**EFM SGBD2 2016   
Compte Rendu**

**I - Partie Théorique : (4 Points)**

1. C’est quoi une transaction ?? Donner un exemple (1pt)

Une transaction est une unité de travail qui décrit une série d'opérations dont les effets respectifs sur la base de données ne s'appliquent seulement si chacune des opérations s'est correctement déroulé sinon toute la transaction est annulée.

2. Rappeler qu’est-ce qu’un trigger et expliquer la différence entre les types INSTEAD OF et FOR (2pts)

Un trigger est une procédure qui s'exécute automatiquement en s'associant à des événements de base de données tel que INSERT, UPDATE, DELETE et qui peut avoir lieu avant, à la place, ou après ces événements.

3. Le Try ... Catch permet de capter les erreurs dont la sévérité est : (1pt)

a) Entre 0 et 10

b) Entre 11 et 19

c) Supérieure ou égale à 20

Le Try ... Catch capte seulement les erreurs dont la sévérité est entre 11 et 19 (Réponse b).

**II - Partie Pratique : (36 Points)**

Soit le schéma relationnel suivant :

create database SociétéVentesDb;

use SociétéVentesDb;

create table Projet (

num\_projet int primary key,

désignation varchar(50),

num\_s int foreign key references Société (num\_s)

on delete set null on update cascade,

budget money

);

create table Vente (

num\_vendeur int foreign key references Employé (matricule),

num\_pièce int foreign key references Pièce (num\_pièce),

num\_projet int foreign key references Projet (num\_projet),

quantité int,

date\_vente date,

primary key (num\_vendeur, num\_pièce, num\_projet)

);

create table Pièce (

num\_pièce int primary key,

désignation varchar(50),

poids float,

num\_s int foreign key references Société (num\_s)

on delete set null on update cascade,

prix\_unitaire money,

quantité\_stockée int

);

create table Société (

num\_s int primary key,

raison\_sociale varchar(50),

ville varchar(50),

surface float,

capital money

);

create table Employé (

matricule int primary key,

nom varchar(50),

prénom varchar(50),

ville varchar(50),

num\_s int foreign key references Société (num\_s)

on delete set null on update cascade,

salaire\_fixe money,

commission money,

date\_naissance date,

date\_embauche date,

fonction varchar(50) foreign key references Fonction (Intitule)

on delete set null on update cascade,

);

create table Fonction (

Intitule varchar(50) primary key,

SalaireMin money,

SalaireMax money,

NbHeures int

);

select name from sys.Tables;

1. Ecrire une fonction qui retourne le montant total des pièces vendues par projet (3pts)

create function montant\_total\_pièces()

returns table

as

return (

select num\_projet as 'Projet', sum(P.prix\_unitaire\*V.quantité) as 'Montant total'

from Pièce P inner join Vente V on P.num\_pièce = V.num\_pièce

group by num\_projet

);

-- Exécuter :

select \* from montant\_total\_pièces();

2. Ecrire une fonction qui calcule le salaire net (salaire fixe + commission) d’un employé passé en paramètre sachant que la commission concerne seulement les vendeurs, pour les autres employés la commission à la valeur « NULL » (4 pts)

create function salaire\_net(@matricule int)

returns money

as

begin

declare @salaire\_fixe money = (select salaire\_fixe from Employé where matricule = @matricule);

declare @comission money = (select commission from Employé where matricule = @matricule);

if @comission is null

return @salaire\_fixe;

return @salaire\_fixe + @comission;

end

-- Exécuter :

select dbo.salaire\_net(6) as 'Salaire net';

3. Ecrire une fonction qui retourne le matricule du meilleur vendeur d’une société passée en paramètre (celui qui a apporté le plus d’argent à la société) (4pts)

create function meilleur\_vendeur(@num\_s int)

returns int

as

begin

declare @T table (

num\_vendeur int,

RevenuTotal money

);

insert into @T

select num\_vendeur, sum(prix\_unitaire\*quantité) as 'RevenuTotal'

from Pièce P inner join Vente V on P.num\_pièce = V.num\_pièce

where num\_s = @num\_s

group by num\_vendeur;

return (select num\_vendeur from @T where RevenuTotal = (select max(RevenuTotal) from @T));

end

-- Exécuter :

select dbo.meilleur\_vendeur(1) as 'Meilleur vendeur';

4. Ecrire une procédure stockée qui renvoie la liste des employés qui vont partir en retraite cette année sachant que : (4pts)

Un employé part à la retraite si :

=> Son âge est au moins 50 ans avec au moins 30 ans d’années de service

Ou bien

=> A l’atteinte de l’âge de 60 ans

create proc sp\_employés\_retraite

as

begin

select \* from Employé where

(datediff(year, date\_naissance, getdate()) >= 50 and datediff(year, date\_embauche, getdate()) >= 30)

or

datediff(year, date\_naissance, getdate()) >= 60

end

-- Exécuter :

exec sp\_employés\_retraite;

5. Ecrire une procédure stockée qui renvoie le nombre d’employés et le nombre de vendeurs d’une société passée en paramètre. (2pts)

create proc sp\_nbr\_employés\_vendeurs

@num\_s int, @nbr\_employés int output, @nbr\_vendeurs int output

as

begin

set @nbr\_employés = (select count(\*) from Employé where fonction <> 'Vendeur');

set @nbr\_vendeurs = (select count(\*) from Employé where fonction = 'Vendeur');

end

-- Exécuter :

declare @nbr\_employés int, @nbr\_vendeurs int

exec sp\_nbr\_employés\_vendeurs 1, @nbr\_employés output, @nbr\_vendeurs output;

select @nbr\_employés as 'Nombre d''employés', @nbr\_vendeurs as 'Nombre de vendeurs';

6. Ecrire une procédure stockée qui affiche pour une société passée en paramètre la liste de ses vendeurs comme suit :

Société : (1pt) …………………………

Ville : ……………………………………. Capital : ……………………

Directeur : ……………………………

Nombre d’employés : ……………

Nombre de vendeurs : ………………

Liste des vendeurs : (5pts)

• Vendeur 1 :

Matricule………….

Nom : …………………

Prénom : ………………

Salaire : ………………

Commission : ……………

Nombre des pièces vendues : ……

• Vendeur 2 :

Matricule : …………

Nom : …………………

Prénom : ………………

Salaire : ………………

Commission : ……………

Nombre des pièces vendues : ……

create proc sp\_vendeurs

@num\_s int

as

begin

declare @ville varchar(max) = (select ville from Société where num\_s = @num\_s);

declare @capital money = (select capital from Société where num\_s = @num\_s);

declare @directeur varchar(max) = (select nom from Employé where num\_s = @num\_s and fonction = 'Directeur');

declare @nbr\_employés int, @nbr\_vendeurs int;

exec sp\_nbr\_employés\_vendeurs @num\_s, @nbr\_employés output, @nbr\_vendeurs output;

print 'Société : ' + convert(varchar, @num\_s);

print ' Ville : ' + @ville + ' Capital : ' + convert(varchar, @capital) + ' DH';

print ' Directeur : ' + @directeur;

print ' Nombre d’employés : ' + convert(varchar, @nbr\_employés);

print ' Nombre de vendeurs : ' + convert(varchar, @nbr\_vendeurs);

print ' Liste des vendeurs : ';

declare CurVen cursor for

select matricule, nom, prénom, salaire\_fixe, commission, count(\*) as nbr\_pièces\_vendues

from Employé E inner join Vente V on E.matricule = V.num\_vendeur

where num\_s = @num\_s

group by matricule, nom, prénom, salaire\_fixe, commission;

declare @matricule int, @nom varchar(max), @prénom varchar(max),

@salaire\_fixe money, @commission money, @nbr\_pièces\_vendues int;

open CurVen;

fetch CurVen into @matricule, @nom, @prénom, @salaire\_fixe, @commission, @nbr\_pièces\_vendues;

declare @compteur int = 0

while @@fetch\_status = 0

begin

set @compteur += 1;

print ' • Vendeur ' + convert(varchar, @compteur) + ' :';

print ' Matricule: ' + convert(varchar, @matricule);

print ' Nom : ' + @nom;

print ' Prénom : ' + @prénom;

print ' Salaire : ' + convert(varchar, @salaire\_fixe);

print ' Commission : ' + convert(varchar, @commission);

print ' Nombre des pièces vendues : ' + convert(varchar, @nbr\_pièces\_vendues);

fetch CurVen into @matricule, @nom, @prénom, @salaire\_fixe, @commission, @nbr\_pièces\_vendues;

end

close CurVen;

deallocate CurVen;

end

-- Exécuter :

exec sp\_vendeurs 3;

7. Ecrire un trigger qui met à jour le stock de la pièce après la suppression d’une vente (2pts)

create trigger vente\_suppr

on Vente

after delete

as

begin

declare @num\_pièce int = (select num\_pièce from deleted);

declare @quantité int = (select quantité from deleted);

update Pièce set quantité\_stockée += @quantité where num\_pièce = @num\_pièce;

end

8. Ecrire un trigger permettant de vérifier avant toute insertion ou modification dans la table VENTE que la quantité demandée est disponible et met à jour le stock Si, ce n'est pas le cas, une exception doit être levée (3pts)

create trigger vente\_màj

on Vente

for insert, update

as

begin try

select \* from inserted;

select \* from deleted;

declare @num\_pièce int = (select top 1 num\_pièce from inserted);

declare @quantité int = (select top 1 quantité from inserted);

declare @quantité\_old int = (select top 1 quantité from deleted);

if @quantité\_old is null

set @quantité\_old = 0;

declare @quantité\_stockée int = (select top 1 quantité\_stockée from Pièce where num\_pièce = @num\_pièce);

if (@quantité\_stockée + (@quantité\_old - @quantité)) < 0

raiserror('Quantité indisponible !', 11, 1);

end try

begin catch

throw;

end catch

9. Ecrire un trigger qui permet de vérifier que le salaire d’un employé correspond bien à ce qui est autorisé par sa fonction. Si, ce n'est pas le cas, une exception doit être levée (4pts)

create trigger salaire\_màj

on Employé

for insert, update

as

begin try

declare @fonction varchar(max) = (select fonction from inserted);

declare @salaire\_fixe money = (select salaire\_fixe from inserted);

declare @SalaireMin money = (select SalaireMin from Fonction where Intitule = @fonction);

declare @SalaireMax money = (select SalaireMax from Fonction where Intitule = @fonction);

if @salaire\_fixe not between @SalaireMin and @SalaireMax

raiserror('Salaire invalide !', 11, 1);

end try

begin catch

throw;

end catch

10. Ecrire un trigger permettant de vérifier avant toute insertion dans la table VENTE que le vendeur correspondant est rattaché à la même société que le projet pour lequel a lieu la vente. Si, ce n'est pas le cas, une exception doit être levée (4pts)

create trigger vendeur\_valide

on Vente

instead of insert

as

begin try

declare @num\_vendeur int = (select num\_vendeur from inserted);

declare @num\_projet int = (select num\_projet from inserted);

declare @num\_s\_vend int = (select num\_s from Employé where matricule = @num\_vendeur);

declare @num\_s\_proj int = (select num\_s from Projet where num\_projet = @num\_projet);

if @num\_s\_vend <> @num\_s\_proj

raiserror('Incorrespondance !', 11, 1);

else

insert into Vente select \* from inserted;

end try

begin catch

throw;

end catch