ORDER BY to Sort Data

by Sophia

WHAT'S COVERED

This lesson explores the ORDER BY clause within a SELECT statement to sort data based on columns, in three parts. Specifically, this lesson will cover:

1. Sorting by One Column
2. Ascending or Descending
3. Cascading Order Sequence

BEFORE YOU START

It can sometimes be useful to sort query results by one or more attributes. For example, you might sort a list of customers and their addresses by ZIP code when planning a delivery driver's route.

1. Sorting by One Column

When we query from a table, the result set is sorted by default based on the order in which the data was entered into the table. In most relational databases, a unique identifier is automatically generated through an auto-incremented value each time a new record is added, and that number is used for the default sort order. In PostgreSQL, this automatically generated identifier is called a **serial**. Often this identifier serves as the primary key in the table. All databases do this, and they all offer a type of "auto-increment" as a way of making sure each record is unique. In the table below, you see a column invoice_id that is the auto-increment for this table.

Now you know why, when you have to return a product or an order over the phone, they always want to know what your order number is. The order number is often the auto-assigned number that serves as the primary key, uniquely identifying your order.

Generally, data is sorted by the primary key in a table. This means when you SELECT * from the invoice table, you will see all the data sorted by the "invoice_id."

```
SELECT *
FROM invoice;
```

invoice_id	customer_id	invoice_date	billing_address
1	2	2009-01-01T00:00:00.000Z	Theodor-Heuss-Straße 34
2	4	2009-01-02T00:00:00.000Z	Ullevålsveien 14
3	8	2009-01-03T00:00:00.000Z	Grétrystraat 63
4	14	2009-01-06T00:00:00.000Z	8210 111 ST NW
5	23	2009-01-11T00:00:00.000Z	69 Salem Street
6	37	2009-01-19T00:00:00.000Z	Berger Straße 10
7	38	2009-02-01T00:00:00.000Z	Barbarossastraße 19
8	40	2009-02-01T00:00:00.000Z	8, Rue Hanovre
9	42	2009-02-02T00:00:00.000Z	9, Place Louis Barthou
10	46	2009-02-03T00:00:00.000Z	3 Chatham Street

The ORDER BY clause is useful when we want to list records in a specific order. For example, if we wanted to sort the result of a SELECT statement based on the billing_country for invoices, we can add an ORDER BY clause to the SELECT statement after the FROM clause:

```
SELECT *
FROM invoice
ORDER BY billing_country;
```

invoice_id	customer_id	invoice_date	billing_address	billing_city	billing_state	billing_country	billing_postal_code	total
403	56	2013-11-08T00:00:00.000Z	307 Macacha Güemes	Buenos Aires		Argentina	1106	9
348	56	2013-03-10T00:00:00.000Z	307 Macacha Güemes	Buenos Aires		Argentina	1106	14
119	56	2010-06-12T00:00:00.000Z	307 Macacha Güemes	Buenos Aires		Argentina	1106	2
142	56	2010-09-14T00:00:00.000Z	307 Macacha Güemes	Buenos Aires		Argentina	1106	4
164	56	2010-12-17T00:00:00.000Z	307 Macacha Güemes	Buenos Aires		Argentina	1106	6
216	56	2011-08-07T00:00:00.000Z	307 Macacha Güemes	Buenos Aires		Argentina	1106	1
337	56	2013-01-28T00:00:00.000Z	307 Macacha Güemes	Buenos Aires		Argentina	1106	2
118	55	2010-05-30T00:00:00.000Z	421 Bourke Street	Sidney	NSW	Australia	2010	1
239	55	2011-11-21T00:00:00.000Z	421 Bourke Street	Sidney	NSW	Australia	2010	2
66	55	2009-10-09T00:00:00.000Z	421 Bourke Street	Sidney	NSW	Australia	2010	6
250	55	2012-01-01T00:00:00.000Z	421 Bourke Street	Sidney	NSW	Australia	2010	14
44	55	2009-07-07T00:00:00.000Z	421 Bourke Street	Sidney	NSW	Australia	2010	4
21	55	2009-04-04T00:00:00.000Z	421 Bourke Street	Sidney	NSW	Australia	2010	2
305	55	2012-08-31T00:00:00.000Z	421 Bourke Street	Sidney	NSW	Australia	2010	9
89	7	2010-01-18T00:00:00.000Z	Rotenturmstraße 4, 1010 Innere Stadt	Vienne		Austria	1010	19

In our results, we get a listing of all the invoices sorted by the billing_country, in alphabetical order. Looking at the rest of the result set, we can see that only the billing_country is in order; the invoice_id is no longer sequential like it was in our first SELECT statement.

Let's look at the customer table and see what the results would look like without ordering the data. For this, let's try this SELECT command:

```
SELECT customer_id, first_name, last_name
FROM customer;
```

customer_id	first_name	last_name
1	Luís	Gonçalves
2	Leonie	Köhler
3	François	Tremblay
4	Bjørn	Hansen
5	František	Wichterlová
6	Helena	Holý
7	Astrid	Gruber
8	Daan	Peeters
9	Kara	Nielsen

We got all the data sorted by customer_id, as that is the first column and the primary key for this table. It also auto-increments so that the customer_id will always be unique.

If we wanted to order it by the first_name, we can simply add the ORDER BY clause:

```
SELECT customer_id, first_name, last_name
FROM customer
ORDER BY first_name;
```

The result set for the customer table would now be sorted by the first_name column:

customer_id	first_name	last_name
32	Aaron	Mitchell
11	Alexandre	Rocha
7	Astrid	Gruber
4	Bjørn	Hansen
39	Camille	Bernard
8	Daan	Peeters
20	Dan	Miller
56	Diego	Gutiérrez
40	Dominique	Lefebvre
10	Eduardo	Martins
30	Edward	Francis

TERM TO KNOW

Serial

In Postgres, the unique identifier assigned to each record as it is entered into a table.

2. Ascending or Descending

By default, when we use the ORDER BY clause in a SELECT statement, it sorts the data in ascending order—in other words, from A to Z, or numerically from smallest to largest. For example, if we wanted to query the invoice table and display the result set based on ORDERED BY the column total, we could sort it like this:

```
SELECT invoice_id, customer_id, total
FROM invoice
ORDER BY total;
```

Query Results

Row count: 412

invoice_id	customer_id	total
174	5	1
384	24	1
167	26	1
62	46	1
265	27	1
286	23	1
328	15	1
160	47	1

This gives us every order that is on the table. All 412 rows show us everyone who has at least one order in the order table. This is good information to know, as we could select this group for a special e-mail, marketing campaign, or even just a customer outreach to see how their experience went with the company. However, there are instances where we may want to sort the data in descending order, from Z to A, or numerically from largest to smallest.

What if we wanted to know who had the most orders on the table, so that we could send them a 10% off coupon or other reward for being a great or loyal customer? We can add the keyword DESC (sort in DESENDING ORDER) to the ORDER BY clause after the column to sort it accordingly.

```
SELECT invoice_id, customer_id, total
FROM invoice
ORDER BY total DESC;
```

This would result in the following:

Query Results

Row count: 412

invoice_id	customer_id	total
404	6	26
299	26	24
194	46	22
96	45	22
89	7	19
201	25	19
88	57	18
313	43	17
306	5	17
103	24	16

You can also use ASC (ASCENDING) in place of DESC to display the result in ascending order. This is where SQL can be interesting with its default behavior of sorting by ascending order naturally. We can use ASC or not in the SELECT command because they both provide the same answer. From an optimization process, using the default behavior to your advantage can make the SELECT query run more efficiently.

These two commands would result in the same result set, in the same order:

```
SELECT *
FROM invoice
ORDER BY total;
```

And

```
SELECT *
FROM invoice
ORDER BY total ASC;
```

3. Cascading Order Sequence

In some cases, you might to specify additional sort levels that will apply in case of a tie in the main sort field. For example, suppose you want a list of customers by last_name, but there are multiple customers with the same last name. In that case, you want the first_name field to be the tiebreaker. If there are any records where both the first and last names are the same, you want to sort by company.

Breaking up the problem into sections can help build the SQL SELECT statement.

1. First, ORDER BY last_name.
2. Then, within the matching last_name values, ORDER BY first_name.
3. Next, within the matching last_name and first_name values, we would ORDER BY the company.

This type of multilevel ordered sequence is called the **cascading order** sequence. It can easily be created by listing the columns, separated by commas, after the ORDER BY clause.

```
SELECT customer_id, last_name, first_name, company
FROM customer
ORDER BY last_name, first_name, company;
```

You can have separate ascending or descending orders per level. For example, if you wanted the last_name order in the above example to be descending but the other two levels ascending, you would change the ORDER BY line to:

```
ORDER BY last_name DESC, first_name, company;
```

This cascading order sequence is useful in helping sort data that can then be used by other systems, spreadsheets, or even database reporting. You will see this a lot when working with database SELECT statements, as they are truly useful in data presentation.

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then, enter in one of the examples above and see how it works. Next, try your own ORDER BY clauses.

Cascading Order

A multilevel sequence used for complex sorting operations.

In this lesson, you learned that PostgreSQL supports **ascending and descending** orderings of query results based on **sorting one column** or more than one column. SELECT can be used with ASCENDING (ASC) and DESCENDING (DESC) clauses that are used to sort query results. ASC arranges the results ascendingly, from lowest to highest. Alternatively, DESC sorts the results from highest to lowest in descending order. A secondary sort order can also be established if multiple rows share the same primary sort column value, as PostgreSQL supports **cascading order sequencing**. This ensures that query results will appear in a consistent and predictable order.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND Faithe Wempen (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR **TERMS OF USE**.

TERMS TO KNOW

Cascading Order

A multilevel sequence used for complex sorting operations.

Serial

In Postgres, the unique identifier assigned to each record as it is entered into a table.