Week#3 - Assignment 1

MS Connect

Vijay Sekhar G

Objectives

- Get comfortable with SQL Server Scripting.
- Start thinking more carefully.
- Solve problems using T-SQL Queries/Stored Procedures.

Reasonable

1. Communicating with colleagues about problem problems in English (or some other spoken language).
2. Discussing the assignment material with others in order to understand it better.
3. Helping a colleagues identify a bug in his or her code, as by viewing, compiling, or running his or her code, even on your own computer.
4. Incorporating snippets of code that you find online or elsewhere into your own code, provided that those snippets are not themselves solutions to assigned problems and that you cite the snippets' origins.
5. Sending or showing code that you've written to someone, possibly a colleagues, so that he or she might help you identify and fix a bug.

Deliverables

- Document your solutions in word file.
- Write proper comments for each line in your source code.
- Document the output of your program.
- Your program should address the problem, there should NOT be any deviations in output.

① There are two tables involved in this problem. The first table is *MovieReleaseDates* which will contain the names of movies along with their release dates.

Sample Data

Table: MovieReleaseDates

```
Movie1      Movie2      Movie3      Movie4      Movie5
----------  ----------  ----------  ----------  ----------
2010-01-20  2010-02-15  2010-02-02  2010-03-31  2010-04-16
2010-01-22  2010-02-16  2010-02-04  2010-04-05  2010-04-18
```

The second table will hold the details of releases that happened in different cities. Here is how the table looks like:

Table: MovieReleasedIn

```
City      MovieNames ReleaseDate
--------- ---------- -----------
Bangalore Movie1     2010-01-22
Bangalore Movie2     2010-02-15
Bangalore Movie3     2010-02-04
Bangalore Movie4     2010-04-05
Bangalore Movie5     2010-04-16
```

The challenge is to find out which movies are released on which dates and in which cities.

Expected Results

```
City      Sun Mon           Tue Wed Thu            Fri            Sat
--------- --- ------------ --- --- -------------- -------------- ---
Bangalore NA  NA           NA  NA  NA             Movie1(01/22)  NA
Bangalore NA  Movie2(02/15)NA  NA  NA             NA             NA
Bangalore NA  NA           NA  NA  Movie3(02/04)  NA             NA
Bangalore NA  Movie4(04/05)NA  NA  NA             NA             NA
Bangalore NA  NA           NA  NA  NA             Movie5(04/16)  NA
```

Rules

1. Output must be sorted in ASCENDING ORDER of City.
2. The days when the movies are NOT AVAILABLE(NA) for the respected cities, should be marked with NA.
3. Column names should respect the desired output shown.

Sample Scripts

Use the TSQL Script given below to generate the source table and fill them with sample data.

```
DECLARE tblMovieReleaseDates TABLE(
Movie1 DATETIME,
Movie2 DATETIME,
Movie3 DATETIME,
Movie4 DATETIME,
Movie5 DATETIME
)
INSERT INTO @tblMovieReleaseDates
SELECT '01/20/2010','02/15/2010','02/02/2010','03/31/2010','0
4/16/2010'
UNION ALL
SELECT '01/22/2010','02/16/2010','02/04/2010','04/05/2010','0
4/18/2010'
SELECT * FROM @tblMovieReleaseDates
GO

DECLARE @tblMovieReleasedIn TABLE(
City VARCHAR(20),
MovieNames VARCHAR(20),
ReleaseDate Datetime
)
INSERT INTO @tblMovieReleasedIn
SELECT 'Bangalore','Movie1','01/22/2010' UNION ALL
SELECT 'Bangalore','Movie2','02/15/2010' UNION ALL
SELECT 'Bangalore','Movie3','02/04/2010' UNION ALL
SELECT 'Bangalore','Movie4','04/05/2010' UNION ALL
SELECT 'Bangalore','Movie5','04/16/2010'

SELECT * FROM @tblMovieReleasedIn
```

② Count the number of mobile phones each person has and generate a summary row.

Sample Data

PersonTable

```
PersonId PersonName
-------- ------------------
1        Deepak Kumar Goyal
2        Niladri Biswas
```

ContactDetail Table

```
PersonId MobileNumber
-------- ----------------------
1        9886551234,9445612356
2        9886334510
```

Expected Results

```
PersonId PersonName          MobileNumber           RecordCount
-------- ------------------- ---------------------  -----------
1        Deepak Kumar Goyal  9886551234,9445612356  2
2        Niladri Biswas      9886334510             1

Total Mobiles:                              3
```

Rules

1. PersonId should be sorted in Ascending Order.
2. If a person does not have any mobile, then his/her Record Count will be 0.
3. If a person does not have any name (Blank or NULL assigned) or Mobile number, his/her record should not be counted.
4. If a person does not have any name (Blank or NULL assigned) but is having Mobile Number, his/her record should not be counted.

Sample Scripts

Use the TSQL Script given below to generate the source table and fill them with sample data.

```
DECLARE @Person TABLE(PersonId INT,PersonName VARCHAR(20))
INSERT INTO @Person
SELECT 1,'Deepak Kumar Goyal' UNION ALL
SELECT 2,'Niladri Biswas'
SELECT * FROM @Person

DECLARE @ContactDetail TABLE(PersonId INT,MobileNumber VARCHAR(100))
INSERT INTO @ContactDetail
SELECT 1, '9886551234,9445612356' UNION ALL
SELECT 2,'9886334510'
SELECT * FROM @ContactDetail
```

③ The problem is all about merging the data based on certain conditions.

Sample Data

```
ID Name
-- ------------------
1  Deepak Kumar Goyal
2  Niladri Biswas
2  Pratik Shaw
3  Sumi Girijan
3  Suresh Beldar
3  Jeeva Baby
```

The problem is that if the Id's are exactly two in number (e.g. count of ID = 2 is exactly two) then the names should be concatenated with 'OR'. When Ids are more than two (e.g. Id # 3 has a count of 3) , the values should be concatenated with 'AND'

Expected Results

```
ID Name
-- -------------------------------------------
1  Deepak Kumar Goyal
2  Niladri Biswas OR Pratik Shaw
3  Sumi Girijan AND Suresh Beldar AND Jeeva Baby
```

Rules

1. Column names should respect the desired output shown.
2. There should be exactly 1 space before and after the OR/AND clause.
3. Output should be in the order of Ascending Id.
4. Names should not be repeated even if it is given in the sample input i.e. if X is appearing more than once in the input for id = 10, the desired output will have only one such record.
5. Names cannot be blank e.g. for id = 10 if the Name field is ''(blank), the program should ignore that.

Sample Scripts

Use the TSQL Script given below to generate the source table and fill them with sample data.

```
DECLARE @tmpData TABLE(
ID INT,
NAME VARCHAR(MAX)
)
INSERT INTO @tmpData (ID,[NAME]) SELECT 1, 'Deepak Kumar
Goyal'
INSERT INTO @tmpData (ID,[NAME]) SELECT 2, 'Niladri Biswas'
INSERT INTO @tmpData (ID,[NAME]) SELECT 2, 'Pratik Shaw'
INSERT INTO @tmpData (ID,[NAME]) SELECT 3, 'Sumi Girijan'
INSERT INTO @tmpData (ID,[NAME]) SELECT 3, 'Suresh Beldar'
INSERT INTO @tmpData (ID,[NAME]) SELECT 3, 'Jeeva Baby'
```

④ The problem is all about finding the factorial of numbers.

Factorial of 3 is 1*2*3 = 6 i.e. the factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n.

Sample Data

```
Nums
-----------
0
1
3
5
10
```

Expected Results

```
Nums        Factorial
----------- -----------
0                     1
1                     1
3                     6
5                   120
10              3628800
```

Rules

1. Nums should be sorted in Ascending Order.
2. The output should be in the same way as it has been shown. Column names should be exactly the same and the result must be sorted in Ascending order of Nums.
3. The program has to be done by a single query and should begin either with a SELECT or WITH statement with no variables, temporary table, table variables permitted.

Sample Scripts

Use the TSQL Script given below to generate the source table and fill them with sample data.

```
DECLARE @Fact TABLE(Nums INT)
INSERT INTO @Fact
SELECT 0 UNION ALL
SELECT 1 UNION ALL
SELECT 3 UNION ALL
SELECT 5 UNION ALL
SELECT 10
SELECT * FROM @Fact
```

⑤ The problem is to find the employees with the second highest salary in each department. However, it is a little more complicated because if two employees have the same salary, you need to list both of them.

Sample Data

EmployeeID	EmployeeName	Department	Salary
1	T Cook	Finance	40000.00
2	D Michael	Finance	25000.00
3	A Smith	Finance	25000.00
4	D Adams	Finance	15000.00
5	M Williams	IT	80000.00
6	D Jones	IT	40000.00
7	J Miller	IT	50000.00
8	L Lewis	IT	50000.00
9	A Anderson	Back-Office	25000.00
10	S Martin	Back-Office	15000.00
11	J Garcia	Back-Office	15000.00
12	T Clerk	Back-Office	10000.00

Expected Results

EmployeeID	EmployeeName	Department	Salary
10	S Martin	Back-Office	15000.00
11	J Garcia	Back-Office	15000.00
2	D Michael	Finance	25000.00
3	A Smith	Finance	25000.00
7	J Miller	IT	50000.00
8	L Lewis	IT	50000.00

Sample Scripts

Use the TSQL Script given below to generate the source table and fill them with sample data.

```
DECLARE @Employees TABLE(
   EmployeeID INT IDENTITY,
   EmployeeName VARCHAR(15),
   Department VARCHAR(15),
   Salary NUMERIC(16,2)
)

INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('T Cook','Finance', 40000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('D Michael','Finance', 25000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('A Smith','Finance', 25000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('D Adams','Finance', 15000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('M Williams','IT', 80000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('D Jones','IT', 40000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('J Miller','IT', 50000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('L Lewis','IT', 50000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('A Anderson','Back-Office', 25000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('S Martin','Back-Office', 15000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('J Garcia','Back-Office', 15000)
INSERT INTO @Employees(EmployeeName, Department, Salary)
VALUES('T Clerk','Back-Office', 10000)
```

⑦ In a large project management application, there is a category of resources which are managed in a special way. During the planning phase you have an activity with a planned start date and a planned finish date and a monthly sequence of estimated effort values for one or more resources. A period always refers to a real calendar month with period 1 being the month of the planned start date. But the first period and last period may not use up a full month.

Sample Data

Activity Table

```
TaskID Planned_Start Planned_Finish
------ ------------- --------------
1      2012-01-19    2012-03-09
```

Effort Table

```
TaskID ResID Period Amount
------ ----- ------ ------
1      1     1      75.0
1      1     2      140.0
1      1     3      50.0
1      1     4      60.0
1      2     2      30.0
1      2     3      10.0
```

When the activity is completed these monthly effort values are transformed into weekly actual values.

You first need to handle situations of there being monthly periods beyond the planned finish date. All the effort amounts in the monthly periods beyond the last period must be added into the period corresponding to the planned finish date.

You must then split the monthly values into weekly values. Each weekly value is calculated in proportion to the number of days of the month in which it resides. A weekly value may need to be generated twice if it crosses the boundary betweeen two months. In that case each weekly value must be proportional to the number of days the week occupies in each month. A week is assumed to start on a Monday and finish on a Sunday with all days being working days. In the output the weekdate must correspond to the Sunday of that week and the monthdate must have the day set to 1.

Expected Results

```
TaskID ResID WeekDate   MonthDate  Amount
------ ----- ---------- ---------- ------
1      1     2012-01-22 2012-01-01 23.1
1      1     2012-01-29 2012-01-01 40.4
1      1     2012-02-05 2012-01-01 11.5
1      1     2012-02-05 2012-02-01 24.1
1      1     2012-02-12 2012-02-01 33.8
1      1     2012-02-19 2012-02-01 33.8
1      1     2012-02-26 2012-02-01 33.8
1      1     2012-03-04 2012-02-01 14.5
1      1     2012-03-04 2012-03-01 48.9
1      1     2012-03-11 2012-03-01 61.1
1      2     2012-02-05 2012-02-01 5.2
1      2     2012-02-12 2012-02-01 7.2
1      2     2012-02-19 2012-02-01 7.2
1      2     2012-02-26 2012-02-01 7.2
1      2     2012-03-04 2012-02-01 3.1
1      2     2012-03-04 2012-03-01 4.4
1      2     2012-03-11 2012-03-01 5.6
```

Rules

1. The output should be ordered by TaskID, ResID, WeekDate, MonthDate.
2. The final amount of effort value must be rounded to one decimal place.
3. There can be gaps in the monthly period sequence.

Sample Scripts

Use the TSQL Script given below to generate the source table and fill them with sample data.

```
IF OBJECT_ID('Activity','U') IS NOT NULL
DROP TABLE Activity
GO

CREATE TABLE Activity(
TaskID INT,
Planned_Start DATE,
Planned_Finish DATE
)
GO
INSERT INTO Activity(TaskID,Planned_Start,Planned_Finish)
SELECT 1,'2012-01-19','2012-03-09'

SELECT * FROM Activity
GO

IF OBJECT_ID('Effort','U') IS NOT NULL
DROP TABLE Effort
GO

CREATE TABLE Effort(
TaskID INT,
ResID INT,
```

```
Period INT,
Amount FLOAT
)
GO

INSERT INTO Effort(TaskID,ResID,Period,Amount)
SELECT 1,1,1,75.0 UNION ALL
SELECT 1,1,2,140.0 UNION ALL
SELECT 1,1,3,50.0 UNION ALL
SELECT 1,1,4,60.0 UNION ALL
SELECT 1,2,2,30.0 UNION ALL
SELECT 1,2,3,10.0

SELECT * FROM Effort
GO
```

⑧ The primary goal of this problem is to demonstrate the power of Recursive CTEs introduced in SQL Server 2005. Recursive CTEs allow you to implement almost any algorithm in a single TSQL statement. This problem is to parse and evaluate arithmetic expressions using TSQL.

Sample Data

```
id  expr       x0   dx    points
--- ---------- ---- ----- ------
1   1+2*(4+x)  0    0.1   4
2   x^2/2      0    0.1   4
3   x^x        0    10    4
```

Expected Results

```
expr        x      value
---------   ------ --------------------------
1+2*(4+x)   0      9
1+2*(4+x)   0.1    9.2
1+2*(4+x)   0.2    9.4
1+2*(4+x)   0.3    9.6
x^2/2       0      0
x^2/2       0.1    0.005
x^2/2       0.2    0.02
x^2/2       0.3    0.045
x^x         0      1
x^x         10     10000000000
x^x         20     1.048576E+26
x^x         30     2.05891132094649E+44
```

Rules

1. “expr” could be any valid expression using single digit constants, the single variable x, the operators + - * / ^ and possibly parentheses.
2. “x0” is the starting position on the x-axis.
3. “dx” is the increment for each value.
4. “points” is the number of values to generate which will be greater than zero.
5. The result should be sorted by expr, x.
6. All calculations should be done using full float precision and number range. The display of numbers in the value column should be in the format generated by a float(53) data type.
7. The maximum number of nested parenthesis will be 64.
8. All expressions will always have correct syntax.
9. The data will never cause overflows or division by zero error.
10. As a hint for a possible algorithm for evaluating expressions, see this article: http://en.wikipedia.org/wiki/Shunting-yard_algorithm.
11. For portions of an expression at the same implied or explicit parenthetical level, the following operator precedence rules apply:
    1. Exponents
    2. Multiplication, Division

3. Addition, Subtraction

12. For more details see http://en.wikipedia.org/wiki/Order_of_operations.

Sample Script

Use the TSQL script given below to generate the sample data for this problem.

```
IF OBJECT_ID('EXPR') IS NOT NULL BEGIN
DROP TABLE EXPR
END
GO
CREATE TABLE EXPR(
id INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
expr VARCHAR(8000),
x0 FLOAT,
dx FLOAT,
points INT
)
GO
INSERT INTO EXPR
SELECT '1+2*(4+X)',0,0.1,4 UNION ALL
SELECT 'X^2/2', 0,0.1,4 UNION ALL
SELECT 'X^X', 0,10,4
```

⑨ The problem is to draw a graph using T-SQL.

Sample Data

```
Seq Data
--- ----
1   2
2   3
3   4
4   4
5   5
6   4
7   3
8   4
9   4
10  4
11  3
12  2
13  2
14  3
15  3
16  4
17  4
18  5
```

Expected Results

```
5|
4|  _/\   __       _/
3| /   \/  \    _/
2|/         \_/
1|
0|____________________
012345678901234567 8
```

Rules

1. The "seq" column controls the flow of the graph. You should process the points in that order.
2. Value of the "data" column should be used to build your graph representation. The graph could start with any value between 0 and 5.
3. The difference between two contiguous readings will only ever be –1, 0 or 1.
4. The highest value the "data" column can hold is 5 and lowest value is 0.
5. There will be no upper limit on the number of rows in the source table.
6. There will be at least 2 rows in the source table.
7. The final result should consist of 7 rows of output consisting of a single VARCHAR column.

Sample Script

Use the following script to generate the sample data.

```
SET NOCOUNT ON
IF OBJECT_ID('GraphData,'U') IS NOT NULL DROP TABLE GraphData
GO
CREATE TABLE GraphData (
Seq INT,
Data INT
)
INSERT INTO GraphData(Seq, Data) SELECT 1, 2
INSERT INTO GraphData(Seq, Data) SELECT 2, 3
INSERT INTO GraphData(Seq, Data) SELECT 3, 4
INSERT INTO GraphData(Seq, Data) SELECT 4, 4
INSERT INTO GraphData(Seq, Data) SELECT 5, 5
INSERT INTO GraphData(Seq, Data) SELECT 6, 4
INSERT INTO GraphData(Seq, Data) SELECT 7, 3
INSERT INTO GraphData(Seq, Data) SELECT 8, 4
INSERT INTO GraphData(Seq, Data) SELECT 9, 4
INSERT INTO GraphData(Seq, Data) SELECT 10, 4
INSERT INTO GraphData(Seq, Data) SELECT 11, 3
INSERT INTO GraphData(Seq, Data) SELECT 12, 2
INSERT INTO GraphData(Seq, Data) SELECT 13, 2
INSERT INTO GraphData(Seq, Data) SELECT 14, 3
INSERT INTO GraphData(Seq, Data) SELECT 15, 3
INSERT INTO GraphData(Seq, Data) SELECT 16, 4
INSERT INTO GraphData(Seq, Data) SELECT 17, 4
INSERT INTO GraphData(Seq, Data) SELECT 18, 5

SELECT * FROM GraphData
```

⑩ The problem is all about generating calendars for given number of months.

Sample Data

```
Mth Yr
--- ----
8   2009
2   1900
10  1959
```

Your job is to take the above table and generate calendars for the months and years given in the table. A calendar should be generated for each row in the table, using a single query (and no temp tables or table variables).

Expected Results

```
+-----------------------------+
|         FEBRUARY 1900       |
|=============================|
| Sun Mon Tue Wed Thu Fri Sat |
|-----------------------------|
|                   1   2   3 |
|   4   5   6   7   8   9  10 |
|  11  12  13  14  15  16  17 |
|  18  19  20  21  22  23  24 |
|  25  26  27  28             |
+-----------------------------+
+-----------------------------+
|         OCTOBER 1959        |
|=============================|
| Sun Mon Tue Wed Thu Fri Sat |
|-----------------------------|
|                   1   2   3 |
|   4   5   6   7   8   9  10 |
|  11  12  13  14  15  16  17 |
|  18  19  20  21  22  23  24 |
|  25  26  27  28  29  30  31 |
+-----------------------------+
+-----------------------------+
|          AUGUST 2009        |
|=============================|
| Sun Mon Tue Wed Thu Fri Sat |
|-----------------------------|
|                           1 |
|   2   3   4   5   6   7   8 |
|   9  10  11  12  13  14  15 |
|  16  17  18  19  20  21  22 |
|  23  24  25  26  27  28  29 |
|  30  31                     |
+-----------------------------+
```

Rules

1. The resulting output is a single 31-character column called Calendar.
2. The Month should be uppercase and should be rendered in the language that is set at runtime.
3. The Month and Year are centered.
4. The Day-Of-The-Week names are the first 3 letters of the days of the week, rendered in the language that is set at runtime. Sunday must be the first column.
5. The calendars must be sorted in order.

Sample Scripts

Use the following script to generate the sample data.

```
DECLARE cal TABLE(
Mth INT,
Yr INT
)
SELECT 8, 2009 UNION ALL
SELECT 2, 1900 UNION ALL
SELECT 10, 1959

SELECT * FROM cal
```