In this lesson we will be looking at the NESTED SELECT and the NESTED SELECT is extremely useful because it lets us nest queries inside other queries.

Let’s see some examples as why NESTED SELECT is useful and how we may go about using it. Let’s say we want to find out what items we want to own in our business which are above the average price of our items. We can do this if we want to figure out if those items sell less. If we want to say it in simple English then it would be like below,

SELECT \* FROM items WHERE price > AVG(price);

It would have been nice if this query would have worked but unfortunately it won’t. Now some might think how about HAVING? Is HAVING not useful in this scenario and unfortunately HAVING is only used in GROUP BY and we do not have any GROUP BY here. We are just selecting from one table. That is why HAVING is not the answer. The answer is the NESTED SELECT.

We need to make sure that the price is grater than average price. So, how do we calculate the Average Price? We do it as below,

SELECT \* FROM items WHERE price >

(SELECT AVG(item.price) FROM items);

|  |  |  |
| --- | --- | --- |
| name  character varying (255) | id  integer | price  numeric |
| Screen | 5 | 275.50 |
| Laptop | 7 | 1500.00 |
| DSLR | 9 | 400.00 |

The query in the parentheses is calculating the average price of items and then it is comparing with the prices of each items and displaying the products those are higher than average price.

Its quite nice but using the WHERE command we can do more advanced filtering based on values of other tables or indeed the same table.

Now we will find out how much above or below of the average price they are. Just so they give us a bit of deviation from the average.

Now we will calculate the price difference of each items with the average price.

SELECT items.name, items.price –

(SELECT AVG(items.price) FROM items)

FROM items;

|  |  |
| --- | --- |
| name  character varying (255) | ?column?  numeric |
| Fountain Pen | -207.4081818181818182 |
| Screen | 56.7918181818181818 |
| Hard Drive | -128.7181818181818182 |
| Pen | -213.7081818181818182 |
| Ink | -213.7081818181818182 |
| Laptop | 1281.291818181818181818 |
| Laptop | -203.7081818181818182 |
| DSLR | 181.2918181818181818 |
| Pencil | -216.708181818181818 |
| Pencil | -118.7081818181818182 |
| Pencil | -216.7081818181818182 |

That how it is. Let’s create a VIEW for luxury items which also tells us the price difference from the average luxury item price. The items those are above $100 will be noted as luxury items. The first thing to do is find out the average selling price of items those are above $100.

SELECT AVG(items.price) FROM items WHERE price > 100;

|  |
| --- |
| avg  numeric |
| 725.166666666666667 |

This is the average price of our expensive items whose individual prices are above $100.

Now we want to select the expensive items and the difference between their price and the average price of expensive items.

SELECT \*, items.price –

(SELECT AVG(items.price) FROM items WHERE price > 100)

FROM items WHERE price > 100;

|  |  |  |  |
| --- | --- | --- | --- |
| name  character varying (255) | id  integer | price  numeric (10,2) | ?column?  numeric |
| Screen | 5 | 275.50 | -449.6666666666666667 |
| Laptop | 7 | 1500.00 | 774.8333333333333333 |
| DSLR | 9 | 400.00 | -325.1666666666666667 |

Here we are selecting all of the items of all the columns from items and we are also selecting *items.price* which is the price of the items. We will subtract the average price of items above $100 from the price of each items. The sub-query inside the parentheses is giving us the average price of items where each item cost above $100 and the last line with the FROM command is getting us those items those each cost above $100.

Finally, we can also save this query as a VIEW,

CREATE VIEW expensive\_items\_diff AS

SELECT \*, items.price –

(SELECT AVG(items.price) FROM items WHERE price > 100)

FROM items WHERE price > 100;

Our query runs successfully, which means our query got saved as a VIEW,

SELECT \* FROM expensive\_items\_diff

|  |  |  |  |
| --- | --- | --- | --- |
| name  character varying (255) | id  integer | price  numeric (10,2) | ?column?  numeric |
| Screen | 5 | 275.50 | -449.6666666666666667 |
| Laptop | 7 | 1500.00 | 774.8333333333333333 |
| DSLR | 9 | 400.00 | -325.1666666666666667 |

Notice that we did not set a name of the last column which is not a best practice so let’s go change it.

DROP VIEW expensive\_items\_diff

CREATE VIEW expensive\_items\_diff AS

SELECT \*, items.price –

(SELECT AVG(items.price) FROM items WHERE price > 100) AS “average\_diff”

FROM items WHERE price > 100;

SELECT \* FROM expensive\_items\_diff

|  |  |  |  |
| --- | --- | --- | --- |
| name  character varying (255) | id  integer | price  numeric (10,2) | average\_diff  numeric |
| Screen | 5 | 275.50 | -449.6666666666666667 |
| Laptop | 7 | 1500.00 | 774.8333333333333333 |
| DSLR | 9 | 400.00 | -325.1666666666666667 |

Also notice that the *price* column was set to be (10,2) which is why the values in that column has two decimal places, whereas nothing was set for the *average\_diff* column which is why it has lot of decimal places. And that is how we create sub-queries.