column named *lead_difference*, which subtracts the value in *standard_sum* from *lead* for each row in the table:

`LEAD(standard_sum) OVER (ORDER BY standard_sum) - standard_sum AS lead_difference`

```
SELECT account_id,
       standard_sum,
       LEAD(standard_sum) OVER (ORDER BY standard_sum) AS lead,
       LEAD(standard_sum) OVER (ORDER BY standard_sum) - standard_sum AS lead_difference
FROM (
SELECT account_id,
       SUM(standard_qty) AS standard_sum
       FROM orders
       GROUP BY 1
     ) sub
```

Each value in *lead_difference* is comparing the row values between the 2 columns (*standard_sum* and *lead*). E.g., for account_id 1901, the value in *lead_difference* will compare the value 0 (*standard_sum* for account_id 1901) with 79 (*lead* for account_id 1901) resulting in 79. This goes on for each row in the table.

What you see after running this SQL code:

account_id	standard_sum	lead	lead_difference
1901	0	79	79
3371	79	102	23
1961	102	116	14
3401	116	117	1
3741	117	123	6
4321	123	148	25
3941	148	149	1
1671	149	183	34

Scenarios for using LAG and LEAD functions

You can use LAG and LEAD functions whenever you are trying to compare the values in adjacent rows or rows that are offset by a certain number.

Example 1: You have a sales dataset with the following data and need to compare how the market segments fare against each other on profits earned.

Market Segment	Profits earned by each market segment
A	$550
B	$500
C	$670
D	$730
E	$982

Example 2: You have an inventory dataset with the following data and need to compare the number of days elapsed between each subsequent order placed for Item A.

Inventory	Order_id	Dates when orders were placed
Item A	001	11/2/2017
Item A	002	11/5/2017
Item A	003	11/8/2017
Item A	004	11/15/2017
Item A	005	11/28/2017

As you can see, these are useful data analysis tools that you can use for more complex analysis!