Abdelhakim EL HAFID
ISTA NTIC Marrakech

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
```