

Programmation SQL

Bade de données des travaux pratiques

Dans tous les TP de la programmation SQL nous allons utiliser la base de données suivant de la gestion des projets.

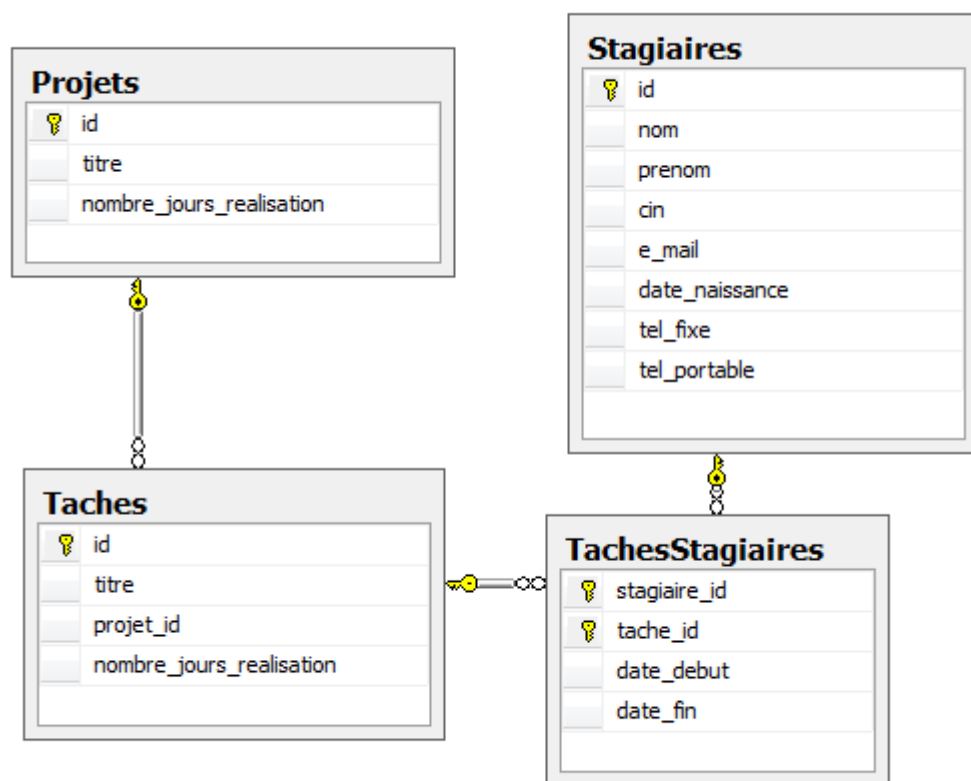


Figure 1 : MPD de gestion des projets

Règles de gestion

- Un projet peut avoir plusieurs tâches
- Le nombre de jours de réalisation d'un projet est égal à la somme des nombres de jours de réalisation de ses tâches.
- Une même tâche peut être affectée à plusieurs stagiaires.
- Le nombre de jour de réalisation d'une tâche et la moyenne de nombre de jours de réalisation de la tâche par les stagiaires.

Comprendre l'intérêt d'utiliser un langage de programmation qui intègre le formalisme SQL

- Limites du langage SQL.
- Intérêt d'utiliser un langage intégrant SQL et une structure de programmation procédurale.
- Identification des traitements qui peuvent être délégués au SGBD.

TP 1 : Création de la base de données

Pour réaliser l'ensemble des travaux pratiques de ce module nous avons besoin de la base de données suivante de la gestion des projets des stagiaires.

Créer la base de données « GestionProjets » avec le modèle logique de données (MLD) suivant :

```
Stagiaires (id, nom, prenom, cin, ville, date_naissance, e_mail, tel_fixe, tel_portable)
```

```
Projets (id, titre, nombre_jours_realisation)
```

```
Taches (id, titre, projet_id, nombre_jours_realisation)
```

```
TachesStagiaires (stagiaire id, tache id, date_debut, date_fin)
```

TP 2 : Création des requêtes Simple

a- Insérer les stagiaires suivants :

Madani Ali	
Nom	Madani
Prénom	Ali
CIN	K380001
Ville	Tanger
Date de naissance	05/06/1995
E-mail	madani.ali@gmail.com
Téléphone Fixe	0539 00 00 01
Téléphone Portable	06 00 00 00 01

Madani Mouad	
Nom	Madani
Prénom	Moad
CIN	K380002
Ville	Tanger
Date de naissance	03/02/1997
E-mail	madani. moad@gmail.com
Téléphone Fixe	0539 00 00 02
Téléphone Portable	06 00 00 00 02

b- Insérer le projet suivant à la base de données

Projets	Tâches	Durée	Stagiaires
Réalisation d'une application de gestion de disponibilité des salles	Réalisation de diagramme de cas d'utilisation	1 jour	Madani Ali
	Réalisation du prototype de l'application	10 jours	Madani Moad
	Réalisation de diagramme de classes	1 jour	Madani Ali

- a- Donner la liste des stagiaires qui habite à la ville de Tanger.
- b- Donner la liste des stagiaires ayant un numéro de téléphone fixe de la région du nord du Maroc
- c- Donner le nombre des tâches de chaque projet
- d- Donner les stagiaires n'ayant aucune tâche
- e- Donner la liste des stagiaires âgés plus de 21 ans

TP 3 : Création des requêtes

- a- Donner la liste des stagiaires ayant les emails valide
- b- Donner la liste des stagiaires ayant un numéro de téléphone valide
- c- Donner le nombre de jour de réalisation de chaque projet

Chapitre 2.4.1 : Écrire des scripts dans le langage procédural du SGBD.

Présentation des instructions du langage de programmation :

- les variables et les types de données ;
- les variables élémentaires et complexes ;
- les structures de contrôle ;
- les conditions ;
- Les EXCEPTIONS : prédéfinies et utilisateur.

Microsoft Transact SQL est un langage de requêtes amélioré par rapport au SQL dont il reprend les bases.

De plus, le Transact SQL prend en compte des fonctionnalités procédurales telles que la gestion des variables, les structures de contrôle de flux, les curseurs, et les lots d'instructions. C'est donc un langage complet qui comporte des instructions, qui manipule des objets SQL, qui admet la programmabilité et qui utilise des expressions.

Expressions

Dans le T-SQL, nous pouvons utiliser des expressions, permettant de mettre en oeuvre l'aspect algorithmique du langage. Les expressions peuvent prendre plusieurs formes.

Constantes

Une constante est une variable, dont la valeur ne peut être changée lors de l'exécution d'instructions T-SQL.

TP 4 : Utilisations des constantes

```
select  
  titre as [Titre],  
  'Projet en cours de réalisation' as [Statut]  
from Projets
```

Results Messages	
Titre	Statut
1 Réalisation d'une application de gestion de disp...	Projet en cours de réalisation

Noms de colonnes

Ils pourront être utilisés comme expressions. La valeur de l'expression étant la valeur stockée dans une colonne pour une ligne donnée.

TP 5 : Utilisation de la colonne comme expression

```
select cast( id as varchar) + '_' + titre as [Titre]
from Projets
```

Results	
Titre	
1 1_Réalisation d'une application de gestion de di...	

Fonctions

On peut utiliser comme expression n'importe quelle fonction, la valeur de l'expression est le résultat retourné par la fonction.

TP 6 : Expression - Fonctions

```
select LOWER('BONJOUR') as BonjourTSQL
select SQRT(9) as racine_carée_9
```

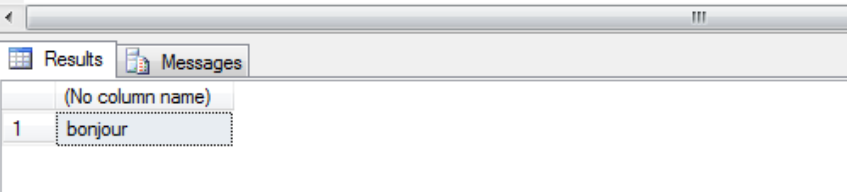
Results Messages	
Bonjour	
1 bonjour	
racine_carée_9	
1 3	

Variables

Les variables peuvent être employées en tant qu'expression ou dans une expression, sous la forme @nom_de_variable ou @@nom_de_variable. La valeur de l'expression est la valeur de la variable.

TP 7 : Expression - Variables

```
declare @x char(10)
select @x= 'BONJOUR'
select LOWER(@x)
```



(No column name)
1 bonjour

Sous-requêtes

Une requête SELECT entre parenthèses peut être employée en tant qu'expression ayant pour valeur le résultat de la requête, soit une valeur unique, soit un ensemble de valeurs.

TP 8 : Expression - Sous-requête

Stockage du nombre de stagiaires dans une variable :

```
declare @nombre_stagiaire int
select @nombre_stagiaire= COUNT(*) from Stagiaires
select @nombre_stagiaire as "Nombre des stagiaires"
```

Expressions booléennes

Elles sont destinées à tester des conditions (IF, WHILE, WHERE, etc.). Ces expressions sont composées de la manière suivante :

expression1 opérateur expression2

Opérateurs

Les opérateurs vont permettre de constituer des expressions calculées, des expressions booléennes ou des combinaisons d'expressions.

Opérateurs arithmétiques

TP 9 : Operateur – addition, soustraction, multiplication et modulo

```
declare @nombre_jour int
select @nombre_jour = sum(DAY( date_fin - date_debut)) from TachesStagiaires;
print @nombre_jour

declare @nom_et_prenom nvarchar(200)
select @nom_et_prenom = nom + ' ' + prenom from Stagiaires where nom = 'Madani'
print @nom_et_prenom
```

Fonctions

Fonctions conversion de types

CAST (exp1 AS types_données)	Permet de convertir une valeur dans le type spécifié en argument
CONVERT (types_données, exp1, style)	Conversion de l'expression dans le type de données spécifié. Un style peut être spécifié dans le cas d'une conversion date ou heure

Fonctions mathématiques

Ces fonctions renvoient une valeur résultant de calculs mathématiques classiques (algèbre, trigonométrie, logarithmes, etc.).

TP 10 : Fonctions mathématiques

```
declare @unReal float
select @unReal = 9.1

print ABS( @unReal)
select @unReal= ABS( @unReal)

print FLOOR(@unReal)
print SQRT(@unReal)
print POWER(@unReal,2)

select FLOOR(@unReal) as 'Partie Entier',
       SQRT(@unReal) as 'Racine carrée' ,
       POWER(@unReal,2) as 'Puissance'
```

Fonctions diverses

```
print RAND(10)
```

Nombre aléatoire compris entre 0 et 1. `Expn` représente la valeur de départ.



Messages
0.71376

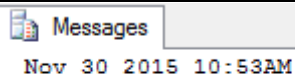
Fonctions date

Les fonctions date manipulent des expressions de type DATETIME, et utilisent des formats représentant la partie de la date à gérer.

Format	Abréviation	Signification
year	yy, yyyy	Année (de 1753 à 9999)
quarter	qq, q	Trimestre (1 à 4)
month	mm, m	Mois (1 à 12)
day of year	dy, y	Jour dans l'année (1 à 366)
day	dd, d	Jour dans le mois (1 à 31)
weekday	dw, ww	Jour dans la semaine (1 Lundi à 7 Dimanche)
hour	hh	Heure (0 à 23)
minute	mi, n	Minute (0 à 59)
seconds	ss, s	Seconde (0 à 59)
millisecond	ms	Milliseconde (0 à 999)

Date et heure système

```
print GETDATE()
```



Messages
Nov 30 2015 10:53AM

DATEPART

```
print DATEPART ("Day", GETDATE ())
```



Instruction If

TP 10 : Utilisation de l'instruction If..Else

Question 1 : Exécuter le code suivant

```
declare @numero_projet int
select @numero_projet = 1;
if exists ( select * from Projets where id = @numero_projet)
begin
    print 'Suppression du projet'
end
else
    print 'le projet ne existe pas'
```

Question 2 : Donner les modifications convenable pour que le script supprimer un projet.

Instruction Case

TP 12 : Instruction Case

Question 1 : Exécuter le code suivant :

```
declare @nombre_jours_realisation int
set @nombre_jours_realisation = 2

print CASE @nombre_jours_realisation
    when '01' then 'un jour'
    when '02' then 'deux jours'
    when '03' then 'trois jours'
    else 'plus de 3 jours'
end
```

Question 2 : Donner le script qui affiche le résultat suivant :

Question avec solution

Results		Messages
Nom du projet		Nombre de jours
1	Réalisation d'une application de gestion de disp...	deux jours

Instruction While

TP 13 : Utilisation de l'instruction While

Question 1 : Exécuter le script suivant :

```

declare @nombre_jour int
declare @i int
set @i = 0

select  @nombre_jour = (SUM( day(date_fin) - day(date_debut) + 1))
from Taches, TachesStagiaires, Projets
where projet_id = 1 and Taches.projet_id = Projets.id
      and TachesStagiaires.id_tache = Taches.id

print @nombre_jour

while @nombre_jour < 20 and @i < 5
begin

    set @i = @i + 1

    print @nombre_jour

    select  @nombre_jour = (SUM( day(date_fin) - day(date_debut) + 1))
    from Taches, TachesStagiaires, Projets
    where projet_id = 1 and Taches.projet_id = Projets.id
          and TachesStagiaires.id_tache = Taches.id

end

```

Question 2 : Donner les modifications convenables pour que la boucle « While » ajout des tâches de deux jours de réalisation au projet numéro 1.

Question 3 : Combien des tâches ajoutées au projet numéro 1.

Question 4 : Réaliser des modifications pour que le script ne comporte que une seule fois la requête « Select »

Chapitre 2.4.2 : Exception

Pour chaque erreur qui survient dans SQL Server, SQL Server produit un message d'erreur. En règle générale, tous les messages possèdent la même structure :

- un numéro d'erreur,
- un indicateur de sévérité,
- un état,
- le numéro de la ligne ayant provoquée l'erreur,
- un message d'explication de l'erreur,

```
declare @a int
set @a = 1/0
Select * from Stagiaires;
```

Message d'erreur :

```
Msg 8134, Level 16, State 1, Line 2
Divide by zero error encountered.
```

```
(2 row(s) affected)
```

La gravité est un indicateur, un chiffre de **0 à 24** (gravité croissante).

Gestion des erreurs dans le code

Il existe deux manières de gérer les erreurs.

- La première consiste à tester la valeur de la variable système @@ERROR,
- la seconde consiste à positionner dans un gestionnaire d'exception TRY le bloc d'instructions à tester, et dans le CATCH, l'erreur à lever.

TP 14 : Exception, Try et Catch

Question 1 : Exécuter le script suivant :

```
declare @a int
set @a = 1/0
Select * from Stagiaires;
```

Question 2 : Utiliser Try et Catch pour capturer l'exception de la question 1

```

Begin Try
  declare @a int
  set @a = 1/0
  Select * from Stagiaires;
End Try
Begin Catch
  print 'Erreur'
End Catch

```

Messages

Erreur

RAISERROR

Il est possible de lever des exceptions personnalisées via l'instruction RAISERROR.

Lorsqu'on veut lever une erreur, on peut soit donner l'identifiant de l'erreur en question, soit lui donner un message particulier. Si on lui donne un message particulier comme nous l'avons fait dans l'exemple ci-dessous, il faut automatiquement lui préciser une gravité et un état.

On peut ajouter une clause WITH à la suite de l'instruction RAISERROR, pour appliquer une des trois options possibles :

- LOG : le message sera consigné dans l'observateur d'évènement Windows.
- NOWAIT : le message sera délivré sans attente à l'utilisateur.
- SETERROR : permet de valoriser @@ERROR et ERROR_NUMBER avec le numéro du message d'erreur.

TP 1 : RaisError

Question 1 : exécuter le code suivant :

```

declare @nombre_stagiaires int
Select @nombre_stagiaires = COUNT(*) from Stagiaires;
if @nombre_stagiaires < 3
  RAISERROR ('Le nombre des stagiaires est inférieur à 25', 11, 1)

```

Messages

Msg 50000, Level 11, State 1, Line 4
Le nombre des stagiaires est inférieur à 25

Question 1 : Ecrire une procédure qui déclenche une exception si le nombre de jours de réalisation d'un projet passé en paramètre, n'est pas correcte.

Variables "système"

SQL Server définit un grand nombre de "variables système" c'est à dire des variables définies par le serveur.

Variable	Description
@@connections	nombre de connexions actives
@@datefirst	premier jour d'une semaine (1:lundi à 7:dimanche)
@@error	code de la dernière erreur rencontrée (0 si aucune)
@@fetch_status	état d'un curseur lors de la lecture (0 si lecture proprement exécutée)
@@identity	dernière valeur insérée dans une colonne auto incrémentée pour l'utilisateur en cours
@@max_connections	nombre maximums d'utilisateurs concurrents
@@procid	identifiant de la procédure stockée en cours
@@rowcount	nombre de lignes concernées par le dernier ordre SQL
@@servername	nom du serveur SGBDR courant
@@spid	identifiant du processus en cours
@@trancount	nombre de transaction en cours

TP 15 : Paramètre output

Question 1 : Créer la procédure « sp_echange » qui permet d'échanger deux variable.

```

Alter procedure sp_echange ( @nombre1 int, @nombre2 int)
as
begin
    declare @nombre3 int
    set @nombre3 = @nombre1
    set @nombre1 = @nombre2
    set @nombre2 = @nombre3
end

```

Question 2 : Donnez un script pour tester la procédure.

```

declare @nombre1 int
declare @nombre2 int
set @nombre1 = 1
set @nombre2 = 2

print 'Avant'
print @nombre1
print @nombre2

exec sp_echange @nombre1, @nombre2

print 'Après'
print @nombre1
print @nombre2

```

Question 3 : Changer la ligne suivant

```

exec sp_echange @nombre1 output , @nombre2 output

```

ensuite exécuter à nouveau le script de teste.

Question 4 : que fait « output » ?

Chapitre 2.4.3 : Manipuler les jeux d'enregistrement.

Définition d'un curseur : implicite, explicite.

- Définition des attributs de curseurs.
- Manipulation du contenu d'un curseur.

L'utilisation de curseurs est une technique permettant de traiter ligne par ligne le résultat d'une requête, contrairement au SQL (SELECT) qui traite un ensemble de lignes.

Syntaxe

```
FETCH
  [ [ NEXT | PRIOR | FIRST | LAST
      | ABSOLUTE { n | @nvar }
      | RELATIVE { n | @nvar }
    ]
  FROM
  ]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

Arguments

NEXT

Renvoie la ligne de résultats immédiatement après la ligne courante et incrémente cette dernière sur la ligne renvoyée. Si FETCH NEXT est la première extraction effectuée sur un curseur, cette instruction renvoie la première ligne dans le jeu de résultats. NEXT est l'option d'extraction du curseur par défaut.

PRIOR

Renvoie la ligne de résultats immédiatement avant la ligne courante, et décrémente cette dernière sur la ligne renvoyée. Si FETCH PRIOR est la première extraction effectuée sur un curseur, aucune ligne n'est renvoyée et le curseur reste placé avant la première ligne.

FIRST

Renvoie la première ligne dans le curseur et la transforme en ligne courante.

LAST

Renvoie la dernière ligne dans le curseur et la transforme en ligne courante.

ABSOLUTE { n | @nvar }

Si n ou @nvar est un nombre positif, retourne la ligne située à n lignes du début du curseur et transforme la ligne retournée en nouvelle ligne actuelle. Si n ou @nvar est un nombre négatif, retourne la ligne située à n lignes de la fin du curseur et transforme la ligne retournée en nouvelle ligne actuelle. Si n ou @nvar est 0, aucune

ligne n'est renvoyée. n doit être une constante entière et @nvar doit être smallint, tinyint ou int.

RELATIVE { n| @nvar }

Si n ou @nvar est un nombre positif, retourne la ligne située à n lignes au-delà de la ligne actuelle et transforme la ligne retournée en nouvelle ligne actuelle. Si n ou @nvar est un nombre négatif, retourne la ligne située à n lignes avant la ligne actuelle et transforme la ligne retournée en nouvelle ligne actuelle. Si n ou @nvar est égal à 0, la ligne actuelle est retournée. Si FETCH RELATIVE est spécifié avec n ou @nvar défini sur un nombre négatif ou égal à 0 lors de la première extraction effectuée sur un curseur, aucune ligne n'est retournée. n doit être une constante entière et @nvar doit être smallint, tinyint ou int.

GLOBAL

Indique que cursor_name fait référence à un curseur global.

cursor_name

Nom du curseur ouvert grâce auquel s'effectue l'extraction. S'il existe deux curseurs, un global et un local, nommés cursor_name, la variable cursor_name fait référence au curseur global si GLOBAL est spécifié et au curseur local si GLOBAL n'est pas spécifié.

@cursor_variable_name

Nom d'une variable de curseur qui fait référence au curseur ouvert à partir duquel l'extraction doit être effectuée.

INTO @variable_name[,...n]

Permet aux données issues des colonnes d'une extraction d'être placées dans des variables locales. Chaque variable de la liste (de gauche à droite) est associée à la colonne correspondante dans le jeu de résultats du curseur. Le type de données de chaque variable doit correspondre ou être une conversion implicite du type de données de la colonne du jeu de résultats correspondante. Le nombre de variables doit correspondre au nombre de colonnes dans la liste de sélection du curseur.

TP 16: Bonjour Cursor

Question 1 : Exécuter le script suivant

```

= Declare @id int
  Declare @titre varchar(100)
  Declare @nombre_jour int

= Declare ptojets_cur cursor for
- select * from Projets;

  open ptojets_cur

  fetch ptojets_cur into @id,@titre,@nombre_jour
  print @@Fetch_STATUS
  print @id
  print @titre

  fetch ptojets_cur into @id,@titre,@nombre_jour
  print @@Fetch_STATUS
  print @id
  print @titre

  close ptojets_cur
- Deallocate ptojets_cur
```

Question 2 : que représente @@Fetch_STATUS

Question 3 : Donnez un script qui affiche tous les tâches (Question avec Correction)

Exercice 1, TP17 : Fonction de correction du nombre de jours de réalisation

Question 1 : Ecrire un script qui permet de modifier les titre des tâches comme suivant :

Titre de la tâche = titre de la tâche + titre du projet

Question 2 : Ecrire un script qui permet de corrige tous les jours de réalisation des tâches à partir les tâches affectés au stagiaires.