Practical SQL Guide for Relational Databases

Book · January 2016

1 author:

Fernando Almeida
Instituto Superior Politécnico Gaya
122 PUBLICATIONS **396** CITATIONS

Some of the authors of this publication are also working on these related projects:

Innovation and Entrepreneurship View project

UX in Web and Mobile Environments View project

Practical SQL Guide for Relational Databases

Fernando Almeida, PhD.

This book offers a short reference tutorial for database engineers and programmers that intends to learn SQL and use it in practice in a MySQL, SQL Server or Oracle databases. This book is organized in 20 chapters and includes an overview about Data Definition Language (DDL) and Data Modeling Language (DML) syntaxes. Each chapter presents some SQL code extracts with proper and argumentative discussion.

I N E S C   T E C

C a m p u s   d a   F E U P
R u a   D r .   R o b e r t o   F r i a s
4 2 0 0   -   4 6 5   P o r t o
P o r t u g a l

T e l .   + 3 5 1   2 2 2   0 9 4   0 0 0

F a x   + 3 5 1   2 2 2   0 9 4   0 5 0

1 / 1 5 / 2 0 1 6

Table of Contents

Index of Figures

Acronyms

CLI - Command-Line Interface

DCL - Data Control Language

DDL - Data Definition Language

DML - Data Modeling Language

I/O - Input/Output

ORA - Oracle

SQL - Structured Query Language

Glossary

MySQL - MySQL is an open source relational database management system (RDBMS) based on Structured Query Language (SQL).

Oracle - object-relational database management system produced and marketed by Oracle Corporation.

SQL Server - SQL Server is a relational database management system (RDBMS) from Microsoft that's designed for the enterprise environment.

1. Introduction

1.1 Contextualization

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

The SQL Standard has gone through a lot of changes during the years, which have added a great deal of new functionality to the standard, such as support for XML, triggers, regular expression matching, recursive queries, standardized sequences and much more.

The SQL language is based on several elements. For the convenience of SQL developers all necessary language commands in the corresponding database management systems are usually executed through a specific SQL command-line interface (CLI). These commands can be grouped in the following areas:

- Clauses - the clauses are components of the statements and the queries;

- Expressions - the expressions can produce scalar values or tables, which consist of columns and rows of data;

- Predicates - they specify conditions, which are used to limit the effects of the statements and the queries, or to change the program flow;

- Queries - a query retrieves data, based on a given criteria;

- Statements - using statements user can control transactions, program flow, connections, sessions, or diagnostics. In database systems the SQL statements are used for sending queries from a client program to a server where the databases are stored. In response, the server processes the SQL statements and returns to the client program. This allows users to execute a wide range of fast data manipulation operations from simple data inputs to more complex queries.

In order to guide students through the elementary syntax of SQL guide, a relational model is presented below.

Products (cod_product, description, unit_price, available_stock, minimal_stock)

Orders (cod_order, date_order, date_delivery, cod_customer -> Customers)

OrdersProducts (cod_product -> Products, cod_order -> Orders, quantity)

Customers (cod_customer, name, address, zip code, country, telephone)

This proposed relational model is normalized in 3NF and it is composed by four tables, which one identified by their primary keys (underlined) and the foreign keys are identified by "->" symbol.

The relational model describes generally a simple scenario of an ecommerce company that has several products and customer. Each customer can create new orders that typically is composed by several products.

1.2 Objectives

This mini books intends to provide a brief reference guide for undergraduate students that intend to learn SQL in the context of their curricular units at university. The book presents the main reference SQL syntax items (DDL and DML) and also presents the most important syntax instructions that should be adopted in order to build relational databases in MySQL, SQL Server or Oracle. This mini book doesn't intend to be a full SQL reference, but it only focus on most critical aspects of the SQL syntax. The DCL syntax is not looked in the context of this publication.

The scripts that will be shown in next chapters were testes using the following databases:

- MySQL 5.6;

- MS SQL Server 2014;

- Oracle Database 11g Release 2.

1.3 Book Structure

The book is organized in 20 chapters as follow:

- Chapter 1 "Introduction" - gives a brief overview about SQL language and organization of the book;

- Chapter 2 "Declaring Tables" - presents the syntax how to declare a new tables in relational databases;

- Chapter 3 "Insert Data" - presents the syntax in order to insert new data in a relational database;

- Chapter 4 "Update Data" - presents the syntax to update data previously stored in a relational database;

- Chapter 5 "Delete Data" - presents the syntax to delete information stored in a relational database;

- Chapter 6 "Remove Tables" - presents the syntax to remove a table from a relational database;

- Chapter 7 "SQL Queries - Basic Structure" - shows the basic structure how a SQL query is organized and structured;

- Chapter 8 "SQL Queries - Comparing Strings" - shows how to compare string contents using only SQL syntax;

- Chapter 9 "SQL Queries - Aggregation Operators" - shows the most relevant aggregation operators that are useful in SQL, such as AVG(), COUNT(), MAX(), MIN() and SUM();

- Chapter 10 "SQL Queries - Scalar Functions" - presents the most useful scalar functions that are available in SQL;

- Chapter 11 "SQL Queries - Grouping Elements" - presents the use of GROUP BY syntax in SQL;

- Chapter 12 "SQL Queries - Ordering Data" - presents the use of "ORDER BY" syntax in SQL;

- Chapter 13 "SQL Queries - Returning Top Elements" - presents an approach to return the top elements of a SQL statement;

- Chapter 14 "SQL Queries - Sub-queries" - shows the use of complex SQL queries adopting a sub-query approach;

- Chapter 15 "SQL Queries - Operator "IN" and "EXISTS" - shows the use of operator in and exists and how to convert an "IN" into "EXISTS" approach;

- Chapter 16 "SQL Queries - Operator "ANY" and "ALL" - shows the use of these two clauses in conjunction with sub-queries;

- Chapter 17 "SQL Queries - Operations with SETS" - shows the use of SQL in order to perform operations with sets;

- Chapter 18 "SQL Queries - Joins" - shows the use of different types of joins, namely inner joins, left outer joins, right outer joins and full outer joins;

- Chapter 19 "SQL Queries - Views" - shows the process of creating views in SQL and how to invoke them;

- Chapter 20 "SQL Queries - System Data" - shows the use of some system data operations in order to get the system date and the logged user;

- "Bibliography" - presents the adopted bibliography for this book;

- "Annex I - Script for MySQL Databases" - presents the full SQL script for a MySQL database;

- "Annex II - Script for MS SQL Server Databases" - presents the full SQL script for a MS SQL Server database;

- "Annex III - Script for Oracle Databases" - presents the full SQL script for an Oracle database.

2. Declaring Tables

The first step when creating a new database application is the process of declaring the tables. In our example we have 4 tables to be created: *Products*, *Orders*, *OrdersProducts* and *Customer*. We will start by the creation of the table Products.

The table Products records all the products that the company has in its catalogue. The script for the creation of this table is given below. The same script can be used to MySQL, SQL Server and Oracle DBMS.

```
create table Products
(
    cod_product    integer,
    description    varchar(50) NOT NULL,
    unit_price     DECIMAL(10,2),
    available_stock integer,
    minimal_stock  integer default 0,
    CONSTRAINT Products_pk PRIMARY KEY (cod_product)
);
```

The primary key is the "cod_products", which is declared in the last line. The description can has a maximum size of 50 characters and cannot be null. The field "unit_price" is declared has a decimal. The maximum number of digits of "unit_price" may be specified in the first parameter; the maximum number of digits to the right of the decimal point is specified in the second parameter. The field "minimal_stock" has a default value of 0, if this information is not given when the user inserts new data.

Then, we will create the table Customers. Like in previous example the same script can be used for all three databases (MySQL, SQL Server and Oracle).

```
create table Customers
(
    cod_customer    integer,
    name    varchar(50) NOT NULL,
    address     varchar(95) Default 'Unknown',
    zip_code char(8),
    country   varchar(40) Default 'Portugal',
    telephone varchar(15),
    CONSTRAINT Customers_pk PRIMARY KEY (cod_customer)
);
```

The primary key is the "cod_customer", which is declared in last line. The address is also a varchar type like name, but if no information is provided it will assume the default value of "unknown". The zip code is a char(8) and it will always reserve a char of size 8, independently of its content. The country of the customer will assume the default value of "Portugal". The others elements have a behavior similar to previous example.

Then, we will create the table Orders. Like before the script for both databases are the same.

```
create table Orders
(
    cod_order    integer,
    date_order date,
    date_delivery date,
    cod_customer integer,
    CONSTRAINT Orders_pk PRIMARY KEY (cod_customer),
    CONSTRAINT Orders_Cust_fk FOREIGN KEY (cod_customer)
                REFERENCES Customers(cod_customer)
);
```

The primary key is the "cod_order". However, the Orders table also has a foreign key in the field "cod_customer". The foreign key is connected to the primary key of the table "Customer" by the field "cod_customer". It is also relevent to highlight the declaration of two variables of the date type. The date format is represented by "YYYY-MM-DD" and its supported range is from '1000-01-01' to '9999-12-31'.

Finally, the script for OrdersProducts is presented. Here the script below only works in MySQL due to the int() type.

```
create table OrdersProducts
(
    cod_product    integer,
    cod_order integer,
    quantity int(2),
    CONSTRAINT OrdersProducts_pk PRIMARY KEY (cod_product, cod_order),
    CONSTRAINT OrdersProducts_Prod_fk FOREIGN KEY (cod_product)
                REFERENCES Products(cod_product),
    CONSTRAINT OrdersProducts_Orders_fk FOREIGN KEY (cod_order)
                REFERENCES Orders(cod_order)
);
```

The primary key is composed by the fields "cod_product" and "cod_order". The quantity is declared an integer where the maximum number of digits is specified in parenthesis. Finally there are two foreign keys: the first one connected to the table Products; and the last one associated to the Orders table. The fields "cod_product" and "cod_order" are simultaneously primary and foreign keys.

To make it work in SQL Server we need to change int() type for decimal(). The full corrected script is given below.

```
create table OrdersProducts
(
    cod_product    integer,
    cod_order integer,
    quantity decimal(2,0),
    CONSTRAINT OrdersProducts_pk PRIMARY KEY (cod_product, cod_order),
    CONSTRAINT OrdersProducts_Prod_fk FOREIGN KEY (cod_product)
                REFERENCES Products(cod_product),
    CONSTRAINT OrdersProducts_Orders_fk FOREIGN KEY (cod_order)
                REFERENCES Orders(cod_order)
);
```

The only change in this script is in "quantity" field. It indicates that only supports decimal

Finally another change was performed to make it work in Oracle databases.

```
create table OrdersProducts
(
    cod_product    integer,
    cod_order integer,
    quantity number(2),
    CONSTRAINT OrdersProducts_pk PRIMARY KEY (cod_product, cod_order),
    CONSTRAINT OrdersProducts_Prod_fk FOREIGN KEY (cod_product)
                REFERENCES Products(cod_product),
    CONSTRAINT OrdersProducts_Orders_fk FOREIGN KEY (cod_order)
                REFERENCES Orders(cod_order)
);
```

The only change in this script is also in "quantity" field. It indicates that only supports numbers with only integer part composed by two digits.

3. Insert Data

The insertion of new data in a database can be done by following two different, but similar, approaches:

- The first form does not specify the column names where the data will be inserted, only their values;
- The second form specifies both the column names and the values to be inserted.

The scripts to insert data are exactly the same for MySQL, SQL Server and Oracle databases.

For the introduction of data in tables "Products" and "Customers" we will adopt the second approach.

```
Insert Into Products (cod_product, description, unit_price)
Values (1, 'Eggs', 2.49);
Insert Into Products (cod_product, description, unit_price)
Values (2, 'Ice Cream', 3.99);
Insert Into Products (cod_product, description, unit_price)
Values (3, 'Soda', 0.65);
Insert Into Products (cod_product, description, unit_price)
Values (4, 'Cheese', 2.89);
Insert Into Products (cod_product, description, unit_price)
Values (5, 'Pork Meat', 3.10);
Insert Into Customers (cod_customer, name)
Values (1, 'Anne');
Insert Into Customers (cod_customer, name)
Values (2, 'Peter');
Insert Into Customers (cod_customer, name)
Values (3, 'Elena');
Insert Into Customers (cod_customer, name)
Values (4, 'Shirley');
Insert Into Customers (cod_customer, name)
Values (5, 'John');
```

In table "Products" we will introduce data related to its code of product, description and unit price. On the other side, we only introduce information regarding the code of customer and its name in table "Customers". In any situation it is mandatory to fill the primary key fields. The other fields will assume the default value if it was declared in the creation table process.

After that the contents of the table "Products" are depicted in Figure 1.

	COD_PRODUCT	DESCRIPTION	UNIT_PRICE	AVAILABLE_STOCK	MINIMAL_STOCK
1	1	Eggs	2,49	(null)	0
2	2	Ice Cream	3,99	(null)	0
3	3	Soda	0,65	(null)	0
4	4	Cheese	2,89	(null)	0
5	5	Pork Meat	3,1	(null)	0

Figure 1 - Contents of table Products

It is also presented in Figure 2 the contents of table "Customers".

Figure 2 - Contents of table Customers

In order to place new data in table "Orders" we will use the script below that gives data values in all fields.

```
Insert Into Orders Values (1, '2015-12-21', '2015-12-21', 1);
Insert Into Orders Values (2, '2015-12-22', '2015-12-23', 1);
Insert Into Orders Values (3, '2015-12-22', '2015-12-27', 2);
Insert Into Orders Values (4, '2015-12-27', '2015-12-30', 3);
Insert Into Orders Values (5, '2015-12-30', '2015-12-31', 3);
```

The customer with code "1" and "3" will have 2 orders recorded in the database. The customer with code "2" will only have one order. Finally the customers with code "4" and "5" don't have any order. It is also important to highlight that the primary field must be always unique.

The contents of the table "Orders" is given in Figure 3.

Figure 3 - Contents of table Orders

Finally, we will adopt the script below to include information regarding each order. This information is stored in table "OrdersProducts".

```
Insert Into OrdersProducts Values (1, 1, 1);
Insert Into OrdersProducts Values (2, 1, 1);
Insert Into OrdersProducts Values (1, 2, 2);
Insert Into OrdersProducts Values (5, 3, 7);
Insert Into OrdersProducts Values (4, 3, 4);
Insert Into OrdersProducts Values (3, 3, 5);
Insert Into OrdersProducts Values (2, 3, 5);
Insert Into OrdersProducts Values (1, 4, 8);
Insert Into OrdersProducts Values (2, 4, 2);
Insert Into OrdersProducts Values (1, 5, 3);
Insert Into OrdersProducts Values (2, 5, 3);
Insert Into OrdersProducts Values (4, 5, 5);
```

The contents of the table "OrdersProducts" after the execution of the above script is given in Figure 4.

Figure 4 - Contents of table OrdersProducts

4. Update 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%.

The contents of the table "Products" after the execution of all previous scripts is shown in Figure 5.

	COD_PRODUCT	DESCRIPTION	UNIT_PRICE	AVAILABLE_STOCK	MINIMAL_STOCK
1	1	Eggs	1,99	25	10
2	2	Ice Cream	3,99	(null)	0
3	3	Soda	0,81	(null)	0
4	4	Cheese	2,89	(null)	0
5	5	Pork Meat	3,1	(null)	0

Figure 5 - Contents of table Products after update

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
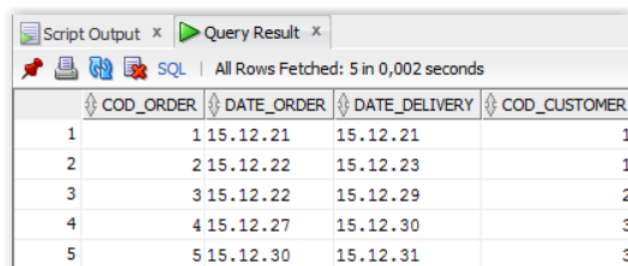   Where name='Peter');

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.

The contents of table "Orders" after the execution of previous scripts is shown in Figure 6.

Figure 6 - Contents of table Orders after update

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

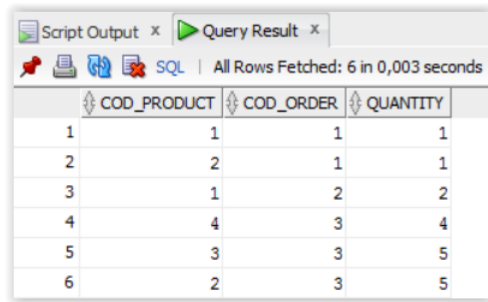The contents of table "OrdersProducts" is given in Figure 7.

Figure 7 - Contents of table OrdersProducts after delete

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.

After the execution of the previous script the content of table "OrdersProducts" is presented in Figure 8.

Figure 8 - Contents of table OrdersProducts after delete

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.

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.

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"]
```

Figure 9 - Basic structure of SQL

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.
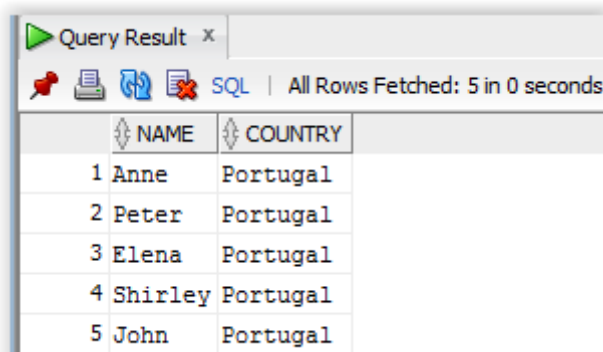
Select * From Customers;

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.

The execution of the above script displays the content of Figure 10.

	NAME	COUNTRY
1	Anne	Portugal
2	Peter	Portugal
3	Elena	Portugal
4	Shirley	Portugal
5	John	Portugal

All Rows Fetched: 5 in 0 seconds

Figure 10 - Contents of table Customer

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.

The execution of the above script shows the following result (Figure 11).
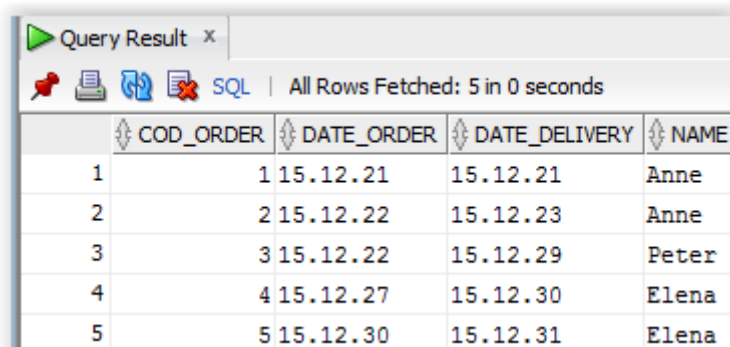

Figure 11 - Contents of all fields in table Products

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.

The result of the execution of above script is given in Figure 12.

Figure 12 - Information regarding orders and customers

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.

The result of the execution of the above script is given in Figure 13.

**Figure 13 - Information regarding 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.

The result of the execution of the above script is given in Figure 14.

Figure 14 - Shows the calculation of marginStock

In next example we use the modulo operation, which is responsible to find the remainder after division of one number by another (sometimes called modulus).

```
select cod_product, description, mod(available_stock, minimal_stock) as modResult
From Products;
```

This instruction calculates the module of division by available stock per minimal stock. It also shows the code of product and description for all products.

The above script only works for MySQL and Oracle databases. To make it work in SQL Server we must use the "%" operator, like it follows.

```
select cod_product, description, available_stock%minimal_stock as modResult
From Products;
```
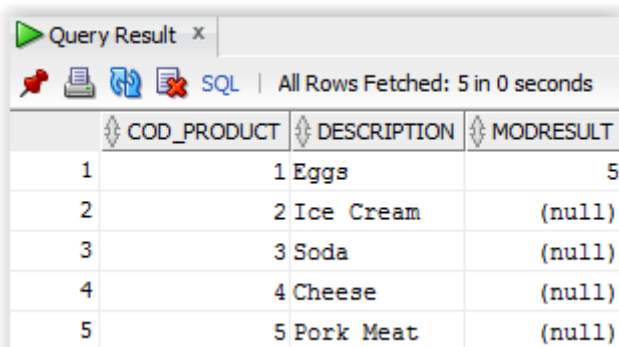
This instruction calculates the module of division by available stock per minimal stock. It also shows the code of product and description for all products.

The result of the execution of the above script is given in Figure 15.

Query Result ×

SQL | All Rows Fetched: 5 in 0 seconds

	COD_PRODUCT	DESCRIPTION	MODRESULT
1	1	Eggs	5
2	2	Ice Cream	(null)
3	3	Soda	(null)
4	4	Cheese	(null)
5	5	Pork Meat	(null)

Figure 15 - Shows the calculation of modResult

8. SQL Queries - Comparing Strings

In order to compare strings to search for a specified pattern we must use the "LIKE" operator. The use of this operator needs the use of wildcards characters that can be used to substitute any other character(s) in a string. The most common wildcards are the following

- "%" - a substitute for zero or more characters;
- "_" - a substitute for a single character.

In the first example we define a wildcard (missing letters) before the pattern.

```
SELECT * FROM Customers
 WHERE name LIKE 'E%';
```
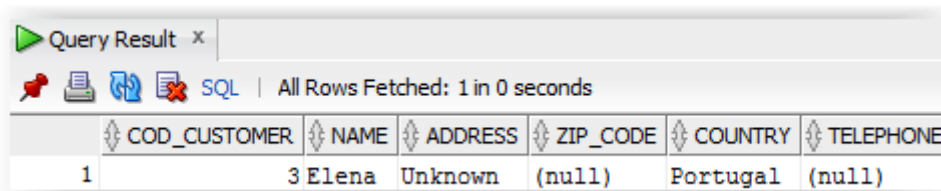*This instruction selects all customers with a name starting with the letter "E".*

In the second example we define a wildcard after the pattern. The statement is very similar to previous.

```
SELECT * FROM Customers
 WHERE name LIKE '%a';
```
This instruction selects all customers with a name ending with the letter "a".

In both situations the only record returned by the instruction is given in example below (Figure 16).

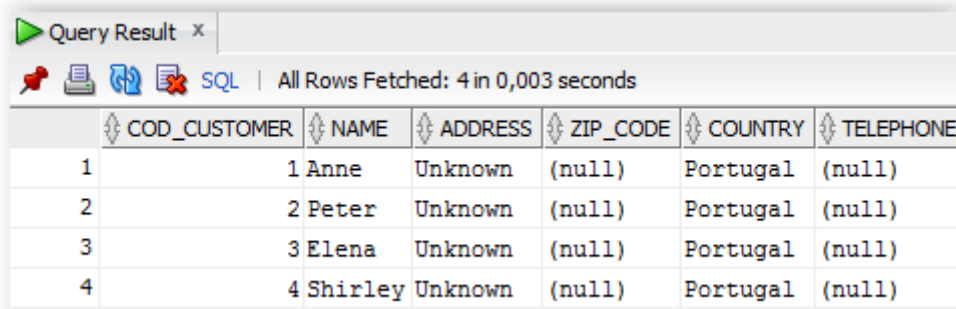|	COD_CUSTOMER	NAME	ADDRESS	ZIP_CODE	COUNTRY	TELEPHONE
1	3	Elena	Unknown	(null)	Portugal	(null)

Figure 16 - Shows all customers ending in "a"

In the third example we define a simultaneous wildcard after and before the pattern.

```
SELECT * FROM Customers
 WHERE name LIKE '%e%';
```
This instruction selects all customers with a name containing the character "e".

The result of the previous example is depicted in Figure 17.

Figure 17 - Shows all customers fields which name contains "e"

Finally we use two examples to demonstrate the use of "_" wildcard.

The first scenario consists only in the substitution of a missing character.

SELECT * FROM Customers
 WHERE name LIKE '_lena';

This instruction selects all customers with a name containing the string "lena" and starting by any character.

The second scenario consists in the substitution of multiple characters.

SELECT * FROM Customers
 WHERE name LIKE 'E_e_a';

This instruction selects all customers with a name containing the string initiated by "E" with a third character equal to "e" and last character equal to "a".

9. SQL Queries - Aggregation Operators

SQL has many built-in functions for performing calculations on data. SQL aggregate functions return a single value, calculated from values in a column. The most common aggregation operators are the following:

- AVG() - returns the average value;
- COUNT() - returns the number of rows;
- MAX() - returns the largest value;
- MIN() - returns the smallest value;
- SUM() - returns the sum.

9.1 AVG() Operator

The AVG() function returns the average value of a numeric column. Here we show three examples using AVG() operator.

Select AVG(unit_price) From Products;
This instruction returns the average of unit price of all products.

If we want to show the price with only two decimal places, we must slight change the previous statement.

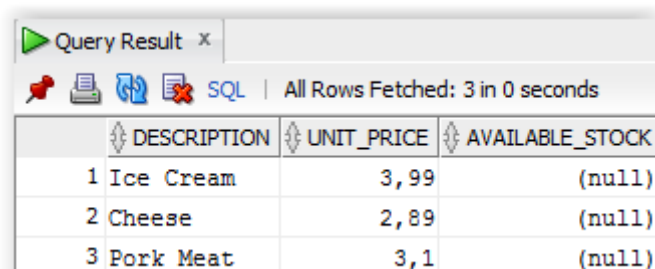Select ROUND(AVG(unit_price),2) From Products;
This instruction returns the average of unit price of all products with 2 decimal places. This is done using the "ROUND" function.

The last example uses the AVG() function as an element of a sub-query.

SELECT description, unit_price, available_stock
FROM Products
WHERE unit_price>(SELECT AVG(unit_price) FROM Products);
This instruction shows the description, unit price and available stock of all products which unit price is higher than the unit price average of all products in the database.

The result of last statement is shown in Figure 18.

Figure 18 - Shows products which price is above average

9.2 COUNT() Operator

The COUNT() function returns the number of rows that matches a specified criteria. Here we show two examples of using COUNT() operator.
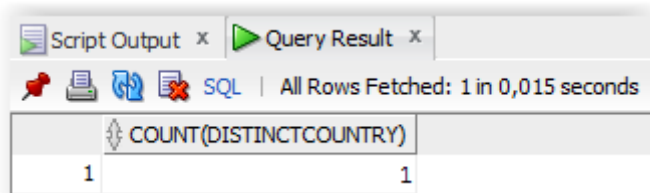
Select Count(*) From Customers;

This instruction returns the number of records in Customers table.

The second example uses simultaneously Count() function and distinct statement.

Select Count(distinct country) From Customers;

This instruction returns the number of distinct countries in Customers table.

The result of the execution of previous statement is given in Figure 19.

Figure 19 - Shows all countries of customers

9.3 MAX() Operator

The MAX() function returns the largest value of the selected column. Here we show two examples using MAX() operator.

SELECT MAX(unit_price) AS HighestPrice FROM Products;

This instruction returns the highest unit price of all products.

In second example we will return simultaneously the maximum and average values.

SELECT MAX(unit_price) AS HighestPrice, AVG(unit_price) AS AveragePrice
FROM Products;

This instruction returns the highest unit price and the average price of all products.

The result of the execution of previous statement is depicted in Figure 20.

Figure 20 - Shows the highest and average price

9.4 MIN() Operator

The MIN() function returns the smallest value of the selected column. Here we show two examples using MIN() operator.

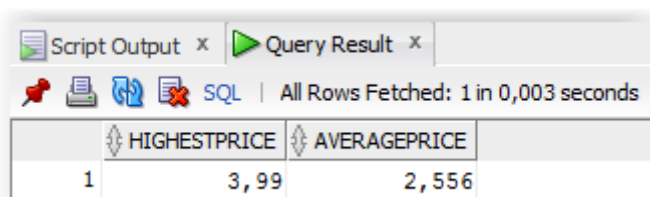SELECT MIN(unit_price) AS SmallestPrice FROM Products;
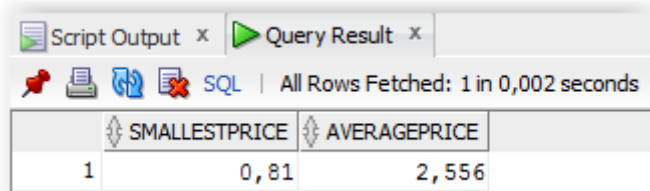
This instruction returns the smallest unit price of all products.

In second example we will return simultaneously the maximum and average values.

```
SELECT MIN(unit_price) AS SmallestPrice, AVG(unit_price) AS AveragePrice
FROM Products;
```
This instruction returns the smallest unit price and the average price of all products.

The result of the execution of previous statement is depicted in Figure 21.

Figure 21 - Shows the smallest and average price

9.5 SUM() Operator

The SUM() function returns the total sum of a numeric column. Here we show two examples using SUM() operator.
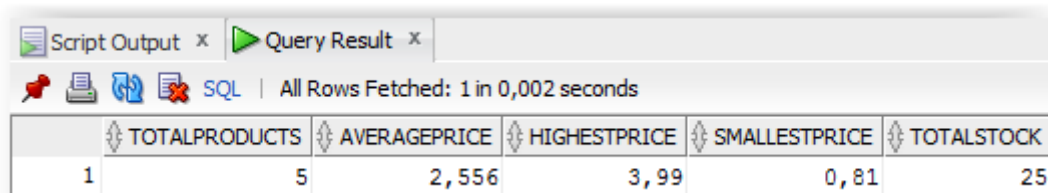
```
SELECT SUM(available_stock) AS TotalStock FROM Products;
```
This instruction returns the sum of available stock for all products.

In second example we show the use of all operators in a single statement.

```
SELECT COUNT(*) as TotalProducts,
    AVG(unit_price) AS AveragePrice,
    MAX(unit_price) AS HighestPrice,
    MIN(unit_price) AS SmallestPrice,
    SUM(available_stock) AS TotalStock
FROM Products;
```
This instruction returns simultaneously the total number of products, average price, highest price, smallest price and total stock.

The result of the above statement is shown in Figure 22.

Figure 22 - Shows the total number of products, average price, highest price, smallest price and total stock

10. SQL Queries - Scalar Functions

SQL scalar functions return a single value, based on the input value. The most useful scalar functions are the following:

- UCASE() - converts a field to upper case;
- LCASE() - converts a field to lower case;
- LEN() - returns the length of a text field;
- ROUND() - rounds a numeric field to the number of decimals specified.

In the following paragraphs we will an example of each scalar functions.

In the first example we will use simultaneously UCASE() and LCASE() functions. However, there is an additional issue, because we must use a different statement for each specific database. We will starting by presenting the statement for MySQL.
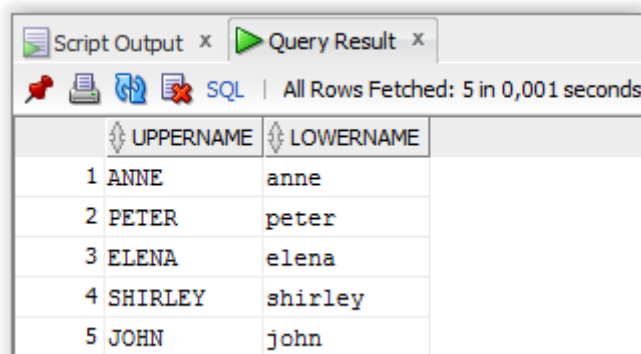
Select UCASE(name) as UPPERName, LCASE(name) as LOWERName
from Customers;
This instruction returns simultaneously the upper name and lower name of customers name.

The statement for SQL Server and Oracle can be the same.

Select UPPER(name) as UPPERName, LOWER(name) as LOWERName
from Customers;
This instruction returns simultaneously the upper name and lower name of customers name.

The result of previous statement is shown in Figure 23.

Figure 23 - Displays the upper and lower names of the customers

In the second example we use the LEN() function to return the size of a text field. This function can only be used in a SQL Server database.
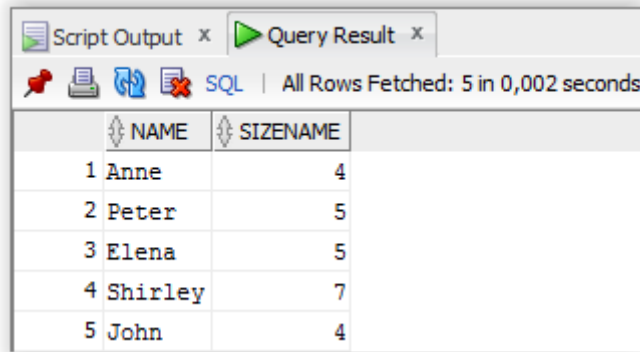
Select name, LEN(name) as SizeName
from Customers;
This instruction shows all the customers' name and for each name presents its size.

In MySQL and Oracle databases we must replace the "LEN()" function by the "LENGTH()" function.

```
Select name, LENGTH(name) as SizeName
from Customers;
```
This instruction shows all the customers' name and for each name presents its size.

The result of previous statement is depicted in Figure 24.
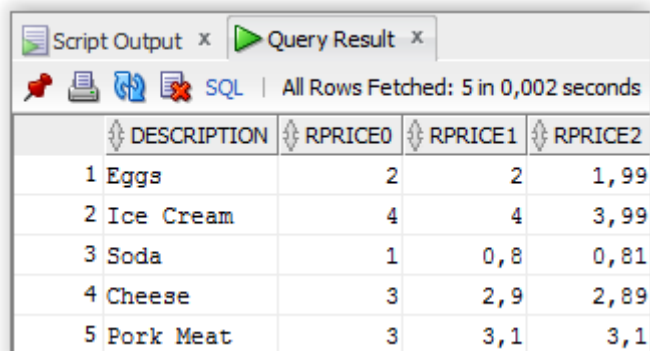
Figure 24 - Shows the size of the customers' name

In the third example we use the round() function. The following statement works well for the three databases.

```
Select description, round(unit_price, 0) as RPrice0,
     round(unit_price, 1) as RPrice1,
     round(unit_price, 2) as RPrice2
From Products;
```
This instruction shows for each product three prices. The first one without any decimal place, the second column with one decimal place, and the third column with two decimal places.

The result of previous statement is given in Figure 25.

Figure 25 - Shows 3 approaches to diplay the price of products

11. SQL Queries - Grouping Elements
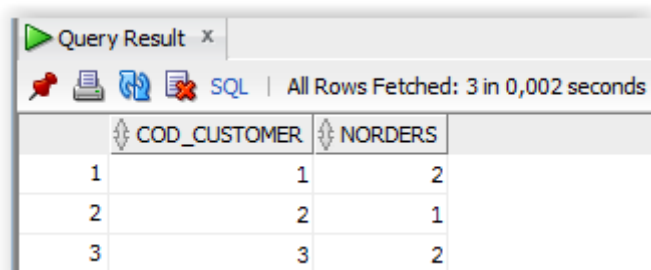
There are two additional SQL statements that are used to group and restrict the number of returned rows. These statement are: "Group By..." and "Having...".

The "Group By..." statement is used in conjunction with the aggregate functions to group the result-set by one or more columns. Here we give 2 examples using the Group By statement.

Select cod_customer, count(cod_customer) as nOrders
From Orders
Group By cod_customer;

This instruction selects the number of orders recorded in the system for each customer.

The result of the previous operation is given in Figure 26.
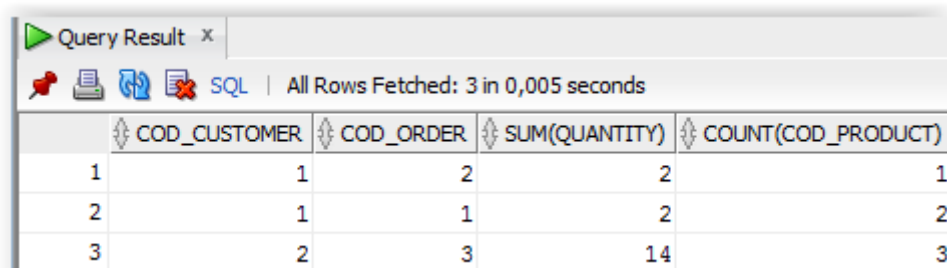
Figure 26 - Counts the number of orders per customer

The second example gives a more complex exercise with more than one table.

select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order;

This instruction shows for each customer and order, the total quantity of itens and the number of products. To perform this operation there is a grouping by code of customer and code of order.

The result of previous statements is depicted in Figure 27.
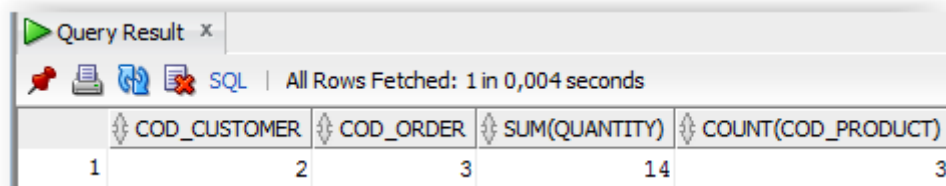
Figure 27 - Shows the quantity of itens and number of products per customer/order

In the last example we use the "Having..." clause to restrict the number of shown elements. The "Having" clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

```
Select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order
Having sum(quantity) > 10;
```

This instruction is very similar to previous and it only adds an additional last line. The "Having" clause only shows lines where the sum of quantity is higher than 10.

The result of the previous operation is depicted in Figure 28.

Figure 28 - Using the clause Having to filter results
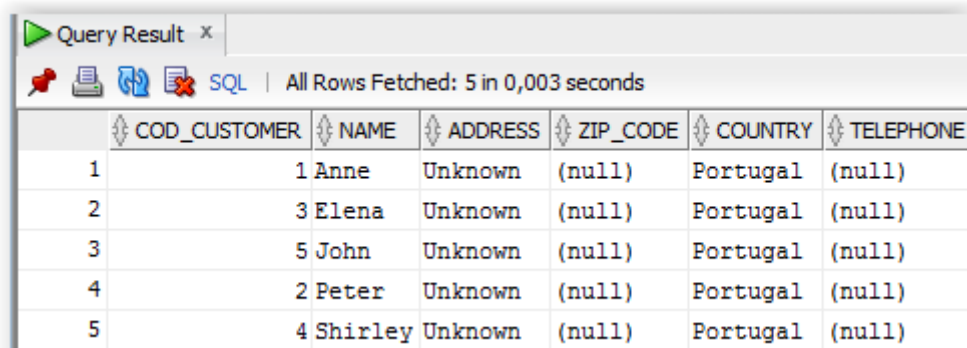
12. SQL Queries - Ordering Data

The ORDER BY keyword is used to sort the result-set by one or more columns. The ORDER BY keyword sorts the records in ascending order by default or using ASC keyword. To sort the records in a descending order, you can use the DESC keyword. Here we show two examples using the "Order By..." syntax.

```
Select *
From Customers
Order By Name;
```

This instruction selects all columns of Customers table ordering the records by name (ascending order).

The result of previous statement is given in Figure 29.
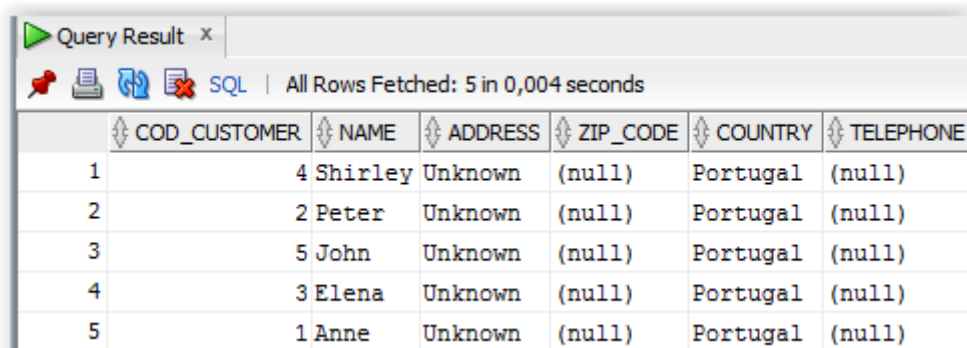
Figure 29 - Order data per name of customer

In second example we use the same Order By operation but now using the inverse order.

```
Select *
From Customers
Order By Name DESC;
```

This instruction selects all columns of Customers table ordering the records by name (descending order).

The result of previous operation is given in Figure 30.

Figure 30 - Ordering name of customer by inverse order

13. SQL Queries - Returning Top Elements

There is an useful statement in SQL that lets the user to return only the first n elements. This clause can be very useful on large tables with thousands of records, which may have deep impact on performance. The syntax is different for each database.

In MySQL it is possible to perform this operation using the syntax below.

SELECT *
FROM Customers
Order By Name
Limit 3;

This instruction shows all fields of Customers table ordered by name. However, it only shows the first three records.

In SQL Server it is possible to perform this operation using the syntax below.

SELECT Top 3 *
FROM Customers
Order By Name;
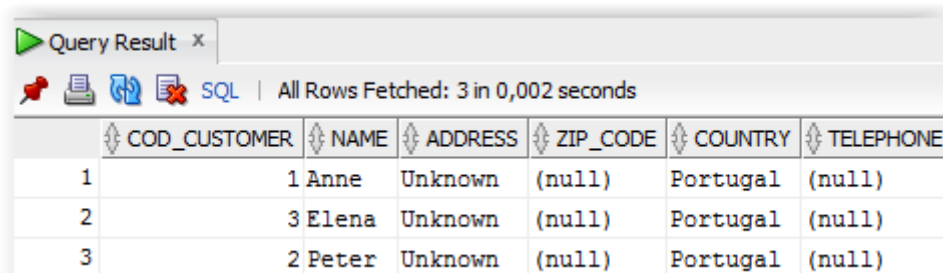
This instruction shows all fields of Customers table ordered by name. However, it only shows the first three records.

Finally, in Oracle the same operation can be done using the syntax below.

SELECT *
FROM Customers
Where RowNum <= 3
Order By Name;

This instruction shows all fields of Customers table ordered by name. However, it only shows the first three records.

In console it shows the result of Figure 31.

Figure 31 - Returning top elements of a query

14. SQL Queries - Sub-queries

Sub-queries are query statements tucked inside of query statements. A sub-query may occur in:

- A SELECT clause;
- A FROM clause;
- A WHERE clause.

Typically a sub-query is usually added within the WHERE clause of another SQL SELECT statement. The syntax model is depicted in Figure 32.

```
SELECT      select_list
FROM        table
WHERE       expr operator

                    (SELECT     select_list
                     FROM       table);
```
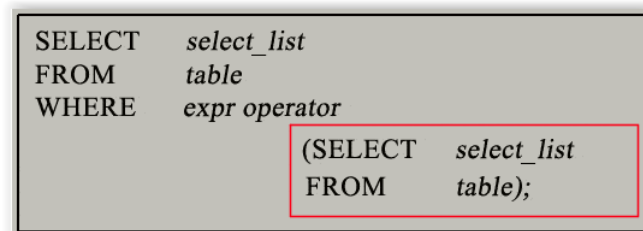
Figure 32 - Structure of a sub-query

The sub-query (inner query) executes once before the main query (outer query) executes. Consequently, the main query (outer query) uses the sub-query result.

In the context of this chapter we give two examples of sub-queries adoption.

In the first example we use always the same table in the inner and outer query.

```
Select description
From Products
Where unit_price = (Select max(unit_price)
                    From Products);
```
This instruction presents the description of the product which has the maximum unit price.

The result of this operation is given in Figure 33.

```
Query Result  x

📌 🖨 🔁 ❌ SQL  |  All Rows Fetched: 1 in 0,031 seconds

   DESCRIPTION
 1 Ice Cream
```
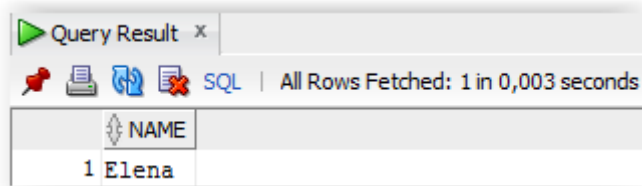
Figure 33 - Shows the product with maximum price

In the second example we use a sub-query that uses another table different from the outer statement.

```
Select name
From Customers
Where cod_customer = (Select max(distinct cod_customer)
                      From Orders
                      Where date_delivery >= '2015-12-26');
```

This instruction presents the name of customer which code of customer is maximum (most recent) and has an order which date of delivery is equal or more recent than 26th December 2015.

The result of this operation is given in Figure 34.
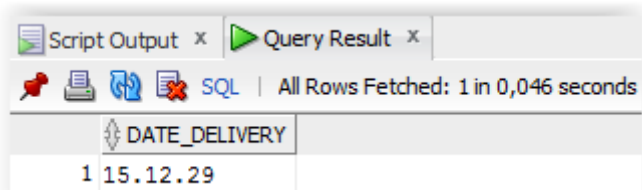
Figure 34 - Shows the customer with max customer code and given delivery date

In the last example we use a sub-query that uses also another table different from the outer statement.

```
Select date_delivery
From Orders
Where cod_order = (Select max(cod_order)
                   From OrdersProducts
                   Where quantity > 3);
```

This instruction presents the delivery date of the last order saved in the database that has an item quantity higher than 3.

The result of this operation is given in Figure 35.

Figure 35 - Shows the delivery date of orders that have specified quantity

15. SQL Queries - Operator "In" and "Exists"
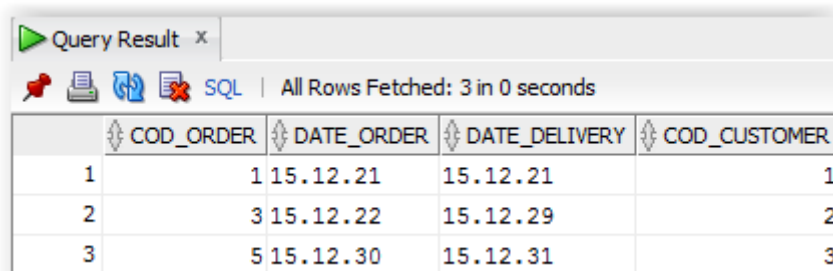
The operator "in" and "exists" can be used in SQL to check the contents of a table. The operator "in" allows users to specify multiple values in a WHERE clause.

In the first example we will use "IN" operator to check numeric values.

SELECT *
FROM Orders
WHERE cod_order IN (1, 3, 5);
This operation shows all fields of records in table Orders with order code equal to "1" or "3" or "5".

The result of this operation is shown in Figure 36.

Figure 36 - Basic approach using the "IN" operator

In the second example we will use "IN" operator to check string values.
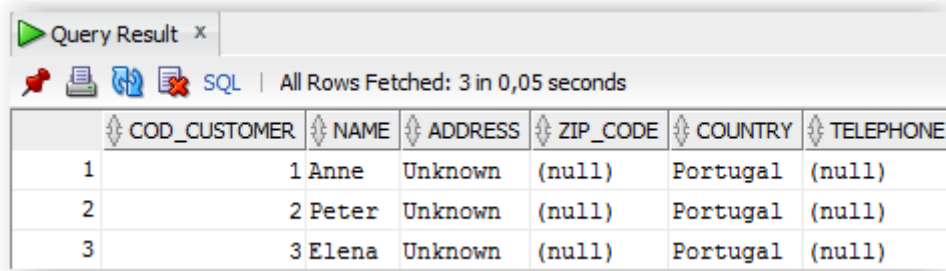
SELECT *
FROM Customers
WHERE name IN ('Peter', 'Elena');
This operation shows all fields of records in table Customers with name equal to "Peter" or "Elena".

The operator "IN" can also be used with sub-queries. One example of this situation is given below.

SELECT *
FROM Customers
WHERE cod_customer IN (SELECT cod_customer
        From Orders);
This operation shows all customers that already have orders in the table. The operator "IN" is used to retrieve several values from Orders table.

The result of this operation is shown in Figure 37.

Figure 37 - Return all customer fields that have orders

The operators "IN" and "EXISTS" offer similar functionality, but the syntax is slight different. The EXISTS checks the existence of a result of a sub-query. The EXISTS sub-query tests whether a sub-query fetches at least one row. When no data is returned then this operator returns 'FALSE'.

In order to demonstrate the use of "EXISTS" operator we use the same previous example, but now using "EXISTS" instead of "IN".
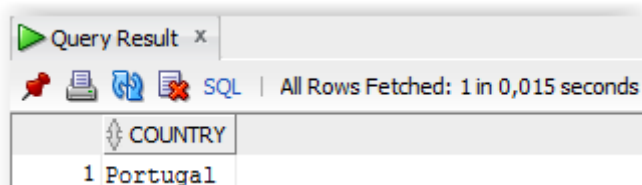
```
SELECT *
FROM Customers C
WHERE EXISTS (SELECT cod_customer
              FROM Orders O
              WHERE C.cod_customer = O.cod_customer);
```
This operation shows all customers that already have orders in the table. The operator "EXISTS" is used to retrieve the customers code of all customers that have orders.

Both operators "IN" and "EXISTS" can be used with the prefix "NOT". An example of such situation is given below.

```
SELECT distinct Country
FROM Customers C
WHERE NOT EXISTS (SELECT cod_customer
                  FROM Orders O
                  WHERE C.cod_customer = O.cod_customer);
```
This instruction shows all distinct countries which there are no orders in the database.

The result of this operation is given in Figure 38.

Figure 38 - Shows customers that don't have orders

16. SQL Queries - Operator "Any" and "All"
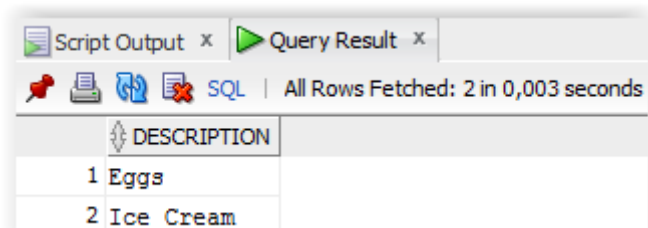
There are two useful operators "ANY" and "ALL" that are typically used with sub-queries.

The operator "ANY" compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row. A simple example using the "ANY" operator is given below.

```
SELECT description
FROM Products
WHERE cod_product = ANY
     (SELECT cod_product
       FROM OrdersProducts
       WHERE Quantity = 1)
```
This instruction shows the description of the products that has a quantity order equal to 1.

The result of the above operation is given in Figure 39.
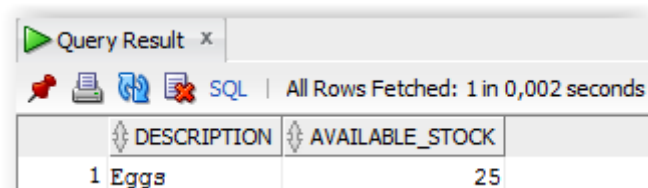
Figure 39 - Shows the use of ANY operator

Below it is a given another example using the same "ANY" operator, but changing the comparing operator to ">=".

```
Select description, available_stock
From Products
Where available_stock >= ANY (Select distinct minimal_stock
                            From Products);
```
This instruction shows the description and available stock of products which available stock is above any minimal stock of all products in database.

The result of the above operation is given in Figure 40.
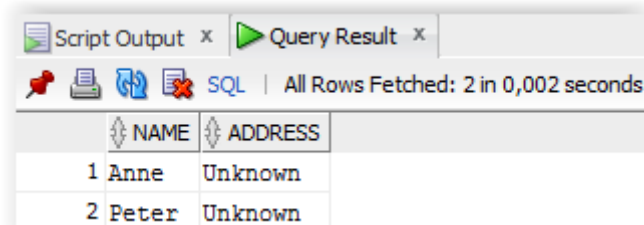
Figure 40 - Shows the use of ANY operator

Another example adopting the "ANY" syntax is given below.

```
Select name, address
From Customers
Where cod_customer < ANY (Select cod_customer From Orders);
```

This instruction shows name and address of customers which code of customer is below than any code of customer that has already orders.

The result of the above operation is given in Figure 41.
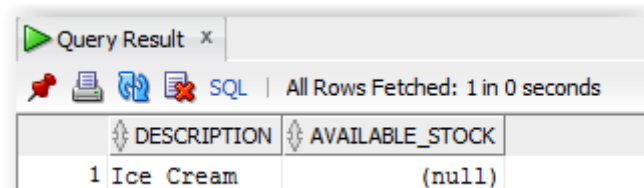
Figure 41 - Shows the use of ANY operator

The ALL is used to select all records of a SELECT statement. It compares a value to every value in a list or results from a query. The ALL must be preceded by the comparison operators and evaluates to TRUE if the query returns no rows. Below it is a given an example.

```
Select description, available_stock
From Products
Where unit_price >= ALL (select distinct unit_price
                        From Products);
```

This instruction shows the description and available stock of products which unit price is higher than price of all products in the database. The "=" signal is used to guarantee that it returns the correct field and match at least one record.

The result of this operation is given in Figure 42.

Figure 42 - Shows the use of ALL operator

17. SQL Queries - Operations with Sets

The SQL offers three operations with sets which can also be found in relational algebra. Considering two simple relations (R1 and R2) these operations are:
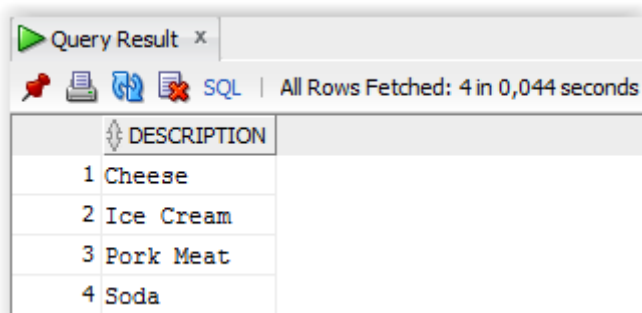
- Union - the relation containing all tuples that appear in R1, R2, or both;
- Intersect - is the relation containing all tuples that appear in both R1 and R2;
- Difference - the relation containing all tuples of R1 that do not appear in R2.

The UNION operator is used to combine the result-set of two or more SELECT statements. The SELECT statement within the UNION must have the same number of columns and have similar or compatible data types.

Select description
From Products
Where cod_product > 3
UNION
Select description
From Products
Where cod_product > 1;

This instruction shows the description of all products which product code is higher than 3 or higher than 1. The UNION statement let join the results of both SQL statements.

The result of previous operation is given in Figure 43.
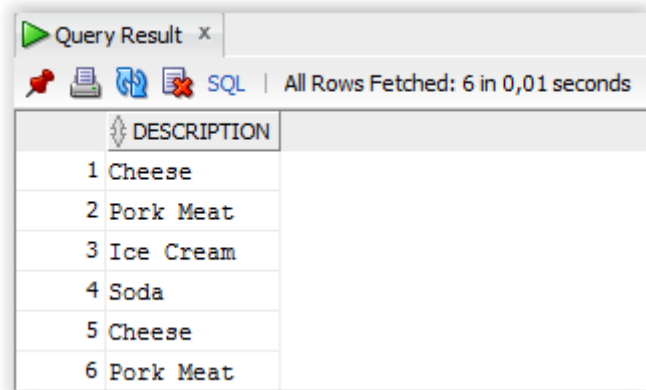
Figure 43 - Shows the use of UNION operator

The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION. An example of such situation is given below.

Select description
From Products
Where cod_product > 3
UNION ALL
Select description
From Products
Where cod_product > 1;

This instruction shows the description of all products which product code is higher than 3 or higher than 1. However, and comparing to previous example, the records returned by both queries are duplicated.

The result of this operation is shown in Figure 44.
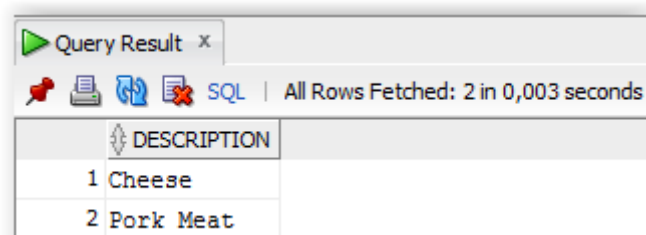
Figure 44 - Shows the use of UNION ALL operator

The SQL INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results. An example of such situation is given below. The script below only works in SQL Server and Oracle databases.

```
Select description
From Products
Where cod_product > 3
INTERSECT
Select description
From Products
Where cod_product > 1;
```

This instruction shows the description of all products which product code is higher than 3 or higher than 1. However, and comparing to previous example, the records returned by both queries are duplicated.

The result of this operation is given in Figure 45.

Figure 45 - Shows the use of INTERSECT operator

The INTERSECT operator isn't supported in MySQL. To make a similar function in MySQL we can use the "IN", "EXISTS" or arithmetic operators. In our previous example we can reach the same functionality by using simple arithmetic operators. An example of this approach is given below.

```
Select description
From Products
Where cod_product > 3 and cod_product > 1;
```
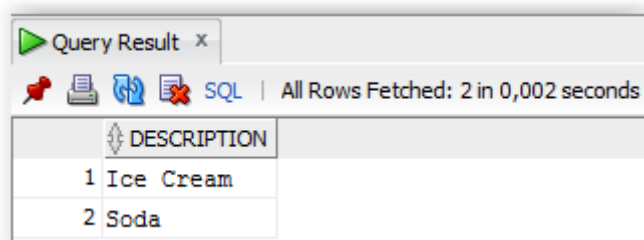
This instruction shows the description of all products which product code is higher than 3 or higher than 1. It uses only arithmetic and boolean operations.

The SQL MINUS clause is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. An example of this situation is given below. This script only works in Oracle.

```
Select description
From Products
Where cod_product > 1
MINUS
Select description
From Products
Where cod_product > 3;
```

This instruction shows the description of all products that have a product code higher than 1 but below or equal to 3.

The result of this operation is given in Figure 46.

Figure 46 - Shows the use of MINUS operator

To make the previous script working in SQL Server we must use the clause "EXCEPT".

```
Select description
From Products
Where cod_product > 1
EXCEPT
Select description
From Products
Where cod_product > 3;
```

This instruction shows the description of all products that have a product code higher than 1 but below or equal to 3.

To make the script work in MySQL we must adopt a similar approach like we did in "INTERSECT" clause. An example of this approach is given below.

```
Select description
From Products
Where cod_product > 1 and cod_product <= 3;
```

This instruction shows the description of all products that have a product code higher than 1 but below or equal to 3.

18. SQL Queries - Joins

SQL joins are used to combine rows from two or more tables. There are in SQL four types of joins:

- Inner join and natural joins - select records that have matching values in both tables;
- Left Outer Join - select records from the first (left-most) table with matching right table records;
- Right Outer Join - select record from the second (right-most) table with matching left table records;
- Full Outer Join - selects all records that match either left or right table records.

18.1 Inner Join and Natural Join

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables. Graphically the inner join is depicted in Figure 47.
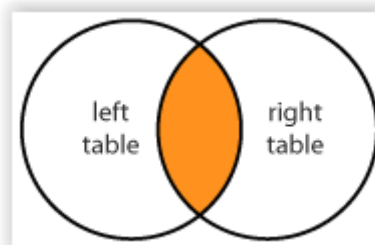
Figure 47 - Structure of inner and natural joins

An example using INNER JOIN syntax is given below.

```
SELECT distinct C.Name, C.Address
FROM Customers C
INNER JOIN Orders O
        ON C.cod_customer=O.cod_customer
ORDER BY c.Name;
```

This instruction shows all the customers' name and addresses that have orders. The inner join elements are explicitly declared by the field customer code in both table. The final result is order alphabetically by customer name.
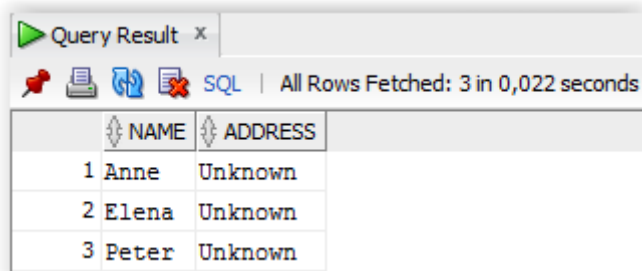
Instead of INNER JOIN we could be used the NATURAL JOIN clause, because the field name of both tables is the same. Then, using the NATURAL JOIN syntax we get the following script.

```
SELECT distinct C.Name, C.Address
FROM Customers C NATURAL JOIN Orders O
ORDER BY c.Name;
```

This instruction shows all the customers' name and addresses that have orders. Using the NATURAL JOIN syntax we don't explicitly declare the join fields. The final result is order alphabetically by customer name.

The above script only works for MySQL and Oracle databases. The SQL Server database doesn't support the NATURAL JOIN clause. Instead of NATURAL JOIN we can easily use the INNER JOIN as used previously.

The result of the execution of the two above scripts is displayed in Figure 48.

18.2 Left Outer Join

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match. Graphically the inner join is depicted in Figure 49.
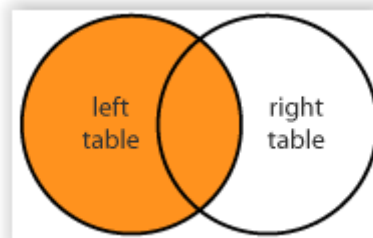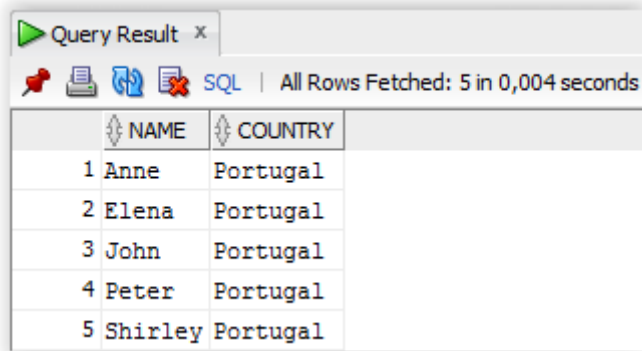
An example using the LEFT OUTER JOIN clause is given below.

```
SELECT distinct C.Name, C.country
FROM Customers C
LEFT JOIN Orders O
      ON C.cod_customer=O.cod_customer
ORDER BY c.Name;
```

This instruction shows all the customers' name and country that have orders or not. Using the LEFT JOIN syntax we show all elements from the first table.

The result of the above operation is given in Figure 50.

Figure 50 - Shows the use of Left Join

18.3 Right Outer Join

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match. Graphically the inner join is depicted in Figure 51.
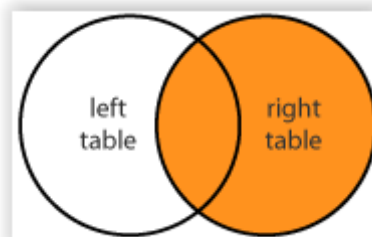
Figure 51 - Structure of Right Outer Join

An example using the RIGHT OUTER JOIN clause is given below.

```
SELECT distinct O.date_order, C.name
FROM Orders O
RIGHT JOIN Customers C
 ON O.cod_customer=C.cod_customer
ORDER BY O.date_order;
```
This instruction shows all the date orders and customers names of all orders, and it also includes the name of customers that don't have any order. This is done using the RIGHT JOIN clause. Finally all elements are ordered by their date.

The result of the above operation is given in Figure 52.

Figure 52 - Shows the use of Right Join

18.4 Full Outer Join

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2). It combines the results from LEFT JOINS and RIGHT JOINS. Graphically the inner join is depicted in Figure 53.
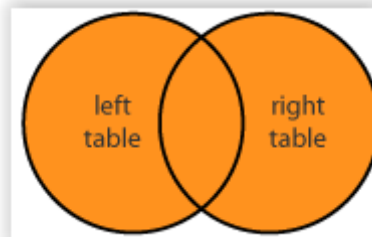
Figure 53 - Structure of Full Outer Join

An example using the FULL OUTER JOIN clause is given below.
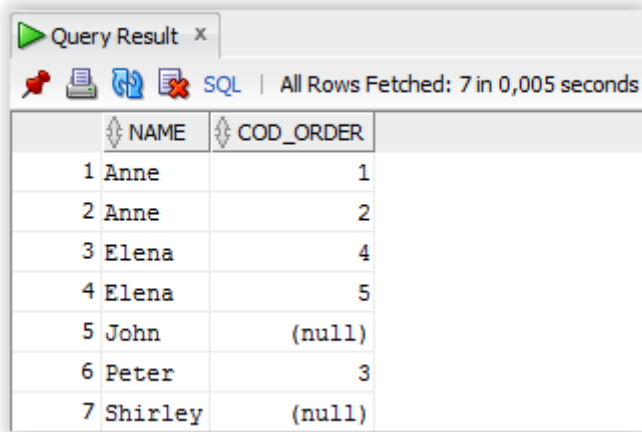
```
SELECT c.name, O.COD_ORDER
FROM Customers C
FULL OUTER JOIN Orders O
        ON C.cod_customer=O.cod_customer
ORDER BY C.Name;
```
This instruction shows all customers' name and the code of orders simultaneously for all customers and orders. The result is ordered by the customers' name.

The above script works well for Oracle and SQL Server databases. However, we don't have FULL JOINS on MySQL, but we can sure emulate them. Attending the above example we can do that in MySQL by adopted the approach below.

```
SELECT c.name, o.cod_order
FROM Customers C
LEFT JOIN Orders O ON C.cod_customer = O.cod_customer
UNION
SELECT c.name, o.cod_order
FROM Customers C
RIGHT JOIN Orders O ON C.cod_customer = O.cod_customer;
```

This instruction shows all customers' name and the code of orders simultaneously for all customers and orders. In order to emulate the functionality of FULL OUTER JOIN we use both LEFT and RIGHT joins.

The result of the above operation is given in Figure 54.

	NAME	COD_ORDER
1	Anne	1
2	Anne	2
3	Elena	4
4	Elena	5
5	John	(null)
6	Peter	3
7	Shirley	(null)

Figure 54 - Shows the use of Full Outer Join

19. SQL Queries - Views

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. An example creating a view is given a below.

Create view Products_BelowStockMin as
Select cod_product, description
From products
Where available_stock < minimal_stock

This instruction creates a view with all code of products and description which stock is below the established minimal stock.

If we want to show the content of a view we can use the following syntax.

Select * from Products_BelowStockMin

This instruction shows all field of the view "Products_BelowStockMin".
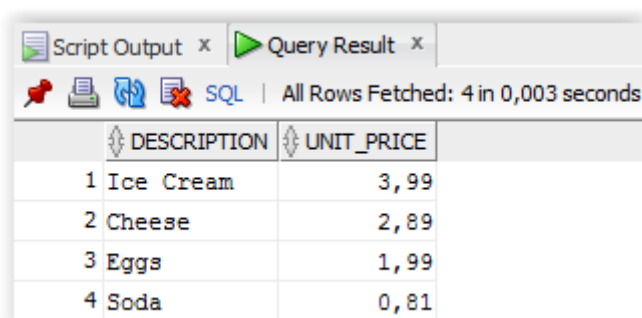
The result of this operation is empty due to the fact that there isn't any product which stock is below the minimal stock.

Other example of creation a view is given below.

Create view descProductsOrders as
Select distinct description, unit_price
From products p, ordersproducts op
Where p.cod_product = op.cod_product
order by unit_price DESC

This instruction creates a view with the description and unit price of all products that have orders submitted. The products in this view are inversed ordered by unit price.

The result after invoking the above view is given in Figure 55.

Figure 55 - Example of creating a view

20. SQL Queries - System Data

There are very useful instructions in SQL that can be used to extract environmental data. One of these situations is the extraction of current date. The approach is different for MySQL, SQL Server and Oracle. In MySQL we can perform this operation using the approach below.

Other example of creation a view is given below.

```
SELECT NOW() AS CurrentDateTime
```
This instruction returns the current date and time.

The same operation can be done in SQL Server using the approach below.

```
SELECT GETDATE() AS CurrentDateTime
```
This instruction returns the current date and time.

Finally, in Oracle we must invoke the DUAL table. It is a table automatically created by Oracle Database along with the data dictionary. Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement. Because DUAL has only one row, the constant is returned only once.

```
SELECT TO_CHAR
   (SYSDATE, 'MM-DD-YYYY HH24:MI:SS') "NOW"
    FROM DUAL;
```
This instruction returns the current date and time.

The result of three previous operations are the same and is depicted in Figure 56.
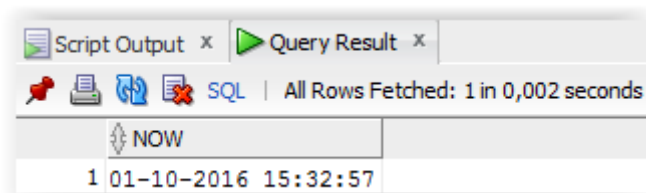
Figure 56 - Get the date from system

Other common operation is to get the current logged user in the database. This operation has also different SQL statements for each database. In MySQL this operation can be done as follows.

```
SELECT CURRENT_USER() AS CurrentLoggedUser
```
This instruction returns the current authenticated user.

The same operation can be done in SQL Server using the approach below. In SQL Server "current user" is seen as a property and not a method.
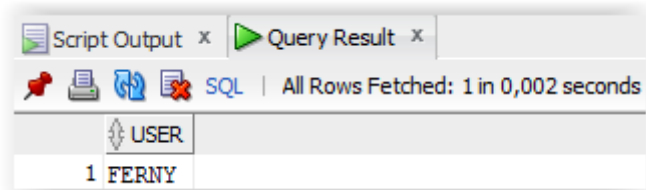
```
SELECT CURRENT_USER AS CurrentLoggedUser
```
This instruction returns the current authenticated user.

Finally, the same operation in Oracle is given below. Like before we need to invoke the DUAL table.

```
select user from dual;
```
This instruction returns the current authenticated user.

The result of the above operation is depicted in Figure 57.

Figure 57 - Get the user logged in system

Bibliography

Database SQL Developer Supplementary Information for Microsoft SQL Server Migrations. (n.d.). Retrieved 01 11, 2016, from https://docs.oracle.com/cd/E10405_01/appdev.120/e10379/ss_oracle_compared.htm

Intro to SQL: Querying and managing data. (n.d.). Retrieved 12 21, 2015, from https://www.khanacademy.org/computing/computer-programming/sql

Selecting from the DUAL Table. (n.d.). Retrieved 7 12, 2016, from https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries009.htm

SQL (Structured Query Language). (n.d.). Retrieved 01 10, 2016, from http://www.ntchosting.com/encyclopedia/databases/structured-query-language/

SQL Basic Query Structure. (n.d.). Retrieved 11 27, 2015, from http://www.mycms.ca/index.cfm/page/sqlqueries.html

SQL Join. (n.d.). Retrieved 12 11, 2015, from http://www.dofactory.com/sql/join

SQL Tutorial. (n.d.). Retrieved 11 18, 2015, from http://www.w3schools.com/sql/

What is SQL? (n.d.). Retrieved 01 10, 2016, from http://www.sqlcourse.com/intro.html

Annex I - Script for MySQL Databases

```
/* Creation of tables */
create table Products
(
    cod_product    integer,
    description    varchar(50) NOT NULL,
    unit_price     DECIMAL(10,2),
    available_stock integer,
    minimal_stock   integer default 0,
    CONSTRAINT Products_pk PRIMARY KEY (cod_product)
);

create table Customers
(
    cod_customer    integer,
    name    varchar(50) NOT NULL,
    address      varchar(95) Default 'Unknown',
    zip_code char(8),
    country   varchar(40) Default 'Portugal',
    telephone varchar(15),
    CONSTRAINT Customers_pk PRIMARY KEY (cod_customer)
);

create table Orders
(
    cod_order    integer,
    date_order date,
    date_delivery date,
    cod_customer integer,
    CONSTRAINT Orders_pk PRIMARY KEY (cod_order),
    CONSTRAINT Orders_Cust_fk FOREIGN KEY (cod_customer)
                REFERENCES Customers(cod_customer)
);

create table OrdersProducts
(
    cod_product    integer,
    cod_order integer,
    quantity int(2),
    CONSTRAINT OrdersProducts_pk PRIMARY KEY (cod_product, cod_order),
    CONSTRAINT OrdersProducts_Prod_fk FOREIGN KEY (cod_product)
                REFERENCES Products(cod_product),
    CONSTRAINT OrdersProducts_Orders_fk FOREIGN KEY (cod_order)
                REFERENCES Orders(cod_order)
);

/* Insert data into tables */
Insert Into Products (cod_product, description, unit_price)
Values (1, 'Eggs', 2.49);
Insert Into Products (cod_product, description, unit_price)
Values (2, 'Ice Cream', 3.99);
Insert Into Products (cod_product, description, unit_price)
Values (3, 'Soda', 0.65);
Insert Into Products (cod_product, description, unit_price)
Values (4, 'Cheese', 2.89);
Insert Into Products (cod_product, description, unit_price)
Values (5, 'Pork Meat', 3.10);

Insert Into Customers (cod_customer, name)
Values (1, 'Anne');
```

```
Insert Into Customers (cod_customer, name)
Values (2, 'Peter');
Insert Into Customers (cod_customer, name)
Values (3, 'Elena');
Insert Into Customers (cod_customer, name)
Values (4, 'Shirley');
Insert Into Customers (cod_customer, name)
Values (5, 'John');

Insert Into Orders Values (1, '2015-12-21', '2015-12-21', 1);
Insert Into Orders Values (2, '2015-12-22', '2015-12-23', 1);
Insert Into Orders Values (3, '2015-12-22', '2015-12-27', 2);
Insert Into Orders Values (4, '2015-12-27', '2015-12-30', 3);
Insert Into Orders Values (5, '2015-12-30', '2015-12-31', 3);

Insert Into OrdersProducts Values (1, 1, 1);
Insert Into OrdersProducts Values (2, 1, 1);
Insert Into OrdersProducts Values (1, 2, 2);
Insert Into OrdersProducts Values (5, 3, 7);
Insert Into OrdersProducts Values (4, 3, 4);
Insert Into OrdersProducts Values (3, 3, 5);
Insert Into OrdersProducts Values (2, 3, 5);
Insert Into OrdersProducts Values (1, 4, 8);
Insert Into OrdersProducts Values (2, 4, 2);
Insert Into OrdersProducts Values (1, 5, 3);
Insert Into OrdersProducts Values (2, 5, 3);
Insert Into OrdersProducts Values (4, 5, 5);

/* Update data previously recorded in tables */
Update Products
Set unit_price = 1.99
Where description = 'Eggs';

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

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

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

/* Delete data from tables */
Delete From OrdersProducts
Where cod_product = 5;

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

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

```
/* SQL Queries - Basic Structure */
Select * From Customers;

Select Name, Country From Customers;

Select distinct Country From Customers;

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

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

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;

select cod_product, description, available_stock - minimal_stock as marginStock
From Products;

select cod_product, description, mod(available_stock, minimal_stock) as modResult
From Products;

/* SQL Queries - Comparing Strings */
SELECT * FROM Customers
 WHERE name LIKE 'E%';

SELECT * FROM Customers
 WHERE name LIKE '%a';

SELECT * FROM Customers
 WHERE name LIKE '%e%';

SELECT * FROM Customers
 WHERE name LIKE '_lena';

SELECT * FROM Customers
 WHERE name LIKE 'E_e_a';

/* SQL Queries - Aggregation Operators */
Select AVG(unit_price) From Products;

Select ROUND(AVG(unit_price),2) From Products;

SELECT description, unit_price, available_stock
FROM Products
WHERE unit_price>(SELECT AVG(unit_price) FROM Products);

Select Count(*) From Customers;

Select Count(distinct country) From Customers;

SELECT MAX(unit_price) AS HighestPrice FROM Products;

SELECT MAX(unit_price) AS HighestPrice, AVG(unit_price) AS AveragePrice
FROM Products;
```

```
SELECT MIN(unit_price) AS SmallestPrice FROM Products;

SELECT MIN(unit_price) AS SmallestPrice, AVG(unit_price) AS AveragePrice
FROM Products;

SELECT SUM(available_stock) AS TotalStock FROM Products;

SELECT COUNT(*) as TotalProducts,
    AVG(unit_price) AS AveragePrice,
    MAX(unit_price) AS HighestPrice,
    MIN(unit_price) AS SmallestPrice,
    SUM(available_stock) AS TotalStock
FROM Products;

/* SQL Queries - Scalar Functions */
Select UCASE(name) as UPPERName, LCASE(name) as LOWERName
from Customers;

Select name, LENGTH(name) as SizeName
from Customers;

Select description, round(unit_price, 0) as RPrice0,
    round(unit_price, 1) as RPrice1,
    round(unit_price, 2) as RPrice2
From Products;

/* SQL Queries - Grouping Elements */
Select cod_customer, count(cod_customer) as nOrders
From Orders
Group By cod_customer;

select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order;

select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order
Having sum(quantity) > 10;

/* SQL Queries - Ordering Data */
Select *
From Customers
Order By Name;

Select *
From Customers
Order By Name DESC;

/* SQL Queries - Returning Top Elements */
SELECT *
FROM Customers
Order By Name
Limit 3;

/* SQL Queries - Sub-queries */
Select description
From Products
Where unit_price = (Select max(unit_price)
```

```
        From Products);

Select name
From Customers
Where cod_customer = (Select max(distinct cod_customer)
          From Orders
            Where date_delivery >= '2015-12-26');

Select date_delivery
From Orders
Where cod_order = (Select max(cod_order)
          From OrdersProducts
            Where quantity > 3);

/* SQL Queries - Operator "In" and "Exists" */
SELECT *
FROM Orders
WHERE cod_order IN (1, 3, 5);

SELECT *
FROM Customers
WHERE name IN ('Peter', 'Elena');

SELECT *
FROM Customers
WHERE cod_customer IN (SELECT cod_customer
            From Orders);

SELECT *
FROM Customers C
WHERE EXISTS ( SELECT cod_customer
FROM Orders O
WHERE C.cod_customer = O.cod_customer);

SELECT distinct Country
FROM Customers C
WHERE NOT EXISTS (SELECT cod_customer
          FROM Orders O
            WHERE C.cod_customer = O.cod_customer);

/* SQL Queries - Operator "Any" and "All" */
SELECT description
FROM Products
WHERE cod_product = ANY
    (SELECT cod_product
     FROM OrdersProducts
     WHERE Quantity = 1)

Select description, available_stock
From Products
Where available_stock >= ANY (select distinct minimal_stock
              From Products);

Select name, address
From Customers
Where cod_customer < ANY (Select cod_customer From Orders);

Select description, available_stock
From Products
Where unit_price >= ALL (select distinct unit_price
              From Products);
```

```
/* SQL Queries - Operations with Sets */
Select description
From Products
Where cod_product > 3
UNION
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 3
UNION ALL
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 3 and cod_product > 1;

Select description
From Products
Where cod_product > 1 and cod_product <= 3;

/* SQL Queries - Joins */
SELECT distinct C.Name, C.Address
FROM Customers C
INNER JOIN Orders O
 ON C.cod_customer=O.cod_customer
ORDER BY c.Name;

SELECT distinct C.Name, C.Address
FROM Customers C NATURAL JOIN Orders O
ORDER BY c.Name;

SELECT distinct C.Name, C.country
FROM Customers C
LEFT JOIN Orders O
 ON C.cod_customer=O.cod_customer
ORDER BY c.Name;

SELECT distinct O.date_order, C.name
FROM Orders O
RIGHT JOIN Customers C
 ON O.cod_customer=C.cod_customer
ORDER BY O.date_order;

SELECT c.name, o.cod_order
FROM Customers C
LEFT JOIN Orders O ON C.cod_customer = O.cod_customer
UNION
SELECT c.name, o.cod_order
FROM Customers C
RIGHT JOIN Orders O ON C.cod_customer = O.cod_customer;

/* SQL Queries - Views */
Create view Products_BelowStockMin as
Select cod_product, description
From products
```

Where available_stock < minimal_stock

Select * from Products_BelowStockMin

Create view descProductsOrders as
Select distinct description, unit_price
From products p, ordersproducts op
Where p.cod_product = op.cod_product
order by unit_price DESC

Select * from descProductsOrders

/* SQL Queries - System Data */
SELECT NOW() AS CurrentDateTime

SELECT CURRENT_USER() AS CurrentLoggedUser

Annex II - Script for MS SQL Server Databases

```
/* Creation of tables */
create table Products
(
   cod_product   integer,
   description    varchar(50) NOT NULL,
   unit_price     DECIMAL(10,2),
   available_stock integer,
   minimal_stock   integer default 0,
   CONSTRAINT Products_pk PRIMARY KEY (cod_product)
);

create table Customers
(
   cod_customer   integer,
   name    varchar(50) NOT NULL,
   address      varchar(95) Default 'Unknown',
   zip_code char(8),
   country   varchar(40) Default 'Portugal',
   telephone varchar(15),
   CONSTRAINT Customers_pk PRIMARY KEY (cod_customer)
);

create table Orders
(
   cod_order    integer,
   date_order date,
   date_delivery date,
   cod_customer integer,
   CONSTRAINT Orders_pk PRIMARY KEY (cod_order),
   CONSTRAINT Orders_Cust_fk FOREIGN KEY (cod_customer)
              REFERENCES Customers(cod_customer)
);

create table OrdersProducts
(
   cod_product    integer,
   cod_order integer,
   quantity decimal(2,0),
   CONSTRAINT OrdersProducts_pk PRIMARY KEY (cod_product, cod_order),
   CONSTRAINT OrdersProducts_Prod_fk FOREIGN KEY (cod_product)
              REFERENCES Products(cod_product),
   CONSTRAINT OrdersProducts_Orders_fk FOREIGN KEY (cod_order)
              REFERENCES Orders(cod_order)
);

/* Insert data into tables */
Insert Into Products (cod_product, description, unit_price)
Values (1, 'Eggs', 2.49);
Insert Into Products (cod_product, description, unit_price)
Values (2, 'Ice Cream', 3.99);
Insert Into Products (cod_product, description, unit_price)
Values (3, 'Soda', 0.65);
Insert Into Products (cod_product, description, unit_price)
Values (4, 'Cheese', 2.89);
Insert Into Products (cod_product, description, unit_price)
Values (5, 'Pork Meat', 3.10);

Insert Into Customers (cod_customer, name)
Values (1, 'Anne');
```

```
Insert Into Customers (cod_customer, name)
Values (2, 'Peter');
Insert Into Customers (cod_customer, name)
Values (3, 'Elena');
Insert Into Customers (cod_customer, name)
Values (4, 'Shirley');
Insert Into Customers (cod_customer, name)
Values (5, 'John');

Insert Into Orders Values (1, '2015-12-21', '2015-12-21', 1);
Insert Into Orders Values (2, '2015-12-22', '2015-12-23', 1);
Insert Into Orders Values (3, '2015-12-22', '2015-12-27', 2);
Insert Into Orders Values (4, '2015-12-27', '2015-12-30', 3);
Insert Into Orders Values (5, '2015-12-30', '2015-12-31', 3);

Insert Into OrdersProducts Values (1, 1, 1);
Insert Into OrdersProducts Values (2, 1, 1);
Insert Into OrdersProducts Values (1, 2, 2);
Insert Into OrdersProducts Values (5, 3, 7);
Insert Into OrdersProducts Values (4, 3, 4);
Insert Into OrdersProducts Values (3, 3, 5);
Insert Into OrdersProducts Values (2, 3, 5);
Insert Into OrdersProducts Values (1, 4, 8);
Insert Into OrdersProducts Values (2, 4, 2);
Insert Into OrdersProducts Values (1, 5, 3);
Insert Into OrdersProducts Values (2, 5, 3);
Insert Into OrdersProducts Values (4, 5, 5);

/* Update data previously recorded in tables */
Update Products
Set unit_price = 1.99
Where description = 'Eggs';

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

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

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

/* Delete data from tables */
Delete From OrdersProducts
Where cod_product = 5;

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

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

```
/* SQL Queries - Basic Structure */
Select * From Customers;

Select Name, Country From Customers;

Select distinct Country From Customers;

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

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

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;

select cod_product, description, available_stock - minimal_stock as marginStock
From Products;

select cod_product, description, available_stock%minimal_stock as modResult
From Products;

/* SQL Queries - Comparing Strings */
SELECT * FROM Customers
 WHERE name LIKE 'E%';

SELECT * FROM Customers
 WHERE name LIKE '%a';

SELECT * FROM Customers
 WHERE name LIKE '%e%';

SELECT * FROM Customers
 WHERE name LIKE '_lena';

SELECT * FROM Customers
 WHERE name LIKE 'E_e_a';

/* SQL Queries - Aggregation Operators */
Select AVG(unit_price) From Products;

Select ROUND(AVG(unit_price),2) From Products;

SELECT description, unit_price, available_stock
FROM Products
WHERE unit_price>(SELECT AVG(unit_price) FROM Products);

Select Count(*) From Customers;

Select Count(distinct country) From Customers;

SELECT MAX(unit_price) AS HighestPrice FROM Products;

SELECT MAX(unit_price) AS HighestPrice, AVG(unit_price) AS AveragePrice
FROM Products;
```

```
SELECT MIN(unit_price) AS SmallestPrice FROM Products;

SELECT MIN(unit_price) AS SmallestPrice, AVG(unit_price) AS AveragePrice
FROM Products;

SELECT SUM(available_stock) AS TotalStock FROM Products;

SELECT COUNT(*) as TotalProducts,
     AVG(unit_price) AS AveragePrice,
     MAX(unit_price) AS HighestPrice,
     MIN(unit_price) AS SmallestPrice,
     SUM(available_stock) AS TotalStock
FROM Products;

/* SQL Queries - Scalar Functions */
Select UPPER(name) as UPPERName, LOWER(name) as LOWERName
from Customers;

Select name, LEN(name) as SizeName
from Customers;

Select description, round(unit_price, 0) as RPrice0,
     round(unit_price, 1) as RPrice1,
     round(unit_price, 2) as RPrice2
From Products;

/* SQL Queries - Grouping Elements */
Select cod_customer, count(cod_customer) as nOrders
From Orders
Group By cod_customer;

select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order;

select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order
Having sum(quantity) > 10;

/* SQL Queries - Ordering Data */
Select *
From Customers
Order By Name;

Select *
From Customers
Order By Name DESC;

/* SQL Queries - Returning Top Elements */
SELECT Top 3 *
FROM Customers
Order By Name;

/* SQL Queries - Sub-queries */
Select description
From Products
Where unit_price = (Select max(unit_price)
          From Products);
```

```
Select name
From Customers
Where cod_customer = (Select max(distinct cod_customer)
           From Orders
             Where date_delivery >= '2015-12-26');

Select date_delivery
From Orders
Where cod_order = (Select max(cod_order)
           From OrdersProducts
           Where quantity > 3);

/* SQL Queries - Operator "In" and "Exists" */
SELECT *
FROM Orders
WHERE cod_order IN (1, 3, 5);

SELECT *
FROM Customers
WHERE name IN ('Peter', 'Elena');

SELECT *
FROM Customers
WHERE cod_customer IN (SELECT cod_customer
             From Orders);

SELECT *
FROM Customers C
WHERE EXISTS ( SELECT cod_customer
FROM Orders O
WHERE C.cod_customer = O.cod_customer);

SELECT distinct Country
FROM Customers C
WHERE NOT EXISTS (SELECT cod_customer
         FROM Orders O
           WHERE C.cod_customer = O.cod_customer);

/* SQL Queries - Operator "Any" and "All" */
SELECT description
FROM Products
WHERE cod_product = ANY
    (SELECT cod_product
     FROM OrdersProducts
     WHERE Quantity = 1)

Select description, available_stock
From Products
Where available_stock >= ANY (select distinct minimal_stock
              From Products);

Select name, address
From Customers
Where cod_customer < ANY (Select cod_customer From Orders);

Select description, available_stock
From Products
Where unit_price >= ALL (select distinct unit_price
            From Products);
```

```
/* SQL Queries - Operations with Sets */
Select description
From Products
Where cod_product > 3
UNION
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 3
UNION ALL
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 3
INTERSECT
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 1
EXCEPT
Select description
From Products
Where cod_product > 3;

/* SQL Queries - Joins */
SELECT distinct C.Name, C.Address
FROM Customers C
INNER JOIN Orders O
 ON C.cod_customer=O.cod_customer
ORDER BY c.Name;

SELECT distinct C.Name, C.country
FROM Customers C
LEFT JOIN Orders O
 ON C.cod_customer=O.cod_customer
ORDER BY c.Name;

SELECT distinct O.date_order, C.name
FROM Orders O
RIGHT JOIN Customers C
 ON O.cod_customer=C.cod_customer
ORDER BY O.date_order;

SELECT c.name, O.COD_ORDER
FROM Customers C
FULL OUTER JOIN Orders O
 ON C.cod_customer=O.cod_customer
ORDER BY C.Name;

/* SQL Queries - Views */
Create view Products_BelowStockMin as
Select cod_product, description
```

```
From products
Where available_stock < minimal_stock

Select * from Products_BelowStockMin

Create view descProductsOrders as
Select distinct description, unit_price
From products p, ordersproducts op
Where p.cod_product = op.cod_product
order by unit_price DESC

Select * from descProductsOrders

/* SQL Queries - System Data */
SELECT GETDATE() AS CurrentDateTime

SELECT CURRENT_USER AS CurrentLoggedUser
```

Annex III - Script for Oracle Databases

```
/* Creation of tables */
create table Products
(
    cod_product    integer,
    description     varchar(50) NOT NULL,
    unit_price      DECIMAL(10,2),
    available_stock integer,
    minimal_stock   integer default 0,
    CONSTRAINT Products_pk PRIMARY KEY (cod_product)
);

create table Customers
(
    cod_customer    integer,
    name     varchar(50) NOT NULL,
    address      varchar(95) Default 'Unknown',
    zip_code char(8),
    country    varchar(40) Default 'Portugal',
    telephone varchar(15),
    CONSTRAINT Customers_pk PRIMARY KEY (cod_customer)
);

create table Orders
(
    cod_order    integer,
    date_order date,
    date_delivery date,
    cod_customer integer,
    CONSTRAINT Orders_pk PRIMARY KEY (cod_order),
    CONSTRAINT Orders_Cust_fk FOREIGN KEY (cod_customer)
                REFERENCES Customers(cod_customer)
);

create table OrdersProducts
(
    cod_product    integer,
    cod_order integer,
    quantity number(2),
    CONSTRAINT OrdersProducts_pk PRIMARY KEY (cod_product, cod_order),
    CONSTRAINT OrdersProducts_Prod_fk FOREIGN KEY (cod_product)
                REFERENCES Products(cod_product),
    CONSTRAINT OrdersProducts_Orders_fk FOREIGN KEY (cod_order)
                REFERENCES Orders(cod_order)
);

/* Insert data into tables */
Insert Into Products (cod_product, description, unit_price)
Values (1, 'Eggs', 2.49);
Insert Into Products (cod_product, description, unit_price)
Values (2, 'Ice Cream', 3.99);
Insert Into Products (cod_product, description, unit_price)
Values (3, 'Soda', 0.65);
Insert Into Products (cod_product, description, unit_price)
Values (4, 'Cheese', 2.89);
Insert Into Products (cod_product, description, unit_price)
Values (5, 'Pork Meat', 3.10);

Insert Into Customers (cod_customer, name)
Values (1, 'Anne');
```

```
Insert Into Customers (cod_customer, name)
Values (2, 'Peter');
Insert Into Customers (cod_customer, name)
Values (3, 'Elena');
Insert Into Customers (cod_customer, name)
Values (4, 'Shirley');
Insert Into Customers (cod_customer, name)
Values (5, 'John');

Insert Into Orders Values (1, '2015-12-21', '2015-12-21', 1);
Insert Into Orders Values (2, '2015-12-22', '2015-12-23', 1);
Insert Into Orders Values (3, '2015-12-22', '2015-12-27', 2);
Insert Into Orders Values (4, '2015-12-27', '2015-12-30', 3);
Insert Into Orders Values (5, '2015-12-30', '2015-12-31', 3);

Insert Into OrdersProducts Values (1, 1, 1);
Insert Into OrdersProducts Values (2, 1, 1);
Insert Into OrdersProducts Values (1, 2, 2);
Insert Into OrdersProducts Values (5, 3, 7);
Insert Into OrdersProducts Values (4, 3, 4);
Insert Into OrdersProducts Values (3, 3, 5);
Insert Into OrdersProducts Values (2, 3, 5);
Insert Into OrdersProducts Values (1, 4, 8);
Insert Into OrdersProducts Values (2, 4, 2);
Insert Into OrdersProducts Values (1, 5, 3);
Insert Into OrdersProducts Values (2, 5, 3);
Insert Into OrdersProducts Values (4, 5, 5);

/* Update data previously recorded in tables */
Update Products
Set unit_price = 1.99
Where description = 'Eggs';

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

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

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

/* Delete data from tables */
Delete From OrdersProducts
Where cod_product = 5;

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

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

```
/* SQL Queries - Basic Structure */
Select * From Customers;

Select Name, Country From Customers;

Select distinct Country From Customers;

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

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

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;

select cod_product, description, available_stock - minimal_stock as marginStock
From Products;

select cod_product, description, mod(available_stock, minimal_stock) as modResult
From Products;

/* SQL Queries - Comparing Strings */
SELECT * FROM Customers
 WHERE name LIKE 'E%';

SELECT * FROM Customers
 WHERE name LIKE '%a';

SELECT * FROM Customers
 WHERE name LIKE '%e%';

SELECT * FROM Customers
 WHERE name LIKE '_lena';

SELECT * FROM Customers
 WHERE name LIKE 'E_e_a';

/* SQL Queries - Aggregation Operators */
Select AVG(unit_price) From Products;

Select ROUND(AVG(unit_price),2) From Products;

SELECT description, unit_price, available_stock
FROM Products
WHERE unit_price>(SELECT AVG(unit_price) FROM Products);

Select Count(*) From Customers;

Select Count(distinct country) From Customers;

SELECT MAX(unit_price) AS HighestPrice FROM Products;

SELECT MAX(unit_price) AS HighestPrice, AVG(unit_price) AS AveragePrice
FROM Products;
```

```
SELECT MIN(unit_price) AS SmallestPrice FROM Products;

SELECT MIN(unit_price) AS SmallestPrice, AVG(unit_price) AS AveragePrice
FROM Products;

SELECT SUM(available_stock) AS TotalStock FROM Products;

SELECT COUNT(*) as TotalProducts,
    AVG(unit_price) AS AveragePrice,
    MAX(unit_price) AS HighestPrice,
    MIN(unit_price) AS SmallestPrice,
    SUM(available_stock) AS TotalStock
FROM Products;

/* SQL Queries - Scalar Functions */
Select UPPER(name) as UPPERName, LOWER(name) as LOWERName
from Customers;

Select name, LENGTH(name) as SizeName
from Customers;

Select description, round(unit_price, 0) as RPrice0,
    round(unit_price, 1) as RPrice1,
    round(unit_price, 2) as RPrice2
From Products;

/* SQL Queries - Grouping Elements */
Select cod_customer, count(cod_customer) as nOrders
From Orders
Group By cod_customer;

select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order;

select O.cod_customer, O.cod_order, sum(quantity), count(cod_product)
From Orders O, OrdersProducts OP
Where O.COD_ORDER = OP.COD_ORDER
Group By O.cod_customer, O.cod_order
Having sum(quantity) > 10;

/* SQL Queries - Ordering Data */
Select *
From Customers
Order By Name;

Select *
From Customers
Order By Name DESC;

/* SQL Queries - Returning Top Elements */
SELECT *
FROM Customers
Where RowNum <= 3
Order By Name;

/* SQL Queries - Sub-queries */
Select description
From Products
Where unit_price = (Select max(unit_price)
```

```
        From Products);

Select name
From Customers
Where cod_customer = (Select max(distinct cod_customer)
            From Orders
            Where date_delivery >= '2015-12-26');

Select date_delivery
From Orders
Where cod_order = (Select max(cod_order)
            From OrdersProducts
            Where quantity > 3);

/* SQL Queries - Operator "In" and "Exists" */
SELECT *
FROM Orders
WHERE cod_order IN (1, 3, 5);

SELECT *
FROM Customers
WHERE name IN ('Peter', 'Elena');

SELECT *
FROM Customers
WHERE cod_customer IN (SELECT cod_customer
            From Orders);

SELECT *
FROM Customers C
WHERE EXISTS (SELECT cod_customer
        FROM Orders O
        WHERE C.cod_customer = O.cod_customer);

SELECT distinct Country
FROM Customers C
WHERE NOT EXISTS (SELECT cod_customer
        FROM Orders O
        WHERE C.cod_customer = O.cod_customer);

/* SQL Queries - Operator "Any" and "All" */
SELECT description
FROM Products
WHERE cod_product = ANY
    (SELECT cod_product
     FROM OrdersProducts
     WHERE Quantity = 1)

Select description, available_stock
From Products
Where available_stock >= ANY (select distinct minimal_stock
                From Products);

Select name, address
From Customers
Where cod_customer < ANY (Select cod_customer From Orders);

Select description, available_stock
From Products
Where unit_price >= ALL (select distinct unit_price
                From Products);
```

```
/* SQL Queries - Operations with Sets */
Select description
From Products
Where cod_product > 3
UNION
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 3
UNION ALL
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 3
INTERSECT
Select description
From Products
Where cod_product > 1;

Select description
From Products
Where cod_product > 1
MINUS
Select description
From Products
Where cod_product > 3;

/* SQL Queries - Joins */
SELECT distinct C.Name, C.Address
FROM Customers C
INNER JOIN Orders O
 ON C.cod_customer=O.cod_customer
ORDER BY c.Name;

SELECT distinct C.Name, C.Address
FROM Customers C NATURAL JOIN Orders O
ORDER BY c.Name;

SELECT distinct C.Name, C.country
FROM Customers C
LEFT JOIN Orders O
 ON C.cod_customer=O.cod_customer
ORDER BY c.Name;

SELECT distinct O.date_order, C.name
FROM Orders O
RIGHT JOIN Customers C
 ON O.cod_customer=C.cod_customer
ORDER BY O.date_order;

SELECT c.name, O.COD_ORDER
FROM Customers C
FULL OUTER JOIN Orders O
 ON C.cod_customer=O.cod_customer
```

```
ORDER BY C.Name;

/* SQL Queries - Views */
Create view Products_BelowStockMin as
Select cod_product, description
From products
Where available_stock < minimal_stock

Select * from Products_BelowStockMin

Create view descProductsOrders as
Select distinct description, unit_price
From products p, ordersproducts op
Where p.cod_product = op.cod_product
order by unit_price DESC

Select * from descProductsOrders

/* SQL Queries - System Data */
SELECT TO_CHAR
    (SYSDATE, 'MM-DD-YYYY HH24:MI:SS') "NOW"
     FROM DUAL;

select user from dual;
```