Nested Queries

Contents

Overview

Nested queries are a type of MySQL command that allows you to query information that has been returned from another query.

They are commonly used to return *unknown* values.

As you might expect, this involves *encapsulation*, or 'wrapping' a `SELECT` statement within another one.

The encapsulated nested query is run first - the data it returns is then 'passed' to the query wrapped around it.

For example, if we wanted to find customer information based on an order, we might write something like this:

```
SELECT customer_id, forename, city
FROM customers
WHERE customer_id=(
    SELECT customer_id
    FROM orders
    WHERE order_id=1
);
```

Here, we have two `SELECT` statements. The encapsulated query is run first, which returns the *unknown* `customer_id` of the customer who made the order with the `order_id` of `1`.

That unknown is then fed into the query that is 'wrapped' around it, which will give us the `customer_id`, `forename` and `city` data that we want.

Views

It is common practice to use a *view* within a database to 'save' the results of a query.

Views can be used for longer queries which may otherwise require several queries to be encapsulated.

A view acts very similarly to a table, with a key difference - a table is *dynamic*, whereas a view is a *snapshot*.

If you update the table(s) that the view was created from, then the content of that view doesn't change along with the table(s) - it just stays the same.

We can save the encapsulated query from the one we used earlier as a view by using the `CREATE VIEW` command:

```
CREATE VIEW result_of_encapsulated_query
AS
    SELECT customer_id
    FROM orders
    WHERE order_id=1
;
```

We could now query that view if we wanted to with a normal `SELECT`:

```
SELECT * from result_of_encapsulated_query;
```

Let's say this is the result of the above query:

customer_id
4

Now we've solved the *unknown* value - we know that the order with an `order_id` of `1` was made by the customer with the `customer_id` of `4`.

We can use that solved value to execute the query 'wrapped' around the outside from before.

We just ask it to `SELECT` by using that `4` we've found in our view instead:

```
SELECT customer_id, forename, city
FROM customers
WHERE customer_id=4;
```

Tutorial

There is no tutorial for this module.

Exercises

In your game shop database, try to work out how to retrieve the *entire record* for the most expensive game.

(*note: if you have not tested aggregate functions yet, refer back to the Aggregate Functions module for context*)

▶ Solution