MySQL Subquery

## **Introduction to the MySQL Subquery**

A MySQL subquery is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE. Also, a subquery can be nested within another subquery.

A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

For example, the following query uses a subquery to return the employees who work in the offices located in the USA.

SELECT

lastName, firstName

FROM

employees

WHERE

officeCode IN (SELECT

officeCode

FROM

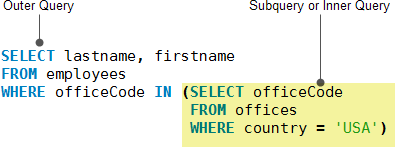
offices

WHERE

country = 'USA');

In this example:

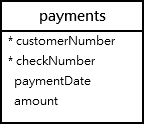
* The subquery returns all office codes of the offices located in the USA.
* The outer query selects the last name and first name of employees who work in the offices whose office codes are in the result set returned by the subquery.



When executing the query, MySQL evaluates the subquery first and uses the result of the subquery for the outer query.

## **Using a MySQL subquery in the WHERE clause**

We will use the table payments in the sample database for the demonstration.



### **MySQL subquery with comparison operators**

You can use comparison operators e.g., =, >, < to compare a single value returned by the subquery with the expression in the WHERE clause.

For example, the following query returns the customer who has the highest payment.

SELECT

customerNumber,

checkNumber,

amount

FROM

payments

WHERE

amount = (SELECT MAX(amount) FROM payments);

mysql subquery with equal operator

Besides the = operator, you can use other comparison operators such as greater than (>), greater than or equal to (>=) less than(<), and less than or equal to (<=).

For example, you can find customers whose payments are greater than the average payment using a subquery:

SELECT

customerNumber,

checkNumber,

amount

FROM

payments

WHERE

amount > (SELECT

AVG(amount)

FROM

payments);

### **mysql subquery with greater than operator**

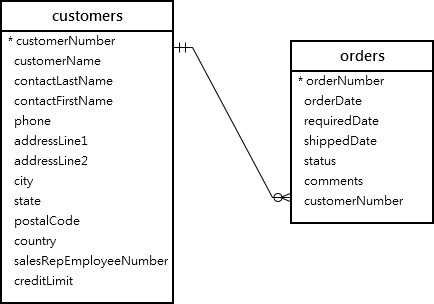
In this example:

* First, get the average payment by using a subquery.
* Then, select the payments that are greater than the average payment returned by the subquery in the outer query.

### **MySQL subquery with IN and NOT IN operators**

If a subquery returns more than one value, you can use other operators such as IN or NOT IN operator in the WHERE clause.

See the following customers and orders tables:



For example, you can use a subquery with NOT IN operator to find the customers who have not placed any orders as follows:

SELECT

customerName

FROM

customers

WHERE

customerNumber NOT IN (SELECT DISTINCT

customerNumber

FROM

orders);

### **mysql subquery not in**

## **MySQL subquery in the FROM clause**

When you use a subquery in the FROM clause, the result set returned from a subquery is used as a temporary table. This table is referred to as a derived table or materialized subquery.

The following subquery finds the maximum, minimum, and average number of items in sale orders:

SELECT

MAX(items),

MIN(items),

FLOOR(AVG(items))

FROM

(SELECT

orderNumber, COUNT(orderNumber) AS items

FROM

orderdetails

GROUP BY orderNumber) AS lineitems;

mysql subquery from clause example

Note that the FLOOR() is used to remove decimal places from the average values of items.

## **MySQL correlated subquery**

In the previous examples, you notice that a subquery is independent. It means that you can execute the subquery as a standalone query, for example:

SELECT

orderNumber,

COUNT(orderNumber) AS items

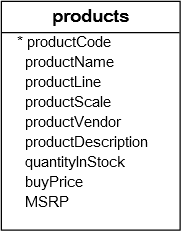
FROM

orderdetails

GROUP BY orderNumber;

Unlike a standalone subquery, a correlated subquery is a subquery that uses the data from the outer query. In other words, a correlated subquery depends on the outer query. A correlated subquery is evaluated once for each row in the outer query.

See the following products table from the sample database:



The following example uses a correlated subquery to select products whose buy prices are greater than the average buy price of all products in each product line.

SELECT

productname,

buyprice

FROM

products p1

WHERE

buyprice > (SELECT

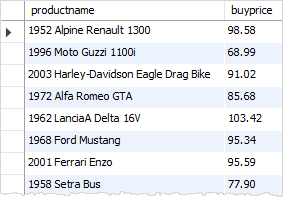
AVG(buyprice)

FROM

products

WHERE

productline = p1.productline);



In this example, both outer query and correlated subquery reference the same products table. Therefore, we need to use a table alias p1 for the products table in the outer query.

Unlike a regular subquery, you cannot execute a correlated subquery independently like this. If you do so, MySQL doesn’t know the p1 table and will issue an error.

SELECT

AVG(buyprice)

FROM

products

WHERE

productline = p1.productline;

For each row in the products (or p1) table, the correlated subquery needs to execute once to get the average buy price of all products in the productline of that row.

If the buy price of the current row is greater than the average buy price returned by the correlated subquery, the query includes the row in the result set.

### **MySQL subquery with EXISTS and NOT EXISTS**

When a subquery is used with the EXISTS or NOT EXISTS operator, a subquery returns a Boolean value of TRUE or FALSE.  The following query illustrates a subquery used with the EXISTS operator:

SELECT

\*

FROM

table\_name

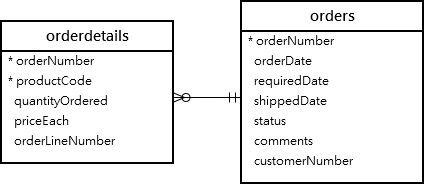
WHERE

EXISTS( subquery );

In the query above, if the subquery returns any rows, EXISTS subquery returns TRUE, otherwise, it returns FALSE.

The EXISTS and NOT EXISTS are often used in the correlated subqueries.

Let’s take a look at the orders and orderdetails tables from the sample database:



The following query finds sales orders whose total values are greater than 60K.

SELECT

orderNumber,

SUM(priceEach \* quantityOrdered) total

FROM

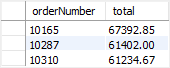
orderdetails

INNER JOIN

orders USING (orderNumber)

GROUP BY orderNumber

HAVING SUM(priceEach \* quantityOrdered) > 60000;



It returns 3 rows, meaning that there are three sales orders whose total values are greater than 60K.

You can use the query above as a correlated subquery to find customers who placed at least one sales order with the total value greater than 60K by using the EXISTS operator:

SELECT

customerNumber,

customerName

FROM

customers

WHERE

EXISTS( SELECT

orderNumber, SUM(priceEach \* quantityOrdered)

FROM

orderdetails

INNER JOIN

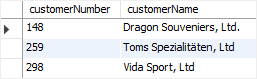
orders USING (orderNumber)

WHERE

customerNumber = customers.customerNumber

GROUP BY orderNumber

HAVING SUM(priceEach \* quantityOrdered) > 60000);



## **Summary**

* A subquery is a query nested within another query (or outer query).
* A correlated subquery depends on the outer query.