**Set 9 - Programming Assignments**

Following points should be taken into consideration while solving the below programming assignments.

1. Proper indentation of the code must be done.
2. Proper naming conventions for all the variables should be there.
3. Exception Handling to be done where ever possible.
4. 100% Logical correctness to be ensured.
5. Appropriate comments are must wherever necessary.
6. As far as possible, the technique used should be effective from performance or query optimization point of view. *(Note 🡪 For some assignments this point may not be applicable)*
7. Test Cases should also be included after every solution. It means that cross-checking of that code should also be the part of solution.
8. Using loop technique of SQL Server display all the multiples of 7 between 31 and 98. Each value has to be shown not one a new line, but each value should be shown concatenated by “-“.

So, the final output will be seen like 1-2-3-4

**Declare @w integer, @s varchar(Max)**

**set @w = 31**

**while @w <= 98**

**begin**

**if @s != ''**

**set @s = @s+cast(@w \* 7 as varchar)+'-'**

**else**

**set @s =cast(@w \* 7 as varchar)+'-'**

**set @w = @w + 1;**

**end**

**print @s**

1. Using Cursors technique check whether the 3rd Clerk’s salary is greater than the 2nd Clerk’s salary. If it is so, then display message as “Third has more salary than the second” otherwise display message as “Second has more salary than the first”

**DECLARE @sal1 int, @sal2 int, @c int**

**set @c = 0**

**Declare SalCheck Cursor**

**for select SAL, lag(SAL,1) over(order by empno) as "Previous"**

**from EMP**

**where job = 'clerk'**

**open SalCheck**

**fetch next from SalCheck**

**into @sal1, @sal2**

**while @@FETCH\_STATUS = 0**

**Begin**

**set @c = @c + 1**

**if(@c = 3)**

**if(@sal1 > @sal2)**

**print 'Third has more salary than the second'**

**else**

**print 'Second has more salary than the third'**

**fetch next from SalCheck**

**into @sal1, @sal2**

**end**

**CLOSE SalCheck;**

**DEALLOCATE SalCheck;**

1. Consider the default emp table. The management has decided to increase salary of employees by 10%. Starting with lowest paid earner and working up. If at any time total of salary exceeds 45000/- then no further employees are to be given an increase. Cursor technique only should be used. Note 🡪Salaries should not get actually updated in the emp table. The changes should only be shown in the output.

**DECLARE @emp\_name varchar(20) ,@emp\_sal int, @Total int**

**set @Total = 0**

**DECLARE Sal\_Raise CURSOR FOR**

**SELECT ENAME,SAL FROM Emp**

**order by SAL;**

**OPEN Sal\_Raise**

**FETCH NEXT FROM Sal\_Raise**

**INTO @emp\_name,@emp\_sal**

**print 'Employee\_ID Employee\_Sal Salary\_Rise'**

**WHILE @@FETCH\_STATUS = 0**

**BEGIN**

**set @Total = @Total + @emp\_sal**

**if @Total < 45000**

**print ' ' + CAST(@emp\_name as varchar(20)) +' '+**

**cast(@emp\_sal as varchar(20)) + ' '+cast((@emp\_sal+@emp\_sal\*0.01) as varchar(20))**

**FETCH NEXT FROM Sal\_Raise**

**INTO @emp\_name,@emp\_sal**

**END**

**CLOSE Sal\_Raise;**

**DEALLOCATE Sal\_Raise;**

1. Create a regular table **Emp\_Coupons** with the two fields Name and Coupon\_No

Enter the following records in it.

John 80

Martin 83

Allen 87

Roger 78

Adams 88

Kim 89

Using Cursor technique check whether the coupon number of the current record is greater than the previous. If any record’s coupon number is less than the previous one then you will display the name of the person whose coupon number is less.

**Declare @name varchar(20), @coupon int, @preCou int**

**set @preCou = 0**

**Declare Coupon\_Check Cursor**

**for select Name, Coupon\_No**

**from Emp\_Coupons**

**open Coupon\_Check**

**fetch next from Coupon\_Check**

**into @name, @coupon**

**while @@FETCH\_STATUS = 0**

**Begin**

**if(@preCou = 0)**

**set @preCou = @coupon**

**else if(@coupon > @preCou)**

**print @name**

**fetch next from Coupon\_Check**

**into @name, @coupon**

**end**

**CLOSE Coupon\_Check;**

**DEALLOCATE Coupon\_Check;**

1. Create a table **Company\_Data** with columns empid, ename and job. There is no primary key or unique key constraint to the empid column. Create a procedure **Add\_Company\_Data** which will take empid as the parameter and adds a new record in that table if the empid with the same value does not exist.

**Create table Company\_Data**

**(EmpId int,**

**Ename varchar(30),**

**Job varchar(20))**

**create procedure Add\_Company\_Data**

**@emp\_id int,**

**@ename varchar(20),**

**@job varchar(20)**

**as**

**begin**

**if (@emp\_id not in (select EmpId from Company\_Data))**

**begin**

**insert into Company\_Data(EmpId,Ename,Job)**

**values (@emp\_id,@ename,@job)**

**end**

**else**

**begin**

**print 'EmpID already exist'**

**end**

**end**

1. Considering the emp table create a procedure **IsHighest** that will take ename as the parameter. The procedure should display whether employee is the highest earner or not.

**create procedure IsHighest (@Ename varchar(20))**

**as**

**select top 1 case**

**when (select sal from EMP where ENAME = @Ename) >= (select max(sal) from EMP) then @Ename+' is the highest earner'**

**when (select sal from EMP where ENAME = @Ename) < (select max(sal) from EMP) then @Ename+' is not the highest earner'**

**end as "Is Highest"**

**from EMP**

**go**

1. Create a function **Yearly\_Raise** that will take the salary, deptno and job as the parameters and raise the salary according to different criteria.

**Criteria Raise**

Clerk employees of deptno 20 earning salary above 1000 🡪 20%

Clerk employees of deptno 20 earning salary less 1000 🡪 15%

Clerk employees of deptno 20 earning salary above 1000 🡪 5%

Clerk employees of deptno 20 earning salary less than 1000 🡪 18%

Clerk employees of deptno 30 having any salary 🡪 10%

For any other job type having any salary 🡪 5%

After creating the function and testing its logic, create a duplicate copy of emp table with all the records. Name of that table will be **Emp\_Raise\_Table**.

Use the function **Yearly\_Raise** to update salaries of the employees in the new table **Emp\_Raise\_Table**.

**create function Yearly\_Raise(@E\_sal int, @E\_deptno int, @E\_job varchar(20))**

**returns int**

**as**

**begin**

**declare @Raise int**

**if(@E\_job = 'Clerk' and @E\_deptno = 10 and @E\_sal > 1000)**

**set @Raise = 20**

**else if(@E\_job = 'Clerk' and @E\_deptno = 10 and @E\_sal < 1000)**

**set @Raise = 15**

**else if(@E\_job = 'Clerk' and @E\_deptno = 20 and @E\_sal > 1000)**

**set @Raise = 5**

**else if(@E\_job = 'Clerk' and @E\_deptno = 20 and @E\_sal < 1000)**

**set @Raise = 18**

**else if(@E\_job = 'Clerk' and @E\_deptno = 30)**

**set @Raise = 10**

**else**

**set @Raise = 5**

**return @Raise**

**end**

**update Emp\_Raise\_Table**

**set sal = sal + sal \* dbo.Yearly\_Raise(sal, deptno, ename)**

1. Create a View whose name will be **Emp\_Dept**. This view should show the empno, ename, sal, job, deptno, dname and loc from both the tables Emp and Dept for the matching deptno.

Create a trigger **DML\_On\_View** which will allow insert, update and delete on **Emp\_Dept** view.

**CREATE TRIGGER DML\_On\_View**

**ON Emp\_Dept**

**Instead Of UPDATE, INSERT, DELETE**

**AS**

**BEGIN**

**DECLARE @Activity NVARCHAR (50)**

**-- update**

**IF EXISTS (SELECT \* FROM inserted) AND EXISTS (SELECT \* FROM deleted)**

**BEGIN**

**SET @Activity = 'UPDATE'**

**print @Activity**

**Declare @sal int, @EMPNO int**

**Set @sal = (Select sal from inserted)**

**set @EMPNO = (Select EMPNO from inserted)**

**update Emp\_Dept**

**set SAL= @sal**

**where EMPNO = @EMPNO**

**END**

**-- insert**

**IF EXISTS (SELECT \* FROM inserted) AND NOT EXISTS(SELECT \* FROM deleted)**

**BEGIN**

**SET @Activity = 'INSERT'**

**print @Activity**

**Insert into Emp (EMPNO, ENAME, SAL, JOB, DEPTNO)**

**select EMPNO, ENAME, SAL, JOB, DEPTNO from inserted**

**END**

**-- delete**

**IF EXISTS (SELECT \* FROM deleted) AND NOT EXISTS(SELECT \* FROM inserted)**

**BEGIN**

**SET @Activity = 'DELETE'**

**print @Activity**

**Declare @ENo int**

**set @ENo = (select EMPNO from deleted)**

**delete from EMP**

**where EmpNo = @Eno**

**END**

**END**

1. Create a table **Inflation\_Data**. It will have 2 columns, namely Year and Inflation\_Amount. Add one row manually using Insert statement in this table. Values will be 2011 for Year column and 2000 for Inflation\_Amount.

Then, create a trigger **Check\_Inflation\_Data** on insert event **Inflation\_Data** table. This trigger should ensure that the year will be greater than all the previous years and Inflation\_Amount will be also greater than all the previous inflation amounts of table **Inflation\_Data.**

**Create trigger Check\_Inflation\_Data**

**on Inflation\_Data**

**for insert**

**as**

**begin**

**Declare @year int, @amo int**

**set @year = (select max(Year) from Inflation\_Data)**

**set @amo = (select max(Inflation\_Amount) from Inflation\_Data)**

**if ((select Year from Inserted) < @year)**

**begin**

**print 'Year Cannot be less than previous'**

**rollback tran**

**end**

**if ((select Inflation\_Amount from Inserted) < @amo)**

**begin**

**print 'Amount cannot be less than previous'**

**rollback tran**

**end**

**end**

**go**

1. Consider the default emp and dept tables. Create a procedure **Show\_Max\_Salary** which will take Dname as an input parameter and Max\_Sal as an output parameter. The procedure should show the highest salary through Max\_Sal parameter for the Dname supplied.

**create procedure Show\_Max\_Salary (@DName varchar(50),@Max\_Sal int output)**

**as**

**set @Max\_Sal = (select max(e.SAL)**

**from EMP E**

**join DEPT D on E.DEPTNO = D.DEPTNO**

**where D.DNAME = @DName)**

**Select @Max\_Sal**

**go**

**declare @x int**

**exec Show\_Max\_Salary 'Research', @x output**