1.4 Update Table data

Data A very useful operation is update previous inserted data. The syntax to perform this operation is the "Update Table". The syntax can be slight different in some situations like it will be presented in next examples.

In the first situation we will update just one field considering the value of other field in the same table.

```
Update Products Set unit_price = 1.99 Where description = 'Eggs';
```

After the execution of this script the unit_price of eggs is 1.99$. The others fields of the tables remains untouched.

In the second situation we will update two fields simultaneously.

```
Update Products Set available_stock = 25, minimal_stock = 10 Where description = 'Eggs';
```

After the execution of this script the available stock and minimal stock of eggs will be changed to 25 and 10, respectively.

In the third situation we will update again the price field, but this time for the soda product.

```
Update Products Set unit_price = unit_price * 1.25 Where description = 'Soda';
```

After the execution of this script the unit price of soda product will be increased by 25%.

Finally, in the fourth situation we will update the date delivery order of the Peter. For that, we will need to use and access two tables: "Orders" and "Customers".

```
Update Orders Set date_delivery = '2015-12-29' Where Orders.cod_customer = (Select cod_customer From Customers Where name='Peter');
```

Practical SQL Guide for Relational Databases 2016 Page 18 After the execution of this script, the delivery date of all orders made by Peter will be 29th December of 2015.

The above script doesn't work in MySQL databases. To make it work in MySQL we shall perform a slight modification.

```
Update Orders O, Customers C Set O.date_delivery = '2015-12-29' Where O.cod_customer = C.cod_customer and C.name='Peter';
```

MySQL needs that both tables are declared in the beginning of "Update" clause. Using this approach we don't anymore to use a sub-query to get match values with Customers table.

1.5 Delete data

The delete instruction in SQL is used to delete data from a table. The most basic syntax is to use this instruction to delete all data from a table. The syntax to perform this operation is exactly the same for all databases (MySQL, SQL Server and Oracle).

```
DELETE FROM Customers; or
DELETE * FROM Customers;
```

The two syntaxes presented are equivalent in terms of functionality and performance. It has the function to delete all data available in Customers table.

If we want to delete a number of elements that accomplishes a given rule, we can add the "Where" clause. We show here one example how to perform it involving only one table.

```
Delete From OrdersProducts Where cod_product = 5;
```

After the execution of the above script the table "OrdersCustomers" will not has any item with code of product equal to 5.

Now we will give two examples how to use the "Delete From..." operation using multiple tables and sub-queries.

```
Delete From OrdersProducts Where cod_order IN (Select cod_order From Orders Where cod_customer = 3);
```

This instruction deletes all itens from OrdersProducts which order was performed by cod_customer equal to 3. The above instruction involves calling OrdersProducts and Orders tables.

If we want to delete all itens from OrdersProducts that its products has no content for the description field we can adopt the approach below.

```
Delete From OrdersProducts Where cod_product IN (Select cod_product From Products Where description is NULL);
```

This instruction tries to delete from OrdersProducts all items which description of the product is null. This instruction is well formed, however it doesn't delete any data because the description field in table Products was declared as not null. Therefore at this point it won't be possible to find any empty description of product.

Finally SQL offers the "Truncate" command to removes all rows from a table. The operation cannot be rolled back and no triggers will be fired. As such, TRUCATE is faster and doesn't use as much undo space as a DELETE, but it should be used carefully. We give an example below how to use it.

```
TRUNCATE TABLE Customers;
```

This instruction would remove all data from Customers table. This instruction is not available in the script of database, because the database contains already orders associated to customers.

It is important to highlight that DROP (that will be seen in next chapter) and TRUNCATE are DDL commands, whereas DELETE is a DML command. As such, DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back.

1.6 Remove Tables

Remove a table from a database is one of the easiest operations in SQL. However, it shouldn't be mix with the instruction "delete from..." that deletes table. In order to remove a table from a database we should use the "Drop Table..." command. This instruction works in all three databases (MySQL, SQL Server and Oracle). DROP TABLE Customers;

This instruction tries to remove the table Customers. However, it is only possible to remove a table if the primary key of this table is not being used by other table. In our database, this instruction will not work because the primary key in table is reference by foreign keys. However, it would be totally possible to remove the "OrdersProducts" table from the system.

```
DROP TABLE OrdersProducts;
```

If used, this instruction would remove the OrdersProducts table from the system.

1.7 SQL Queries - Basic Structure

The basic structure of a SQL query is composed by the elements below.

```
SELECT field1 [,"field2",etc]
FROM table
[WHERE "condition"]
[GROUP BY "field"]
[ORDER BY "field"]
```

Only the two initial clauses are mandatory ("Select" and "From"). The others elements are optional. One of the most used queries in SQL is to show all the contents of a table. It can be used liked in the example below.

```
Select * From Customers;
```

This instruction shows all content of table Customers. It shows all attributes of the table. It is possible to choose only the fields to be shown, like in the example below.

```
Select Name, Country From Customers;
```

This instruction shows the content of fields Name and Country of table Customers.

If we only want to show the distinct country of customers, we can use the instruction "distinct" in Select clause, like it is shown below.

```
Select distinct Country From Customers;
```

This instruction shows the content of Country, but only distinct elements. In this situation only the country "Portugal" is shown.

We can use also the "Where" clause to restrict the elements of a table that will be shown. An example of this approach is shown below.

```
Select * From Products Where Available_stock is not NULL;
```

This instruction shows all products that have stock in warehouse. It is possible to use several conditions in a "Where" clause like it is shown below.

```
Select * From Products Where (Available_stock is NULL and unit_price > 1.00) or minimal_stock > 0;
```

This instruction shows all products which there is stock in warehouse and price is higher than 1€. It also includes in the result all field which defined minimal_stock is positive.

In the "From" clause we can use more than one table. We show this situation using two examples.

```
Select O.cod_order, O.date_order, O.DATE_DELIVERY, C.name From Orders O, Customers C Where O.cod_customer = C.COD_CUSTOMER;
```

This instruction shows the code of the order, date of order, date of delivery and name of the customer for all orders available in the database. It is particular relevant to look for the "Where" clause that is mandatory and it needs to be used to guarantee the join between the Orders and Customers table.

In the next example we use three tables in "From" clause.

```
Select OP.COD_PRODUCT, OP.QUANTITY, C.name From Orders O, Customers C, OrdersProducts OP Where O.cod_customer = C.COD_CUSTOMER and O.COD_ORDER = OP.COD_ORDER and O.COD_ORDER = 3;
```

This instruction shows the code of the code of products, quantity of each item and customer name for the order which code is equal to "3". This instruction needs to use the Orders, OrdersProducts and Customers tables. Like in the example before, in the "Where" clause we explicitly details the joint conditions between the attributes of these three tables.

Finally we demonstrate how to use SQL with some basic math operations. We start by using the operator "-" in the example below.

```
Select cod_product, description, available_stock - minimal_stock as marginStock From Products;
```

This instruction calculates the marginStock for all products and shows it with the code of product and description.