Exercises Program Structures

1. Example

```
DECLARE @sum FLOAT
SELECT @sum=SUM(Price)
FROM otherBooks
```

Declaration of the variable @sum of type FLOAT. @sum is assigned the sum of the Price of the “otherBooks” table.

```
DECLARE @avg FLOAT
SELECT @avg=AVG(Price)
FROM otherBooks
```

Declaration of the variable @avg of type FLOAT. @avg is assigned the average Price of the "otherBooks" table.

```
DECLARE @priceNb INT
SELECT @priceNb=COUNT(Price)
FROM otherBooks
```

Declaration of the @priceNb variable of type INT. @priceNb is assigned the number of prices in the Price column of the “otherBooks” table.

```
DECLARE @nbLines int
SELECT @nbLines=COUNT(*)
FROM otherBooks
```

Declaration of the variable @nbLines of type INT. @nbLines is assigned the number of rows in the "otheBooks" table.

Display the values of the following variables :

SELECT @nbLines AS "Nb Lines", @priceNb AS "Nb Price",@sum AS "sum of Price",@avg AS "Average Price"

NB. In the example, **COUNT(*) counts the number of lines** in the table, while **COUNT(Price) counts the number of different NULL values in the Price column.**

The COUNT, AVG, SUM, MAX, and MIN aggregate functions eliminate the NULL values before computing (with the exception of COUNT(*)).

Reminder : in a query : SELECT … FROM … GROUP BY …

- ➢ The SELECT clause can not specify a column which is not included in the grouping defined by the GROUP BY clause.
- ➢ The HAVINGA clause is only evaluated after the execution of the aggregate function.

2. Display the publisher ID, and for each the type of books whose Average Book Price is between the Average Price of psychology books and the Average Price of business books. You must declare and set the values for the average price of both genre of books in separate variables.

3. Display the max and min Price of each books, with each book's respective title.

4. With reference to the Pubs database :
   a. declare the variable @publ set the text 'Binnet & Hardley' as its value.
   b. assign to the variable @nb the number of books published by the publisher @publ
   c. assign to the variable @m the average price of the books published the the publisher @publ
   d. display the values of @nb and @m
   e. display the books published by @publ whose Price is below the average Price.
   f. assign the books returned in e to a new variable, and display that variable.
   g. display the books published by @publ whose Price is above the Average Price.
   h. assign the books returned in g to a new variable, and display that variable.

5. Use the case statement for the following :

Write SQL code that tests the contents of a variable @a and displays, either :

- 10 if @a is 10; 20 if @a is 20 and 100 otherwise, or
- 'Less than 10', 'less than 20' or 'greater than 20' depending on whether @a is smaller than 10, between 10 and 20 or greater than 20.

(Use an intermediary variable to store the value of the result string.)

6. Calculate the amount of sales for a given store. Depending on the sun, return one of the following messages :

Less than $ 1000: 'Store to close'

Between $ 1000 and $ 1500: 'it can work'

Above $ 1500: 'Ok, that's good'

7. WHILE loop

What does this code do?

```
DECLARE @i INT = 0
WHILE @i<20
    BEGIN
        IF @i%2!=0
            BEGIN
                SET @i = @i + 1
                CONTINUE
            END
        PRINT 'loop ' + CONVERT(CHAR,@i)
        SET @i = @i + 1
    END
```

8. Write SQL code that produces the following output :

```
Orders with quantities between 0 and 10:
OrderNb              Store_ID Quantity
-------------------- ---------- --------
722a                 6380       3
6871                 6380       5
D4482                7067       10
TQ456                7896       10
423LL930             8042       10

Orders with quantities between 10 and 20 :
OrderNb              Store_ID Quantity
-------------------- ---------- --------
D4482                7067       10
TQ456                7896       10
423LL930             8042       10
423LL922             8042       15
P3087a               7131       15
QQ2299               7896       15
P2121                7067       20
P2121                7067       20
N914008              7131       20
P3087a               7131       20

Orders with quantities between 20 and 30 :
OrderNb              Store_ID Quantity
```

```
-------------------- ---------- --------
P2121                7067       20
P2121                7067       20
N914008              7131       20
P3087a               7131       20
P3087a               7131       25
P3087a               7131       25
P723                 8042       25
N914014              7131       25
QA879.1              8042       30

Orders with quantities between 30 and 40 :
OrderNb          Store_ID Quantity
-------------------- ---------- --------
QA879.1              8042       30
X999                 7896       35
P2121                7067       40

Orders with quantities between 40 and 50 :
OrderNb          Store_ID Quantity
-------------------- ---------- --------
P2121                7067       40
A2976                7066       50

Orders with quantities between 50 and 60 :
OrderNb          Store_ID Quantity
-------------------- ---------- --------
A2976                7066       50
```

9. (BONUS) Write T-SQL code that performs the following tasks :

- Delete the TempBooks table if it exists
- create a new TempBook table with two columns : the book's title and its advance pay (include only books with non NULL advances). Use the INTO clause as shown below to create a table based on the results of a query :

```
SELECT title AS 'Books', advance AS 'Advance' INTO TempBooks
FROM titles WHERE advance IS NOT NULL
```

- increase the advance in increments of 10% until the advance exceeds 10000. Use **UPDATE (see the Glossary.docx document)**
- display the TempBooks table
- delete the TempBooks table