<https://www.sqlshack.com/use-window-functions-sql-server/>

<https://dzone.com/articles/difference-between-rownumber>

<http://sqlfiddle.com/#!17/b789e/59/0>

<https://blog.jooq.org/2014/08/12/the-difference-between-row_number-rank-and-dense_rank/>

<https://blog.jooq.org/2013/11/03/probably-the-coolest-sql-feature-window-functions/>

-- find second highest salary in below table dept wise

Emp deptno salary

"A" ,10,1000

"B" ,10 ,2000

"C" ,10 ,3000

"D",20,7000

"E" ,20 ,9000

"F" ,20 ,8000

"G" ,30 ,17000

"H" ,30 ,15000

"I" ,30 ,30000

CREATE TABLE employee (emp Varchar(255), deptno INT,salary INT);

INSERT INTO employee VALUES("A" ,10,1000), ("B" ,10 ,2000),("C" ,10 ,3000),("D",20,7000),

("E" ,20 ,9000),("F" ,20 ,8000),("G" ,30 ,17000),("H" ,30 ,15000),("I" ,30 ,30000);

select \* from employee;

-- below query will return dept no and second highest salary

select e1.deptno, max(e1.salary) as maxs

from employee e1

where e1.salary < (select max(salary)

from employee e2

where e2.deptno = e1.deptno

)

group by e1.deptno;

select e3.\* from employee e3 join

(select e1.deptno, max(e1.salary) as maxs

from employee e1

where e1.salary < (select max(salary)

from employee e2

where e2.deptno = e1.deptno

)

group by e1.deptno

) result on result.deptno = e3.deptno and result.maxs = e3.salary;

<http://sqlfiddle.com/#!17/b789e/2>

create table EMPLOYEES

(EmpID char(4) unique Not null,

Ename varchar(10),

Job varchar(9),

MGR char(4),

Hiredate date,

Salary decimal(7,2),

Comm decimal(7,2),

DeptNo char(2) not null,

Primary key(EmpID),

FOREIGN KEY(MGR)

REFERENCES EMPLOYEES (EmpID));

insert into EMPLOYEES values (7839,'King','President',null,'17-Nov-11',5000,null,10);

insert into EMPLOYEES values (7698,'Blake','Manager',7839,'01-May-11',2850,null,30);

insert into EMPLOYEES values (7782,'Clark','Manager',7839,'02-Jun-11',2450,null,10);

insert into EMPLOYEES values (7566,'Jones','Manager',7839,'02-Apr-11',2975,null,20);

insert into EMPLOYEES values (7654,'Martin','Salesman',7698,'28-Feb-12',1250,1400,30);

insert into EMPLOYEES values (7499,'Allen','Salesman',7698,'20-Feb-11',1600,300,30);

insert into EMPLOYEES values (7844,'Turner','Salesman',7698,'08-Sep-11',1500,0,30);

insert into EMPLOYEES values (7900,'James','Clerk',7698,'22-Feb-12',950,null,30);

insert into EMPLOYEES values (7521,'Ward','Salesman',7698,'22-Feb-12',1250,500,30);

insert into EMPLOYEES values (7902,'Ford','Analyst',7566,'03-Dec-11',3000,null,20);

insert into EMPLOYEES values (7369,'Smith','Clerk',7902,'17-Dec-10',800,null,20);

insert into EMPLOYEES values (7788,'Scott','Analyst',7566,'09-Dec-12',3000,null,20);

insert into EMPLOYEES values (7876,'Adams','Clerk',7788,'12-Jan-10',1100,null,20);

insert into EMPLOYEES values (7934,'Miller','Clerk',7782,'23-Jan-12',1300,null,10);

select e.\* from EMPLOYEES e ,EMPLOYEES ee where e.MGR =ee.EmpID and ee.EmpID ='7839'

select e.\* from EMPLOYEES e inner join EMPLOYEES ee on e.MGR =ee.EmpID where ee.EmpID ='7839'

SELECT EmpID,Ename,Salary, RANK() OVER(ORDER BY Salary desc), DENSE\_RANK() OVER(ORDER BY Salary desc),ROW\_NUMBER(ORDER BY Salary desc) FROM EMPLOYEES

SELECT EmpID , Ename, Salary, DENSE\_RANK() OVER(ORDER BY Salary desc) FROM EMPLOYEES

SELECT EmpID,Ename,Salary, RANK() OVER(ORDER BY Salary desc), DENSE\_RANK() OVER(ORDER BY Salary desc),ROW\_NUMBER() OVER(ORDER BY Salary desc) FROM EMPLOYEES

SELECT EmpID,Ename,Salary, RANK() OVER(ORDER BY Salary desc), DENSE\_RANK() OVER(ORDER BY Salary desc),ROW\_NUMBER() OVER(ORDER BY Salary desc,Hiredate asc) FROM EMPLOYEES

**RANK (), DENSE\_RANK (), ROW\_NUMBER ()** these are called window function.

These window function are used with **Over ()** clause

Inside over clause we have to use order by at least with one column.

Partition by clause is optional inside Over () clause

SELECT EmpID, Ename, Job, Salary, RANK () OVER(partition by Job ORDER BY Salary desc) as rank FROM EMPLOYEES ee

In above query we cannot put any condition in where clause directly it will throw error window function cannot be used in where clause. So to use the rank in where clause we have use inline view as below.

select EmpID,Ename,rank from (SELECT EmpID,Ename,Job,Salary,RANK() OVER(partition by Job ORDER BY Salary desc) as rank FROM EMPLOYEES ) ee where ee.rank=3

The red part of the query is used as inline view.

**Get the department with 2nd highest sum of salary (or nth highest salary)**

Select job, summ from (select Job, sum(salary) summ, rank() over(order by sum(salary)) as rank from EMPLOYEES ee group by Job) eee where eee.rank=3

**Get Department wise 2nd highest salary (or nth highest salary)**

select EmpID,Ename,rank from (SELECT EmpID,Ename,Job,Salary,RANK() OVER(partition by Job ORDER BY Salary desc) as rank FROM EMPLOYEES ) ee where ee.rank=3

# [SubQuery Example in SQL – Correlated vs Noncorrelated](http://javarevisited.blogspot.in/2012/07/subquery-example-in-sql-correlated-vs.html)

**SubQuery in SQL** is a query inside another query. Some time to get a particular information from database you may need to fire two separate sql queries, subQuery is a way to combine or join them in single query. SQL query which is on inner part of main query is called inner query while outer part of main query is called outer query. for example in below sql query

**SELECT** name **FROM** City **WHERE** pincode **IN** (**SELECT** pincode **FROM** pin **WHERE** zone='west')

section not highlighted is **OUTER query** while section highlighted with grey is **INNER query**. In this SQL tutorial we will see both Correlated and non correlated sub-query and there examples, some *differences between correlated and noncorrelated subqueries* and finally *subquery vs join* which is classic debatable topic in SQL. By the way this SQL tutorial is next in series of SQL and database articles in Javarevisited like [truncate vs delete](http://javarevisited.blogspot.com/2011/10/how-to-use-truncate-and-delete-command.html) and [10 examples of  SELECT queries](http://javarevisited.blogspot.com/2011/10/selct-command-sql-query-example.html). If you are new here then you may find those examples interesting.

**SubQuery Rules in SQL**

Like any other concept in SQL, subquery also has some rules and you can only embed one query inside another by following rules :

1. subquery can be used in insert statement.

2. subquery can be used in select statement as column.

3. subquery should always return either a scaler value if used with where clause or value from a column if used with IN or NOT IN clause.

Before going to understand **non-correlated  and correlated subquery**, let’s see the table and data which we are going to use in this example. Until you have an understanding of how table look like and what kind of data it stores its little difficult to understand queries. In this subquery example we will use two table Stock and Market. Stock holds different stocks and Market holds all stock exchanges in the world.

mysql> select \* from stock;

+---------+-------------------------+--------------------+

| RIC     | COMPANY                 | LISTED\_ON\_EXCHANGE |

+---------+-------------------------+--------------------+

| 6758.T  | Sony                    | T                  |

| GOOG.O  | Google Inc              | O                  |

| GS.N    | Goldman Sachs Group Inc | N                  |

| INDIGO  | INDIGO Airlines         | NULL               |

| INFY.BO | InfoSys                 | BO                 |

| VOD.L   | Vodafone Group PLC      | L                  |

+---------+-------------------------+--------------------+

6 rows in set (0.00 sec)

mysql> select  from Market;

+------+-------------------------+---------------+

| RIC  | NAME                    | COUNTRY       |

+------+-------------------------+---------------+

| T    | Tokyo Stock Exchange    | Japan         |

| O    | NASDAQ                  | United States |

| N    | New York Stock Exchange | United States |

| BO   | Bombay Stock Exchange   | India         |

+------+-------------------------+---------------+

4 rows in set (0.00 sec)

## Noncorrelated subquery in SQL

There are two kind of subquery in SQL one is called non-correlated and other is called correlated subquery. In non correlated subquery, **inner query doesn't depend on outer query** and can run as stand alone query.*Subquery used along-with IN or NOT IN sql clause is good examples of Noncorrelated subquery in SQL*. Let's a **noncorrelated subquery example** to understand it better.

**NonCorrelated Subquery Example:**

Let’s see the query  “Find all stocks from Japan”, If we analyze this query we know that stock names are stored in Stock table while Country name is stored in Market table, so we need to fire two query first to get RIC for Japanese market and than all stocks which is listed on that Market. we can combine these two queries into one sql query by using subquery as shown in below example:

mysql> **SELECT** COMPANY **FROM** Stock **WHERE** LISTED\_ON\_EXCHANGE = (**SELECT** RIC **FROM** Market **WHERE** COUNTRY='Japan');  
+*---------+*  
| COMPANY |  
+*---------+*  
| Sony    |  
+*---------+*  
1 row **IN** **SET** (0.02 sec)

Here part which is inside bracket is called inner query or subquery. As you see in this example of subquery, **inner query can run alone**and its not depended on outer query and that's why its called **NonCorrelated query**.

**NonCorrelated Subquery Example with IN Clause SQL**

NonCorrelated subquery are used along-with IN and NOT IN clause. here is an example of subquery with IN clause in SQL.

SQL query: Find all stocks from United States and India

mysql> **SELECT** COMPANY **FROM** Stock **WHERE** LISTED\_ON\_EXCHANGE **IN** (**SELECT** RIC **FROM** Market **WHERE** COUNTRY='United States' **OR** COUNTRY= 'INDIA');  
+*-------------------------+*  
| COMPANY                 |  
+*-------------------------+*  
| Google Inc              |  
| Goldman Sachs **GROUP** Inc |  
| InfoSys                 |  
+*-------------------------+*

When Subquery is used along-with *IN or NOT IN Clause* it returns result from one column instead of Scaler value.

## Correlated SubQuery in SQL

**Correlated subqueries** are the one in which **inner query or subquery reference outer query**. Outer query needs to be executed before inner query. One of the most common *example of correlated subquery* is using keywords exits and not exits. An important point to note is that **correlated subqueries are slower queries** and one should avoid it as much as possible.

**Example of Correlated Subquery in SQL**

Here is an example of Correlated subquery “Return all markets which has at least one stock listed on it.”

mysql> **SELECT** m.NAME **FROM** Market m **WHERE** m.RIC = (**SELECT** s.LISTED\_ON\_EXCHANGE **FROM** Stock s **WHERE** s.LISTED\_ON\_EXCHANGE=m.RIC);  
  
+*-------------------------+*  
| NAME                    |  
+*-------------------------+*  
| Tokyo Stock Exchange    |  
| NASDAQ                  |  
| New York Stock Exchange |  
| Bombay Stock Exchange   |  
+*-------------------------+*  
4 rows **IN** **SET** (0.00 sec)

Here inner query will execute for every Market as RIC will be changed for every market.

## Difference between Correlated and NonCorrelated Subquery

Now we have seen correlated and noncorrelated subqueries and there example its much easier to understand *difference between correlated vs noncorrelated queries*. By the way this is also one of the popular sql interview question and its good to know few differences:

1.In case of **correlated subquery** inner query depends on outer query while in case of noncorrelated query inner query or subquery doesn't depends on outer query and run by its own.

2.In case of correlated subquery, outer query executed before inner query or subquery while in case of NonCorrelated subquery inner query executes before outer query.

3.Correlated Sub-queries are slower than non correlated subquery and should be avoided in favor of sql joins.

4.Common example of correlated subquery is using exits and not exists keyword while non correlated query mostly use IN or NOT IN keywords.

## SubQuery vs Join in SQL

Any information which you retrieve from database using subquery can be retrieved by using different types os joins also. Since SQL is flexible and it provides different way of doing same thing. Some people find SQL Joins confusing and subquery specially noncorrelated more intuitive but in terms of performance SQL Joins are more efficient than subqueries.

**Important points about SubQuery in DBMS**

1.Almost whatever you want to do with subquery can also be done using join, it just matter of choice

subquery seems more intuitive to many user.

2.Subquery normally return an scaler value as result or result from one column if used along with

IN Clause.

3.You can use subqueries in four places: subquery as a column in select clause,

4.In case of correlated subquery outer query gets processed before inner query.

Read more: <http://javarevisited.blogspot.com/2012/07/subquery-example-in-sql-correlated-vs.html#ixzz58oSJlNIM>

# Second Highest Salary in MySQL and SQL Server - LeetCode Solution

Write a SQL query to get the second highest salary from the Employee table.  
  
+----+--------+  
| Id | Salary |  
+----+--------+  
| 1  | 100    |  
| 2  | 200    |  
| 3  | 300    |  
+----+--------+  
  
For example, given the above Employee table, the second highest salary is 200. If there is no second highest salary, then the query should return NULL. You can write SQL query in any of your favorite database e.g. MySQL, Microsoft SQL Server or Oracle. You can also use database specific feature e.g. TOP, LIMIT or ROW\_NUMBER to write SQL query, but you must also provide a generic solution which should work on all database. In fact, there are *several ways to find second highest salary* and you must know couple of them e.g. in MySQL without using LIMIT keyword, in SQL Server without using TOP and in Oracle without using RANK and ROWNUM. Once you solve the problem, Interviewer will most likely increase the difficulty level by either moving to Nth salary direction or taking away this buit-in utilities.

## Second Highest Salary in MySQL without LIMIT

Here is a generic SQL query to find second highest salary, which will also work fine in MySQL. This solution uses subquery to first exclude the maximum salary from the data set and then again finds maximum salary, which is effectively the second maximum salary from the Employee table.

**SELECT** **MAX**(salary) **FROM** Employee **WHERE** Salary NOT **IN** ( **SELECT** **Max**(Salary) **FROM** Employee);

This will return 200 in our case.  
  
Here is another solution which uses sub query but instead of IN clause it uses < operator

**SELECT** **MAX**(Salary) **From** Employee **WHERE** Salary **<** ( **SELECT** **Max**(Salary) **FROM** Employee);

You can use this SQL query if Interviewer ask you to get second highest salary in MySQL without using LIMIT.  You can also use distinct keyword if your [Employee table may contain duplicate salary](http://javarevisited.blogspot.sg/2012/12/how-to-find-duplicate-records-in-table-mysql-query-example.html), In this example there is no such record, so I have not used distinct.

### Second Highest Salary using Correlated SubQuery

Previous SQL query was also using subquery but it was non-correlated, this solution will use correlated subquery. This is also generic solution to *find Nth highest salary in Employee table*. For each record processed by outer query, inner query will be executed and will return how many records has records has salary less than the current salary. If you are looking for second highest salary then your query will stop as soon as inner query will return 2. 

**SELECT** Id, Salary

**FROM** Employee e

**WHERE** 2**=**(**SELECT** **COUNT**(DISTINCT Salary) **FROM** Employee p

**WHERE** e.Salary**<=**p.Salary)

By the way, If you don't know difference between correlated and non-correlated sub-query, see [here](http://javarevisited.blogspot.sg/2012/07/subquery-example-in-sql-correlated-vs.html).

## Second Maximum Salary in MySQL using LIMIT

MySQL has a special keyword called LIMIT which can be used to limit the result set e.g. it will allow you to see first few rows, last few rows or range of rows. You can use this keyword to find the second, third or Nth highest salary. Just use order by clause to sort the result set then print the second salary as shown below :

**SELECT** Salary **FROM** (**SELECT** Salary **FROM** Employee **ORDER BY** salary **DESC** **LIMIT** 2) **AS** Emp **ORDER BY** salary **LIMIT** 1;

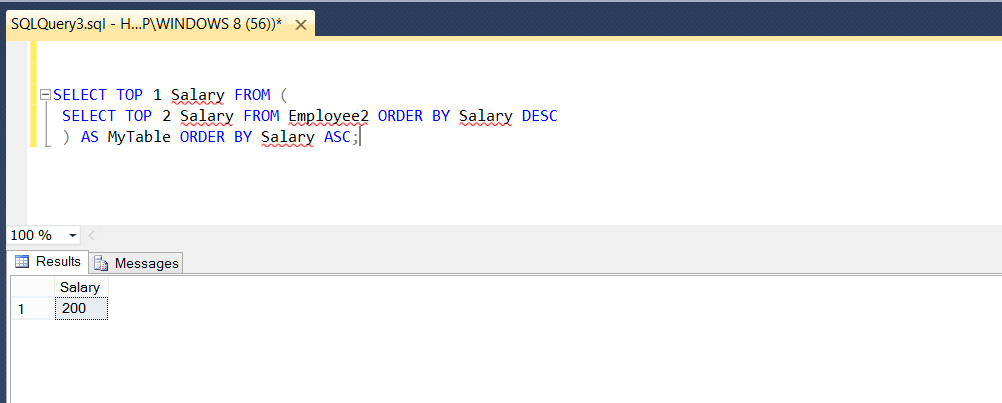
In this solution, we have first sorted all salaries form Employee table in decreasing order, so that 2 highest salaries come at top of the result set. After that we took just two records by using LIMIT 2. Again we did the same thing but this time we sort the result set on ascending order, so that second highest salary comes at top. Now we print that salary by using LIMIT 1. Simple and easy, right?

## Second Highest Salary using SQL Server Top Keyword

Just like MySQL has LIMIT keyword, which is immensely helpful in [sorting and paging](http://javarevisited.blogspot.sg/2012/12/oracle-pagination-sql-query-example-for-java.html), Microsoft SQL Server also has a special keyword called TOP, which as name suggest prints top records from result set. You can print top 10 records by saying TOP 10. I frequently use this keyword to see the data from a large table, just to understand columns and data inside it. Here is the SQL query to find second maximum salary in SQL Server :

**SELECT** TOP 1 Salary **FROM** ( **SELECT** TOP 2 Salary **FROM** Employee **ORDER BY** Salary **DESC**) **AS** MyTable **ORDER BY** Salary **ASC**;

Here is the output of above query running on Microsoft SQL Server 2014 :

[](http://2.bp.blogspot.com/-oy2kF1SVgx0/VLnwAewaY1I/AAAAAAAACYc/JxS5T73IXHo/s1600/How+to+find+Second+Highest+Salary+in+SQL+Server.png)

Now It's time to apply the knowledge you have learned so far. Solve following SQL queries at your convenience :

1. Write SQL query to get third highest salary from Employee table?
2. How do you find 4th highest salary in MySQL without using LIMIT keyword?
3. Write SQL query to find second highest salary in Oracle database using ROWNUM?
4. How to find Nth highest salary in SQL Server without using TOP keyword?
5. Find second highest salary in Oracle using rank?
6. How to find top 3 salary in Oracle without using ROW\_NUMBER or RANK()?

That's all about **different ways to find Second highest Salary in MySQL and SQL Server**.  We have seen examples to get second highest salary in MySQL by using LIMIT and without using LIMIT. Similarly in MSSQL we know how to get second highest salary by using TOP and without using TOP keyword. I have left the Oracle database for you as an exercise. If you able to find solution of all above SQL queries in quick time and feeling bore again, checkout my post about [Top 20 SQL queries from Interviews](http://java67.blogspot.sg/2013/04/10-frequently-asked-sql-query-interview-questions-answers-database.html) for some more fun.  
  
**P.S.** - If you are looking for online training/course to learn SQL from scratch, I suggest you joining [Introduction to SQL by Jon Flanders](http://www.shareasale.com/m-pr.cfm?merchantID=53701&userID=880419&productID=546411704). It's one of the best sourse to learn SQL fundamentals e.g. join, subquery, aggregate functions, window functions, gropuing data, advanced filtering and SQL query optimization.

One of the most common SQL interview questions is to find the Nth highest salary of employee, where N could be 2, 3, 4 or anything e.g. find the [second highest salary in SQL](http://java67.blogspot.com/2015/01/second-highest-salary-in-mysql-and-sql-server.html). Sometimes this question is also twisted as to find the *nth minimum salary in SQL*. Since many Programmers only know the easy way to solve this problem e.g. by using SQL IN clause, which doesn't scale well, they struggle to write the SQL query when Interviewer keep asking about 4th highest, 5th highest and so on. In order to solve this problem effectively, you need to know about some key concepts like correlated subquery, window functions like **ROW\_NUMER()**, **RANK()** and **DENSE\_RANK()**etc. Once you know the generic logic to solve this problem, you can tackle all those variations by yourself.  
  
  
In this article, I'll show you 4 ways to solve this problem e.g. by using the correlated subquery, using **ROW\_NUMBER()**, using TOP in SQL SERVER and by using LIMIT keyword in MySQL. Btw, if you are new to SQL and just learning these query from interviews sake then I suggest you to first read a good book on SQL e.g. [Head First SQL](http://www.amazon.com/Head-First-SQL-Brain-Learners/dp/0596526849?tag=javamysqlanta-20). It will help you to build your fundamentals.

## Sample table and data for Nth Highest Salary Problem

Before solving this problem we need some sample data to visualize the problem better, let's create employee table with some data.  
  
Use below query to create table and build schema:

-- creating Employee table in Oracle

CREATE TABLE Employee (name varchar(10), salary int);

-- inserting sample data into Employee table

INSERT INTO Employee VALUES ('Rick', 3000);

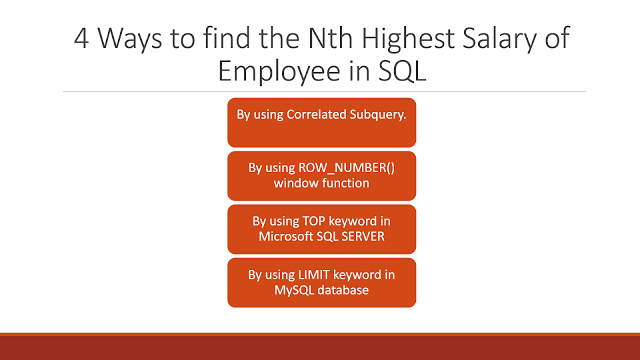
INSERT INTO Employee VALUES ('John', 4000);

INSERT INTO Employee VALUES ('Shane', 3000);

INSERT INTO Employee VALUES ('Peter', 5000);

INSERT INTO Employee VALUES ('Jackob', 7000);

## Nth highest salary using correlated subquery

[](http://www.amazon.com/Head-First-SQL-Brain-Learners/dp/0596526849?tag=javamysqlanta-20)

One of the most common ways to solve this problem of finding the Nth maximum salary from Employee table is by using the [correlated subquery](http://javarevisited.blogspot.com/2012/07/subquery-example-in-sql-correlated-vs.html). This is a special type of subquery where the subquery depends upon the main query and execute for every row returned by the main query.  It's slow but it can solve problems which are difficult to solve otherwise. Let's see the *SQL query to find the Nth highest salary using the Correlated subquery*.  
  
**SQL Query:**

SELECT name, salary

FROM #Employee e1

WHERE N-1 = (SELECT COUNT(DISTINCT salary) FROM #Employee e2

WHERE e2.salary > e1.salary)

for the 2nd maximum you can replace N with 2, and for 3rd maximum replace N with 3, here is the output:  
  
**2nd highest salary:**

SELECT name, salary

FROM #Employee e1

WHERE N-1 = (SELECT COUNT(DISTINCT salary) FROM #Employee e2

WHERE e2.salary > e1.salary)SELECT name, salary

FROM #Employee e1

WHERE 2-1 = (SELECT COUNT(DISTINCT salary) FROM #Employee e2

WHERE e2.salary > e1.salary)

Result:

name salary

Peter 5000

**3rd highest salary:**

SELECT name, salary

FROM #Employee e1

WHERE 3-1 = (SELECT COUNT(DISTINCT salary) FROM #Employee e2

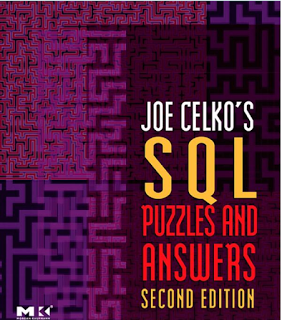
WHERE e2.salary > e1.salary)

Result:

name salary

John 4000

**Explanation :**  
The **distinct**keyword is there to deal with duplicate salaries in the table. In order to find the Nth highest salary, we are only considering unique salaries. Highest salary means no salary is higher than it, Second highest means only one salary is higher than it, 3rd highest means two salaries are higher than it, similarly Nth highest salary means N-1 salaries are higher than it.  
  
**Pros :**  
1) The generic solution works in all database including Oracle, MySQL, SQL SERVER and PostgreSQL.  
  
**Cons :**  
1) Slow, because the inner query will run for every row processed by the outer query.  
  
See [SQL Puzzles and Answers](http://www.amazon.com/Puzzles-Answers-Edition-Kaufmann-Management/dp/0123735963?tag=javamysqlanta-20) book for more of such SQL queries for practicing and improving your SQL query skill.

[](http://www.amazon.com/Puzzles-Answers-Edition-Kaufmann-Management/dp/0123735963?tag=javamysqlanta-20)

## Nth highest salary in SQL SERVER using TOP keyword

You can use the TOP keyword to find the Nth highest salary in SQL SERVER. This is also faster than the previous solution because here we are calculating Nth maximum salary without a subquery.

SELECT TOP 1 salary

FROM (

SELECT DISTINCT TOP N salary

FROM #Employee

ORDER BY salary DESC

) AS temp

ORDER BY salary

**Explanation:**  
By default ORDER BY clause print rows in ascending order, since we need the highest salary at the top, we have used ORDER BY DESC, which will display salaries in descending order. Again DISTINCT is used to remove duplicates. The outer query will then pick the top most salary, which would be your Nth highest salary.  
  
3rd highest salary in SQL SERVER

SELECT TOP 1 salary

FROM (

SELECT DISTINCT TOP 3 salary

FROM #Employee

ORDER BY salary DESC

) AS temp

ORDER BY salary

Result:

salary

4000

Here is [another example](http://javarevisited.blogspot.com/2012/12/how-to-find-second-highest-or-maximum-salary-sql.html)where we have used the TOP keyword to find the second highest salary in Microsoft SQL SERVER 2008.

## Nth maximum salary in MySQL using LIMIT keyword

Similar to TOP, MySQL also supports a LIMIT keyword, which provides pagination capability. You can find the nth highest salary in MySQL without using subquery as shown below:

SELECT salary FROM Employee ORDER BY salary DESC LIMIT N-1, 1

2nd highest salary in MySQL without subquery:

SELECT salary FROM Employee ORDER BY salary DESC LIMIT 1,1

salary

5000

3rd highest salary in MySQL using LIMIT clause:

SELECT salary FROM Employee ORDER BY salary DESC LIMIT 2,1

salary

4000

Nth highest salary in MySQL using LIMIT clause:

SELECT salary FROM Employee ORDER BY Salary DESC LIMIT n-1,1

**Explanation :**  
The benefit of this approach is that it's faster than correlated query approach but its vendor dependent. This solution will only work in MySQL database.

## Nth highest salary in Oracle using ROW\_NUMBER() function

SELECT \* FROM (

SELECT e.\*,

ROW\_NUMBER() OVER (ORDER BY salary DESC) rn

FROM Employee e

)

WHERE rn = N; /\*N is the nth highest salary\*/

Here is the [2nd highest salary in Oracle using ROW\_NUMBER()](http://javarevisited.blogspot.com/2015/11/2nd-highest-salary-in-oracle-using-rownumber-rank-example.html) window function:

SELECT \* FROM (

SELECT e.\*,

ROW\_NUMBER() OVER (ORDER BY salary DESC) rn

FROM Employee e

)

WHERE rn = 2;

Output

NAME SALARY RN

Peter 5000 2

and here is 3rd highest salary in Oracle:

SELECT \* FROM (

SELECT e.\*,

ROW\_NUMBER() OVER (ORDER BY salary DESC) rn

FROM Employee e

)

WHERE rn = 3;

By the above code has a problem. It is not handling duplicate salaries properly. For example, in our table we have two employees with salary 3000, that's our 4th highest salary, but above code will print the same salary, albeit different employee for both 4th and 5th maximum as shown below:

SELECT \* FROM (

SELECT e.\*,

ROW\_NUMBER() OVER (ORDER BY salary DESC) rn

FROM Employee e

)

WHERE rn = 5;

Result:

NAME SALARY RN

Shane 3000 5

In oracle, you can also use SQL statements to build schema and run sample SQL.  
  
You can also do the same thing by using **RANK()** window function in Oracle, but that's for another day. This is more than enough to answer the SQL interview question, the print nth highest salary of an employee in the Oracle.  
  
  
That's all about **how to find the nth highest salary in SQL**. The easiest way to find nth maximum/minimum salary is by using the correlated subquery, but it's not the fastest way. Better ways are database dependent e.g. you cause TOP keyword in SQL SERVER, LIMIT keyword in MySQL and ROW\_NUMBER() window function in Oracle to calculate the nth highest salary. The normal subquery way is good for the second maximum but after that, it become really nested and cluttered.

SQL subqueries are basic tools if you want to communicate effectively with relational databases. In this article, I provide five subquery examples demonstrating how to use scalar, multirow, and correlated subqueries in the WHERE, FROM/JOIN, and SELECT clauses.

A **subquery**, or nested query, is a query placed within another SQL query. When requesting information from a database, you may find it necessary to include a subquery into the SELECT, FROM , JOIN, or WHERE clause. However, you can also [use subqueries when updating the database](https://learnsql.com/blog/subqueries-in-insert-update-delete-statements/) (i.e. in INSERT, UPDATE, and DELETE statements).

There are several types of SQL subqueries:

* **Scalar subqueries** return a single value, or exactly one row and exactly one column.
* **Multirow subqueries** return either:
  + One column with multiple rows (i.e. a list of values), or
  + Multiple columns with multiple rows (i.e. tables).
* **Correlated subqueries**, where the inner query relies on information obtained from the outer query.

You can read more about the [different types of SQL subqueries](https://learnsql.com/blog/sql-subquery-types/) elsewhere; here, I want to focus on examples. As we all know, it’s always easier to grasp new concepts with real-world use cases. So let’s get started.

If you’re interested in learning more about SQL but have no prior knowledge of programming or databases, take a look at our [SQL Basics](https://learnsql.com/course/sql-queries?itm_source=lsqlBlog&itm_campaign=_default&itm_medium=text&itm_content=course-sql-queries-7) course.

## 5 Subquery Examples in SQL

Let’s say we run an art gallery. We have a database with four tables: paintings, artists, collectors, and sales. You can see the data stored in each table below.

| **paintings** | | | |
| --- | --- | --- | --- |
| id | name | artist\_id | listed\_price |
| 11 | Miracle | 1 | 300.00 |
| 12 | Sunshine | 1 | 700.00 |
| 13 | Pretty woman | 2 | 2800.00 |
| 14 | Handsome man | 2 | 2300.00 |
| 15 | Barbie | 3 | 250.00 |
| 16 | Cool painting | 3 | 5000.00 |
| 17 | Black square #1000 | 3 | 50.00 |
| 18 | Mountains | 4 | 1300.00 |

| **artists** | | |
| --- | --- | --- |
| id | first\_name | last\_name |
| 1 | Thomas | Black |
| 2 | Kate | Smith |
| 3 | Natali | Wein |
| 4 | Francesco | Benelli |

| **collectors** | | |
| --- | --- | --- |
| id | first\_name | last\_name |
| 101 | Brandon | Cooper |
| 102 | Laura | Fisher |
| 103 | Christina | Buffet |
| 104 | Steve | Stevenson |

| **sales** | | | | | |
| --- | --- | --- | --- | --- | --- |
| id | date | painting\_id | artist\_id | collector\_id | sales\_price |
| 1001 | 2021-11-01 | 13 | 2 | 104 | 2500.00 |
| 1002 | 2021-11-10 | 14 | 2 | 102 | 2300.00 |
| 1003 | 2021-11-10 | 11 | 1 | 102 | 300.00 |
| 1004 | 2021-11-15 | 16 | 3 | 103 | 4000.00 |
| 1005 | 2021-11-22 | 15 | 3 | 103 | 200.00 |
| 1006 | 2021-11-22 | 17 | 3 | 103 | 50.00 |

Now let’s explore this data using SQL queries with different types of subqueries.

### Example 1 - Scalar Subquery

We’ll start with a simple example: We want to list paintings that are priced higher than the average. Basically, we want to get painting names along with the listed prices, but only for the ones that cost more than average. That means that we first need to find this average price; here’s where the scalar subquery comes into play:

|  |
| --- |
| SELECT name, listed\_price  FROM paintings  WHERE listed\_price > (      SELECT AVG(listed\_price)      FROM paintings  ); |

Our subquery is in the WHERE clause, where it filters the result set based on the listed price. This subquery returns a single value: the average price per painting for our gallery. Each listed price is compared to this value, and only the paintings that are priced above average make it to the final output:

| **name** | **listed\_price** |
| --- | --- |
| Pretty woman | 2800.00 |
| Handsome man | 2300.00 |
| Cool painting | 5000.00 |

If this seems a bit complicated, you may want to check out our interactive [SQL Basics](https://learnsql.com/course/sql-queries) course and brush up on your essential SQL skills.

### Examples 2 – Multirow Subquery

Now let’s look into subqueries that return one column with multiple rows. These subqueries are often included in the WHERE clause to filter the results of the main query.

Suppose we want to list all collectors who purchased paintings from our gallery. We can get the necessary output using a multirow subquery. Specifically, we can use an inner query to list all collectors’ IDs present in thesalestable – these would be IDs corresponding to collectors who made at least one purchase with our gallery. Then, in the outer query, we request the first name and last name of all collectors whose ID is in the output of the inner query. Here’s the code:

|  |
| --- |
| SELECT first\_name, last\_name  FROM collectors  WHERE id IN (      SELECT collector\_id      FROM sales  ); |

And here’s the output:

| **first\_name** | **last\_name** |
| --- | --- |
| Laura | Fisher |
| Christina | Buffet |
| Steve | Stevenson |

Interestingly, we could get the same result without a subquery by using an INNER JOIN (or just JOIN). This join type returns only records that can be found in both tables. So, if we join the collectors and the salestables, we’ll get a list of collectors with corresponding records in the sales table. **Note:** I have also used the DISTINCT keyword here to remove duplicates from the output.

Here’s the query:

|  |
| --- |
| SELECT DISTINCT collectors.first\_name, collectors.last\_name  FROM collectors  JOIN sales    ON collectors.id = sales.collector\_id; |

You can read more about [choosing subquery vs. JOIN](https://learnsql.com/blog/subquery-vs-join/) elsewhere in our blog.

### Example 3 – Multirow Subquery with Multiple Columns

When a subquery returns a table with multiple rows and multiple columns, that subquery is usually found in the FROM or JOIN clause. This allows you to get a table with data that was not readily available in the database (e.g. grouped data) and then join this table with another one from your database, if necessary.

Let’s say that we want to see the total amount of sales for each artist who has sold at least one painting in our gallery. We may start with a subquery that draws on the sales table and calculates the total amount of sales for each artist ID. Then, in the outer query, we combine this information with the artists’ first names and last names to get the required output:

|  |
| --- |
| SELECT    artists.first\_name,    artists.last\_name,    artist\_sales.sales  FROM artists  JOIN (      SELECT artist\_id, SUM(sales\_price) AS sales      FROM sales      GROUP BY artist\_id    ) AS artist\_sales    ON artists.id = artist\_sales.artist\_id; |

We **assign a meaningful alias to the output of our subquery** (artist\_sales). This way, we can easily refer to it in the outer query, when selecting the column from this table, and when defining the join condition in the ON clause. **Note:** Databases will throw an error if you don't provide an alias for your subquery output.

Here’s the result of the query:

| **first\_name** | **last\_name** | **sales** |
| --- | --- | --- |
| Thomas | Black | 300 |
| Kate | Smith | 4800 |
| Natali | Wein | 4250 |

So, within one short SQL query, we were able to calculate the total sales for each artist based on the raw data from one table (sales), and then join this output with the data from another table (artists).

Subqueries can be quite powerful when we need to combine information from multiple tables. Let’s see what else we can do with subqueries.

### Example 4 – Correlated Subquery

The following example will demonstrate how subqueries:

* Can be used in the SELECT clause, and
* Can be correlated (i.e. the main or outer query relies on information obtained from the inner query).

For each collector, we want to calculate the number of paintings purchased through our gallery. To answer this question, we can use a subquery that counts the number of paintings purchased by each collector. Here’s the entire query:

|  |
| --- |
| SELECT    first\_name,    last\_name,    (      SELECT count(\*) AS paintings      FROM sales      WHERE collectors.id = sales.collector\_id    )  FROM collectors; |

Notice how the inner query in this example actually runs for each row of the collectors table:

* The subquery is placed in the SELECT clause because we want to have an additional column with the number of paintings purchased by the corresponding collector.
* For each record of the collectors table, the inner subquery calculates the total number of paintings purchased by a collector with the corresponding ID.

Here’s the output:

| **first\_name** | **last\_name** | **paintings** |
| --- | --- | --- |
| Brandon | Cooper | 0 |
| Laura | Fisher | 2 |
| Christina | Buffet | 3 |
| Steve | Stevenson | 1 |

As you see, the output of the subquery (i.e. the number of paintings) is different for each record and depends on the output of the outer query (i.e. the corresponding collector). Thus, we are dealing with a correlated subquery here.

Check out [this guide](https://learnsql.com/blog/correlated-sql-subquery-5-minutes/) if you want to learn how to write correlated subqueries in SQL. For now,  let’s have one more correlated subquery example.

### Example 5 – Correlated Subquery

This time, we want to show the first names and the last names of the artists who had zero sales with our gallery. Let’s try to accomplish this task using a correlated subquery in the WHERE clause:

|  |
| --- |
| SELECT first\_name, last\_name  FROM artists  WHERE NOT EXISTS (    SELECT \*    FROM sales    WHERE sales.artist\_id = artists.id  ); |

Here is what's going on in this query:

* The outer query lists basic information on the artists, first checking if there are corresponding records in the sales
* The inner query looks for records that correspond to the artist ID that is currently being checked by the outer query.
* If there are no corresponding records, the first name and the last name of the corresponding artist are added to the output:

| **first\_name** | **last\_name** |
| --- | --- |
| Francesco | Benelli |

In our example, we have only one artist without any sales yet. Hopefully, he’ll land one soon.

Learn how to create one query temporary tables (known as Common Table Expressions) with our [Recursive Queries](https://learnsql.com/course/common-table-expressions?itm_source=lsqlBlog&itm_campaign=_default&itm_medium=text&itm_content=course-common-table-expressions-3) course.

## It’s Time to Practice SQL Subqueries!

In this article, I’ve covered several SQL subquery examples to give you a general understanding of how subqueries can be leveraged in SQL. However, often [common table expressions (CTEs) can do better than subqueries](https://learnsql.com/blog/reasons-to-use-ctes/).

If you want to practice SQL subqueries and other foundational SQL topics, try our [SQL Basics](https://learnsql.com/course/sql-queries) interactive course. It includes 129 coding challenges on querying multiple tables, aggregating and grouping data, joining tables, writing subqueries, and much more.

Want to become a SQL master? Check out our [SQL From A to Z](https://learnsql.com/track/sql-from-a-to-z) learning track. It goes beyond the basics and includes 7 interactive courses covering standard SQL functions, SQL data manipulation language (DML), basic SQL reports, window functions, common table expressions (CTEs), and GROUP BY extensions.

Thanks for reading, and happy learning!