

Base de données

Table des matières

Section 1 – Analyse d’un système d’information	4
Section 2 – Langage SQL et SGBD	5
Partie 0 : Notions de base	6
Installation	7
TD 1 : Installation de SQL Server	7
TD 2 : Création d’une base de données	7
TD 3 : Utilisation d’un schéma de la base de données	7
TD 4 : Exécution des requêtes SQL	7
Chapitre 0.2 : Avant de commencer	8
Création de la table des stagiaires	8
Insérer des groupes et leurs stagiaires :	9
Modifier un stagiaire :	10
Supprimer un stagiaire	10
Rechercher un stagiaire	11
Afficher tous les stagiaires avec le nom de leurs groupes.	11
Produit cartésien	12
Jointure	13
Partie 1 - LDD	16
Chapitre 2.1.1 : Création de la base de données	16
Création de la base de données	16
Chapitre 2.1.2 : Création des tables	18
Contraintes d’intégrité :	20
Contraintes Unique	20
Contraintes Check	21

Contraintes Primary Key	21
Contraintes Foreign Key	21
Supprimer une table :.....	22
Chapitre 2.1.3 : Modifier une table	24
Ajouter une colonne :.....	24
Modifier une colonne :	24
Ajouter une contrainte d'intégrité :	25
Supprimer une colonne :	28
Supprimer une contrainte :.....	28
Chapitre 2.1.2 : Séquences	30
Partie 2 : Langage de manipulation des données : LMD	32
Chapitre 2.2.1 : Insert, Delete, Update	33
Insertion d'enregistrement : « Insert »	33
Modification d'enregistrement : « Update ».....	35
Suppression d'enregistrement : « Delete ».....	35
Chapitre 2.2.2 : Manipulation des données avec le type : « DateTime »	36
Chapitre 2.2.3 : Interrogation des données	39
Extraction des données : « Select ».....	39
Duplicate : « distinct »	40
Select avec expression :	40
Ordonnancement : « Order By »	41
Concaténation : « + »	42
Opérateurs intégrés :	42
L'opérateur « IN » :	42
L'opérateur « LIKE » :	42
Statistiques sur les données (Sum, Min, Max, Avg, Count)	43
Regroupement : « GROUP BY »	43
L'élimination de certains groupes par « HAVING »:	44
Chapitre 2.2.4 : Sous interrogations.....	45
Sous-interrogations mono-lignes	45
Sous-interrogations multi-lignes : (IN, ANY, ALL)	46

Utilisation de « IN » :	46
Utilisation de « ANY » :	46
Utilisation de « ALL » :	47
Chapitre 2.2.5 : Jointure	49
Produit Cartésien avec SQL	49
Jointure relationnelle :	50
Équijointure :	50
InÉquijointure :	51
Jointure Syntaxe ANSI	51
Produit cartésien	51
Jointure.....	51
Jointure externe.....	51
Partie 2.3.0 : Optimisation	52
Chapitre 2.3.1 : Création des vues.....	53
Simplification :	53
Sécurité d'accès :	53
Chapitre 3.1 : Index	54
Avant la création d'index :	54
Création des index :	55
Après la création de l'index	55
Chapitre 3.3 : Transaction.....	56
Chapitre 3.4 : Créer des packages sur le SGBD.....	57
Section 3 – Programmation SQL	58

Section 1 – Analyse d'un système d'information

Section 2 – Langage SQL et SGBD

Partie 0 : Notions de base

L'objectif de cette partie est de résumer les notions de base du langage SQL. Nous résumons les 11 chapitres que nous avons réalisés dans ce module. Les chapitres sont organisés selon la composition du langage SQL.

Ce langage est composé de trois parties majeures et distinctes :

- LMD (langage de manipulation des données) : il permet de consulter ou de modifier le contenu de la base de données.
- LDD (langage de définition des données) : il permet de modifier la structure de la base de données.
- LCD (langage de contrôle des données) : il permet de gérer les privilèges, ou les différents droits des utilisateurs sur la base de données.

Voici les principaux ordres employés :

LMD	LDD	LCD
SELECT	CREATE	GRANT
INSERT	ALTER	REVOKE
DELETE	RENAME	
UPDATE	DROP	

Installation

TD 1 : Installation de SQL Server

<https://www.youtube.com/watch?v=o51V6Y9jERM>

TD 2 : Création d'une base de données

Créer la base de données « GestionStagiaires »

```
Groupe (Id, Nom)  
Stagiaire (Id, Nom, IdGroupe)
```

https://www.youtube.com/watch?v=_vrcimJE4ag

TD 3 : Utilisation d'un schéma de la base de données

Créez la base de données en utilisant le schéma de la base de données

<https://www.youtube.com/watch?v=KYBmhna53G4>

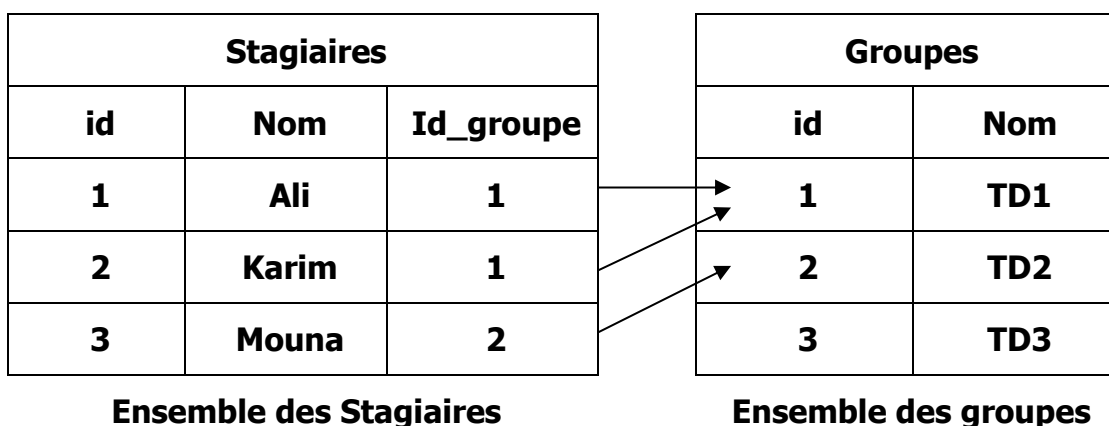
TD 4 : Exécution des requêtes SQL

Chapitre 0.2 : Avant de commencer

L'objectif de ce chapitre est d'apprendre les connaissances initiales du langage SQL pour simplifier la rentrée dans le détail des différentes parties du langage SQL.

Nous allons travailler sur un ensemble des stagiaires et leurs groupes. Par exemple, nous allons apprendre comment déclarer l'existence des tables, ajouter, supprimer ou modifier un stagiaire et rechercher toutes les informations d'un stagiaire qui sont réparties sur plusieurs ensembles.

Dans chaque chapitre, nous travaillons sur un exemple unique pour simplifier la réalisation pratique de différents exemples. Dans ce chapitre, nous prenons deux ensembles : l'ensemble des stagiaires et l'ensemble de leurs groupes.



Création de la table des stagiaires

Pour déclarer l'existence de l'ensemble des stagiaires nous devons déclarer les tables stagiaires avec la requête « Create table ». Il permet de créer une table en déclarant ses attributs :

Requête 1 : Création de la table « Stagiaires »

```
Create table Stagiaires (  
  Id Number,  
  Nom Varchar,
```



```
Ville Varchar,  
Id_groupe Number  
);
```

Par même raisonnement nous créons la table « Groupes » :

Requête 2 : Création de la table « Groupes »

```
Create table Groupes (  
  Id Number,  
  Nom Varchar  
);
```

Résultat d'exécution des deux tables : deux tables vide

Stagiaires		
id	Nom	Id_groupe

Ensemble des Stagiaires

Groupes	
id	Nom

Ensemble des groupes

Insérer des groupes et leurs stagiaires :

Pour ajouter un stagiaire à notre table nous utilisons la requête « Insert ». Cette requête permet d'insérer une ligne ou un enregistrement dans la table.

Mais avant d'insérer un stagiaire nous devons d'abord insérer les groupes.

Requête 3 : Exemple de requête pour insérer le groupe TDI1

```
Insert into Groupes values (1, 'TDI1') ;
```

Maintenant, nous pouvons ajouter le stagiaire « Karim » au groupe TDI1 avec la requête suivante :

Requête 4 : Ajouter le stagiaire « Karim » à la table « Stagiaires »

```
Insert into Stagiaires values (2, 'Karim',1) ;
```

Résultat d'exécution des deux tables :

Stagiaires			Groupes	
id	Nom	Id_groupe	id	Nom
2	Karim	1	1	TD1

Ensemble des Stagiaires

Ensemble des groupes

Modifier un stagiaire :

La requête « UPDATE », permet de mettre à jour un ou plusieurs enregistrements. Par exemple pour changer le nom de stagiaire numéro 1 à « Mouna » nous utilisons la requête suivante :

Requête 5 : Changement du nom du stagiaire "Ali" à "Moad"

```
Update Stagiaires set Nom= 'Moad' where id=1 ;
```

Résultat d'exécution des deux tables :

Stagiaires			Groupes	
id	Nom	Id_groupe	id	Nom
1	<u>Moad</u>	1	1	TD1
2	Karim	1	2	TD2
3	Mouna	2	3	TD3

Ensemble des groupes

Ensemble des groupes

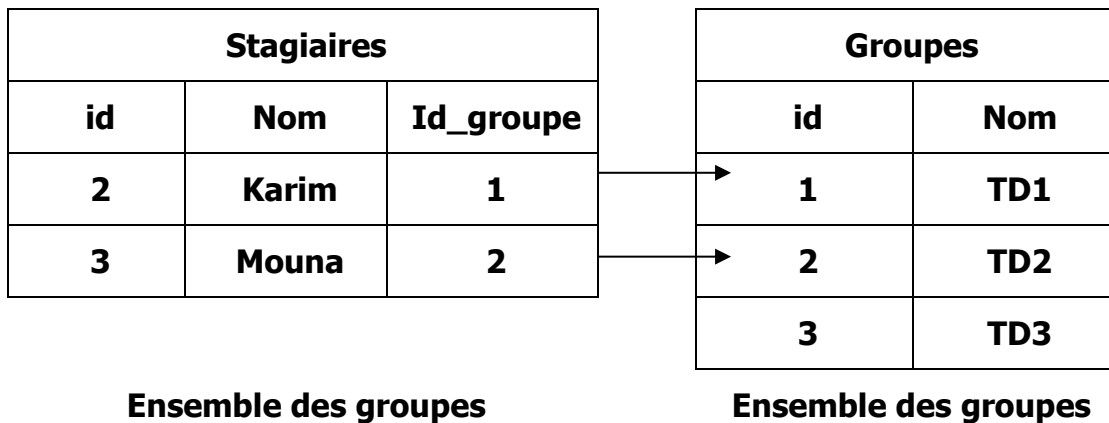
Supprimer un stagiaire

La requête « DELETE » permet de supprimer des enregistrements. Pour supprimer le stagiaire numéro 1 nous devons utiliser la requête suivant :

Requête 6 : Suppression du stagiaire "Mouna" du numéro 1

```
Delete from Stagiaires where id=1;
```

Résultat d'exécution des deux tables :



Rechercher un stagiaire

La requête « SELECT » permet de chercher des données d'une ou plusieurs tables.

Par exemple, pour chercher la liste de tous les stagiaires, nous utilisons la requête « Select » sur la table stagiaire. Le symbole « * » désigne que nous voudrions afficher tous les informations de la table stagiaire.

Requête 7 : Affichage de tous les stagiaires.

```
Select * from Stagiaires;
```

Maintenant, nous voudrions afficher que les stagiaires qui appartiennent au groupe numéro 1. Pour cela nous devons ajouter la condition « id_groupe = 1 » avec la clause « Where ».

Requête 8 : Affichage de tous les stagiaires de groupe numéro 1

```
Select Nom from Stagiaires where Id_groupe = 1;
```

Afficher tous les stagiaires avec le nom de leurs groupes.

Pour afficher par exemple la liste des stagiaires avec les noms de leurs groupes, nous devons utiliser la table stagiaire et la table groupe en même temps dans la clause « From ».

Requête 9 : Produit entre la table Stagiaires et la table Groupes

```
Select * FROM Stagiaire, Groupe;
```

Résultat d'exécution :

id	Nom	Id_groupe	id	Nom
2	Karim	1	3	TD2
2	Karim	1	2	TD3
2	Karim	1	1	TD1
3	Mouna	2	1	TD1
3	Mouna	2	2	TD2
3	Mouna	2	3	TD3

Mais dans la logique du modèle relationnelle, cette requête ne donne pas la bonne solution. Il nous donne un tableau avec deux types d'information, le premier type concerne la solution cherchée, c'est-à-dire la liste des stagiaires avec les noms de leurs groupes. Le deuxième type concerne des informations incorrectes. La question, ici, est d'où viennent ces informations incorrectes ?

Pour répondre à cette question, nous allons rappeler d'abord la notion de produit cartésien entre deux ensembles. Ensuite nous allons appliquer cette notion sur le produit entre la table « Stagiaires » et la table « Groupes ».

Produit cartésien

Le but du produit cartésien est de croiser les données d'un ou plusieurs ensembles, de manière à obtenir toutes les combinaisons possibles.

Exemple 1 : Produit cartésien entre deux ensembles A et B

Soit deux ensembles A et B. l'ensemble A comprend quatre bulles (trois rouges et un bleu) (voir Figure 1 : Ensemble A).

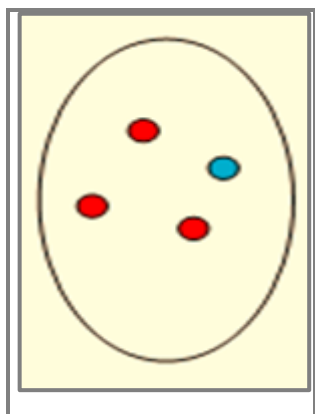


Figure 1 : Ensemble A

L'ensemble B comprend trois triangles (deux bleus et l'autre rouge) (voir Figure 2 : Ensemble B)

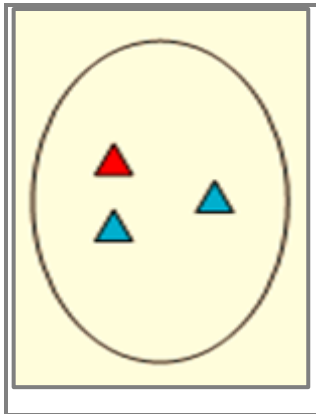


Figure 2 : Ensemble B

Le produit cartésien de l'ensemble A et B donne un ensemble de douze éléments (voir Figure 3 : Produit cartésien de l'ensemble A et B).

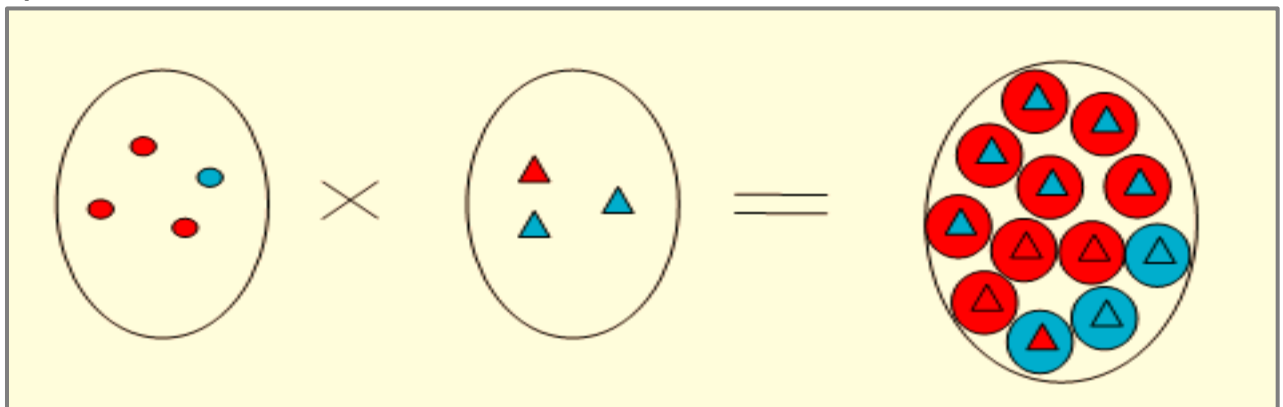


Figure 3 : Produit cartésien de l'ensemble A et B

Remarque

Si nous voulons garder dans le résultat de produit que les éléments qui ont une couleur homogène (deux éléments de même couleur). Nous aurons seulement **deux bleus** et **trois rouges**.

Jointure

Pour chercher nos informations dans la base de données nous devons parfois d'utiliser plusieurs tables. Parce qu'au départ dans la phase de la conception nous avons groupé les données dans des entités suivant le modèle relationnel.

Pour trouver les listes des stagiaires et leurs groupes nous devons chercher l'information dans les : Stagiaires et Groupes.

Dans SQL nous utilise la requête SQL suivante :

```
Select * FROM Stagiaires , Groupes ;
```

La clause « From », ici, réalise le produit cartésien de l'ensemble A (Stagiaires) et l'ensemble B (Groupes).

La **jointure** est aussi un **produit cartésien**. C'est une opération permettant de combiner des informations venant de plusieurs tables.

Exemple 2 : Exemple de jointure entre la table Stagiaires et la Tables Groupes

La jointure est formulée par la requête suivante :

```
Select * FROM Stagiaires , Groupes ;
```

Le résultat de cette requête est aussi un tableau avec 5 colonnes et 6 lignes (voir Figure 4 : Résultat de requête de jointure).

id	Nom	Id_groupe	id	Nom
2	Karim	1	3	TD2
2	Karim	1	2	TD3
2	Karim	1	1	TD1
3	Mouna	2	1	TD1
3	Mouna	2	2	TD2
3	Mouna	2	3	TD3

Figure 4 : Résultat de requête de jointure entre Stagiaires et Groupes

Remarque

La multiplication des 2 tableaux nous donne toutes les combinaisons possibles mais sauf les 3 résultats en bleu qui sont corrects d'où vient la nécessité de **la condition de jointure**.

Si vous remarquez bien, la condition qui vérifie les lignes bleues est : **id_groupe de la table stagiaire = id groupe de la table groupe**

Dans SQL, pour ajouter une condition de jointure nous utilisons la clause « Where » :

```
Select *FROM Stagiaire, Groupe where Stagiaire.id_groupe = Groupe.id ;
```

Autre façon d'écrire cette requête avec les **Alias** :

```
Select *from Stagiaire s,Groupe g where s.id_groupe=g.id ;
```

Le résultat de cette requête est comme suivant, qui est maintenant un résultat correct.

id	Nom	Id_groupe	id	Nom
2	Karim	1	1	TD1
3	Mouna	2	2	TD2

Partie 1 - LDD

Langage de définition de données

Il permet de définir les objets (Tables, View, Function,...) de la base de données.

Chapitre 2.1.1 : Création de la base de données

Nous allons aborder dans ce chapitre les ordres du langage de définition de données. Nous allons commencer par l'ordre « Create Database ».

Pour les exemples, nous allons utiliser la base de données « **GestionStagiaires** ».

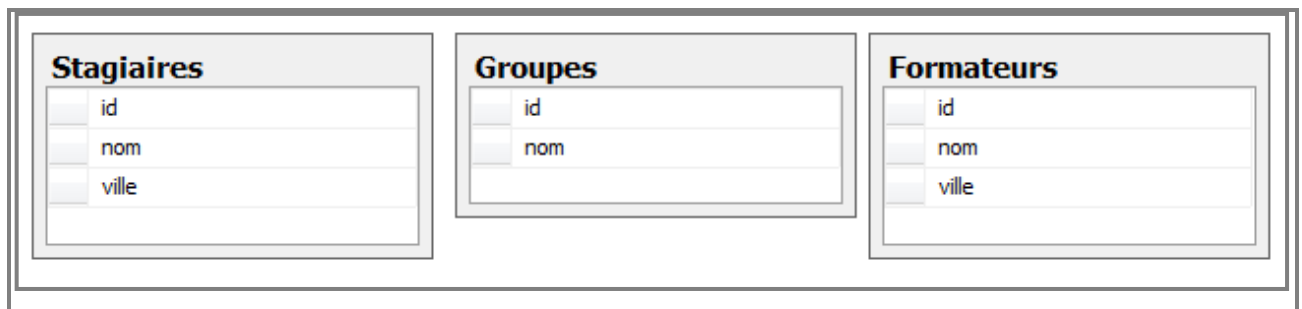


Figure 5 : Base de données des exemples utilisés dans le chapitre 2, Gestion des stagiaires

Création de la base de données

Pour créer une base de données nous utilisons la requête « Create DataBase ».

Requête 10 : Création de la base de données « GestionStagiaires »

```
CREATE DATABASE GestionStagiaires;
```

Résultat d'exécution sous SQL Server 2008 :

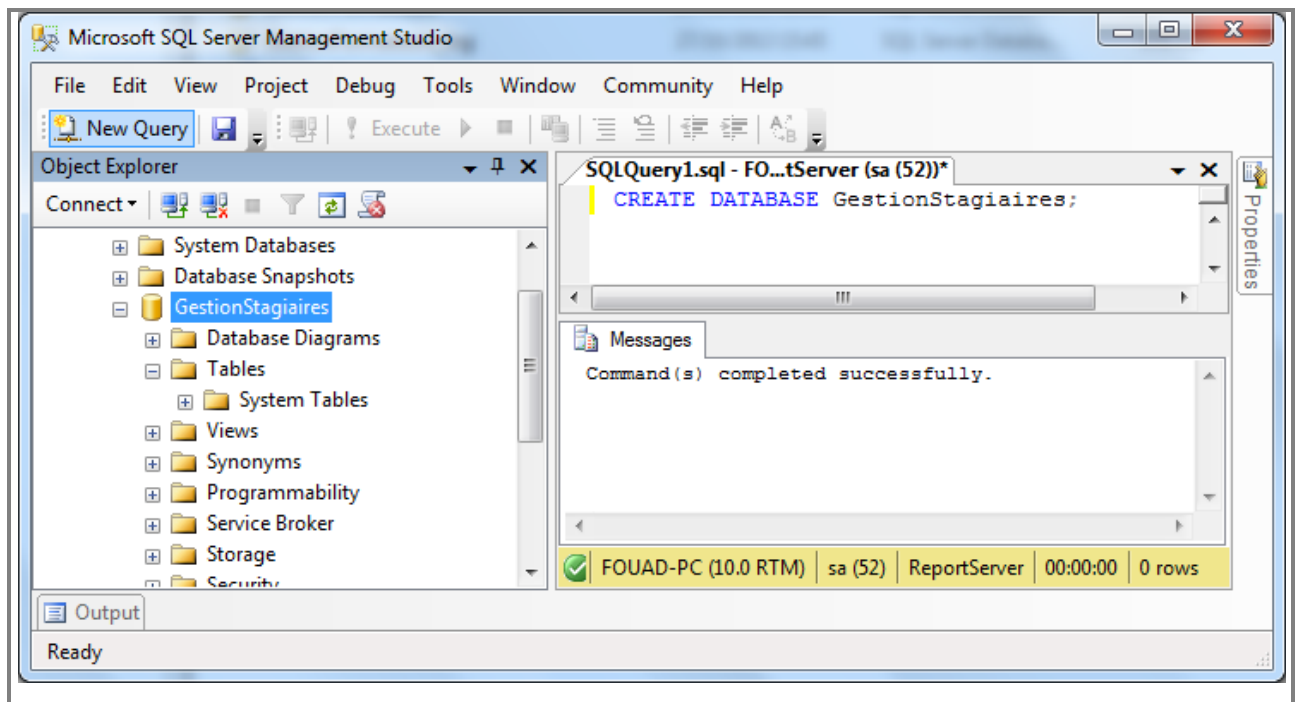


Figure 6 : Création de la base de données - Gestion des stagiaires

Remarque

Pour créer une base de données, il faut être connecté en tant qu'administrateur système, ou avoir la permission d'utiliser CREATE DATABASE, et être dans la base de données système master.

Quelques messages d'erreur :

Msg 5170, Level 16, State 1, Line 1 Cannot create file 'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\GestionStagiaires.mdf' because it already exists. Change the file path or the file name, and retry the operation.

Chapitre 2.1.2 : Création des tables

Pour créer une table nous utilisons la requête « Create Table ».

Requête 11 : Création de la table « Groupes »

```
create table Groupes(  
id int,  
nom varchar(50)  
);
```

Résultat d'exécution sous SQL Server 2008 :

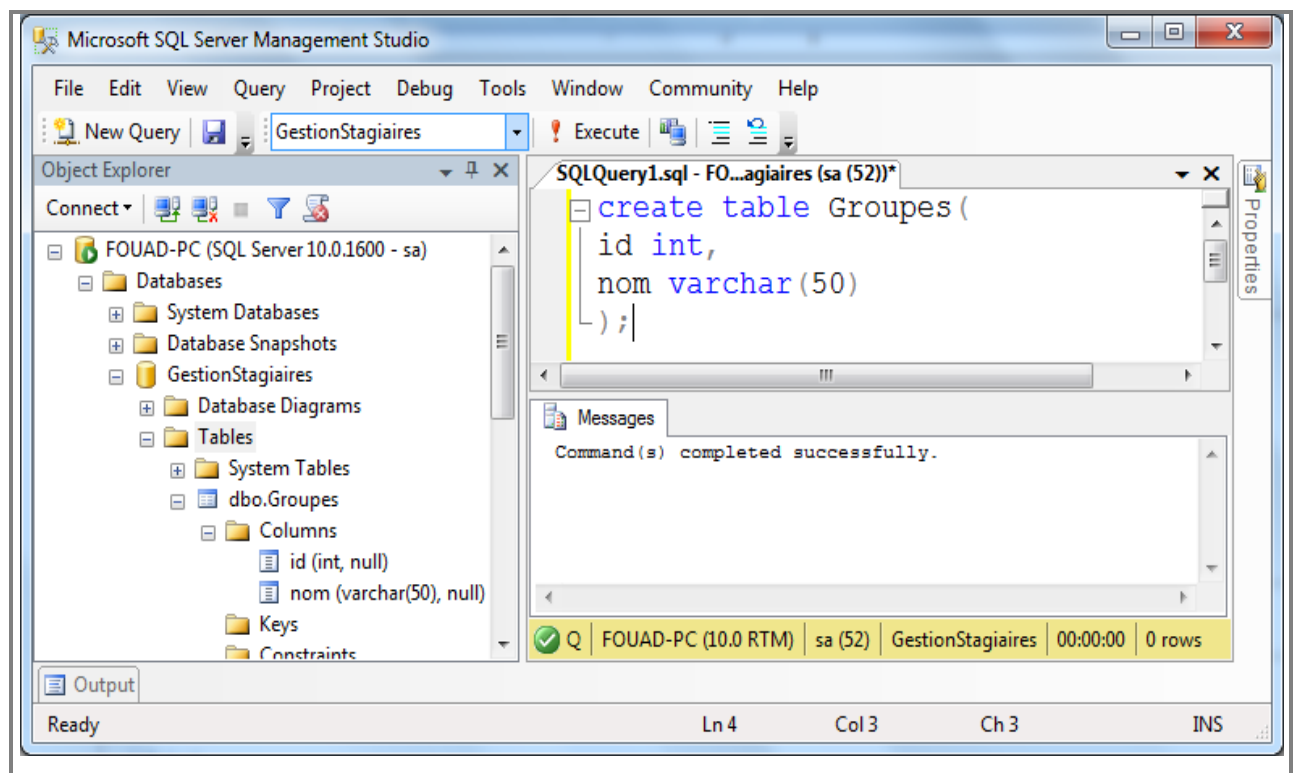


Figure 7 : Création de la table « Groupes »

Requête 12 : Création de la table « Stagiaires »

```
create table Stagiaires(  
id int ,  
nom varchar(50) not null,
```

```
ville varchar(50)
);
```

Résultat d'exécution sous SQL Server 2008

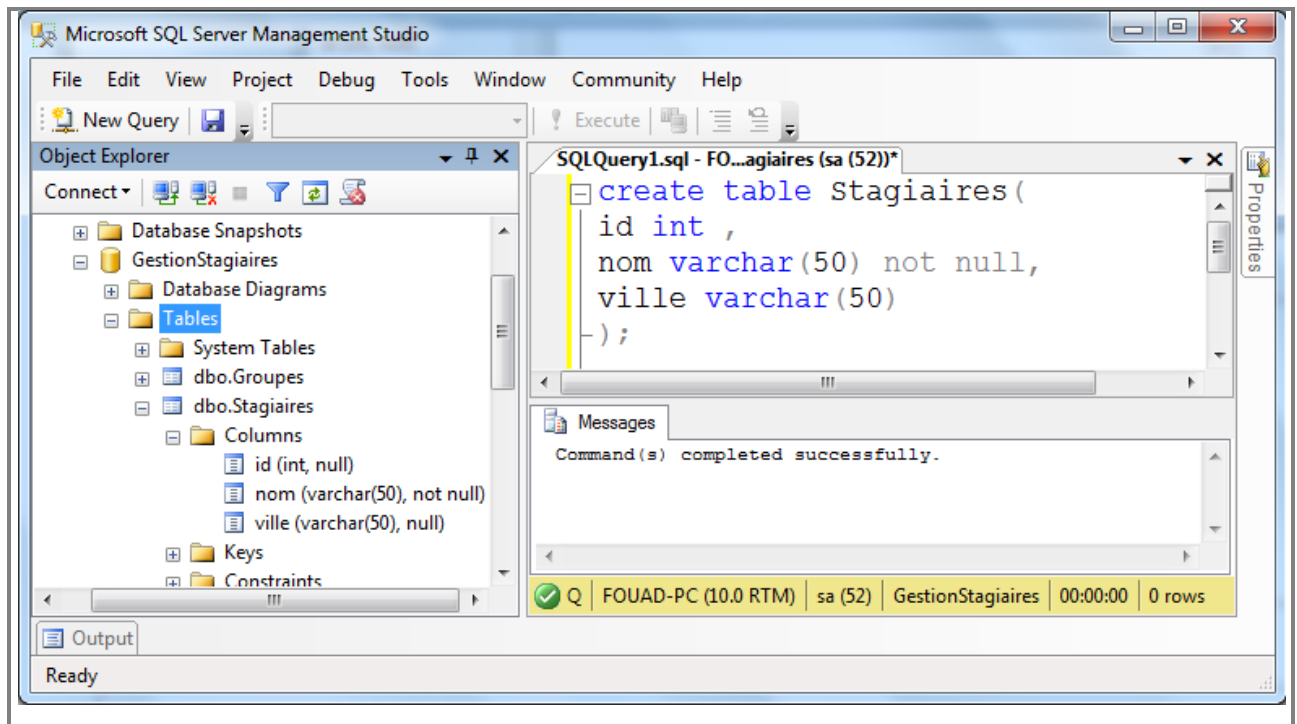


Figure 8 : Création de la table « Stagiaires »

Requête 13 : Création de la table « Formateurs »

```
create table Formateurs(
id int,
nom varchar(50),
ville varchar(50) default 'tanger'
);
```

Résultat d'exécution sous SQL Server 2008

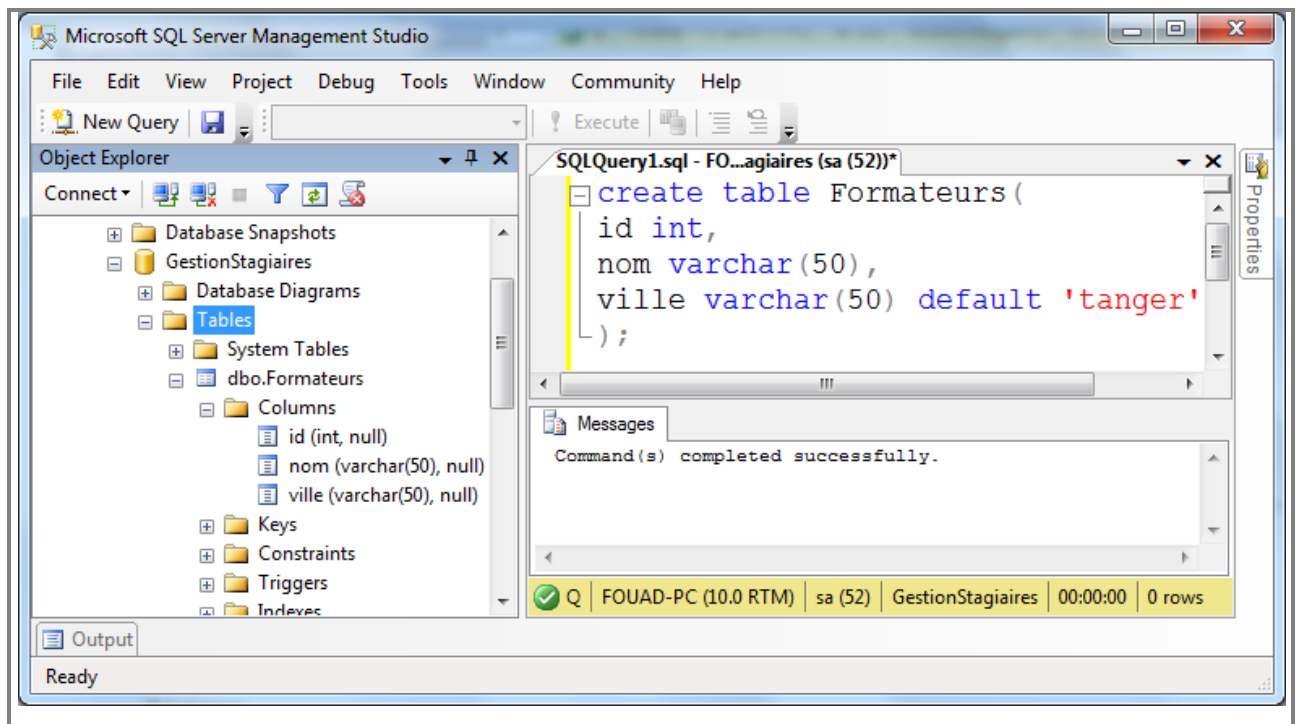


Figure 9 : Création de la table « Formateurs »

Contraintes d'intégrité :

Il y a quatre types des contraintes d'intégrité :

- Unique.
- Check.
- Primary Key.
- Foreign Key.

Contraintes Unique

La contrainte **Unique** : impose une valeur distincte au niveau de la table.

Requête 14 : Création de la table « Formateurs » avec la contrainte « Unique »

```
Create table Formateurs(
id int,
nom varchar(50),
constraint un_nom Unique(nom)
);
```

Contraintes Check

La contrainte **Check** : impose un domaine de valeurs ou une condition entre les colonnes.

Requête 15 : Création de la table « Formateurs » avec la contrainte « Check »

```
Create table Formateurs(  
id int,  
nom varchar(50),  
Constraint nn_nom Check(nom is not null),  
);
```

Contraintes Primary Key

La contrainte « Primary Key » déclare la clé primaire de la table, les colonnes ne peuvent être ni nul ni identique.

Requête 16 : Création de la table « Groupe » avec la contrainte « Primary Key »

```
Create table Groupes(  
id int,  
nom varchar(50),  
Constraint pk_groupe Primary key(id)  
);
```

Contraintes Foreign Key

La contrainte « Foreign Key » déclare une clé étrangère entre une table enfant et une table père.

Requête 17 : Création de la table « Stagiaires » avec la contrainte « Foreign Key »

```
Create table Stagiaires(  
id int,  
nom varchar(50),  
note float,
```

```
filiere varchar(50),  
id_groupe int,  
Constraint fk_stagiaire_groupe Foreign key (id_groupe) References  
Groupe(id)  
);
```

Remarque :

Tous les 4 contrainte peuvent être regroupés dans une même requête SQL

Requête 18 : Création de la table « Stagiaires » avec plusieurs contraintes d'intégrités.

```
Create table Stagiaires(  
id int,  
nom varchar(50),  
note_bac float,  
filiere varchar(50),  
id_groupe int,  
Constraint n_note Check(note_bac between 0 and 20),  
Check(filiere='tdi'or filiere='tri'),  
Constraint pk_stagiaire Primary key(id),  
Constraint fk_stagiaire_groupe Foreign key(id_groupe) References  
Groupes(id)  
);
```

Supprimer une table :

Requête 19 : Suppression de la table « formateurs »

```
Drop table formateurs ;
```

Requête 20 : Suppression de la table « groupes »

```
Drop table Groupes ;
```

On ne peut pas supprimer cette table car elle est référencée par une contrainte foreign key dans la table « Stagiaires ».

Chapitre 2.1.3 : Modifier une table

Ajouter une colonne :

Requête 21 : Ajoute de la colonne ville à la table stagiaire en utilisation de « Alter table »

```
Alter table Stagiaire Add ville varchar(50);
```

Requête 22 : Insertion de la colonne « id_groupe » à la table stagiaire

```
Alter table Stagiaires Add id_groupe int;
```

Résultat d'exécution sous SQL Server 2008

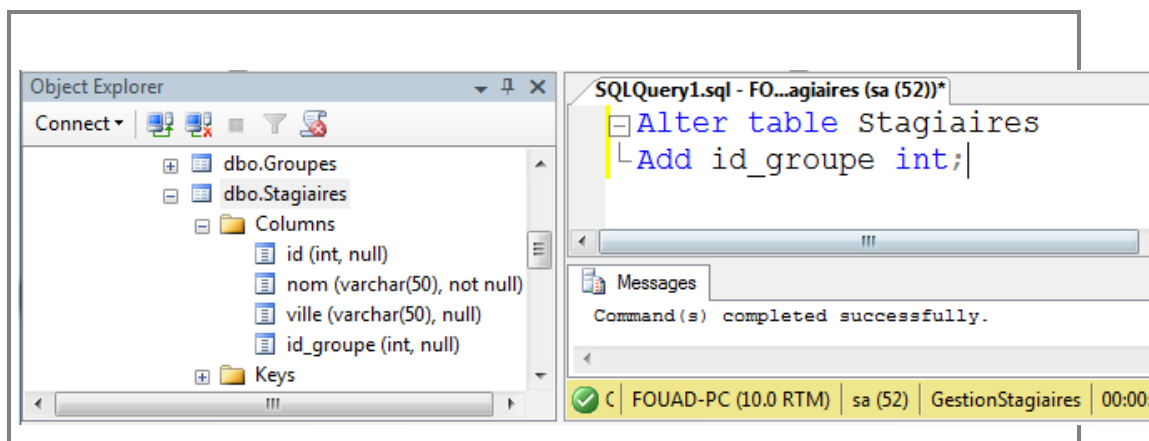


Figure 10 : Ajouter la colonne "id_groupe" à la table « Stagiaires »

Modifier une colonne :

Nous changeons dans cette requête le nombre des caractères et nous ajoutons la contrainte « not null ».

Requête 23 : Modification de la colonne ville

```
Alter table Stagiaires Alter column ville varchar(100) not null;
```

Requête 24 : Ajouter la contrainte « not null » au colonne « id » de la table « stagiaires »


```
Alter table Stagiaires Alter column id int not null;
```

Résultat d'exécution sous SQL Server 2008

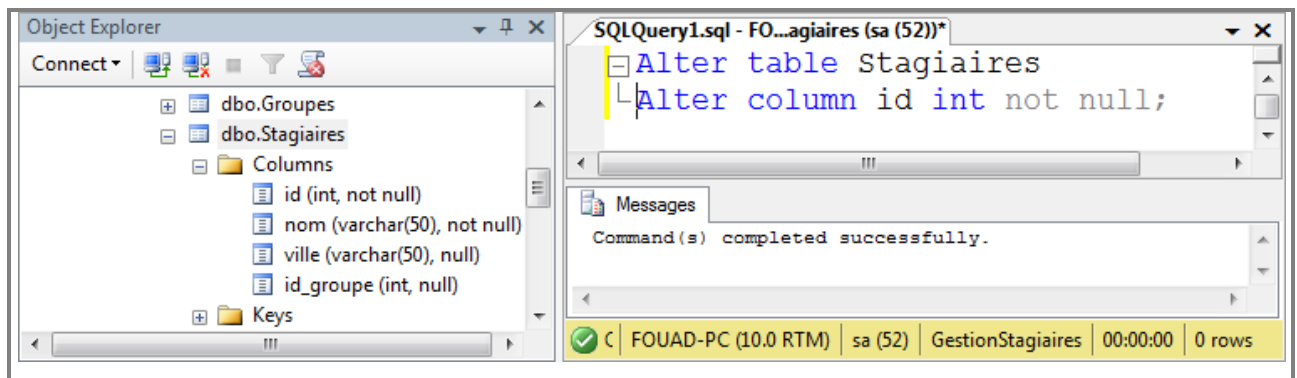


Figure 11 : Ajouter la contrainte « not null » au colonne Id de la table « Stagiaires »

Requête 25 : Ajouter la contrainte « not null » au colonne « id » de la table « groupes »

```
Alter table Stagiaires Alter column id int not null;
```

Résultat d'exécution sous SQL Server 2008

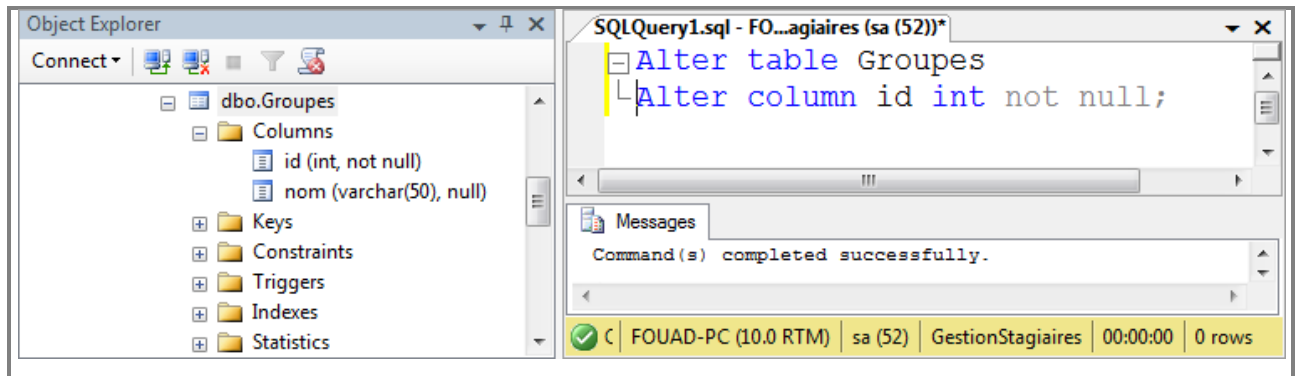


Figure 12 : Ajouter la contrainte « not null » au colonne « id » de la table « Groupees »

Ajouter une contrainte d'intégrité :

Exemple : Ajouter une contrainte « Check » dans la table « Stagiaires »

```
Alter table Stagiaires Add Constraint cc_ville Check(ville='tanger' or  
ville='rabat');
```

Résultat d'exécution sous SQL Server 2008

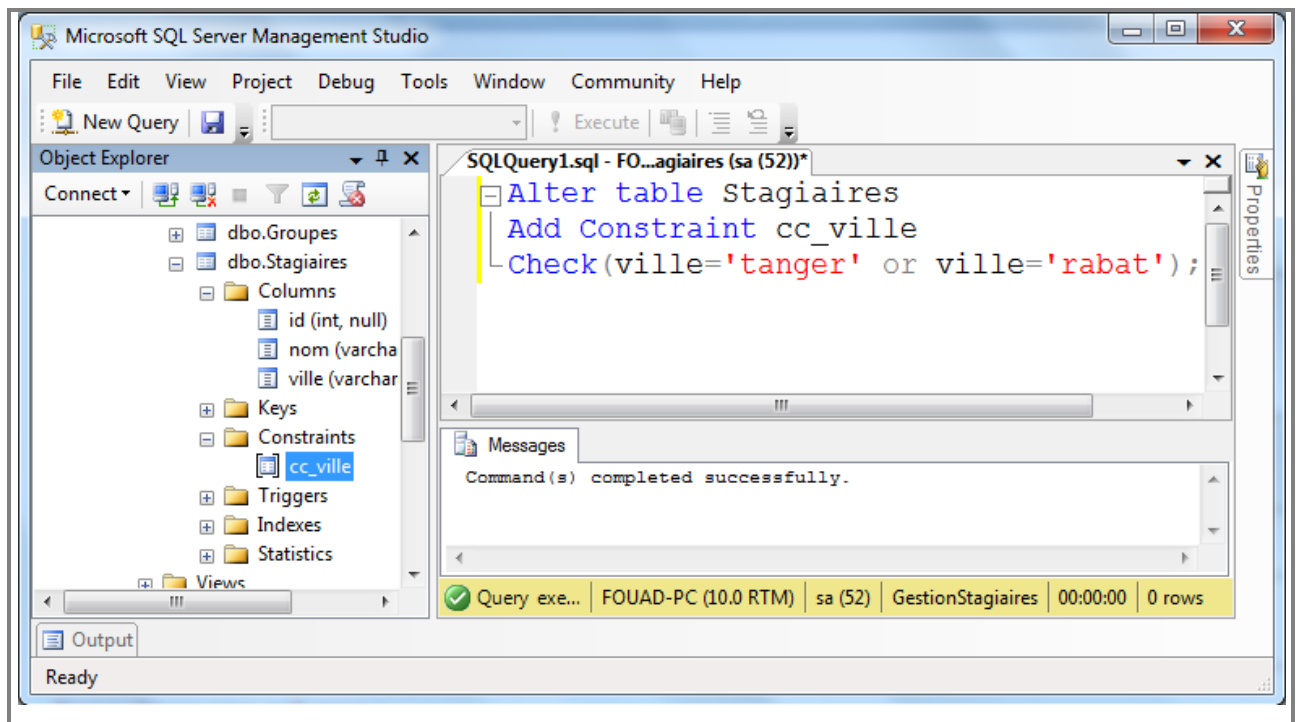


Figure 13 : Ajouter la contrainte Check au champ "Ville" de la table « Stagiaires »

Exemple : Ajouter une contrainte Clé primaire à la table Stagiaires

```
Alter table Stagiaires Add Constraint pk_stagiaire Primary key(id);
```

Résultat d'exécution sous SQL Server 2008

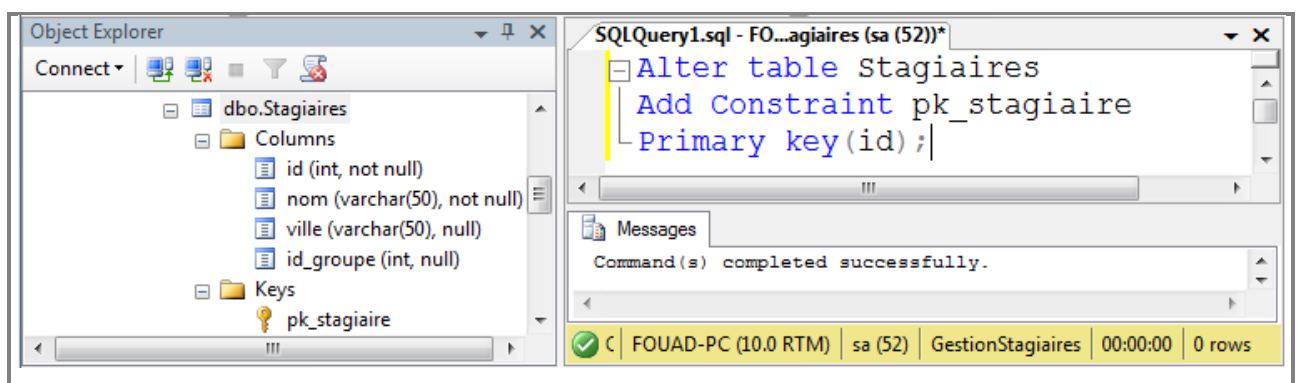


Figure 14 : Ajouter une contrainte Clé primaire à la table « Stagiaires »

Exemple : Ajouter une contrainte Clé primaire à la table Groupes

```
Alter table Groupes Add Constraint pk_groupes Primary key(id);
```

Résultat d'exécution sous SQL Server 2008

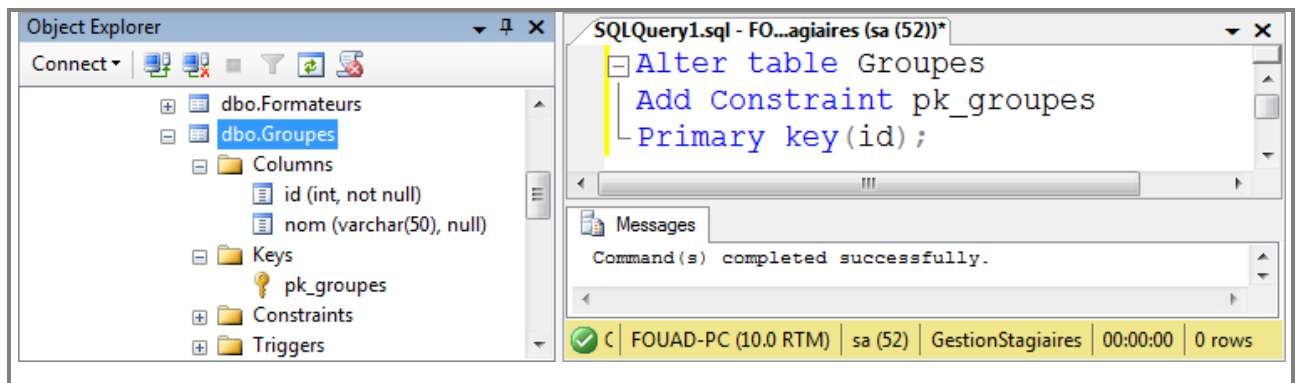


Figure 15 : Ajouter une contrainte Clé primaire à la table « Groupes »

Exemple : Ajouter une contrainte « Foreign key » entre la table stagiaires et groupes

```
Alter table Stagiaires
Add Constraint fk_stagiaires_groupes
Foreign key(id_groupe) References Groupes(id);
```

Résultat d'exécution sous SQL Server 2008

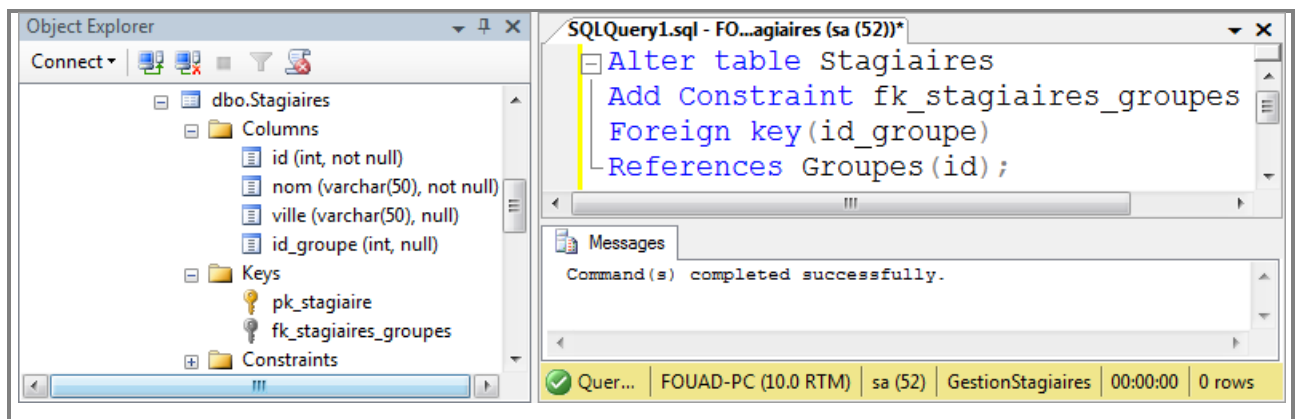


Figure 16 : Ajouter une contrainte « Foreign key » entre la table « Stagiaires » et « Groupes »

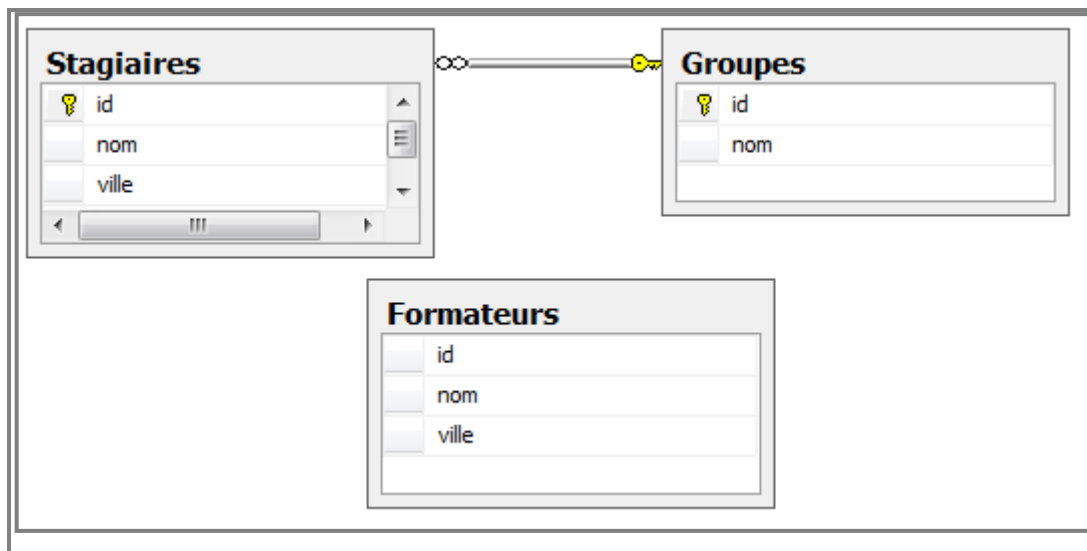


Figure 17 : Base de données de gestion des stagiaires, version 3

Supprimer une colonne :

Exemple : Supprimer la colonne « prenom ».

```
Alter table Stagiaires Drop column prenom;
```

Remarque : Supprimer la colonne « Ville ».

```
Alter table Stagiaire Drop column ville;
```

On ne peut pas supprimer cette colonne car elle liée à la contrainte « cc_ville ».

Supprimer une contrainte :

Exemple : Supprimer la contrainte « cc_ville ».

```
Alter table Stagiaire Drop Constraint cc_ville;
```


Chapitre 2.1.2 : Séquences

La syntax de création de séquence ne diponible que dans SQL Server 2012. Pour réaliser ce principe nous allons utiliser la syntax 'Identity'.

✗ 2005

✗ 2008

✓ 2012

Création de la table « Mention » avec l'auto-incrémentation du champ Id.

Requête 26 : Création d'une table avec Id « auto-incrémenté », Syntax de « Identity »

```
Create table Mention (  
id int Identity( 1 ,2 ),  
nom varchar(50),  
nmaxfloat,  
nmin float
```

);

Partie 2 : Langage de manipulation des données : LMD

Chapitre 2.2.1 : Insert, Delete, Update

Nous allons aborder dans ce chapitre les ordres du langage de manipulation de données. Nous allons commencer par l'ordre Insert.

Insertion d'enregistrement : « Insert »

Trois formes d'insertion :

- ⇒ Insérer tous les attributs
- ⇒ Insérer avec le choix d'attributs à utiliser
- ⇒ Insérer plusieurs enregistrements

Requête d'insertion avec l'utilisation de toutes les colonnes de la table :

Pour ajouter le stagiaire 'Ali' de numéro 1 et de groupe numéro 1 nous utilisons la requête suivant :

Requête 27 : Ajouter le groupe TDI4

```
Insert into groupes values(1,'TDI4');
```

Résultat d'exécution

id	nom
1	TDI4

Requête 28 : Ajouter le stagiaire « Ali » au groupe TDI4

```
Insert into stagiaires values(1,'Ali','Tanger',1);
```

Résultat d'exécution

id	nom	ville	id_groupe
1	Ali	Tanger	1

Requête d'insertion avec l'utilisation quelques colonnes de la table :

Insertion du stagiaire 'Karim' de numéro 1 sans spécifier son groupe.

```
Insert into Stagiaires(Id, Nom) values (2,'Karim');
```

Résultat d'exécution

	id	nom	ville	id_groupe
1	1	Ali	Tanger	1
2	2	Karim	NULL	NULL

Remarque :

SQL server va affecter, ici, la valeur « Null » au colonne « Id_groupe ».

Nous disons que la valeur **Null** , **ici, est implicite**. par ce que c'est le SQL Server qui a affecté cette valeur.

Requête d'insertion de plusieurs stagiaires :

```
Insert into Stagiaires (id,nom) values (3,'Mouna'),(4,'Karima') ;
```

Résultat d'exécution

	id	nom	ville	id_groupe
1	1	Ali	Tanger	1
2	2	Karim	NULL	NULL
3	3	Mouna	NULL	NULL
4	4	Karima	NULL	NULL

Remarque :

Les valeurs peuvent être explicitement **Null** ou **Default** c'est-à-dire que l'utilisateur peut choisir de donner des valeurs comme null ou bien par défaut (ici nous avons inséré le 1^{er} stagiaire avec un **nom Null** et le 2^{ème} avec un **nom par défaut**).

```
Insert into Stagiaires (id,nom,ville)  
values (5,'Mouhamed',Null),(6,'Ali',Default);
```

Résultat d'exécution

	id	nom	ville	id_groupe
1	1	Ali	Tanger	1
2	2	Karim	NULL	NULL
3	3	Mouna	NULL	NULL
4	4	Karima	NULL	NULL
5	5	Mouhamed	NULL	NULL
6	6	Ali	NULL	NULL

Modification d'enregistrement : « Update »

Exemple : Mise à jour d'un stagiaire

Nous voudrions modifier le nom et le prénom de stagiaire numéro 1 ainsi que son groupe.

```
Update Stagiaires  
Set nom = 'Moad', ville = 'Rabat'  
where id=2 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe
1	1	Ali	Tanger	1
2	2	Moad	Rabat	NULL
3	3	Mouna	NULL	NULL
4	4	Karima	NULL	NULL
5	5	Mouhamed	NULL	NULL
6	6	Ali	NULL	NULL

Remarque

Si nous éliminons la condition « Where » la requête va modifier le nom et le prénom de tous les stagiaires.

Suppression d'enregistrement : « Delete »

Exemple : Supprimer un stagiaire

La requête consiste à supprimer le stagiaire numéro 3.

```
Delete from Stagiaires where id = 3;
```

Résultat d'exécution

	id	nom	ville	id_groupe
1	1	Ali	Tanger	1
2	2	Moad	Rabat	NULL
3	4	Karima	NULL	NULL
4	5	Mouhamed	NULL	NULL
5	6	Ali	NULL	NULL

Chapitre 2.2.2 : Manipulation des données avec le type : « DateTime »

Pour traiter ce type de données, nous ajoutons la colonne « DateNaissance » de type « DateTime » à la table « Stagiaires » avec l'utilisation de requête « Alter table ».

Exemple 3 : Utilisation de type DateTime dans SQL Server 2008"

Requête 29 : Ajoute de la colonne « DateNaissance » à la table Stagiaires

```
Alter table Stagiaires add DateNaissance Datetime
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	NULL
2	2	Moad	Rabat	NULL	NULL
3	4	Karima	NULL	NULL	NULL
4	5	Mouhamed	NULL	NULL	NULL
5	6	Ali	NULL	NULL	NULL

Requête 30 : Mise à jour de la date de naissance avec date et heur : minute : second

```
Update Stagiaires Set DateNaissance = '20130918 13:22:06' where id =1 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	2013-09-18 13:22:06.000
2	2	Moad	Rabat	NULL	NULL
3	4	Karima	NULL	NULL	NULL
4	5	Mouhamed	NULL	NULL	NULL
5	6	Ali	NULL	NULL	NULL

Requête 31 : Mise à jour de la date de naissance avec date et heur : minute

```
Update Stagiaires Set DateNaissance = ' 20130918 13 :20' where id = 2 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	2013-09-18 13:22:06.000
2	2	Moad	Rabat	NULL	2013-09-18 13:20:00.000
3	4	Karima	NULL	NULL	NULL
4	5	Mouhamed	NULL	NULL	NULL
5	6	Ali	NULL	NULL	NULL

Requête 32 : Mise à jour de la date de naissance avec seulement la date

```
Update Stagiaires Set DateNaissance = '2013/09/14' where id =4 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	2013-09-18 13:22:06.000
2	2	Moad	Rabat	NULL	2013-09-18 13:20:00.000
3	4	Karima	NULL	NULL	2013-09-14 00:00:00.000
4	5	Mouhamed	NULL	NULL	NULL
5	6	Ali	NULL	NULL	NULL

Requête 33 : Recherche par date

```
Select * from Stagiaires  
where DateNaissance between '20130918 12 :00' and '20130918 14 :00' ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	2013-09-18 13:22:06.000
2	2	Moad	Rabat	NULL	2013-09-18 13:20:00.000

Requête 34 : Mise à jour la date de naissance par date d'aujourd'hui

```
Update Stagiaires Set DateNaissance = CURRENT_TIMESTAMP where id  
=5 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	2013-09-18 13:22:06.000
2	2	Moad	Rabat	NULL	2013-09-18 13:20:00.000
3	4	Karima	NULL	NULL	2013-09-14 00:00:00.000
4	5	Mouhamed	NULL	NULL	2013-10-27 18:36:09.090
5	6	Ali	NULL	NULL	NULL

Requête 35 : Recherche par jour

```
Select *from Stagiaires where Day(DateNaissance)=14 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	4	Karima	NULL	NULL	2013-09-14 00:00:00.000

Requête 36 : Recherche par Mois

```
Select *from Stagiaires where Month (DateNaissance)=9 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	2013-09-18 13:22:06.000
2	2	Moad	Rabat	NULL	2013-09-18 13:20:00.000
3	4	Karima	NULL	NULL	2013-09-14 00:00:00.000

Requête 37 : Recherche par Année

```
Select * from Stagiaires where Year (DateNaissance)=2013 ;
```

Résultat d'exécution

	id	nom	ville	id_groupe	DateNaissance
1	1	Ali	Tanger	1	2013-09-18 13:22:06.000
2	2	Moad	Rabat	NULL	2013-09-18 13:20:00.000
3	4	Karima	NULL	NULL	2013-09-14 00:00:00.000

Chapitre 2.2.3 : Interrogation des données

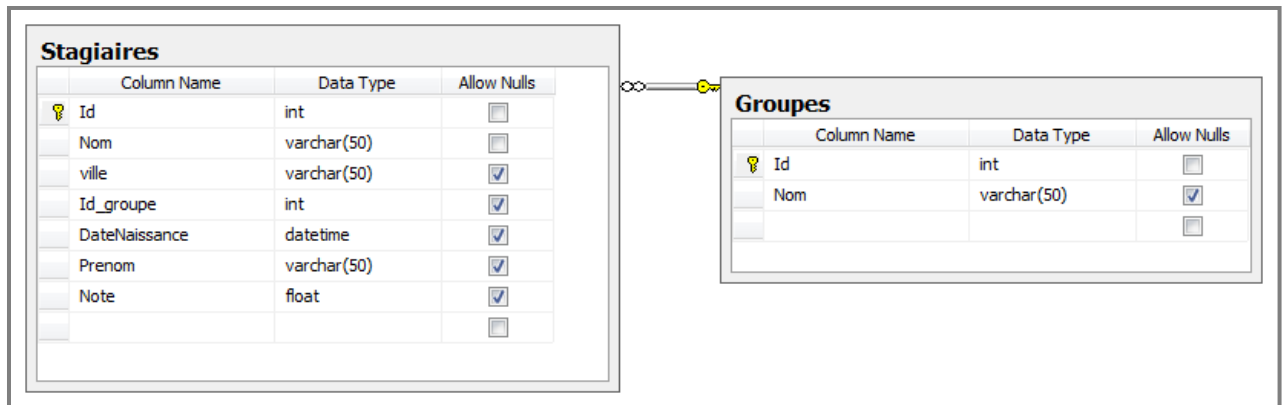


Figure 18 : Base de données d'exemples de chapitre 4

	Id	Nom	Prenom	ville	DateNaissance	Note	Id_groupe
1	1	Madani	Ismail	Tanger	1995-09-18 13:22:06.000	9,5	1
2	2	Madani	Mouhssine	Rabat	1995-09-18 13:20:00.000	17	1
3	4	Wahabi	Imaran	Tanger	1995-09-14 00:00:00.000	14	1
4	5	Wahabi	Youssef	Tanger	1995-10-27 18:36:09.090	11,5	2
5	6	Andaloussi	Yasimina	Rabat	1995-10-27 18:36:09.090	13	2

Figure 19 : Données de la table « Stagiaires », Chapitre 4

	Id	Nom
1	1	TDI4
2	2	TDI5

Figure 20 : Données de la table « Groupes », Chapitre 4

Extraction des données : « Select »

Pour extraire ou chercher des informations dans la base de données, nous utilisons la requête « Select ».

Requête 38 : afficher la liste de tous les stagiaires

```
Select * from stagiaires ;
```

Résultat d'exécution

	Id	Nom	ville	Id_groupe	DateNaissance	Prenom	Note
1	1	Madani	Tanger	1	1995-09-18 13:22:06.000	Ismail	9,5
2	2	Madani	Rabat	1	1995-09-18 13:20:00.000	Mouhssine	17
3	4	Wahabi	Tanger	1	1995-09-14 00:00:00.000	Imaran	14
4	5	Wahabi	Tanger	2	1995-10-27 18:36:09.090	Youssef	11,5
5	6	Andaloussi	Rabat	2	1995-10-27 18:36:09.090	Yasimina	13

Requête 39 : afficher les noms de tous les stagiaires

```
Select nom from stagiaires ;
```

Résultat d'exécution

	nom
1	Madani
2	Madani
3	Wahabi
4	Wahabi
5	Andaloussi

Duplicate : « distinct »

Si on a plusieurs données similaires à l'instant de l'affichage du résultat, distinct nous permet d'éliminer les doublants.

Requête 40 : Utilisation de « distinct » pour éliminer les informations redoublantes.

```
Select distinct(nom) from stagiaires;
```

Résultat d'exécution

	nom
1	Andaloussi
2	Madani
3	Wahabi

Select avec expression :

Cet exemple nous montre qu'on peut modifier les données d'une colonne et les mettre dans une nouvelle colonne qui prend un nouveau nom qu'on appelle, ici « NotePlus »

Requête 41 : Afficher la note augmentée par 10 %.

```
Select Nom,Prenom,note,(note * 0.1 + note) as NotePlus from Stagiaires;
```

Résultat d'exécution

	Nom	Prenom	note	NotePlus
1	Madani	Ismail	9,5	10,45
2	Madani	Mouhssine	17	18,7
3	Wahabi	Imaran	14	15,4
4	Wahabi	Youssef	11,5	12,65
5	Andaloussi	Yasimina	13	14,3

Ordonnancement : « Order By »

Requête 42 : Affichage de toutes stagiaires avec ordre par défaut, croissant

```
Select Nom,Prenom, Note from stagiaires ORDER BY note;
```

Résultat d'exécution

	Nom	Prenom	Note
1	Madani	Ismail	9,5
2	Wahabi	Youssef	11,5
3	Andaloussi	Yasimina	13
4	Wahabi	Imaran	14
5	Madani	Mouhssine	17

La clause **ORDER BY** est utilisée dans une instruction **SELECT** pour trier les données d'une table (ou plusieurs tables) en fonction d'une ou plusieurs colonnes.

Par défaut, le rangement se fera par ordre croissant ou par ordre alphabétique.

Avec le mot clé **ASC**, le rangement se fera dans l'ordre ascendant ↗. Avec le mot clé **DESC**, le rangement se fera dans l'ordre descendant ↘.

Requête 43 : Affichage de toutes stagiaires avec ordre croissant des notes

```
Select Nom,Prenom, Note from stagiaires ORDER BY note ASC;
```

Résultat d'exécution

	Nom	Prenom	Note
1	Madani	Ismail	9,5
2	Wahabi	Youssef	11,5
3	Andaloussi	Yasimina	13
4	Wahabi	Imaran	14
5	Madani	Mouhssine	17

Requête 44 : Affichage de toutes stagiaires avec ordre décroissante des notes

```
Select Nom,Prenom, Note from stagiaires ORDER BY note DESC;
```

Résultat d'exécution

	Nom	Prenom	Note
1	Madani	Mouhssine	17
2	Wahabi	Imaran	14
3	Andaloussi	Yasimina	13
4	Wahabi	Youssef	11,5
5	Madani	Ismail	9,5

Concaténation : « + »

C'est associer des informations venant de plusieurs colonnes de type chaîne de caractère dans une nouvelle colonne qui porte un nouveau nom qu'on appelle **Alias**.

Requête 45 : Affichage de Nom Complet (Nom + Prénom) de tous les stagiaires.

```
Select nom+' '+prenom as NomComplet , ville from Stagiaires;
```

Résultat d'exécution

	NomComplet	ville
1	Madani Ismail	Tanger
2	Madani Mouhssine	Rabat
3	Wahabi Imaran	Tanger
4	Wahabi Youssef	Tanger
5	Andaloussi Yasimina	Rabat

Opérateurs intégrés :

L'opérateur « IN » :

Permet de rechercher une valeur qui se trouve dans un ensemble de données.

Requête 46 : Afficher les stagiaires dont la ville = Tanger ou Casa

```
Select * from Stagiaires where Ville IN ('Tanger','Casa');
```

Résultat d'exécution

	Id	Nom	ville	Id_groupe	DateNaissance	Prenom	Note
1	1	Madani	Tanger	1	1995-09-18 13:22:06.000	Ismail	9,5
2	4	Wahabi	Tanger	1	1995-09-14 00:00:00.000	Imaran	14
3	5	Wahabi	Tanger	2	1995-10-27 18:36:09.090	Youssef	11,5

L'opérateur « LIKE » :

Permet de rechercher des chaînes de caractères avec des conditions selon le besoin, on utilise le symbole « % » pour remplacer un alphabet ; par exemple si on veut

rechercher les stagiaires dont le nom contient 'A' au milieu, la syntaxe sera comme suit :

Requête 47 : Utilisation de l'opérateur «like» pour chercher des stagiaires

```
select * from Stagiaires where nom LIKE ('%m%') ;
```

Résultat d'exécution

	Id	Nom	ville	Id_groupe	DateNaissance	Prenom	Note
1	1	Madani	Tanger	1	1995-09-18 13:22:06.000	Ismail	9,5
2	2	Madani	Rabat	1	1995-09-18 13:20:00.000	Mouhssine	17

Statistiques sur les données (Sum, Min, Max, Avg, Count)

Les requêtes utilisant les fonctions **AVG**, **COUNT**, **SUM**, **MAX** et **MIN** ne renvoient qu'une seule valeur.

- **SUM ()** : somme des valeurs d'une colonne
- **AVG ()** : moyenne des valeurs d'une colonne
- **MAX ()** : maximum des valeurs d'une colonne
- **MIN ()** : minimum des valeurs d'une colonne.
- **COUNT ()** : compte le nombre des lignes.

Requête 48 : Afficher la note moyenne de tous les stagiaires

```
select AVG (note) as NoteMoyenne from Stagiaires;
```

Résultat d'exécution

	NoteMoyenne
1	13

Requête 49 : Afficher la note maximale de tous les stagiaires

```
select MAX(note) as NoteMaximumfrom Stagiaires;
```

Résultat d'exécution

	NoteMaximum
1	17

Regroupement : « GROUP BY »

Requête 50 : Afficher les notes maximums de toutes les villes

```
Select Ville , max(note) as NoteMax From Stagiaires GROUP BY ville
```

Résultat d'exécution

	Ville	NoteMax
1	Rabat	17
2	Tanger	14

Remarque

Tous les attributs placés derrière la clause **SELECT** doivent être présents derrière la clause **GROUP BY**.

L'élimination de certains groupes par « **HAVING** »:

Lorsqu'on obtient le résultat d'une requête issue d'un regroupement, on peut souhaiter ne sélectionner que certains groupes parmi d'autres. Pour cela, il faut ajouter à **GROUP BY** la clause **HAVING** qui permet la sélection de groupes parmi d'autres.

Requête 51 : Afficher les notes maximums supérieures à 10 de toutes les villes en utilisation de « Having »

```
Select Ville , max(note) as NoteMax From Stagiaires GROUP BY ville  
HAVING max(note) > 15 ;
```

Résultat d'exécution

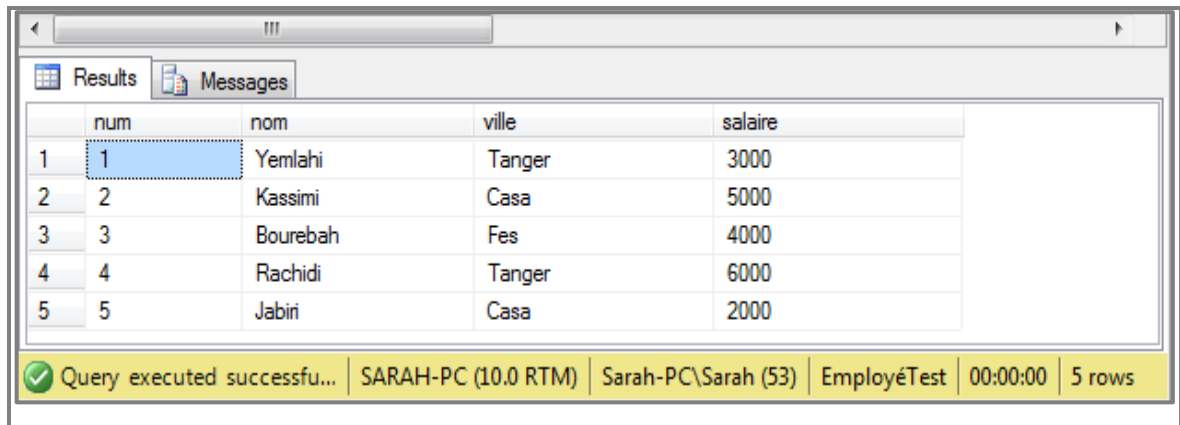
	Ville	NoteMax
1	Rabat	17

Remarque

Il ne pas confondre **HAVING** et **WHERE** : **HAVING** permet la sélection de groupes à la suite d'une requête avec regroupement alors que **WHERE** permet de sélectionner des lignes pour construire la requête.

Chapitre 2.2.4 : Sous interrogations

Prenant l'exemple de la table **Employé** pour bien comprendre :



	num	nom	ville	salaire
1	1	Yemlahi	Tanger	3000
2	2	Kassimi	Casa	5000
3	3	Bourebah	Fes	4000
4	4	Rachidi	Tanger	6000
5	5	Jabiri	Casa	2000

Query executed successfully... SARAH-PC (10.0 RTM) | Sarah-PC\Sarah (53) | EmployéTest | 00:00:00 | 5 rows

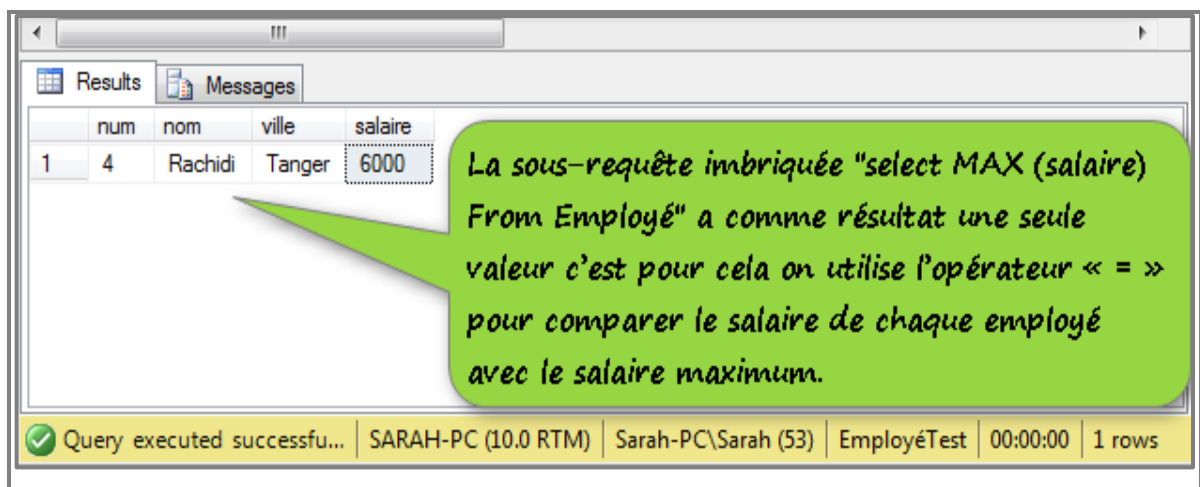
Figure 1 : Table Employé

Sous-interrogations mono-lignes

Requête 52 : Afficher la liste des employés ayant le salaire maximal.

```
Select * From Employé where salaire =(select MAX(salaire) From Employé) ;
```

Résultat :



	num	nom	ville	salaire
1	4	Rachidi	Tanger	6000

La sous-requête imbriquée "select MAX (salaire) From Employé" a comme résultat une seule valeur c'est pour cela on utilise l'opérateur « = » pour comparer le salaire de chaque employé avec le salaire maximum.

Query executed successfully... SARAH-PC (10.0 RTM) | Sarah-PC\Sarah (53) | EmployéTest | 00:00:00 | 1 rows

Figure 2 : Résultat sous-interrogations mono-lignes.

Sous-interrogations multi-lignes : (IN, ANY, ALL)

Utilisation de « IN » :

Le principe consiste à construire une sous-requête qui donne un résultat équivalent à une liste.

Ensuite la requête principale permet de sélectionner des lignes dans la liste précédemment construite.

Requête 53 : Utilisation sous-interrogation avec l'opérateur « IN »

```
Select * from Employé where salaire IN (Select MAX(salaire) as SalaireMax from Employé GROUP BY ville) ;
```

Résultat d'exécution sous SQL Server 2008 :

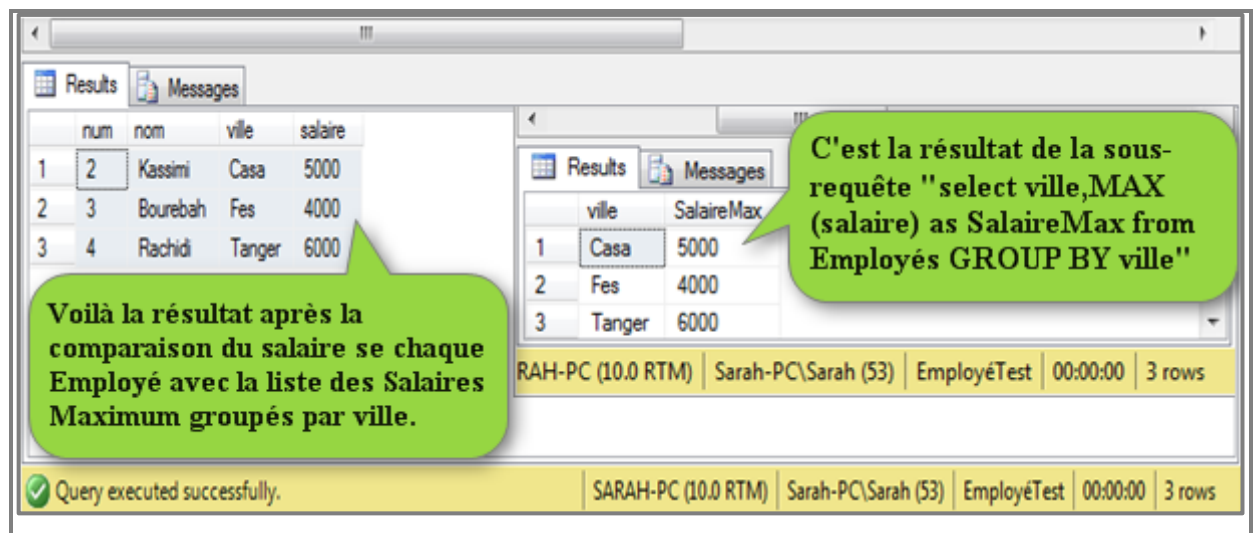


Figure 3 : Résultat de sous-interrogations IN

Utilisation de « ANY » :

Nous voudrions afficher la liste des employés ayant le salaire supérieur au moins d'un salaire maximale d'une ville.

Requête 54 : Utilisation sous-interrogation avec l'opérateur « Any »

```
Select * From Employé where salaire > ANY(Select MAX (salaire) from Employés GROUP BY ville) ;
```

Résultat d'exécution sous SQL Server 2008 :

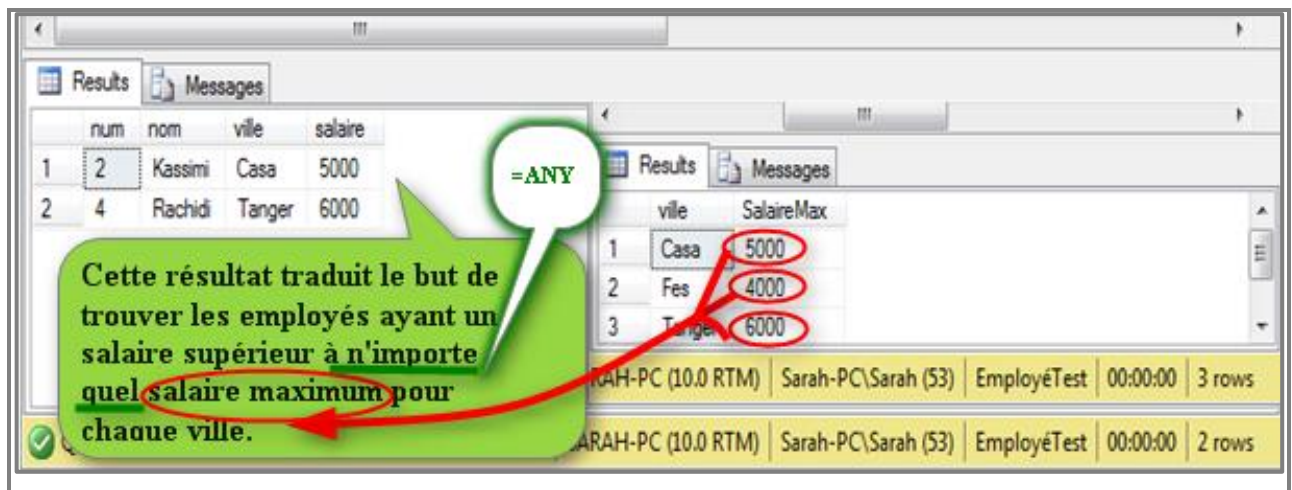


Figure 4 : Résultat de sous-interrogations ANY

Utilisation de « ALL » :

Exemple :

Nous voudrions afficher la liste des employés ayant le salaire inférieur de tous les salaires maximums de toutes les villes.

Requête 55 : Utilisation sous-interrogation avec l'opérateur « All »

```
Select * From Employés where salaire < All(select MAX (salaire) from Employés GROUP BY ville) ;
```

Résultat d'exécution sous SQL Server 2008 :

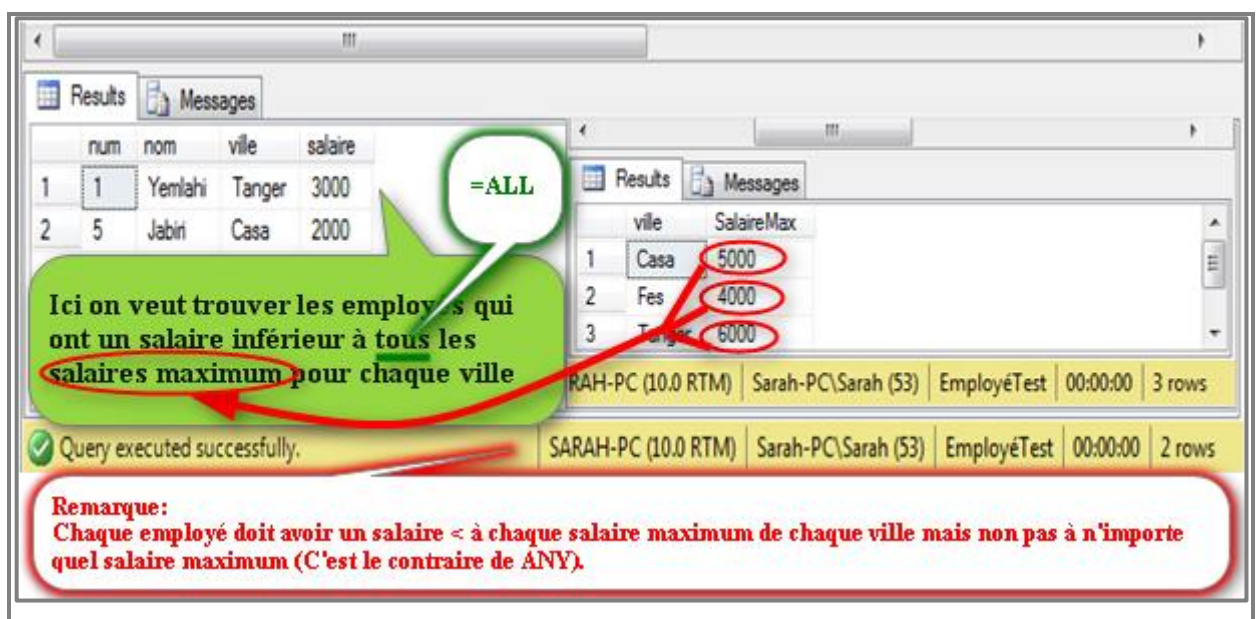


Figure 21 : Résultat de sous-interrogations ALL

Chapitre 2.2.5 : Jointure

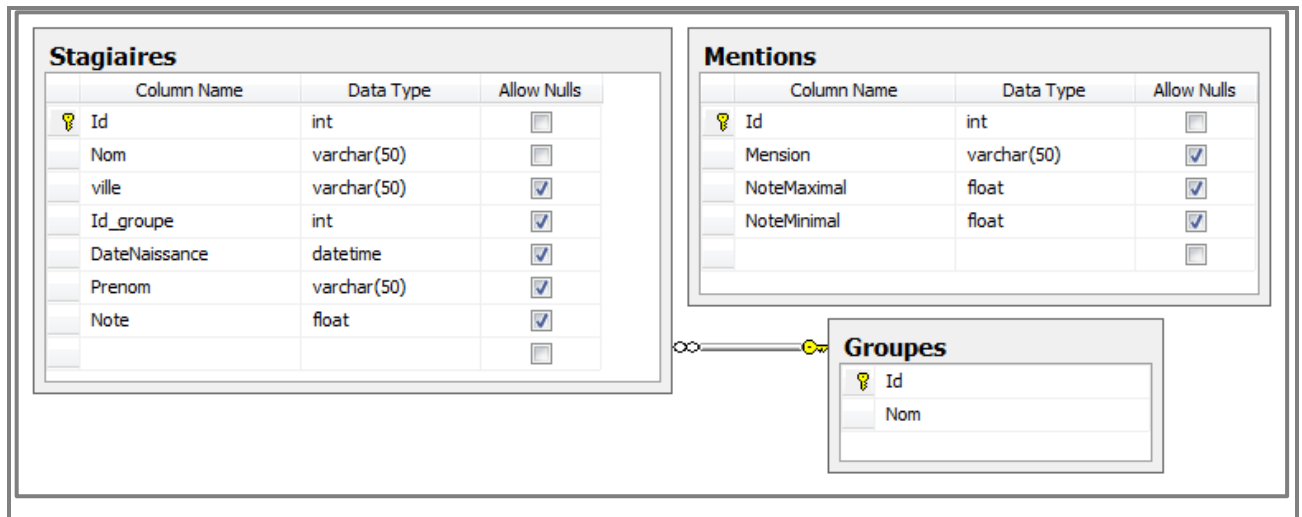


Figure 22 : Base de données des exemples de chapitre 6

	Id	Nom	ville	Id_groupe	DateNaissance	Prenom	Note
1	1	Madani	Tanger	1	1995-09-18 13:22:06.000	Ismail	9,5
2	2	Madani	Rabat	1	1995-09-18 13:20:00.000	Mouhssine	17
3	4	Wahabi	Tanger	1	1995-09-14 00:00:00.000	Imaran	14
4	5	Wahabi	Tanger	2	1995-10-27 18:36:09.090	Youssef	11,5
5	6	Andaloussi	Rabat	2	1995-10-27 18:36:09.090	Yasimina	13

Figure 23 : Données de la table « Stagiaires », chapitre 6

	Id	Mention	NoteMaximal	NoteMinimal
1	1	Non validé	0	10
2	2	Passable	10	12
3	3	Assez bien	12	14
4	4	Bien	14	17
5	5	Trés bien	17	20

Figure 24 : Données de la table « Mentions »

Produit Cartésien avec SQL

Exemple du produit cartésien entre deux tables **Stagiaire** et **Groupe** :

Requête SQL :

```
Select s.Prenom,s.Id_groupe,g.Id, g.Nom from Stagiaires s,GROUPES g
```

Résultat d'exécution

	Prenom	Id_groupe	Id	Nom
1	Ismail	1	1	TDI4
2	Mouhssine	1	1	TDI4
3	Imaran	1	1	TDI4
4	Youssef	2	1	TDI4
5	Yasimina	2	1	TDI4
6	Ismail	1	2	TDI5
7	Mouhssine	1	2	TDI5
8	Imaran	1	2	TDI5
9	Youssef	2	2	TDI5
10	Yasimina	2	2	TDI5

Remarque :

La multiplication des deux tableaux nous donnent toutes les combinaisons possibles mais sauf les trois résultats où on a **s.id=g.id_groupe** qui sont correctes d'où vient la notion de **la condition de jointure**.

Jointure relationnelle :

Il est caractérisé par une seule clause « **From** » contenant les tables et alias .

Équijointure :

Elle utilise l'opérateur **égalité** dans la clause de jointure.

Requête 56 : Exemple de « équijointure »

```
Select s.Prenom,s.Id_groupe,g.Id, g.Nom from Stagiaires s,GROUPES g
where s.id_groupe = g.id;
```

Résultat d'exécution

	Prenom	Id_groupe	Id	Nom
1	Ismail	1	1	TDI4
2	Mouhssine	1	1	TDI4
3	Imaran	1	1	TDI4
4	Youssef	2	2	TDI5
5	Yasimina	2	2	TDI5

InÉquijointure :

Il fait intervenir tout type d'opérateur <>, <, >, >=, like,in,Between...

Exemple :

Nous voudrions afficher la liste des stagiaires avec leur mention.



Requête 57 : Exemple de « In Equijointure »

```
Select Nom,Prenom,Mension from Stagiaires ,Mentions where note >=
NoteMaximal and note <= NoteMinimal;
```

Résultat d'exécution

	Nom	Prenom	Mension
1	Madani	Ismail	Non validé
2	Wahabi	Youssef	Passable
3	Wahabi	Imaran	Assez bien
4	Andaloussi	Yasimina	Assez bien
5	Madani	Mouhssine	Bien
6	Wahabi	Imaran	Bien
7	Madani	Mouhssine	Trés bien

Jointure Syntaxe ANSI

Produit cartésien

```
SELECT liste_colonne FROM nomtable CROSS JOIN nomtable [
.... ]
```

Jointure

```
SELECT liste_colonne FROM nomtable INNER JOIN nomtable ON
```

Jointure externe

```
SELECT liste_colonne FROM nomtable {LEFT|RIGHT|FULL}OUTER
JOIN nomtable
ON nomtable.nomcolonne opérateur nomtable.nomcolonne [...]
```

Partie 2.3.0 : Optimisation

Chapitre 2.3.1 : Création des vues

Une **vue** « VIEW » est un SELECT qui est transformé en une table virtuelle. C'est tout simplement une requête SELECT que nous stockons dans notre base de donnée sous forme d'une table virtuelle et que nous allons pouvoir appeler comme si c'était une table.

Donc, c'est un outil très intéressant pour encapsuler un SELECT qui pourrait être un peu complexe avec beaucoup de jointure ou bien un SELECT qu'on exécute très régulièrement.

Requête 58 : Exemple de création d'un view

```
Create VIEW ListeStagiaires as Select Nom, id from Stagiaires ;
```

Résultat d'exécution

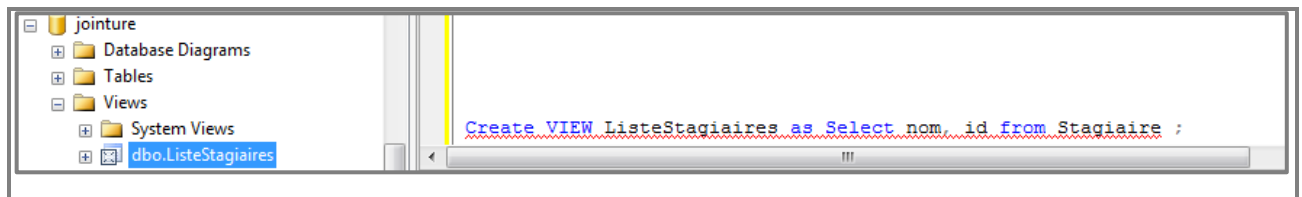


Figure 25 : Exécution sous SQL des vues

Les vues servent aussi à **simplifier** et à **sécuriser**:

Simplification :

Les structures des tables deviennent peu pratiques à manipuler.

Sécurité d'accès :

Il est possible de cacher des lignes et des colonnes.

Chapitre 3.1 : Index

L'objectif des index est de permettre un accès plus rapide à l'information (Select, Update, Insert).

Avant la création d'index :

Par exemple, le temps d'exécution de la requête suivante :

```
Select * from Stagiaire where nom= 'Ali' ;
```

Est : 196 ms

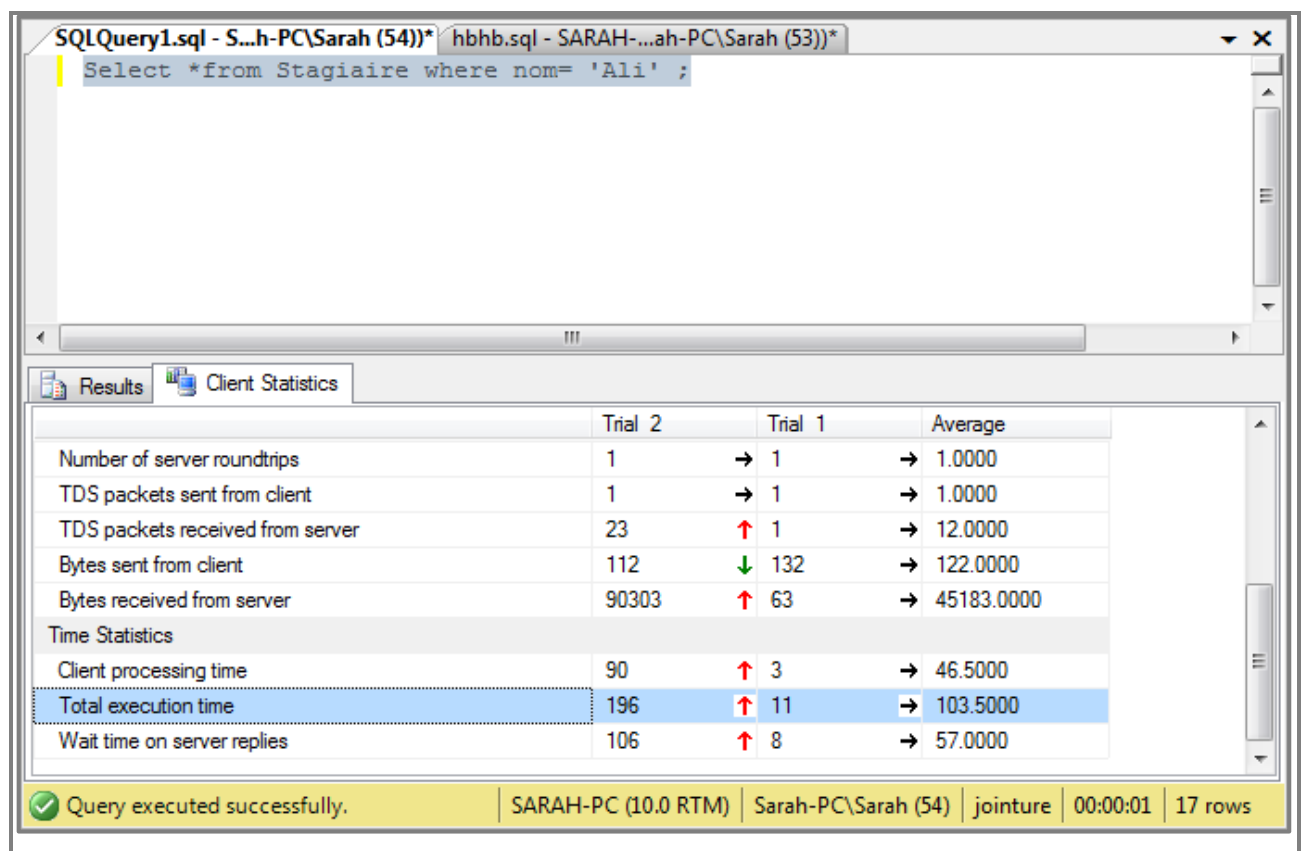


Figure 26 : Temps d'exécution de la requête

Création des index :

Requête 59 : Exemple de création d'un index

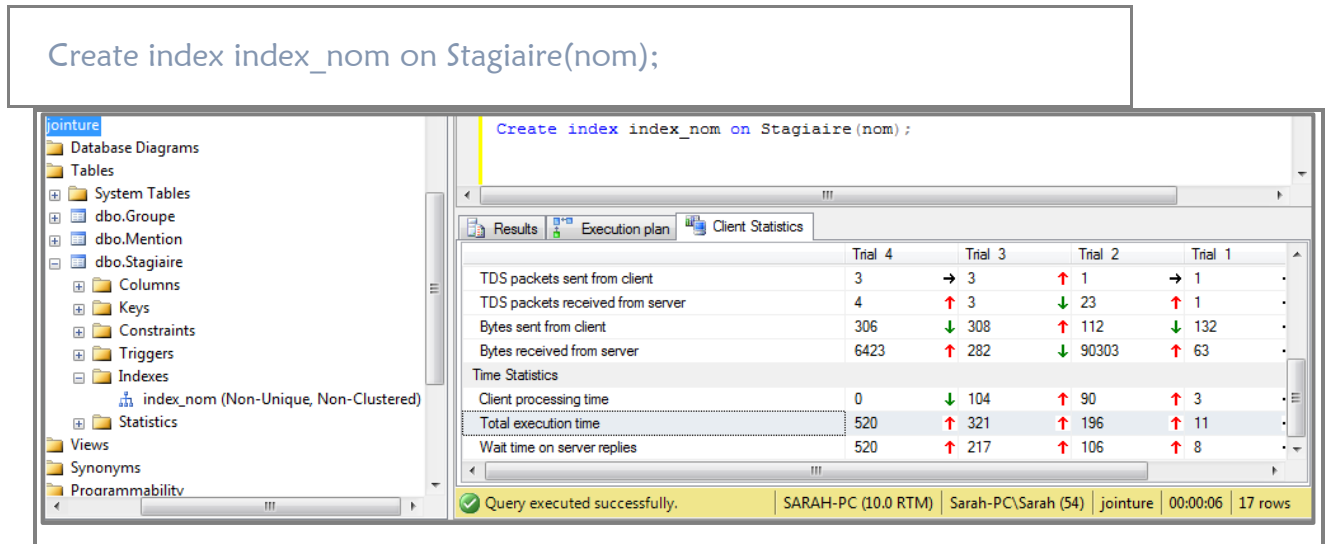


Figure 27 : Création d'index

Après la création de l'index

Le temps d'exécution est passé de 520 ms à 121 ms.

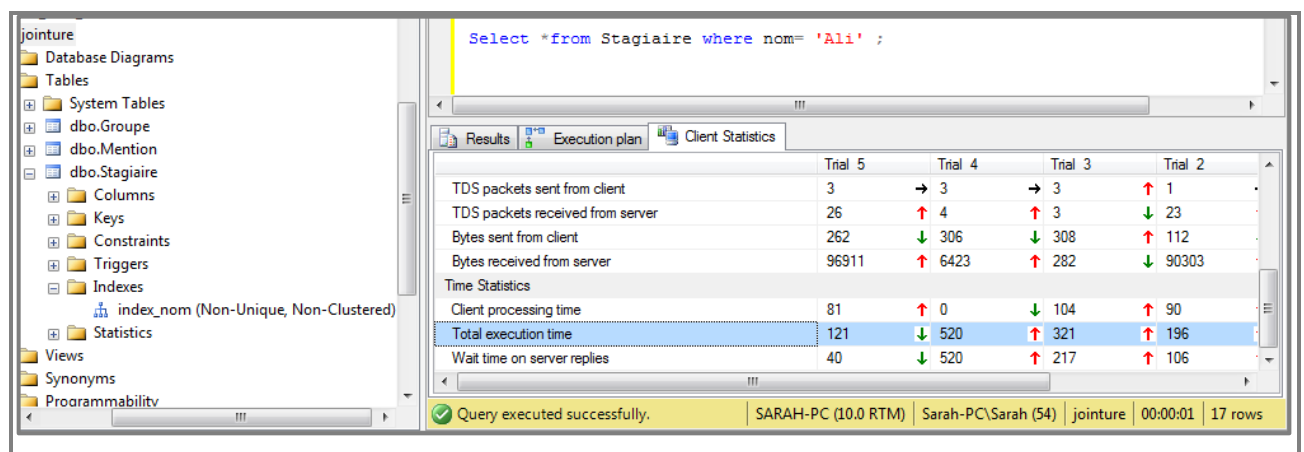


Figure 28 : Après la réaction de l'index

Chapitre 3.3 : Transaction

Le modèle le plus simple d'une transaction est celui du transfert d'un compte vers un autre.

```
Update Client Set Compte1=Compte1-500 where id=1 ;
```

```
Update Client Set Compte2=Compte2+500 where id=1 ;
```

Le mécanisme transactionnel empêche une erreur en validant les opérations faites depuis le début de la transaction si une panne survient au cours de cette même transaction.

Exemple :

```
Begin Transaction Modification  
save Transaction p1  
Update stg set Nom='A' where id=1;  
save Transaction p2  
Update stg set Nom='B' where id=1;  
Roll back Transaction p2  
Commit Transaction Modification
```

Remarque :

Seules les instructions LMD (Select, Insert, Update, Delete) sont prises en compte dans une transaction mais (Create, Alter, Drop, Grant) ne sont pas prises en compte.

Chapitre 3.4 : Créer des packages sur le SGBD.

- Description du formalisme à respecter.
- Présentation des packages standard.
- Programmation des packages.

Comprendre l'intérêt des packages.

- Utilité des packages dans le déploiement d'applications.

Connaître le modèle Événementiel

Limites des contraintes d'intégrité et utilité des triggers.

- Événements sur les objets de la base de données.

Section 3 – Programmation SQL