Unknown date
Unknown author

MySQL Subquery

Summary: in this tutorial, we will show you how to use the MySQL subquery to write complex queries and explain the correlated subquery concept.

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

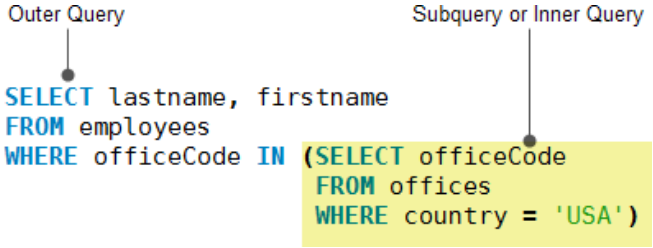
A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

The following query returns employees who work in offices located in the USA.

```
SELECT
    lastName, firstName
FROM
    employees
WHERE
    officeCode IN (SELECT
            officeCode
        FROM
            offices
        WHERE
            country = 'USA');
```

In this example:

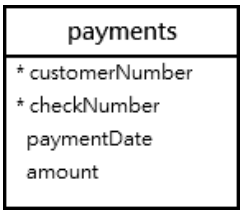
- The subquery returns all *office codes* of the offices located in the USA.
- The outer query selects the last name and first name of employees who work in the offices whose office codes are in the result set returned by the subquery.



When the query is executed, the subquery runs first and returns a result set. Then, this result set is used as an input for the outer query.

MySQL subquery in `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 maximum payment.

```
SELECT
    customerNumber,
    checkNumber,
    amount
FROM
    payments
WHERE
    amount = (SELECT MAX(amount) FROM payments);
```

Try It Out

	customerNumber	checkNumber	amount
▶	141	JE105477	120166.58

In addition to the equality operator, you can use other comparison operators such as greater than (>), less than(<).

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);
```

Try It Out

	customerNumber	checkNumber	amount
▶	112	HQ55022	32641.98
	112	ND748579	33347.88
	114	GG31455	45864.03
	114	MA765515	82261.22
	114	NR27552	44894.74
	119	LN373447	47924.19
	119	NG94694	49523.67
	121	DB889831	50218.95
	121	MA302151	34638.14

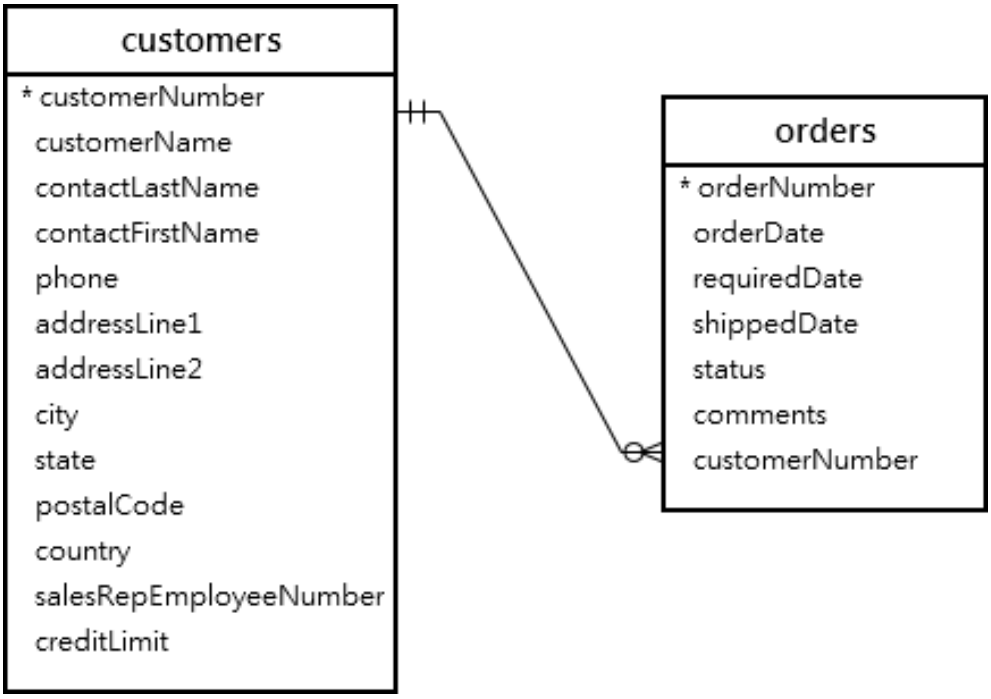
In this example:

- First, use a subquery to calculate the average payment using the `AVG` aggregate function.
- Then, query 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);
```

Try It Out

	customername
▶	Havel & Zbyszek Co
	American Souvenirs Inc
	Porto Imports Co.
	Asian Shopping Network, Co
	Natürlich Autos
	ANG Resellers
	Messner Shopping Network
	Franken Gifts, Co
	BG&E Collectables

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
```

```
10      GROUP BY orderNumber) AS lineitems;
```

Try It Out

	MAX(items)	MIN(items)	FLOOR(AVG(items))
▶	18	1	9

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:

```
1 SELECT
2     orderNumber,
3     COUNT(orderNumber) AS items
4 FROM
5     orderdetails
6 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.

In the following query, we select products whose buy prices are greater than the average buy price of all products in each product line.

```
1   SELECT
2       productname,
3       buyprice
4   FROM
5       products p1
6   WHERE
7       buyprice > (SELECT
8               AVG(buyprice)
9           FROM
10              products
11          WHERE
12              productline = p1.productline)
```

Try It Out

	productname	buyprice
▶	1952 Alpine Renault 1300	98.58
	1996 Moto Guzzi 1100i	68.99
	2003 Harley-Davidson Eagle Drag Bike	91.02
	1972 Alfa Romeo GTA	85.68
	1962 LanciaA Delta 16V	103.42
	1968 Ford Mustang	95.34
	2001 Ferrari Enzo	95.59
	1958 Setra Bus	77.90

The inner query executes for every product line because the product line is changed for every row. Hence, the average buy price will also change. The outer query filters only products whose buy price is greater than the average buy price per product line from the subquery.

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:

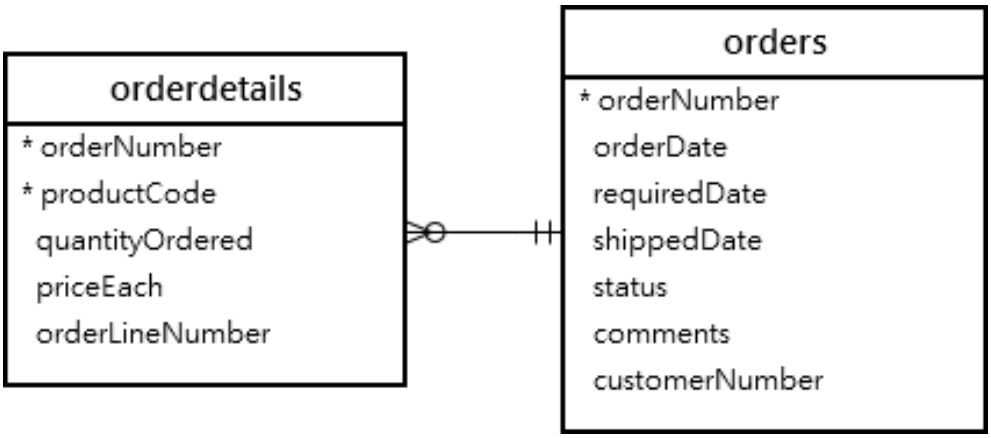
```
1 SELECT
```

```
2    *
3 FROM
4    table_name
5 WHERE
6    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.

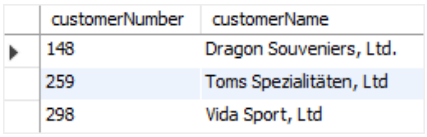
```
1 SELECT
2    orderNumber,
3    SUM(priceEach * quantityOrdered) total
4 FROM
5    orderdetails
6        INNER JOIN
7    orders USING (orderNumber)
8 GROUP BY orderNumber
9 HAVING SUM(priceEach * quantityOrdered) > 60000;
```

orderNumber	total
10165	67392.85
10287	61402.00
10310	61234.67

It returns 3 rows, meaning that there are 3 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:

```
1   SELECT
2      customerNumber,
3      customerName
4   FROM
5      customers
6   WHERE
7      EXISTS( SELECT
8            orderNumber, SUM(priceEach * quantityOrdered)
9         FROM
10           orderdetails
11              INNER JOIN
12           orders USING (orderNumber)
13        WHERE
14           customerNumber = customers.customerNumber
15        GROUP BY orderNumber
16        HAVING SUM(priceEach * quantityOrdered) > 60000);
```



customerNumber	customerName
148	Dragon Souveniers, Ltd.
259	Toms Spezialitäten, Ltd
298	Vida Sport, Ltd

In this tutorial, we have shown you how to use MySQL subquery and correlated subquery to construct more complex queries.

- Was this tutorial helpful?
- YesNo
