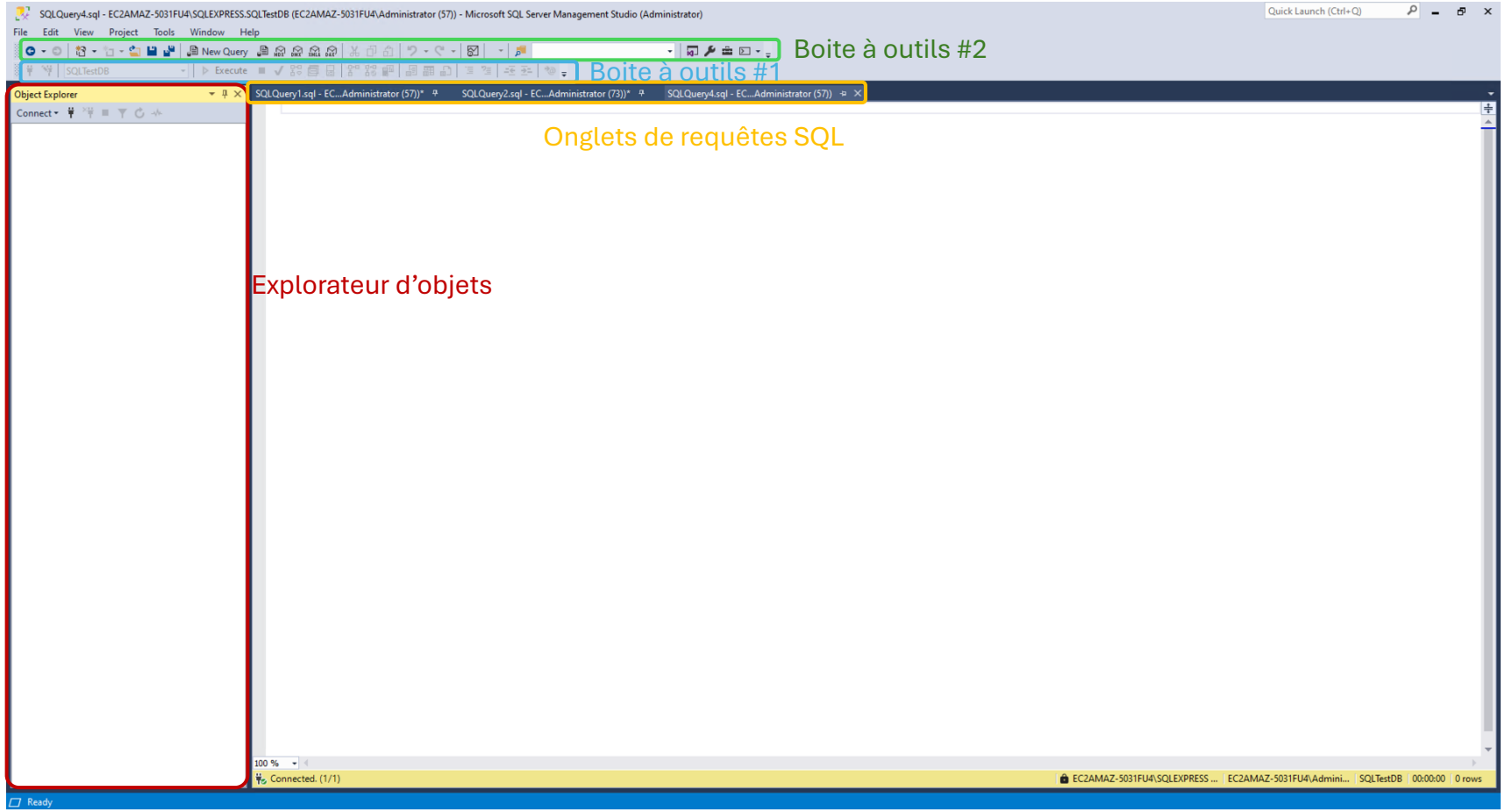
Microsoft®
SQL Server®

Cours TD2-3-4 Microsoft SQL Server

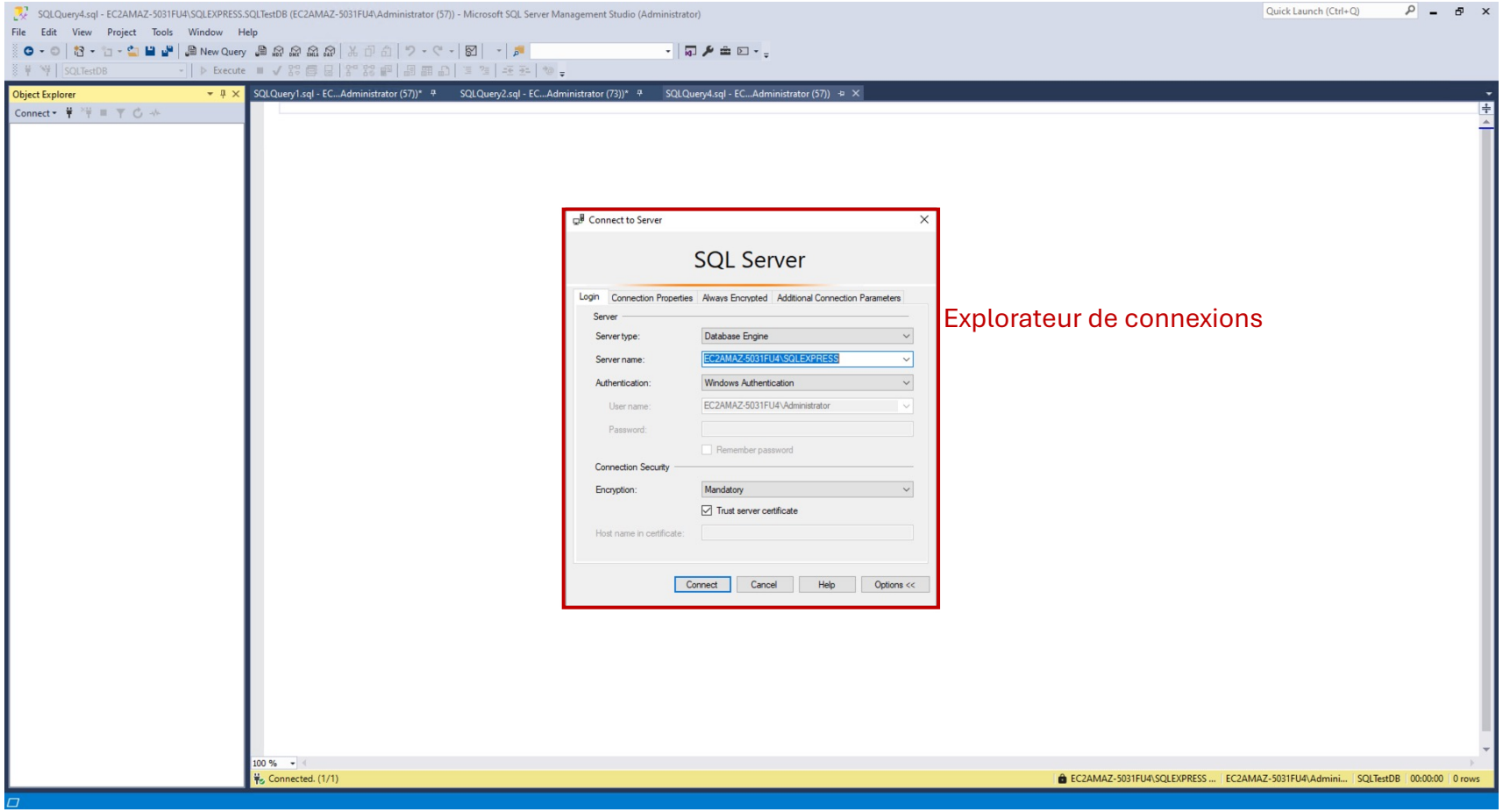
Mohamed BA / EPF-AFRICA

SOMMAIRE

I. SQL SERVER MANAGEMENT STUDIO

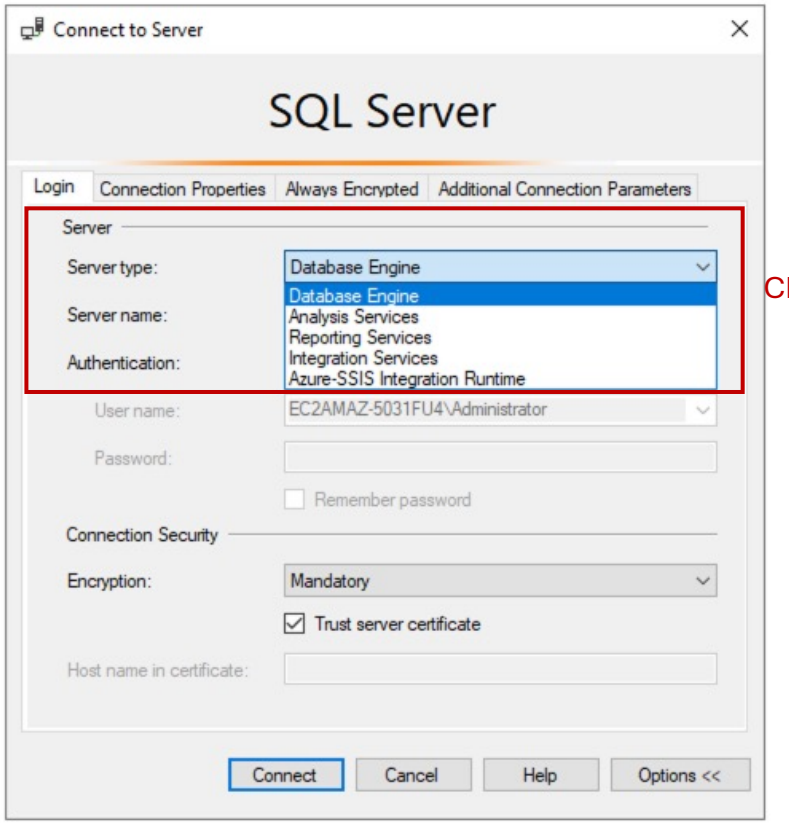


I. SQL SERVER MANAGEMENT STUDIO



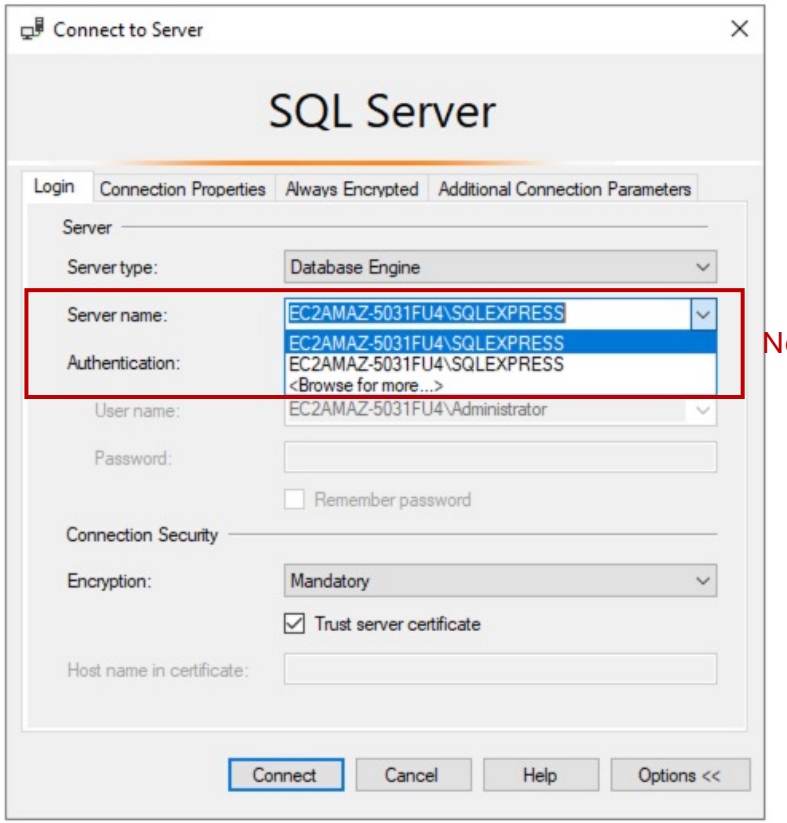
Explorateur de connexions

I. SQL SERVER MANAGEMENT STUDIO



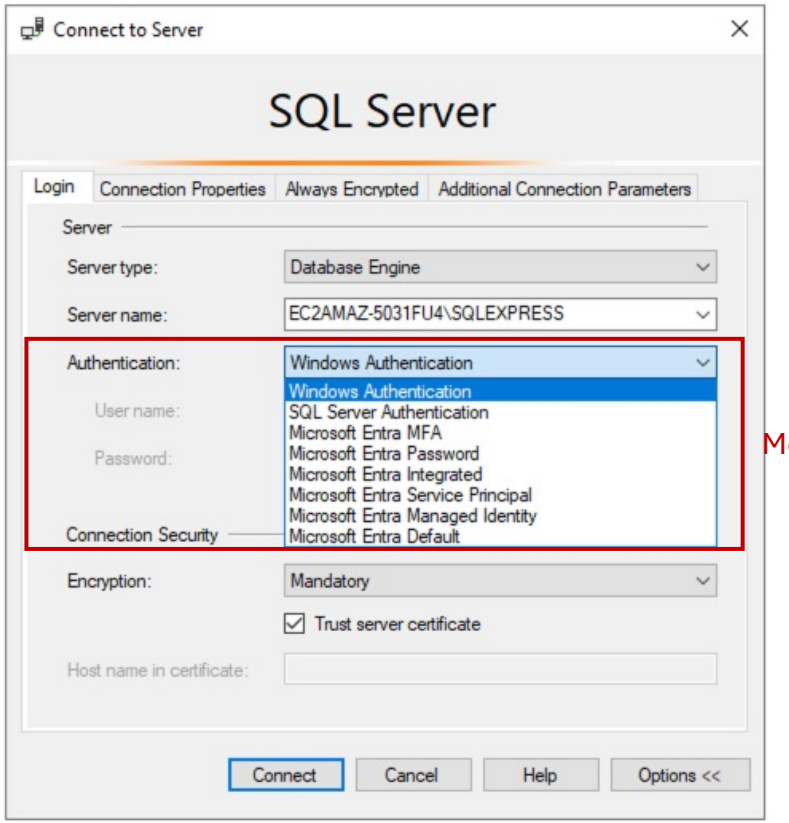
Choix du type de serveur

I. SQL SERVER MANAGEMENT STUDIO



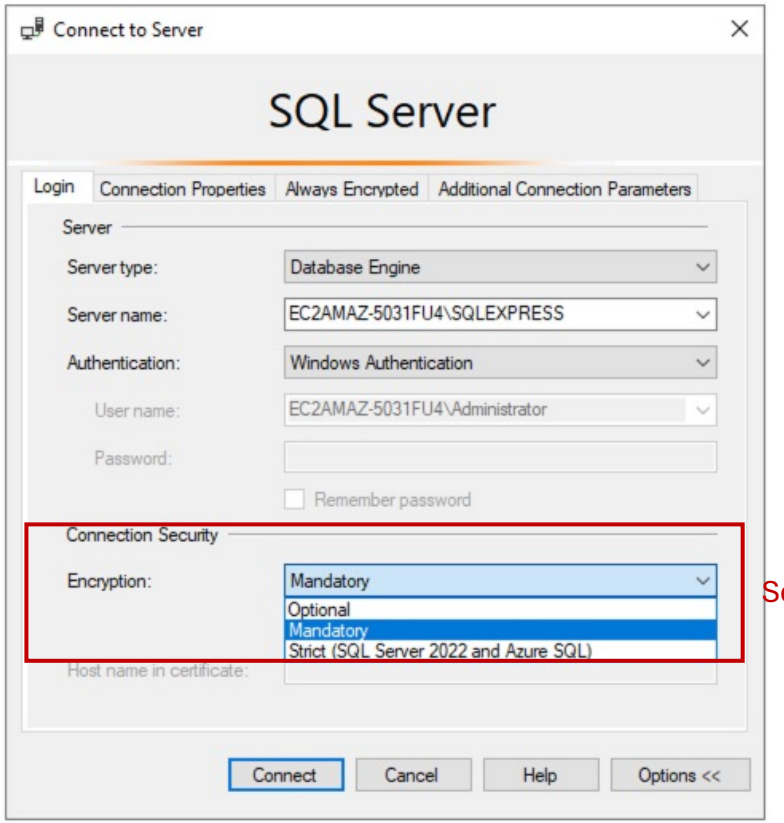
Nom de serveur

I. SQL SERVER MANAGEMENT STUDIO



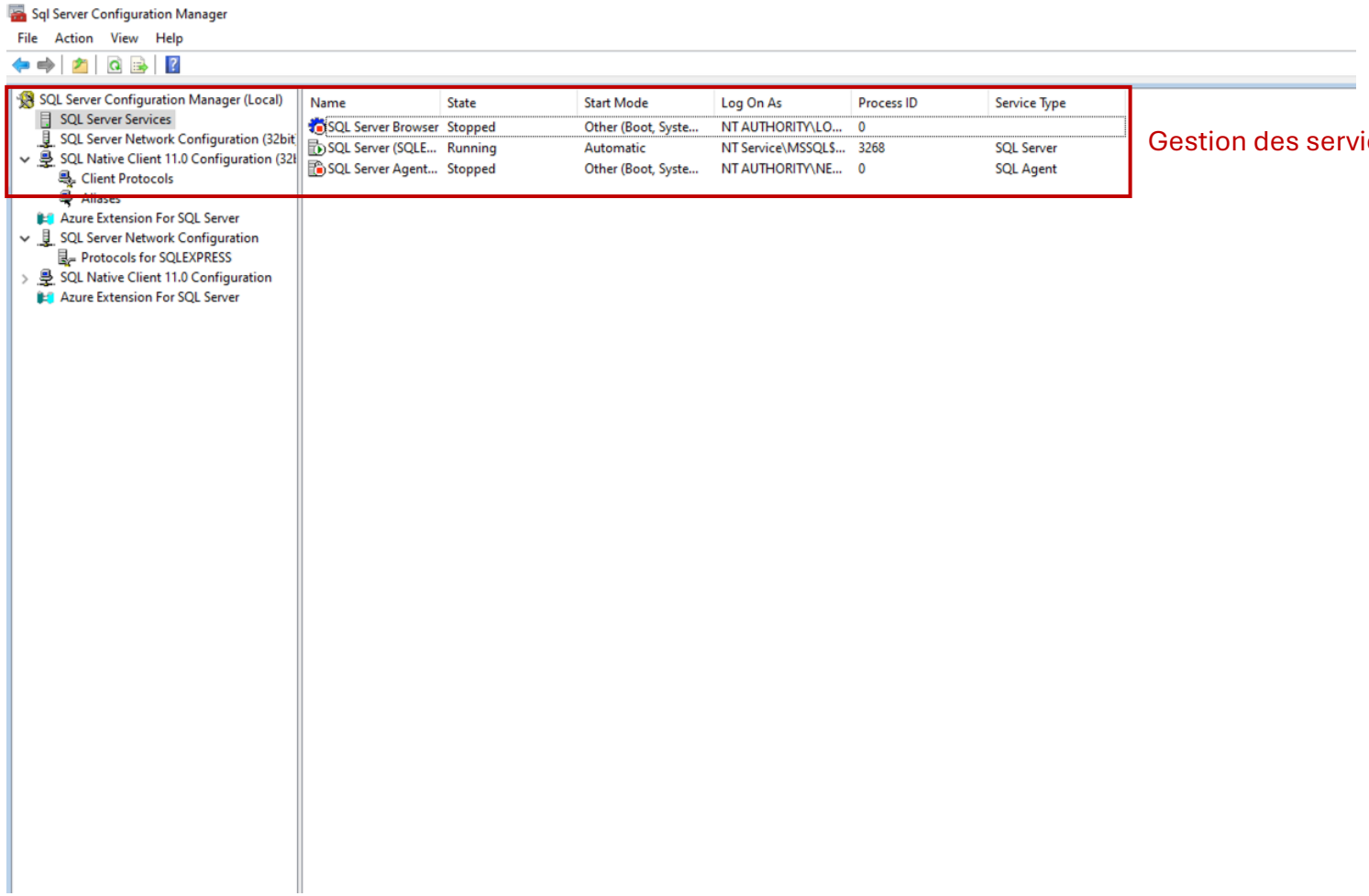
Mode d'authentification

I. SQL SERVER MANAGEMENT STUDIO



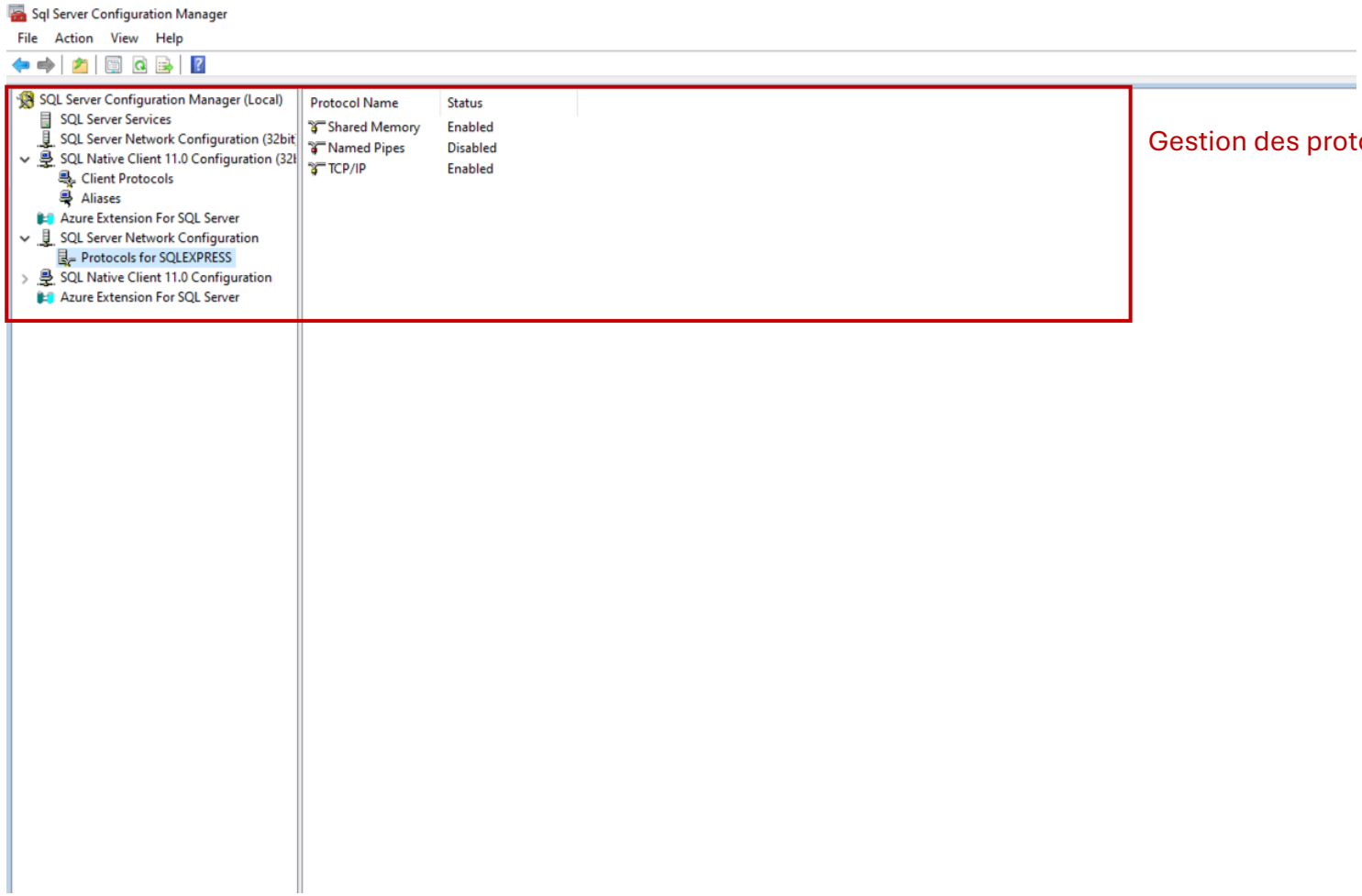
Sécurité & Encryption

II. SQL SERVER CONFIGURATION MANAGER



Gestion des services SQL Server

II. SQL SERVER CONFIGURATION MANAGER



Gestion des protocoles de connexion SQL Server

III. MODES DE CONNEXION

Durant l'installation, nous avons sélectionné un mode d'authentification pour le moteur de base de données. Deux modes sont possibles : le mode d'authentification Windows et le mode mixte.

❖ Authentification Windows

L'authentification Windows est le mode d'authentification par défaut. Elle est plus sécurisée que l'authentification SQL Server. L'authentification Windows utilise la Technology LAN Manager (NTLM) ou le protocole de sécurité Kerberos, met en œuvre des stratégies de mot de passe portant sur la validation de la complexité des mots de passe forts et prend en charge le verrouillage des comptes et l'expiration des mots de passe.
L'authentification Windows crée le compte « sa » pour l'authentification SQL Server, mais il est désactivé.

Étant donné que le compte sa est connu et qu'il est souvent la cible d'utilisateurs malveillants, n'activez pas le compte sa à moins que votre application n'en ait besoin.
N'attribuez jamais de mot de passe vide ou faible au compte sa.

III. MODES DE CONNEXION

❖ Authentification Mixte (Windows/SQL Server)

Si vous sélectionnez le mode d'authentification mixte (mode SQL Server et Authentification Windows) au cours de l'installation, vous devez fournir, puis confirmer, un mot de passe fort pour le compte d'administrateur système SQL Server intégré appelé sa. Le compte sa se connecte à l'aide de l'authentification SQL Server.

❖ Authentification SQL Server

Les connexions sont créées dans SQL Server et ne sont pas basées sur des comptes d'utilisateur Windows. Le nom d'utilisateur et le mot de passe sont créés à l'aide de SQL Server et stockés dans SQL Server. Les utilisateurs qui recourent à l'authentification SQL Server doivent fournir leurs informations d'identification (nom de connexion et mot de passe) chaque fois qu'ils se connectent. Lorsque vous utilisez l'authentification SQL Server, vous devez définir des mots de passe forts pour tous les comptes SQL Server.

III. MODES DE CONNEXION

❖ Inconvénients de l'authentification SQL Server

- Si un utilisateur est un utilisateur de domaine Windows qui dispose d'une connexion et d'un mot de passe pour Windows, il doit toujours fournir un autre nom de connexion et un autre mot de passe (SQL Server) pour se connecter. Pour de nombreux utilisateurs, il est difficile de gérer plusieurs noms et plusieurs mots de passe. Devoir fournir des informations d'identification SQL Server à chaque connexion de l'utilisateur à la base de données peut s'avérer fastidieux.
- L'authentification SQL Server ne peut pas utiliser le protocole de sécurité Kerberos.
- Windows offre des stratégies de mot de passe supplémentaires qui ne sont pas disponibles pour les connexions SQL Server.
- Le mot de passe de connexion chiffré de l'authentification SQL Server doit être passé sur le réseau au moment de la connexion. Certaines applications qui se connectent automatiquement conservent le mot de passe au niveau du client. Il s'agit de points d'attaque supplémentaires.

III. MODES DE CONNEXION

❖ Avantages de l'authentification SQL Server

- Permet à SQL Server de prendre en charge des applications anciennes et des applications tierces qui requièrent l'authentification SQL Server.
- Permet à SQL Server de prendre en charge des environnements contenant des systèmes d'exploitation mixtes, où tous les utilisateurs ne sont pas authentifiés par un domaine Windows.
- Permet aux utilisateurs de se connecter à partir de domaines inconnus ou non approuvés. Par exemple, une application où les clients établis se connectent via des connexions SQL Server assignées pour obtenir l'état de leurs commandes.
- Permet à SQL Server de prendre en charge les applications web où les utilisateurs créent leur propre identité.
- Permet aux développeurs de logiciels de distribuer leurs applications à l'aide d'une hiérarchie d'autorisations complexe basée sur des connexions SQL Server connues et prédéfinies.

III. MODES DE CONNEXION

❖ TP2 : Gestion des utilisateurs & des connexions

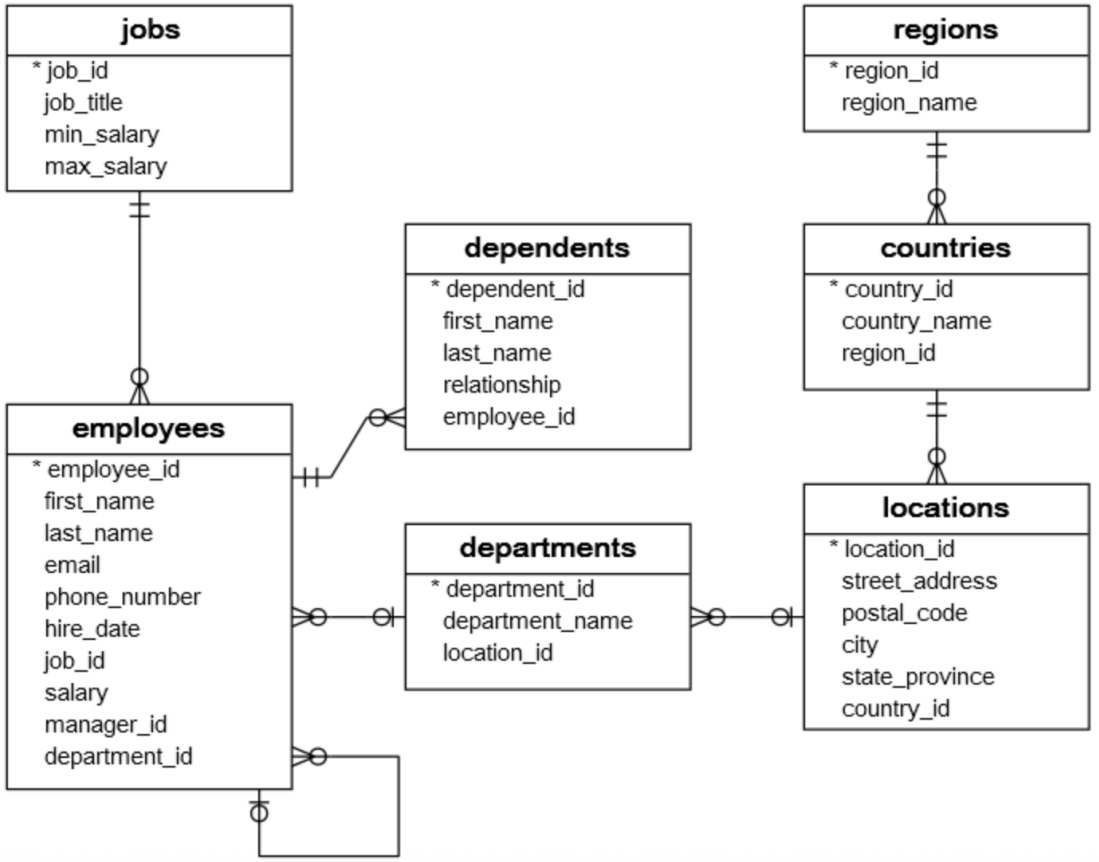
- Créer un utilisateur SQL Server.
- Se connecter avec le compte d'utilisateur créé.
- Créer un utilisateur Windows.
- Se connecter avec le compte d'utilisateur créé.
- Configurer la connexion via TCP/IP 1433.
- Créer une règle de pare-feu.
- Se connecter au serveur SQL.

IV. GESTION DES BASES DE DONNEES

Sous-langage SQL	Commandes	Exemples
Data Definition Language (DDL)	Commandes de définition du schéma de la base de données : création, modification et suppression de tables de la base de données ; définition des clés primaires, des clés étrangères et des contraintes	CREATE TABLE, DROP TABLE
Data Manipulation Language (DML)	Commandes de manipulation des données : modification, insertion et suppression d'ensembles de données	INSERT, UPDATE
Data Query Language (DQL)	Commandes d'interrogation et de préparation des données	SELECT
Data Control Language (DCL)	Commandes de gestion des autorisations	GRANT, REVOKE
Transaction Control Language (TCL)	Commandes de contrôle des transactions	COMMIT, ROLLBACK

IV. GESTION DES BASES DE DONNEES

Le diagramme de base de données ci-après illustre la base de données ***hrdb*** sur les indices de ressources humaines.



IV. GESTION DES BASES DE DONNEES

❖ TP3 : Création de bases de données

- Créer une base de données (avec assistant).
- Créer la base de données ***hrdb*** (T-SQL).

Script T-SQL :

```
CREATE DATABASE hrdb
CONTAINMENT = PARTIAL
ON  PRIMARY
( NAME = N'hrdb', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.SQLEXPRESS\MSSQL\DATA\hrdb.mdf',
SIZE = 5120KB , FILEGROWTH = 1024KB )
LOG ON
( NAME = N'hrdb_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.SQLEXPRESS\MSSQL\DATA\hrdb_log.ldf',
SIZE = 2048KB , FILEGROWTH = 10%)
GO
```

IV. GESTION DES BASES DE DONNEES

A. Sauvegarde & Restauration

TP4 : Sauvegarder & Restaurer une base de données

❖ Sauvegarde

Script T-SQL :

```
USE hrdb;
GO
BACKUP DATABASE hrdb
TO DISK = 'path_to_location\DB_backup_filename.bak'
WITH FORMAT,
MEDIANAME = 'SQLServerBackups',
NAME = 'Full Backup_of_DB_name';
GO
-----------------------------------------------------------------
BACKUP DATABASE hrdb
TO DISK = 'path_to_location\DB_backup_filename.bak '
WITH INIT -- écrase toutes sauvegardes existantes / -- STATS / -- COMPRESSION
```

IV. GESTION DES BASES DE DONNEES

A. Sauvegarde & Restauration
TP5 : Restaurer une sauvegarde de base de données

❖ Restauration

Script T-SQL :

```
RESTORE DATABASE hrdb
FROM DISK = 'path_to_location\DB_backup_filename.bak'
WITH RECOVERY
```

Restauration vers une autre base de données

```
RESTORE DATABASE hrdbCopy
FROM DISK = ‘C:\tmp\hrdb.bak'
WITH MOVE 'hrdb’ TO 'C:\Data\hrdbCopy.mdf',
MOVE ‘hrdb_Log' TO 'C:\Log\hrdbDBCopy.ldf'
```

V. GESTION DES DROITS & RÔLES

❖ TP6 : Création d'utilisateurs et octroi de droits & rôles

- Créer une connexion d'utilisateur SQL Server.

Script T-SQL : CREATE LOGIN userlogin WITH PASSWORD = 'mot_de_passe'; ou
CREATE LOGIN [<domainName>\<loginName>] FROM WINDOWS;

- Créer un compte utilisateur de base de données pour la base de données *hrdb*

Script T-SQL : *USE hrdb;*
GO
CREATE USER userlogin FOR LOGIN userlogin; ou
CREATE USER userlogin WITH PASSWORD = 'mot_de_passe';

- Accorder les droits requis sur la base de données créée au point précèdent.

- Se connecter avec les comptes utilisateurs créés.

V. GESTION DES DROITS & RÔLES

- Créer un rôle depuis la base de données *TestContainedDB*

Script T-SQL : USE hrdb;
GO
CREATE ROLE rolehrdb;
GRANT CREATE TABLE, CREATE VIEW to rolehrdb;

- Révoquer (retirer) des droits a un rôle

Script T-SQL : USE hrdb;
GO
REVOKE SELECT ON SCHEMA :: Sales TO *rolehrdb;*

V. GESTION DES DROITS & RÔLES

- Créer un rôle sur la base base de données utilisée au point précèdent

Script T-SQL : USE hrdb;
```
GO
CREATE ROLE rolehrdb2;
```

- Ajouter des droits au rôle crée au point précèdent.

Script T-SQL : USE hrdb;
```
GO
GRANT CREATE TABLE, CREATE VIEW to rolehrdb2;
```

- Assigner le rôle à l'utilisateur précédemment créé.

Script T-SQL : USE hrdb;
```
GO
ALTER ROLE rolehrdb2 ADD MEMBER user;
```

VI. TYPES DE DONNEES

❖ Chaines de caractères

Data type	Description	Max char length	Storage
char(n)	Fixed-length non-Unicode character data (n must be between 1 and 8000)	8,000	n bytes (uses one byte for each character)
varchar(n)	Variable-length non-Unicode character data (n must be between 1 and 8000)	8,000	n bytes + 2 bytes
varchar(max)	Variable-length non-Unicode character data		up to 2 GB
nchar(n)	Fixed-length Unicode character data (n must be between 1 and 4000)	4,000	2 * n bytes (uses two bytes for each character)
nvarchar(n)	Variable-length Unicode character data (n must be between 1 and 4000)	4,000	2 * n bytes + 2 bytes (uses two bytes for each character)
nvarchar(max)	Variable-length Unicode character data		up to 2 GB
binary(n)	Fixed-length binary data (n must be between 1 and 8000)	8,000	n bytes
varbinary(n)	Variable-length binary data (n must be between 1 and 8000)	8,000	actual length of data entered + 2 bytes
varbinary(max)	Variable-length binary data	2GB	

VI. TYPES DE DONNEES

❖ Données numériques

Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

VI. TYPES DE DONNEES

❖ Données de type dates

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

VI. TYPES DE DONNEES

❖ Autres types de données

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

VII. LES CONTRAINTES

- *NOT NULL* - Garantit qu'une colonne ne peut pas contenir de valeur NULL.
- *UNIQUE* - Garantit que toutes les valeurs d'une colonne sont différentes.
- *PRIMARY KEY* - Combinaison de NOT NULL et d'UNIQUE. Identifie de manière unique chaque ligne d'une table.
- *FOREIGN KEY* - Empêche les actions qui détruiraient les liens entre les tables.
- *CHECK* - Garantit que les valeurs d'une colonne satisfont à une condition spécifique.
- *DEFAULT* - Définit une valeur par défaut pour une colonne si aucune valeur n'est spécifiée.
- *CREATE INDEX* - Permet de créer et de récupérer très rapidement des données de la base de données.

VIII. LES TABLES

❖ TP7 : Création de tables

- Créer la table *regions*

```
USE hrdb
CREATE TABLE regions (
region_id INT IDENTITY(1,1) PRIMARY KEY,
region_name VARCHAR (25) DEFAULT NULL
);
```

VIII. LES TABLES

- Créer la table *countries*

```
USE hrdb
CREATE TABLE countries (
country_id CHAR (2) PRIMARY KEY,
country_name VARCHAR (40) DEFAULT NULL,
region_id INT NOT NULL,
FOREIGN KEY (region_id) REFERENCES regions (region_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

VIII. LES TABLES

- Créer la table *locations*

```
USE hrdb
CREATE TABLE locations (
location_id INT IDENTITY(1,1) PRIMARY KEY,
street_address VARCHAR (40) DEFAULT NULL,
postal_code VARCHAR (12) DEFAULT NULL,
city VARCHAR (30) NOT NULL,
state_province VARCHAR (25) DEFAULT NULL,
country_id CHAR (2) NOT NULL,
FOREIGN KEY (country_id) REFERENCES countries (country_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

VIII. LES TABLES

- Créer la table *jobs*

```
USE hrdb
CREATE TABLE jobs (
job_id INT IDENTITY(1,1) PRIMARY KEY,
job_title VARCHAR (35) NOT NULL,
min_salary DECIMAL (8, 2) DEFAULT NULL,
max_salary DECIMAL (8, 2) DEFAULT NULL
);
```

VIII. LES TABLES

- Créer la table *departments*

```
USE hrdb
CREATE TABLE departments (
department_id INT IDENTITY(1,1) PRIMARY KEY,
department_name VARCHAR (30) NOT NULL,
location_id INT DEFAULT NULL,
FOREIGN KEY (location_id) REFERENCES locations (location_id) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

VIII. LES TABLES

- Créer la table *employees*

```
USE hrdb
CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
first_name VARCHAR (20) DEFAULT NULL,
last_name VARCHAR (25) NOT NULL,
email VARCHAR (100) NOT NULL,
phone_number VARCHAR (20) DEFAULT NULL,
hire_date DATE NOT NULL,
job_id INT NOT NULL,
salary DECIMAL (8, 2) NOT NULL,
manager_id INT DEFAULT NULL,
department_id INT DEFAULT NULL,
FOREIGN KEY (job_id) REFERENCES jobs (job_id) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (department_id) REFERENCES departments (department_id) ON DELETE CASCADE ON UPDATE
CASCADE,
FOREIGN KEY (manager_id) REFERENCES employees (employee_id)
);
```

VIII. LES TABLES

- Créer la table *dependents*

```
CREATE TABLE dependents (
dependent_id INT IDENTITY(1,1) PRIMARY KEY,
first_name VARCHAR (50) NOT NULL,
last_name VARCHAR (50) NOT NULL,
relationship VARCHAR (25) NOT NULL,
employee_id INT NOT NULL,
FOREIGN KEY (employee_id) REFERENCES employees (employee_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

VIII. LES TABLES

- Créer une copie de la table *jobs*

```
CREATE TABLE jobs_copy AS
  SELECT job_id, job_title, max_salary
  FROM jobs;

CREATE TABLE jobs_copy2 AS
  SELECT job_id, job_title, min_salary
  FROM jobs
WHERE max_salary=1000;
```

VIII. LES TABLES

- Supprimer une table

```
DROP TABLE jobs_copy2;
```

- Modifier une table

```
ALTER TABLE jobs_copy
ADD min_salary DECIMAL (8, 2) DEFAULT NULL
ADD category INT NOT NULL;

ALTER TABLE jobs_copy
DROP COLUMN category;
```

VIII. LES TABLES

- Peupler les tables regions & countries

```
/*Data for the table regions */
SET IDENTITY_INSERT regions ON;

INSERT INTO regions(region_id,region_name) VALUES (1,'Europe');
INSERT INTO regions(region_id,region_name) VALUES (2,'Amerique');
INSERT INTO regions(region_id,region_name) VALUES (3,'Asie');
INSERT INTO regions(region_id,region_name) VALUES (4,'Moyen Orient et Afrique');
SET IDENTITY_INSERT regions OFF;

/*Data for the table countries */
INSERT INTO countries(country_id,country_name,region_id) VALUES ('AR','Argentine',2);
INSERT INTO countries(country_id,country_name,region_id) VALUES ('AU','Australie',3);
INSERT INTO countries(country_id,country_name,region_id) VALUES ('BE','Belgium',1);
INSERT INTO countries(country_id,country_name,region_id) VALUES ('SN','Senegal',4);
INSERT INTO countries(country_id,country_name,region_id) VALUES ('CA','Canada',2);
```

VIII. LES TABLES

- Peupler la table locations

```
/*Data for the table locations */
SET IDENTITY_INSERT locations ON;
INSERT INTO locations(location_id,street_address,postal_code,city,state_province,country_id) VALUES
(1400,'Dakar Liberte 6','17700','Dakar','Dakar','SN');
INSERT INTO locations(location_id,street_address,postal_code,city,state_province,country_id) VALUES
(1500,'Dakar Plateau','17700','Dakar','Dakar','SN');
INSERT INTO locations(location_id,street_address,postal_code,city,state_province,country_id) VALUES
(1700,'2004 Charade Rd','98199','Buenos Aires','Buenos Aires','AR');
INSERT INTO locations(location_id,street_address,postal_code,city,state_province,country_id) VALUES
(1800,'147 Spadina Ave','M5V 2L7','Toronto','Ontario','CA');
INSERT INTO locations(location_id,street_address,postal_code,city,state_province,country_id) VALUES
(2400,'8204 Arthur St',NULL,'Bruxelles',NULL,'BE');
SET IDENTITY_INSERT locations OFF;
```

VIII. LES TABLES

- Peupler la table jobs

```
/*Data for the table jobs */
SET IDENTITY_INSERT jobs ON;
INSERT INTO jobs(job_id,job_title,min_salary,max_salary) VALUES (1,'Comptable',4200.00,9000.00);
INSERT INTO jobs(job_id,job_title,min_salary,max_salary) VALUES (2,'Gestionnaire de
compte',8200.00,16000.00);
INSERT INTO jobs(job_id,job_title,min_salary,max_salary) VALUES (3,'Assistant
Administratif',3000.00,6000.00);
INSERT INTO jobs(job_id,job_title,min_salary,max_salary) VALUES
(4,'Informaticien',20000.00,40000.00);
SET IDENTITY_INSERT jobs OFF;
```

VIII. LES TABLES

- Peupler la table departments

```
/*Data for the table departments */
SET IDENTITY_INSERT departments ON;
INSERT INTO departments(department_id,department_name,location_id) VALUES (1,'Administration',1700);
INSERT INTO departments(department_id,department_name,location_id) VALUES (2,'Marketing',1800);
INSERT INTO departments(department_id,department_name,location_id) VALUES (3,'Achats',1700);
INSERT INTO departments(department_id,department_name,location_id) VALUES (4,'Ressources Humaines',2400);
INSERT INTO departments(department_id,department_name,location_id) VALUES (5,'Logistique',1500);
INSERT INTO departments(department_id,department_name,location_id) VALUES (6,'IT',1400);
INSERT INTO departments(department_id,department_name,location_id) VALUES (9,'Comptabilite',1700);
INSERT INTO departments(department_id,department_name,location_id) VALUES (10,'Finance',1700);
INSERT INTO departments(department_id,department_name,location_id) VALUES (11,'Conformite',1700);
SET IDENTITY_INSERT departments OFF;
```

VIII. LES TABLES

- Peupler la table employees

```
/*Data for the table employees */
SET IDENTITY_INSERT employees ON;
INSERT INTO
employees(employee_id,first_name,last_name,email,phone_number,hire_date,job_id,salary,manager_id,department_id)
VALUES (100,'Paul','NDIAYE','paul.ndiaye@test.com','774505050','1987-06-17',4,24000.00,NULL,9);
INSERT INTO
employees(employee_id,first_name,last_name,email,phone_number,hire_date,job_id,salary,manager_id,department_id)
VALUES (101,'Oceanie','DIOUF','oceanie.diouf@test.com','764504030','1989-09-21',1,17000.00,100,9);
INSERT INTO
employees(employee_id,first_name,last_name,email,phone_number,hire_date,job_id,salary,manager_id,department_id)
VALUES (102,mohamed’,BA’,’mohamed.ba@test.com','',’1983-01-13',2,19000.00,100,9);
INSERT INTO
employees(employee_id,first_name,last_name,email,phone_number,hire_date,job_id,salary,manager_id,department_id)
VALUES (103,'Marietou','SECK','marietou.seck@test.com','','1993-06-13',3,19000.00,100,9);
SET IDENTITY_INSERT employees OFF;
```

VIII. LES TABLES

- Peupler la table dependents

```
/*Data for the table dependents */
SET IDENTITY_INSERT dependents ON;
INSERT INTO dependents(dependent_id,first_name,last_name,relationship,employee_id) VALUES
(1,'Martin','NDIAYE','Child',100);
INSERT INTO dependents(dependent_id,first_name,last_name,relationship,employee_id) VALUES
(2,'Corine','NDIAYE','Child',100);
INSERT INTO dependents(dependent_id,first_name,last_name,relationship,employee_id) VALUES
(3,'Eve','DIOUF','Child',101);
INSERT INTO dependents(dependent_id,first_name,last_name,relationship,employee_id) VALUES
(4,'Doudou','NDIAYE','Child',103);
INSERT INTO dependents(dependent_id,first_name,last_name,relationship,employee_id) VALUES
(5,'Fatima','BA','Child',102);
INSERT INTO dependents(dependent_id,first_name,last_name,relationship,employee_id) VALUES
(6,'Mia','BA','Child',102);
SET IDENTITY_INSERT dependents OFF;
```

VIII. LES TABLES

❖ Ordre d’exécution d’une requête SQL

1. FROM/JOIN
Les clauses FROM et/ou JOIN sont exécutées en premier pour déterminer les données d’intérêt.

2. WHERE
La clause WHERE est exécutée pour filtrer les enregistrements qui ne répondent pas aux contraintes.

3. GROUP BY
La clause GROUP BY est exécutée pour regrouper les données en fonction des valeurs d'une ou plusieurs colonnes.

4. HAVING
La clause HAVING est exécutée pour supprimer les enregistrements groupés créés qui ne répondent pas aux contraintes.

VIII. LES TABLES

Ordre d’exécution d’une requête SQL

5. SELECT
La clause SELECT est exécutée pour dériver toutes les colonnes et expressions souhaitées.

6. ORDER BY
La clause ORDER BY est exécutée pour trier les valeurs dérivées par ordre croissant ou décroissant.

7. LIMIT/OFFSET
Enfin, les clauses LIMIT et/ou OFFSET sont exécutées pour conserver ou ignorer un nombre spécifié de lignes.

VIII. LES TABLES

- Créer une table a partir des attributs d'une autre table

```
SELECT dependent_id, first_name, last_name
INTO dependents_copy
FROM dependents
WHERE dependent_id BETWEEN 2 AND 4;
```

- Supprimer le contenu d'une table sans la supprimer (afin d'en garder la structure)

```
TRUNCATE TABLE dependents_copy;
```

VIII. LES TABLES

❖ Les Index

Les index permettent de récupérer les données de la base de données plus rapidement. Les utilisateurs ne peuvent pas les voir ; ils servent uniquement à accélérer les recherches et les requêtes.

- Créer un index pour la table countries sur la colonne country_name

```
CREATE INDEX indexcountries ON countries (country_name)
```

- Supprimer un index

```
DROP INDEX countries.indexcountries
```

La mise à jour d'une table avec index prend plus de temps que celle d'une table sans index (car les index doivent également être mis à jour). Par conséquent, créez des index uniquement sur les colonnes qui feront l'objet de recherches fréquentes.

VIII. LES TABLES

❖ Les déclencheurs (Triggers)

Un trigger (déclencheur) est un objet de base de données qui exécute automatiquement un bloc de code en réponse à un événement (INSERT, UPDATE, DELETE) sur une table ou une vue.

AFTER triggers *(le plus courant)*
Se déclenchent après l'exécution d'un INSERT, UPDATE ou DELETE.
Utilisé pour valider ou enregistrer des actions une fois la modification réussie.

INSTEAD OF triggers
Se déclenche à la place de l'action INSERT, UPDATE ou DELETE.
Utile pour intercepter et modifier les opérations, souvent sur des vues.

DDL triggers *(moins courant)*
Se déclenche sur des événements DDL comme CREATE, ALTER, DROP, etc.
Utile pour auditer ou restreindre la modification de la structure de la base.

Un trigger est une procédure stockée SPECIALE car elle n'accepte pas de paramètres.

VIII. LES TABLES

❖ Les déclencheurs (Triggers)

- Créer une table logs

```
USE hrdb
CREATE TABLE logs (
TableModifiee VARCHAR (40) DEFAULT NULL,
DateModif DATE DEFAULT NULL,
TypeModif VARCHAR (40) DEFAULT NULL
);
```

- Créer un trigger pour les requêtes *INSERT* effectuées sur la table countries

```
CREATE TRIGGER trigger_countries
ON countries
AFTER INSERT
AS
BEGIN
   INSERT INTO logs (TableModifiee, DateModif, TypeModif)
   VALUES ('countries', GETDATE(), 'INSERT')
END
```

VIII. LES TABLES

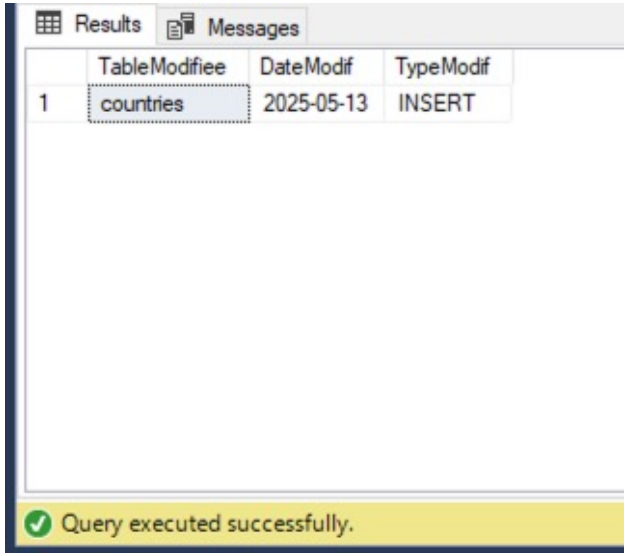
❖ Les déclencheurs (Triggers)

- Effectuer un INSERT dans la table countries

```
INSERT INTO countries(country_id,country_name,region_id) VALUES ('RW','Rwanda',4);
```

- Interroger la table logs

```
SELECT * FROM logs;
```



VIII. LES TABLES

❖ Les procédures stockées

Une procédure stockée est un code SQL préparé que vous pouvez enregistrer afin de pouvoir le réutiliser à l'infini. Si vous répétez une requête SQL à plusieurs reprises, enregistrez-la en tant que procédure stockée, puis appelez-la pour l'exécuter. A la différence des déclencheurs, vous pouvez transmettre des paramètres à une procédure stockée afin qu'elle agisse en fonction des valeurs de paramètre transmises.

- Créer une procédure

```
CREATE PROCEDURE GenerateCustomerInvoice
    @OrderId INT
AS
BEGIN
    -- Déclaration de la variable intermédiaire pour stocker le ClientID
    DECLARE @CustomerId INT;
    -- Déclaration de la variable intermédiaire pour stocker le Prix total
    DECLARE @Price DECIMAL;
    -- Déclaration de la variable pour stocker l'identifiant de la facture
    DECLARE @CreatedId INT;
```

VIII. LES TABLES

```
-- 1. Récupérer le CustomerId de la commande
SELECT @CustomerId = c.Id
FROM Customer c
INNER JOIN Order o
ON o.CustomerId = c.Id
WHERE o.Id = @OrderId;

-- 2. Récupérer le prix total de la commande
SELECT @Price = SUM(p.Price)
FROM Product p
INNER JOIN Order o
ON p.OrderId = o.Id
WHERE o.Id = @OrderId;


-- Vérification : si le client n'existe pas, on arrête l'exécution
IF @CustomerId IS NULL
BEGIN
    RETURN;
END
```

VIII. LES TABLES

```
-- 3. Insérer une nouvelle commande dans la table Commandes
    INSERT INTO Invoice(CustomerId, TotalPrice)
    VALUES (@CustomerId , @Price);
-- 4. Sauvegarde de l'identifiant de la facture créé
SET @CreatedId = SCOPE_IDENTITY();
-- 5. On retourne la facture créée
    SELECT * FROM Invoice WHERE Id = @CreatedId;

END;
```

- Exécuter la procédure

```
EXEC GenerateCustomerInvoice @OrderId = 1;
```

- Modifier la procédure

```
ALTER PROCEDURE GenerateCustomerInvoice
            -- Liste des paramètres
AS
BEGIN
            -- Contenu de la procédure
END;
```

IX. FONCTIONS & OPERATEURS

- Interroger la table regions

```
SELECT * FROM regions;
```

- Récupérer la liste des pays dont le country_id commence par la lettre ‘A’

```
SELECT * FROM countries WHERE country_id LIKE ‘A%’;
```

- Récupérer la masse salariale (depuis la table employees).

```
SELECT SUM(salary) AS masse_salariale FROM employees;
```

- Récupérer le salaire moyen (depuis la table employees).

```
SELECT AVG(salary) AS average_salary FROM employees;
```

IX. FONCTIONS & OPERATEURS

- Requête personnalisée #1

```
SELECT COUNT(*) AS nombre_employes FROM employees WHERE job_id=1;
```

- Requête personnalisée #2

```
SELECT first_name, last_name, salary FROM employees WHERE salary >5000 AND salary <20000
ORDER BY salary;
```

- Requête personnalisée #3

```
SELECT first_name, last_name, salary FROM employees WHERE salary BETWEEN 5000 AND 20000
ORDER BY salary;
```

- Requête personnalisée #4

```
SELECT ROUND(SUM(total_price), 2) AS total_sales1, ROUND(SUM(total_price), -2) AS total_sales2
FROM sales;
```

IX. FONCTIONS & OPERATEURS

- Requête personnalisée sur les produits #5

SELECT first_name, last_name FROM employees WHERE first_name LIKE '%ariet%';

- Requête personnalisée sur les ventes #6

```
SELECT first_name, last_name FROM dependents WHERE first_name LIKE '_o%';
```

- Requête personnalisée sur les ventes #7
- SELECT department_name, SUM(salary) total_salary *FROM* employees e *INNER JOIN* departments d *ON* d.department_id = e.department_id *GROUP BY* department_name *HAVING SUM*(salary) > 30000 *ORDER BY* total_salary;

IX. FONCTIONS & OPERATEURS

- Mettre a jour une table

```
SET IDENTITY_INSERT employees ON;
UPDATE employees SET first_name='Etan',last_name='BA',email='etan.ba@test.com' WHERE employee_id=102;
SET IDENTITY_INSERT employees OFF;
```

- Supprimer un enregistrement

```
SET IDENTITY_INSERT employees ON;
DELETE FROM products WHERE product_id=101;
SET IDENTITY_INSERT employees OFF;
```

IX. FONCTIONS & OPERATEURS

- CASE

```
SELECT
  first_name,
  last_name,
  CASE
    WHEN salary < 3000 THEN ‘Salaire faible'
    WHEN salary >= 3000 AND salary <= 5000 THEN ‘Salaire moyen'
    WHEN salary > 5000 THEN ‘Salaire eleve'
  END salary_ranking
FROM
  employees
ORDER BY
  first_name;
```

IX. FONCTIONS & OPERATEURS

- Fonctions basées sur les dates

```
SELECT sale_id, DATEDIFF(NOW(), sale_date) AS days_since_sale
FROM Sales;
GO
```

- Formatage de la date selon le patern dd-MM-yy

```
SELECT FORMAT (getdate(), 'dd-MM-yy') as date;
GO

SELECT FORMAT (getdate(), 'dd-MM-yyyy') as date;
GO

SELECT FORMAT (CURRENT_TIMESTAMP, 'dd-MM-yyyy') as date;
GO
```