SQL Limit: The Complete Guide to SQL Row Limiting and Top-N Queries

4 Comments / Last updated: June 10, 2023

Do you need to get the top rows in an SQL query?

Do you need to perform pagination, or find the top 5 or 10 results from a query?

You can do that in SQL using a "row limiting" feature.

Learn how to use this SQL limit feature in this article.

SQL Cheat Sheets:
Get my SQL Cheat Sheets for Oracle, SQL Server, MySQL, and Postgres

Table of Contents

Table of Contents

Here's what's included in this guide.

What is the "SQL Limit" or Row Limiting Feature?

SELECT queries in SQL allow you to return results from database tables.

You can use the ORDER BY clause to order data however you like. You can also use the WHERE clause to filter data.

But what if you wanted to only return a certain number of rows?

Some common examples are:

- Find the top 10 highest-paid employees
- Find the top 20 most profitable customers
- Find the top 3 users on the system

These are also called "top-N queries" because you're trying to find the top number of records from a result set. This could be top 1, top 3, top 5, top 10, or top any number.

These queries are hard to do with just an ORDER BY and WHERE clause alone.

The good news is you don't have to use another programming language like Java or .Net. You don't even have to use PL/SQL.

You can limit the rows in an SQL query using keywords that already exist in SQL.

I'll explain how to do that in this guide.

↑ Table of Contents

Sample Data

Before we get started, here is some sample data that we'll use throughout this guide. You can find the SQL script for this here on my GitHub repository.

```
CREATE TABLE customer_revenue (
  customer_id NUMBER(4),
  revenue NUMBER(10)
);

INSERT INTO customer_revenue (customer_id, revenue) VALUES (1, 109470);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (2, 9384760);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (3, 1852828);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (4, 3596297);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (5, 82495);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (6, 935191);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (7, 654796);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (8, 4341421);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (9, 3378075);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (10, 5131750);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (11, 4431791);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (12, 245583);
INSERT INTO customer_revenue (customer_id, revenue) VALUES (13, 3596297);
```

Once you run this statement on an SQL database, your table will look like this:

CUSTOMER_ID	REVENUE
1	109470
2	9384760
3	1852828
4	3596297
5	82495
6	935191
7	654796
8	4341421
9	3378075
10	5131750
11	4431791
12	245583
13	3596297

This SQL is designed to work on Oracle. To get it working on other databases, you may just need to change the data types in the CREATE TABLE statement.

Now, let's take a look at how you perform row limiting on different vendors, before looking at how you can do it in Oracle.

MySQL SQL Limit Feature: The SQL LIMIT Keyword

In MySQL, you can use the LIMIT clause to restrict the number of rows returned by a SELECT query. You provide two parameters: the offset number, and the count (the maximum number of rows to be returned).

The syntax of this in MySQL is:

```
SELECT columns
FROM table
LIMIT offset, count;
```

↑ Table of Contents

We can use this on our sample database. Let's say we wanted to find the top 5 customers by revenue in descending order. Our query would look like this:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
LIMIT 5;
```

The results would look like this:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297

This table shows the top 5 results ordered by revenue in descending order. The LIMIT clause in MySQL is easy to use and is the most common way of limiting the top results in MySQL. For more information on the MySQL row limiting feature, look at the official documentation.

SQL Server SQL Limit Feature: The SQL Top Keyword

The way to perform row limiting in SQL Server is different from doing it in MySQL.

In SQL Server, you use the SQL TOP keyword rather than LIMIT. The SQL TOP keyword goes at the start of the query in the SELECT clause.

The syntax of a SELECT query that uses TOP is:

```
SELECT TOP number|percent columns
FROM table;
```

The word TOP goes just after the word SELECT, just like the DISTINCT keyword.

You can then specify either the number of rows to display or the percentage of rows from the result set to display.

Let's look at an example using our sample data. If we wanted to find the same data as the earlier example (the top 5 customers ordered by revenue in descending order), our SQL query would look like this:

```
SELECT TOP 5 customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC;
```

The output for this query would be:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297

It's the same result as the MySQL LIMIT example. It shows the top 5 customers by revenue.

Both the MySQL and SQL Server features for row limiting or showing the top-N queries are simple. What about Oracle?

SQL Cheat Sheets:

Get my SQL Cheat Sheets for Oracle, SQL Server, MySQL, and Postgres

↑ Table of Contents

Oracle Row Limiting Clause

Up until Oracle 12c, performing row limiting or top-N queries was harder. There was no keyword you could use, so you had to use one of several solutions. I've outlined those later in this guide if you're not using 12c or higher.

If you are using Oracle 12c, performing top-N and SQL limit queries is a lot easier with the introduction of the Row Limiting clause. (Related: Oracle 12c New Features for Developers)

The syntax for this clause looks a little complicated (as shown in the official documentation), but it's simple to use when you know what data you want to see:

```
SELECT columns
FROM table
[ OFFSET offset { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ { rowcount | percent PERCENT } ]
{ ROW | ROWS } { ONLY | WITH TIES } ]
```

If you want to write a top-N query, the query would look simple. Using our earlier example of finding the top 5 customers by revenue, our query would look like this:

```
SELECT customer_id, revenue
FROM customer_revenue
FETCH FIRST 5 ROWS ONLY;
```

The result looks like this:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297

So, to write a top-N SQL query in Oracle, you simply add FETCH FIRST n ROWS ONLY to the end of your SQL query and substitute n for the number of rows you want to return.

Oracle WITH TIES

The WITH TIES option lets you include rows that have the same value for row number N. If we run our query from earlier (with ROWS ONLY), it shows the same results:

```
SELECT customer_id, revenue
FROM customer_revenue
FETCH FIRST 5 ROWS ONLY;
```

CUSTOMER_ID	REVENUE
2	9384760
10	5131750

11	4431791
8	4341421
4	3596297

However, there are two rows with the same revenue that could appear in position 5. To show them both, we use the WITH TIES option:

↑ Table of Contents

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
FETCH FIRST 5 ROWS WITH TIES;
```

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297
13	3596297

This shows 6 rows because both customer_id 4 and 13 have the same revenue.

Fetch the Top X Percent of Rows in Oracle

You can use the Oracle row limiting clause to get the top percentage of rows. This is done using the PERCENT keyword within this clause.

Let's take a look at this query:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
FETCH FIRST 25 PERCENT ROWS ONLY;
```

We've added the PERCENT keyword into the FETCH clause to indicate we only want to see the top 25% of rows. This top 25% is calculated by ordering the revenue in descending order, as specified by the ORDER BY clause.

The results of this query are:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421

There are 4 rows shown because the table has 13 rows. 25% of 13 is 3.25, which is rounded up to 4.

You can use the same concept to get the last percentage of rows. Rather than changing the FIRST keyword to LAST, you can change the ORDER BY to sort in the opposite direction. This example shows ORDER BY revenue ASC instead of ORDER BY revenue DESC:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue ASC
FETCH FIRST 25 PERCENT ROWS ONLY;
```

This will show the lowest 25% of customers according to their revenue.

CUSTOMER_ID	REVENUE
5	82495

1	109470
12	245583
7	654796

↑ Table of Contents

SQL Pagination in Oracle

Pagination is a feature in many applications that allows you to show different pages of results.

For example, when searching in Google, you see results 1 to 10. When you click on the next page, you see results 11-20.

This is easy to do in Oracle SQL with the row limiting clause. We can simply use the OFFSET keyword within this clause.

With our sample query, say we wanted to find the top 3 customers by revenue. Our query would look like this:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
FETCH FIRST 3 ROWS ONLY;
```

The results would be:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791

If we then wanted to find “page 2” of this data, or see results 4 to 6, our query would be:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
OFFSET 3 ROWS FETCH FIRST 3 ROWS ONLY;
```

This includes the words OFFSET 3 ROWS, which means that the “FETCH FIRST 3 ROWS” should start after the first 3 rows, therefore getting rows 4 to 6.

The results would be:

CUSTOMER_ID	REVENUE
8	4341421
4	3596297
13	3596297

You can change the parameters in the OFFSET and FETCH clause to define how many results you want per page.

For example, if you want 10 results per page, your queries would look like this:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
FETCH FIRST 10 ROWS ONLY;
```

And for the second page:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
OFFSET 10 ROWS FETCH FIRST 10 ROWS ONLY;
```

You can and should use bind variables for the OFFSET and FETCH values, but I'll write about that in another post.

↑ Table of Contents

Select the Top 1 Row in Oracle SQL

To find the top 1 row in Oracle SQL, you can use the FETCH parameter and specify FETCH FIRST 1 ROWS ONLY. As long as your ORDER BY clause shows how you want to order your data, it will work.

An example query would look like this:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
FETCH FIRST 1 ROWS ONLY;
```

The result shown is:

CUSTOMER_ID	REVENUE
2	9384760

It shows the customer with the highest revenue.

Select the Top 100 Rows in Oracle SQL

To find the top 100 rows in a query in Oracle SQL, you can use the FETCH parameter and specify FETCH FIRST 100 ROWS ONLY. Add an ORDER BY clause to your query to define how the data is ordered, and the data will be displayed.

The query could look like this:

```
SELECT customer_id, revenue
FROM customer_revenue
ORDER BY revenue DESC
FETCH FIRST 100 ROWS ONLY;
```

The results of this are:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297
13	3596297
9	3378075
3	1852828
6	935191
7	654796
12	245583
1	109470
5	82495

This result only includes 13 rows, but that's because there are 13 rows in the table. If there are more in your table, the result will only show 100 records.

↑ Table of Contents

Row Limiting Without the FETCH Clause

There are a few ways to do row limiting and top-N queries without the FETCH clause in Oracle. This could be because you're not working on a 12c or 18c database. Perhaps you're running Oracle Express 11g, or using an 11g database at work.

There are a few ways to do this.

Using an Inline View with ROWNUM

You can use an inline view with the ROWNUM pseudocolumn to perform top-N queries. This is one of the most common ways to do row limiting without the FETCH clause.

The syntax looks like this:

```
SELECT columns
FROM (
  SELECT columns
  FROM table
  ORDER BY col DESC
)
WHERE ROWNUM <= rownumber;
```

You just need to specify the columns to view, the column to order by, and the number of rows to limit it to.

If we want to see the top 5 customers by revenue using our sample data, our query would look like this:

```
SELECT customer_id, revenue
FROM (
  SELECT customer_id, revenue
  FROM customer_revenue
  ORDER BY revenue DESC
)
WHERE ROWNUM <= 5;
```

The results would be the same as the earlier examples:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297

This works because the data is ordered before ROWNUM is applied.

However, this is the kind of query **we want to avoid**:

```
SELECT customer_id, revenue
FROM customer_revenue
WHERE ROWNUM <= 5
ORDER BY revenue DESC;
```

This will perform the limiting on the row number before the ordering, and give us these results:

CUSTOMER_ID	REVENUE
2	9384760

4	3596297
3	1852828
1	109470
5	82495

↑ Table of Contents

It has limited the results to the first 5 rows it has found in the table, and then performed the ordering. This will likely give you results you are not expecting. This is why we use a subquery such as an inline view.

Pagination with ROWNUM in Oracle

We can use this method to perform pagination of records in SQL. While it looks a little messier than the FETCH and OFFSET methods, it does work well:

```
SELECT customer_id, revenue
FROM (
  SELECT customer_id, revenue, ROWNUM AS rnum
  FROM (
    SELECT customer_id, revenue
    FROM customer_revenue
    ORDER BY revenue DESC
  )
  WHERE ROWNUM <= 10
)
WHERE rnum > 5;
```

This query does a few things:

- It includes two nested inline views. The innermost view gets the data and orders it by the revenue.
- The next inline view limits the results of the innermost view where the ROWNUM is less than or equal to 10.
- The inline view then calculates the ROWNUM of these new results and labels it as “rnum”
- The outer query then limits the entire result set to where the rnum is greater than 5.

So, in theory, it should show us rows 6 to 10. This is the result from this query:

CUSTOMER_ID	REVENUE
13	3596297
9	3378075
3	1852828
6	935191
7	654796

It shows rows 6 to 10 when ordering by the revenue in descending order.

Using a WITH Clause with ROWNUM

You can use the WITH clause to write the earlier query. For example, to find the top 5 customers by revenue, you can write the query that orders the customers in the WITH clause, and select from that.

This works in a similar way to an inline view.

The query would look like this:

```
WITH customers_ordered AS (
  SELECT customer_id, revenue
  FROM customer_revenue
  ORDER BY revenue DESC
)
SELECT customer_id, revenue
FROM customers_ordered
WHERE ROWNUM <= 5;
```

This will show the same results:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297

↑ Table of Contents

Using RANK and DENSE_RANK for Top-N Queries

You can use the RANK function in Oracle to find the top-N results from a query. It's a bit more complicated, as you'll need to use RANK as an analytic function, but the query works.

The query to find the top 5 customers by revenue would look like this:

```
SELECT customer_id, revenue
FROM (
  SELECT customer_id, revenue,
  RANK() OVER (ORDER BY revenue DESC) AS revenue_rank
  FROM customer_revenue
)
WHERE revenue_rank <= 5;
```

The ordering is done within the parameters of the RANK function, and the limiting is done using the WHERE clause.

The results of this query are:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297
13	3596297

This shows 6 results because customer_id 4 and 13 have the same revenue. This is because the RANK function does not exclude rows where there are ties.

You can also use the DENSE_RANK function instead of the RANK function.

```
SELECT customer_id, revenue
FROM (
  SELECT customer_id, revenue,
  DENSE_RANK() OVER (ORDER BY revenue DESC) AS revenue_rank
  FROM customer_revenue
)
WHERE revenue_rank <= 5;
```

The difference between these two functions is that the rank numbers that are assigned do not include gaps. However, it still shows us the top 6 rows as there is a tie.

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297

13	3596297

Using ROW_NUMBER For Top-N Queries

↑ Table of Contents

You can use the Oracle analytic function ROW_NUMBER to write top-N queries. It's similar to the RANK function. The ROW_NUMBER function assigns a unique number for each row returned, but can be used over a window of data (just like all analytic queries). It can also work in SQL Server.

A query to find the top 5 customers by revenue using ROW_NUMBER would look like this:

```
SELECT customer_id, revenue
FROM (
  SELECT customer_id, revenue,
  ROW_NUMBER() OVER (ORDER BY revenue DESC) AS revenue_row
  FROM customer_revenue
) WHERE revenue_row <= 5;
```

It looks very similar to the RANK and DENSE_RANK methods.

The results of this query are:

CUSTOMER_ID	REVENUE
2	9384760
10	5131750
11	4431791
8	4341421
4	3596297

Conclusion

There are many reasons that you may have to limit the number of rows in SQL, such as using pagination or for top-N queries. It's simple to do this in MySQL and SQL Server.

In Oracle, you can use the FETCH clause to do this which makes it easy to do. There are other methods in Oracle you can use for pagination and top-N queries that work but aren't as simple.

SQL Cheat Sheets:

Get my SQL Cheat Sheets for Oracle, SQL Server, MySQL, and Postgres

← Previous Post

Next Post →

4 thoughts on "SQL Limit: The Complete Guide to SQL Row Limiting and Top-N Queries"

MARGARET
NOVEMBER 28, 2018 AT 3:45 AM

Can't say enough 'thank you' for putting this information together for us (developers).
Your examples are very helpful in understanding concepts.

Reply