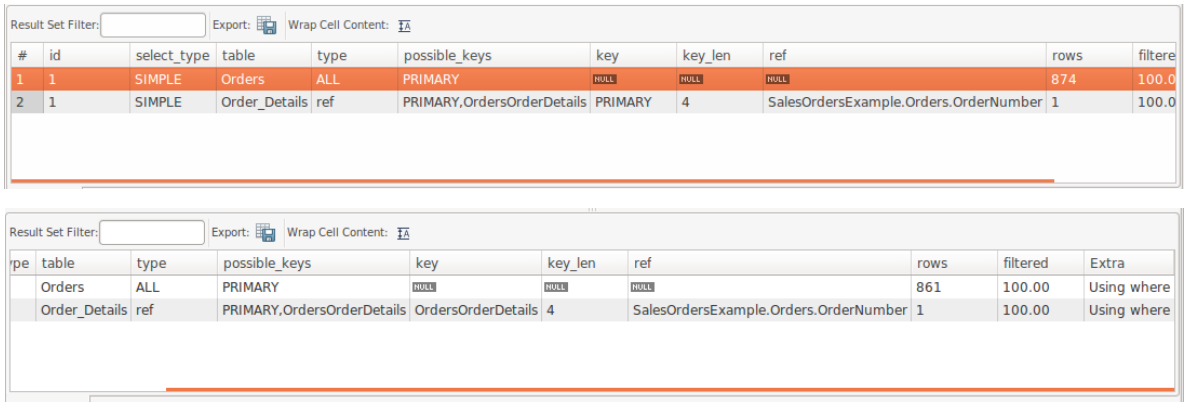
Subtask 10.2.1

```
EXPLAIN EXTENDED
SELECT * FROM Orders NATURAL JOIN Order_Details WHERE QuotedPrice > 1000 AND
OrderDate BETWEEN '2012-10-01' AND '2012-10-31';
```



Result Set Filter: Export: Wrap Cell Content:

#	id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtere
1	1	SIMPLE	Orders	ALL	PRIMARY	NULL	NULL	NULL	874	100.0
2	1	SIMPLE	Order_Details	ref	PRIMARY,OrdersOrderDetails	PRIMARY	4	SalesOrdersExample.Orders.OrderNumber	1	100.0

Result Set Filter: Export: Wrap Cell Content:

pe	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	Orders	ALL	PRIMARY	NULL	NULL	NULL	861	100.00	Using where
	Order_Details	ref	PRIMARY,OrdersOrderDetails	OrdersOrderDetails	4	SalesOrdersExample.Orders.OrderNumber	1	100.00	Using where

Describe in your own words how the DBMS is fetching the rows.

It finds all the October Orders rows, then filters the order details rows according to the two conditions at the same time. They match the chosen Order rows and whether they have a quoted price of over 1000. Applying both restrictions means each row in the OrderDetails table has to be looked at only once.

For each table, does it look through the data rows or access them through an index?

- For the Orders table, it is searched sequentially, which is expensive.
- For the Order_Details, the DBMS uses an index on the key column to find the matching rows.

How does this influence the number of rows examined?

It would not require all tuples of both Orders and OrderDetails to be examined.

According to the output, which table was accessed first?

We bring the Orders row first, we can tell from the foreign-priamry key match which rows of OrderDetails we don't need. The restriction on the OrderDetails (the price) helps reduce the numbers further

Why do you think the DBMS decided to access this table first instead of the other one (both have restrictions)?

```
SELECT COUNT(*) FROM Orders WHERE OrderDate BETWEEN '2012-10-01' AND '2012-10-31';
```

#	count(*)
1	147

```
SELECT COUNT(*) FROM Order_Details WHERE QuotedPrice > 1000;
```

Result Set Filter:

#	COUNT(*)
1	734

We can use the count function to find out how many tuples matched each condition. We can see that the Orders table has a lower number of matching tuples. So the number of tuples that it would need to match in the Order Details is lower than the other way around.

If you need to find out which columns are accessed through an index, run

```
SHOW indexes in Order_Details;
```

#	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment
1	Order_Details	0	PRIMARY	1	OrderNumber	A	2294	NULL	NULL		BTREE	
2	Order_Details	0	PRIMARY	2	ProductNumber	A	4588	NULL	NULL		BTREE	
3	Order_Details	1	OrdersOrderDetails	1	OrderNumber	A	4588	NULL	NULL		BTREE	
4	Order_Details	1	ProductsOrderDetails	1	ProductNumber	A	99	NULL	NULL		BTREE	

The WHERE clause is very selective, so the DBMS decided to use indexes. The columns that are accessed through an index are OrderNumber and ProductNumber.