**********
Introduction to SQL, Session 4, Part 1
The SELECT statement ORDER BY clause
**********

---
Session 4 Part 1 Slide 1
---

Hello, this is Professor Teresa Pelkie.

This is Session 4, Part 1, the SELECT statement ORDER BY clause.

---
Retrieving data from a datbase - sorting
Session 4 Part 1 Slide 2
---

In this session, we'll be looking at how you can sort data when you retrieve it with a SELECT statement.

In most applications, your users will expect to see data in a certain order. You've probably seen this on some web sites, where you can sort items by price, by size, or by other factors.

In this session, we'll cover the following:

* Part 1, the SELECT statement ORDER BY clause. The ORDER BY clause is easy to get started with.

* Part 2, sorting in ascending and descending sequence. You'll see how you can get the data to appear in the sequence that you need.

* Part 3, sorting when NULL values are present. This will be one of the first times we've had to work with NULL values, and you'll see how they work with the ORDER BY clause.

Once again, we'll be working with the "world" schema and its tables. This is the schema that was introduced in Lab 1.

---
How is data sorted by default?
Session 4 Part 1 Slide 3
---

First of all, we'll look at how data is returned from a SELECT statement when there is no ORDER BY clause.

In this example, you can see data in the world.country table. There is no ORDER BY clause.

The data seems to be sorted by the Code. The first few

country names are sorted, but you start to see names
further down in the list that are not in alphabetical order.

---

How is data sorted by default?
Session 4 Part 1 Slide 4

---

The rule is, the SQL language does not specify or require
that data be returned in any particular order if the
ORDER BY clause is not present.

That means that data can be returned in any order. Also,
there is no guarantee that if you use a SELECT without
an ORDER BY, that you will get data back in the same
default order. New rows might have been added, or
existing rows deleted or changed.

The bottom line is, when you do not have an ORDER BY
clause, you cannot predict how the data will be
sorted when you work with it. It might seem to be
in the right order, but you cannot depend upon it.

---

Introducing the ORDER BY clause
Session 4 Part 1 Slide 5 (IntroSQL_S04P01S05)

---

The good news is, the ORDER BY clause is really easy
to use and it provides exactly what you need: a guaranteed
sort order.

As you can see on this slide,the ORDER BY clause
follows the column specifications and the FROM clause.

In this example, the SELECT is a "SELECT ALL", using
the asterisk character.

Notice that the ORDER BY clause is written as two
separate words, ORDER and BY. If you forget to put
a space between the words, you'll get an error
when you try to run it.

To use the ORDER BY, you just put the column name
that you want to use for the sort.

You can see the results of this sort: all of the country
names are now in ascending order. The default sort
order is ascending, or alphabetical.

---

Another way to specify a column
Session 4 Part 1 Slide 6

---

This is the same SELECT and in fact, it is the same
ORDER BY. But this time, instead of using the
column name in the ORDER BY clause, we are using

the column position.

In this SELECT ALL, the Name column is returned as column 2. Keep in mind, when you use a SELECT ALL, the columns are returned in the order that they are defined in the table.

This type of ORDER BY, using the column number instead of the name, is so difficult to work with that you will almost never see it in production applications.

We will not use this type of ORDER BY any more in this course. I only wanted to show this to you now, so that you would be aware that you can do this. If you ever see it again, you'll know what is going on.

---

Multiple column ORDER BY
Session 4 Part 1 Slide 7

---

The true power of ORDER BY comes into play when you specify more than one column in the ORDER BY list.

In this SELECT, we are specifying the ORDER BY clause using the Continent and Name columns.

In the Result Grid, you can see that the data is first sorted by Continent. In this case, you see that Asia is showing up for all of the countries.

Next, the data is sorted by the Name column. So what we end up with is a sorted list of countries that are sorted by the continent first, then the country.

---

Multiple column ORDER BY
Session 4 Part 1 Slide 8

---

Here's another example or an ORDER BY, this time, using three columns. The columns are Continent, Region and Name.

You can specify as many columns in the ORDER BY list as you need. If necessary, you can include all of the columns in the table.

---

ORDER BY column not in SELECT list
Session 4 Part 1 Slide 9

---

Finally, here's an example of an ORDER BY that you might never see in a production application.

Look closely at the SELECT statement. This is not a SELECT ALL, instead, three specific column names are listed as the columns to include in the results.

The columns to include are Name, Continent and Region.

Now look at the ORDER BY clause. The ORDER BY column is Population, which was not specified in the SELECT list.

As you can see in the Result Grid, this is a valid SELECT statement. What you can't see is that the data is sorted in order by population, from the least populated country to the most populated. You can't see that because the Population column is not included.

In most applications, it would be pretty unusual to see a SELECT statement that uses an ORDER BY like this. Most of the time, if you're sorting by a column, you want to see the column in the results.

But if you need to sort by a column but don't care about the data in the column, you can use an ORDER BY like this.

---

Up Next
Session 4 Part 1 Slide 10

---

In the next video, we'll look at how to use the Ascending and Descending options in the ORDER BY clause.

**********

Introduction to SQL, Session 4, Part 2
Sorting in ascending and descending sequence

**********

---

Session 4 Part 2 Slide 1

---

Hello, this is Professor Teresa Pelkie.

This is Session 4, Part 2, sorting in ascending and descending sequence.

---

What is the default sort order in the ORDER BY clause?
Session 4 Part 2 Slide 2

---

In the previous video, we had our first look at the ORDER BY clause. You saw how you could use ORDER BY to put data that is retrieved by a SELECT statement into a sorted order.

By default, when you include an ORDER BY clause, each column in the ORDER BY is sorted into ascending order.

As you can see on this slide, the ORDER BY clause is for the Population column. That means that when the

data is retrieved, it is shown in the Result Grid
in order by the population, from the lowest number
to the highest number.

If the ORDER BY clause is for a character column
instead of a numeric column, the default sort order
is alphabetic.

When you write a SELECT statement with an ORDER BY
clause, the default sort order is ascending. There
is also a keyword, ASC that you can use to specify
ascending order. In actual practice, you will probably
never see the ascending keyword used.

It is not a mistake to include it in your code,
but this is one of those times in programming
when the default is so widely known and accepted
that it is practically never specified.

---

The DESC keyword changes the sort order
Session 4 Part 2 Slide 3

---

You can sort in the other order using the DESC
or descending, keyword.

This keyword is easy to use. You just put it after
the column name in the ORDER BY clause, as shown
on this slide. You have to put at least one space
between the column name and the keyword.

So now you see that the Result Grid shows countries
in order by population, with the country with the
greatest population listed first. When you use
descending order, the greatest numeric value
is shown first. If you scroll through the Result Grid,
you'll see that as you scroll, the population numbers
decrease.

If you use the descending keyword with a character
column, the data is displayed in reverse alphabetical
order.

---

Using DESC with multiple columns
Session 4 Part 2 Slide 4

---

You can use the descending keyword when you have more
than one column in the ORDER BY clause.

On this slide, there are three columns in the ORDER BY.
The last column is Population, and once again, it has
the descending keyword specified for it.

So you now see data in the Result Grid, by continent,
in ascending order, then region, in ascending order,
and finally population, in descending order.

---

Using DESC with multiple columns
Session 4 Part 2 Slide 5

---

You can use the descending keyword on as many of the ORDER BY columns as you want.

This slide is the same as the previous, with the same three columns in the ORDER BY clause.

This time, the descending keyword is used for the Region and Population columns. So the data is displayed by continent in ascending order, then region in descending order, then population in descending order.

Notice that Region is a character column and Population is a numeric column. It is perfectly valid to use the descending keyword on different data types.

This wraps up our quick look at the ascending and descending sequences in the ORDER BY clause.

When you develop a SELECT statement that has more than one column in the ORDER BY, you should run some examples of the select and look at what is returned in the Result Grid.

You'll want to be especially careful when you use the descending keyword, and when you have more than one column. Be sure to check the grid to make sure that the data is being returned in the order that you need for your application.

---

Up Next
Session 4 Part 2 Slide 6

---

In the next video, we'll look at how to to work with nulls when you are sorting data.

**********

Introduction to SQL, Session 4, Part 3
Sorting when NULL values are present

**********

---

Session 4 Part 3 Slide 1

---

Hello, this is Professor Teresa Pelkie.

This is Session 4, Part 3, sorting when NULL values are present.

---

Using a NULL capable column in ORDER BY
Session 4 Part 3 Slide 2

---

We'll finish up this session by looking at what happens when you use an ORDER BY clause and there are NULL values in the ORDER BY column list.

On this slide, you can see that we are running a SELECT statement that specifies a column list. The four columns that are to be retrieved are Name, which is the country name, Continent, Region and Independence Year, which has an abbreviated column name.

The ORDER BY clause specifies three columns: Continent, Region and Independence Year.

In the definition for the country table, the Independence Year column is defined as a NULL capable column. It makes sense to allow NULL for this column, since there are some countries that are not considered to be independent.

When used like this, for a year value, NULL indicates that there is no value that can be assigned.

You can see the results of using ORDER BY with a NULL capable column. In this example, data is sorted first by the continent, then the region, then the independence year.

When the default sort order of ASCENDING is used, the NULL values for the Independence Year appear first in the sorted order.

Notice that NULL appears before negative values. China and Japan are shown as having negative independence years, or "B.C." years. The two countries with NULL values appear before the "B.C." years.

---

Using a NULL capable column in ORDER BY
Session 4 Part 3 Slide 3

---

You can use the DESC keyword with a NULL capable column.

This is the same SELECT statement and the same ORDER BY as on the previous slide. This time, the Independence Year sort order will be in descending sequence.

In this example, the countries with NULL values will appear last in the sort. Descending sequence means that values will be sorted from the greatest to the least.

---

Sorting an ENUM column type
Session 4 Part 3 Slide 4

---

We're going to take a look at another issue that you might run in to when using ORDER BY.

Take a look at the Continent column in the Result Grid.

Now look at the ORDER BY clause. In that clause, Continent is listed as the first column, meaning that it has the first sort order. That is, the data in the Result Grid is to be sorted first into continent order, then within continent by region, then within region by independence year.

You can see that the Continent column data is not sorted into the expected order. Given the data shown here, South America, Antarctica and Oceania, Antarctica should have appeared last.

What is going on here? Is the database broken?

---

## Sorting an ENUM column type
## Session 4 Part 3 Slide 5

---

It turns out that the Continent column is defined in the table as an ENUM datatype.

An ENUM datatype is defined with an enumeration, or list, of valid values that can be used with the column. In this enumeration, the continent names are listed in this order: Asia, Europe, North America, Africa, Oceania, Antarctica, South America.

That list is not in alphabetical order. When you define a column as an ENUM datatype, you are not required to specify the values in any particular order.

Now, there are a lot of issues with using an ENUM datatype, but we won't go into those issues in this course. The ENUM datatype is not defined in the SQL standard, so if you are working with something other than MySQL, it may not be available.

The important thing to understand is that MySQL does have an ENUM datatype, and you might have to work with it in your applications.

---

## Sorting an ENUM column type
## Session 4 Part 3 Slide 6

---

There is a "work around" that lets you sort an ENUM datatype column into the expected order.

The solution is to use the CAST function, as shown in the ORDER BY clause on this slide. In this example, the CAST

function is used to cast, or convert, the Continent column values to character datatypes.

Now it is true that the values that were specified for the ENUM, which you saw on the previous slide, are character data. For example, you saw values like Asia, Africa and North America.

Without going into a lot of details about how MySQL works internally, it is sufficient to say that MySQL stores a value in an ENUM column as an integer. So in the Continent ENUM, Asia was defined as the first value in the ENUM, so it is assigned the numeric value 1. Europe is assigned the value 2, and so on.

So if you want to use an ORDER BY on an ENUM column and end up with a "natural order" sort, that is, you want to see data sorted into the order you would expect, you have to tell MySQL to convert the stored integer value back to the character value. The character value is then used in the ORDER BY sort.

As shown on this slide, the data is now displayed in the Result Grid in the order that you would expect, given that we asked for the Continent to be sorted in descending order.

---

Up Next
Session 4 Part 3 Slide 7

---

In the next session, we'll start working with the SELECT statement's WHERE clause. Using the WHERE clause, you'll be able to include just the data that you want.

** END **