ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Système D'information Décisionnel

Dr. Abdelali El Gourari

Doctor of Computer Science specializing in Artificial Intelligence and Embedded Systems at Cadi Ayyad University, Morocco. His primary research interests are applying artificial intelligence to adaptive education systems, particularly remote-controlled experiments.

mail

a.elgourari@emsi.ma
a.elgourari.ced@uca.ac.ma

Websites

ai-labs.uca.ma
elgourari.com

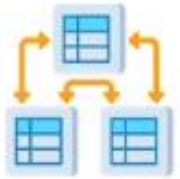
Présentez vous

Introduction Générale

Introduction aux bases de données

Base de données

Une base de données est un ensemble d'informations qui est organisé de manière à être facilement accessible. Elle est utilisée par les organisations comme méthode de stockage, de gestion de l'information.



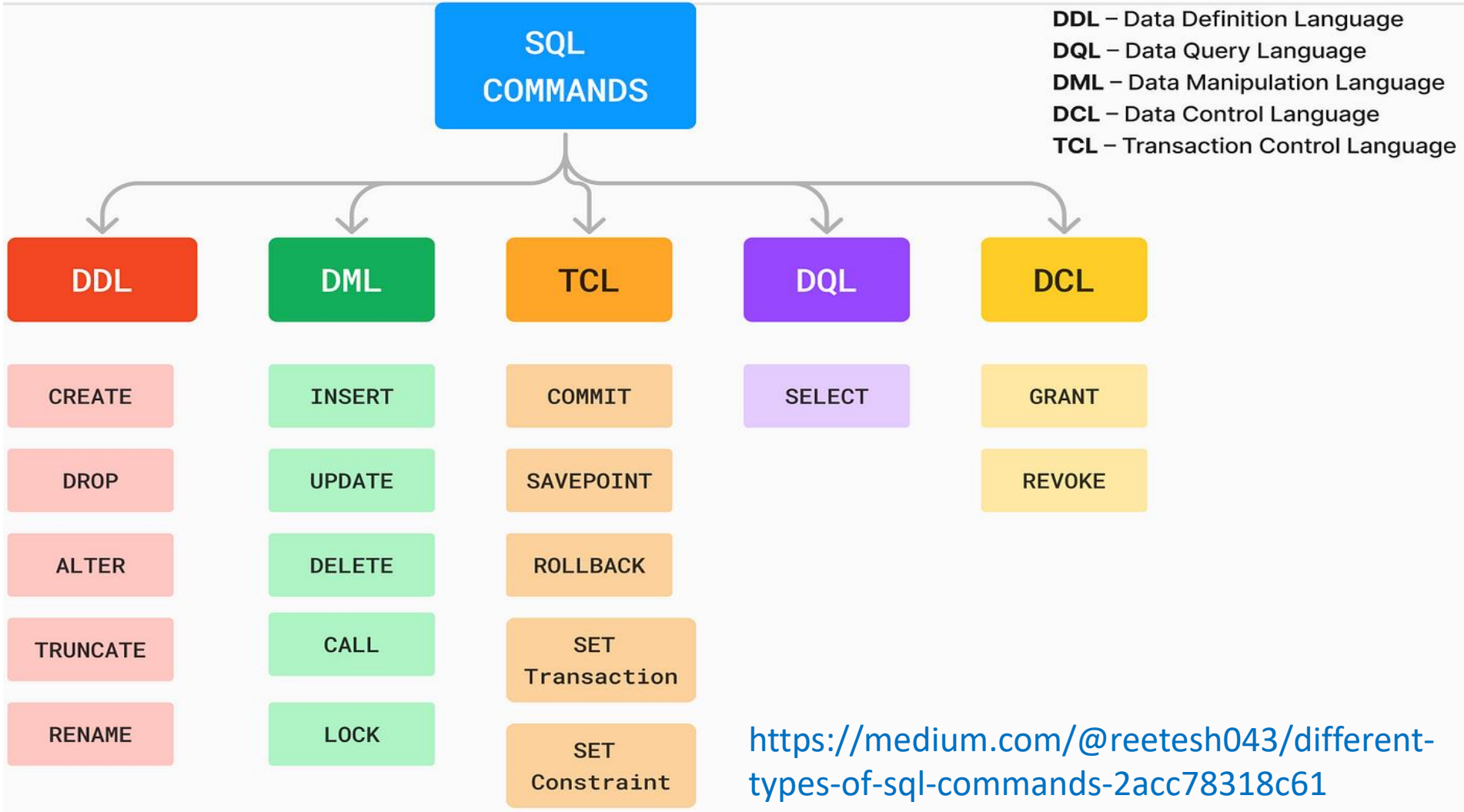
SGBD

Un système de gestion de base de données est un logiciel servant à gérer des données sockées dans une base de données, en garantissant la qualité, la pérennité et la confidentialité des informations, tout en cachant la complexité des opérations.

SQL

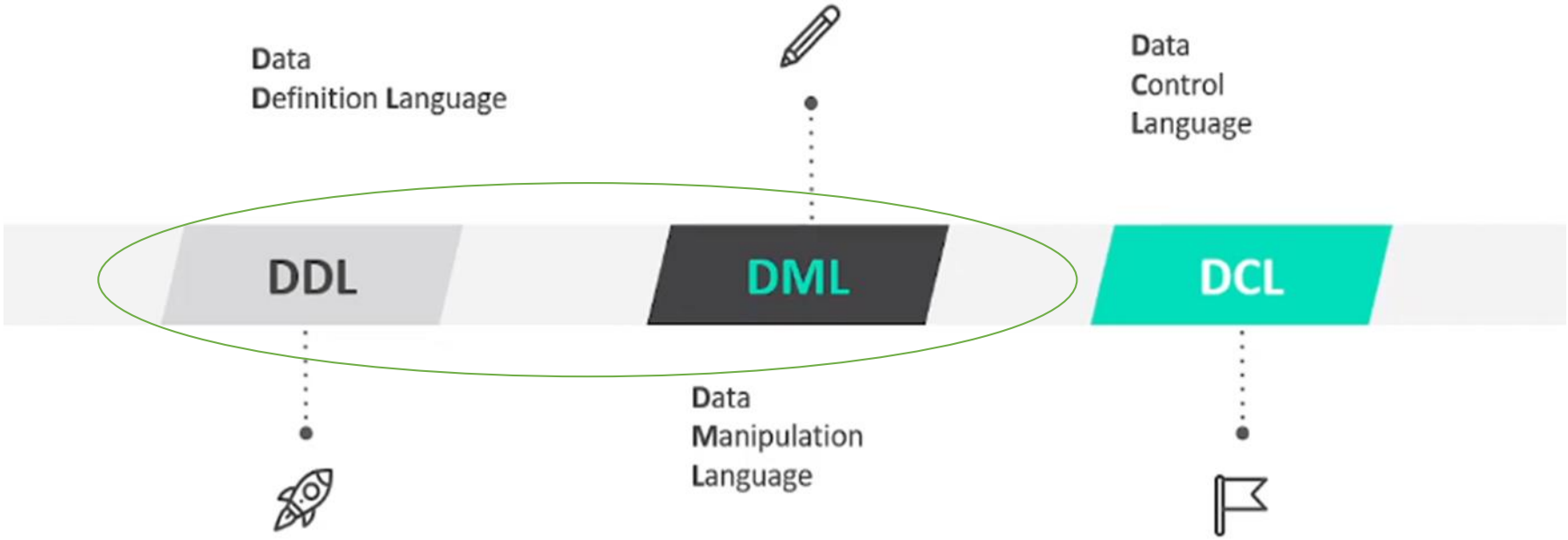
Le SQL (Structured Query Language) est un langage permettant de communiquer avec une base de données en respectant un certain nombre de règles.

Introduction aux bases de données



https://medium.com/@reetesh043/different-types-of-sql-commands-2acc78318c61

Introduction aux bases de données

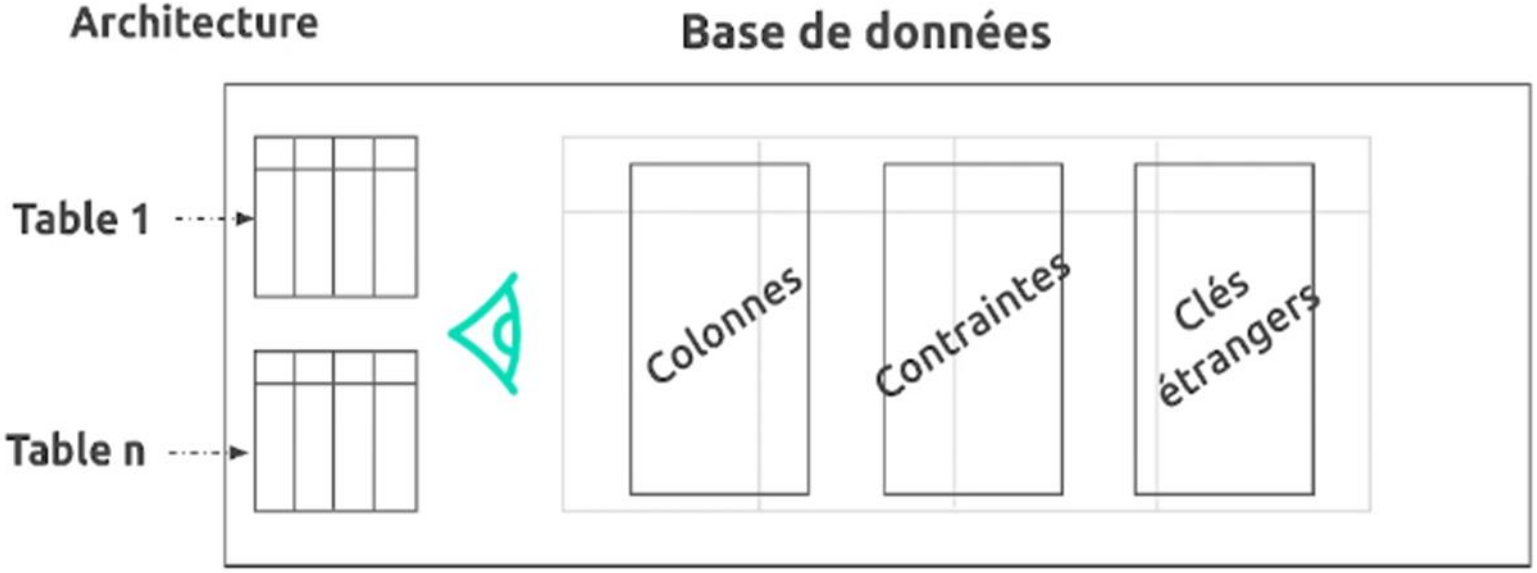


Introduction aux bases de données

Base de données

Une base de données est une collection organisée de données. Elle permet de stocker, gérer et récupérer des informations de manière efficace.

Architecture



Microsoft SQL Server

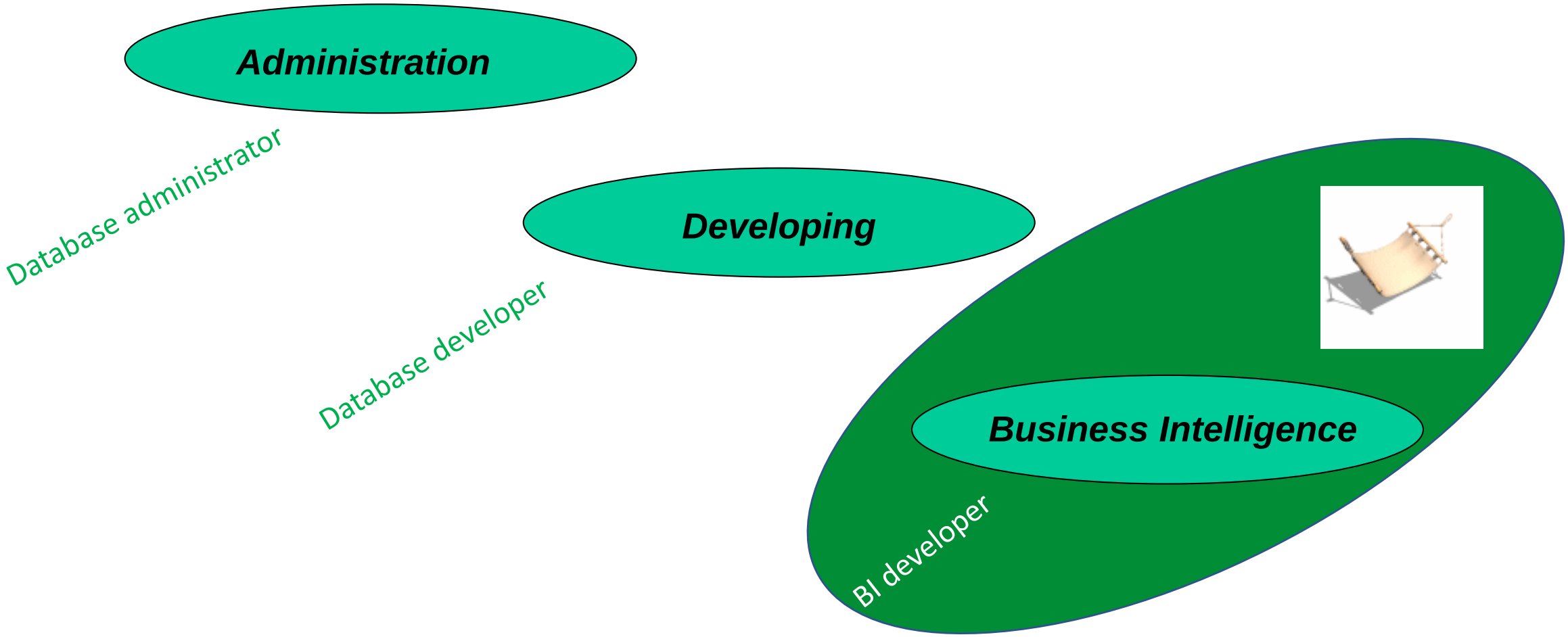
- ✓ **M**icrosoft **SQL Server** est un Système de gestion de base de données (SGBD) relationnel et transactionnel développé et commercialisé par **Microsoft**.

- ✓ **M**icrosoft **SQL Server** utilise le langage **T-SQL** (**Transact-SQL**) pour ses requêtes, c'est une implémentation de SQL qui prend en charge les procédures stockées et les déclencheurs.

- ✓ Il faut savoir que, tous les **SGBDs** relationnels (**Oracle**, **MS SQL Server**, **MySQL**, **SQLite**, **DB2**, **PostgreSQL** ..) utilisent un SQL standard.

- ✓ TOUS les **SGBDs** offrent une interface ou un logiciel de gestion des bases de données. Pour Oracle, c'est SQL Developer. Pour MS SQL Server c'est **SQL Server Management Studio**, pour MySQL c'est MySQL Workbench, pour SQLite c'est SQLite DB Browser.

Microsoft SQL Server Tracks



Qu'est-ce que la Business Intelligence ?

L’informatique décisionnelle, aussi appelée business intelligence (BI), désigne un ensemble de méthodes, de moyens et d’outils informatiques utilisés pour piloter une entreprise et aider à la prise de décision : tableaux de bord, rapports analytiques et prospectifs.

Catégories de traitement de données:

OLTP, OLAP et Big Data

OLTP et OLAP sont des catégories de traitement de données, le premier dédié aux **systèmes opérationnels(OLTP)** et le second aux **systèmes décisionnels(OLAP)**. Nous allons les découvrir et positionner par rapport à elles le concept de **Big Data**.

Catégories de traitement de données:

OLTP, OLAP et Big Data

Online Transaction Processing (OLTP)

Catégorie de systèmes de traitement de données visant à gérer en temps réel des requêtes potentiellement concurrentes à des fins de gestion d'un **système opérationnel**.

Online Analytical Processing (OLAP)

Catégorie de systèmes de traitement de données visant à répondre à des requêtes multi-dimensionnelles sur les données à des fins d'**analyse**.

Catégories de traitement de données:

OLTP, OLAP et Big Data

Big Data

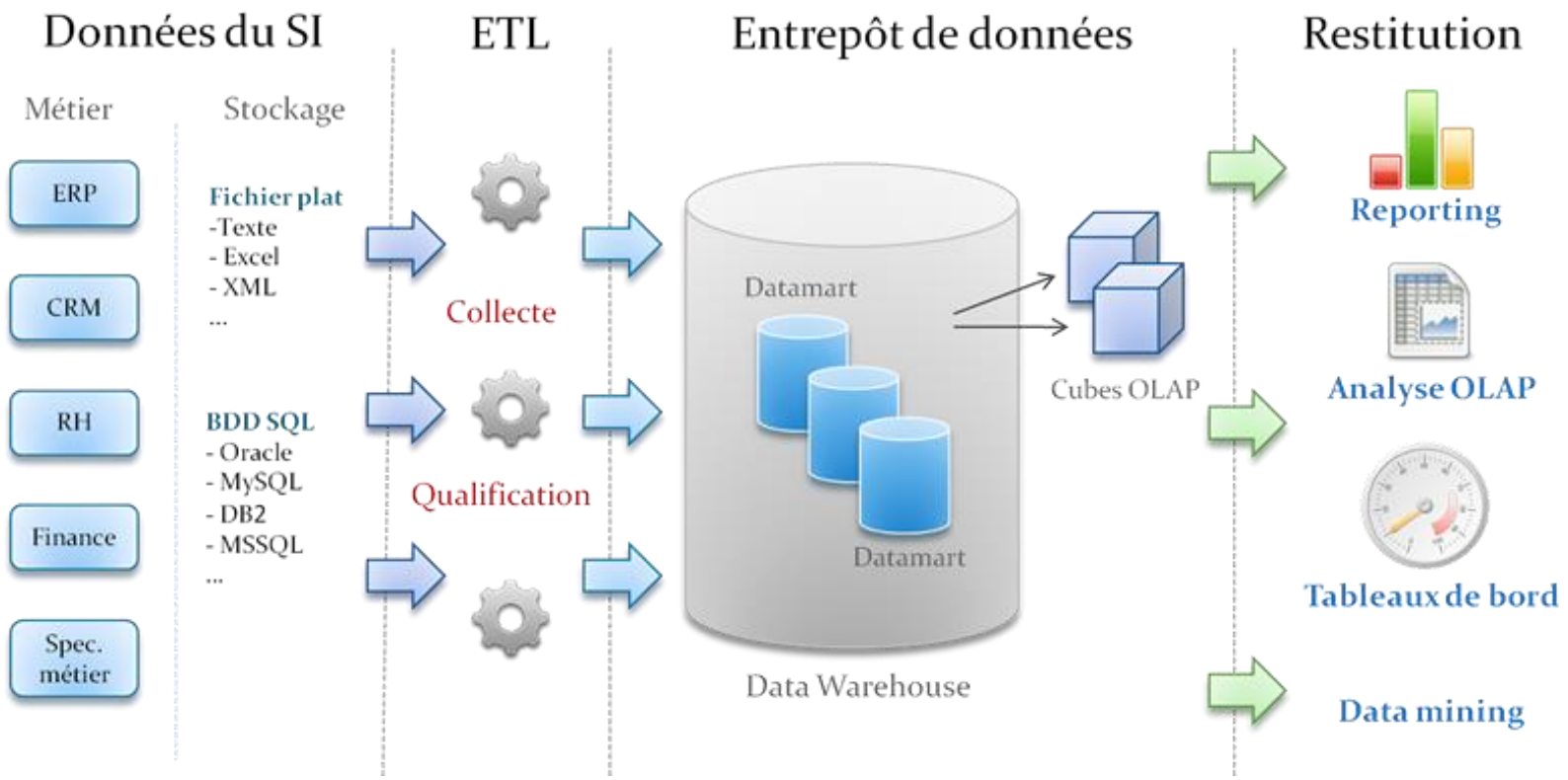
Ensemble des technologies pouvant gérer des **volumes très importants** de données (au-delà de la centaine de **To**).

Système d'information décisionnel

Définition

Le système d'information décisionnel est **la pierre** angulaire de l'**entreprise** afin de produire des **tableaux de bord** pour aider **la prise de décision**.

Système d'information décisionnel



Vue globale d'un système d'information décisionnel

https://formations.imt-atlantique.fr/bi/bi_architectures.html

Système d'information décisionnel

Un système d’information décisionnel est structuré en trois zones principales :

La zone ETL (pour Extract, Transform, Load) où l’on effectue les traitements sur les données : cette zone doit être réservée aux développeurs.

La zone de stockage des données : historiquement nous avions ici un entrepôt de données (data warehouse) reposant sur une technologie OLAP.

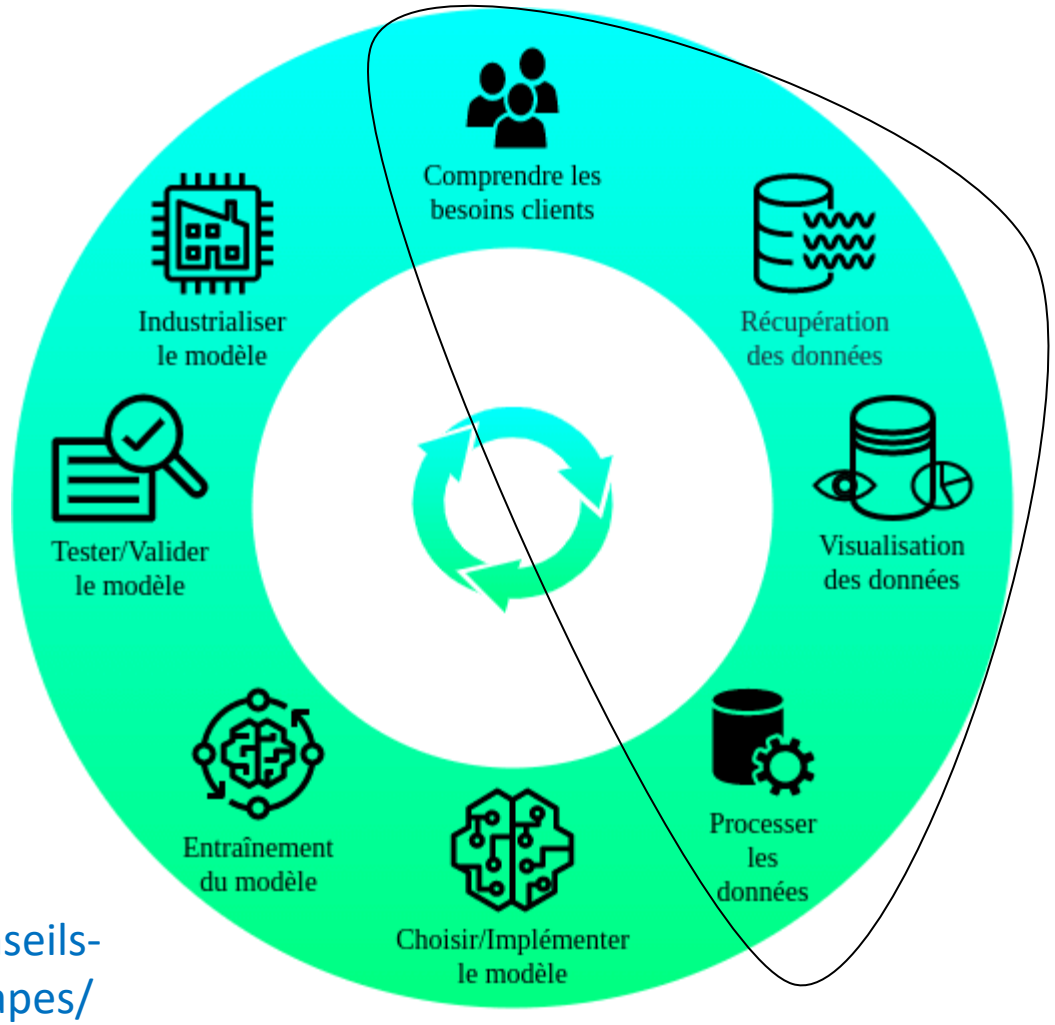
La zone de restitution des données qui couvre tous les outils qui génèrent des rapports ou des tableaux de bord.

Ingénierie des données

L'ingénierie des données est le développement, la mise en œuvre et la maintenance de systèmes et de processus qui prennent en charge les données brutes et produisent des informations cohérentes et de haute qualité

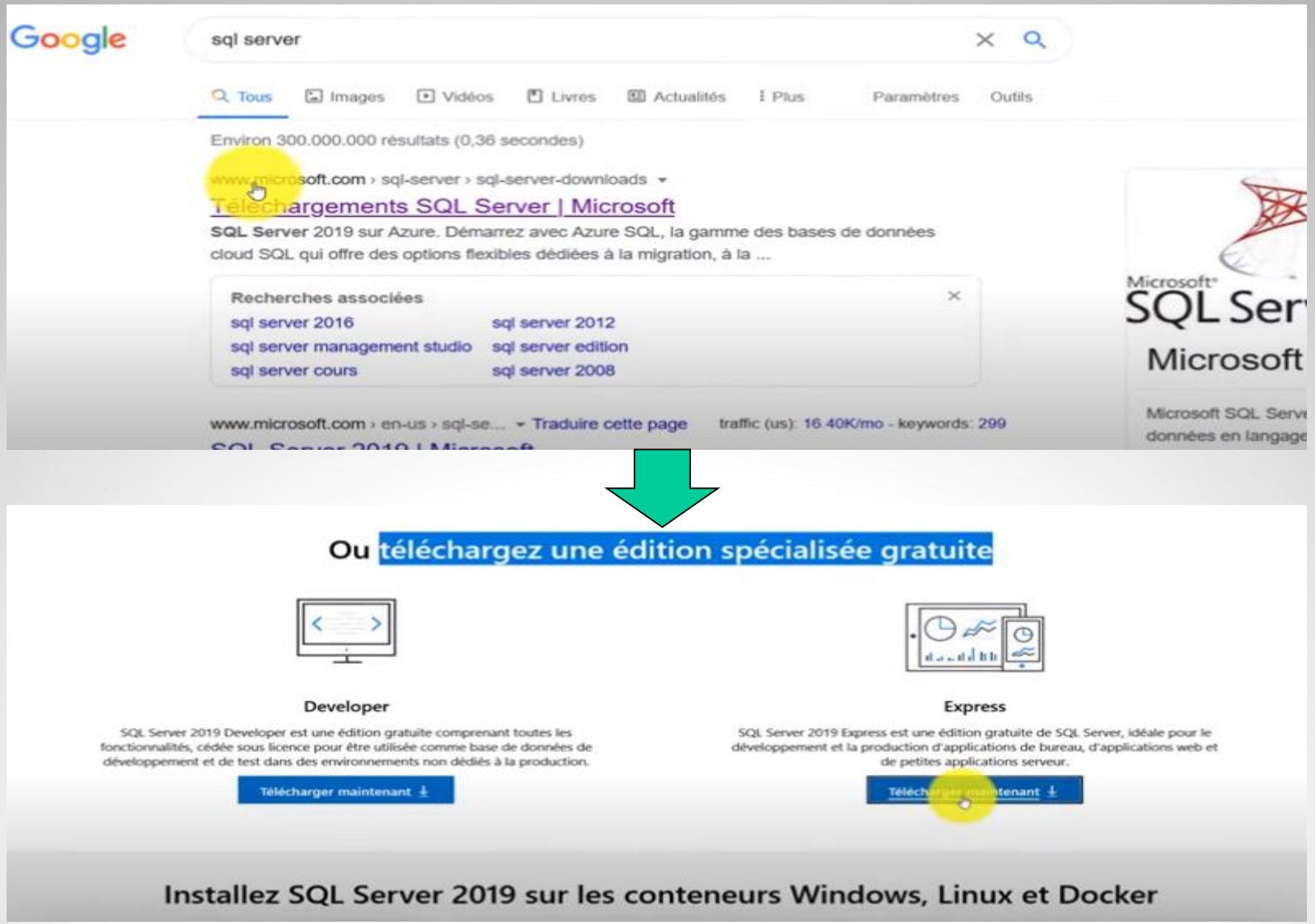
Cycle de vie de l'ingénierie des données

Le cycle de vie d'un projet de Machine Learning

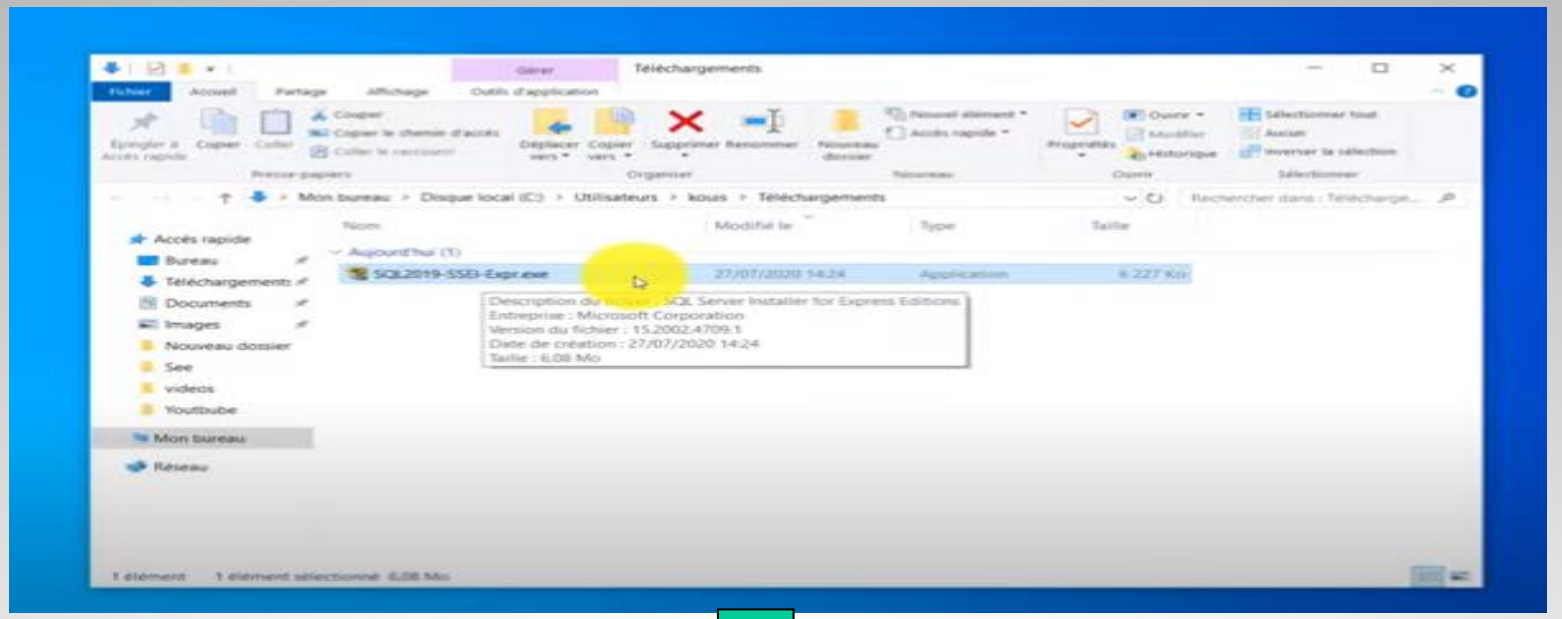


https://kaizen-solutions.net/kaizen-insights/articles-et-conseils-de-nos-experts/cycle-de-vie-projet-machine-learning-8-etapes/

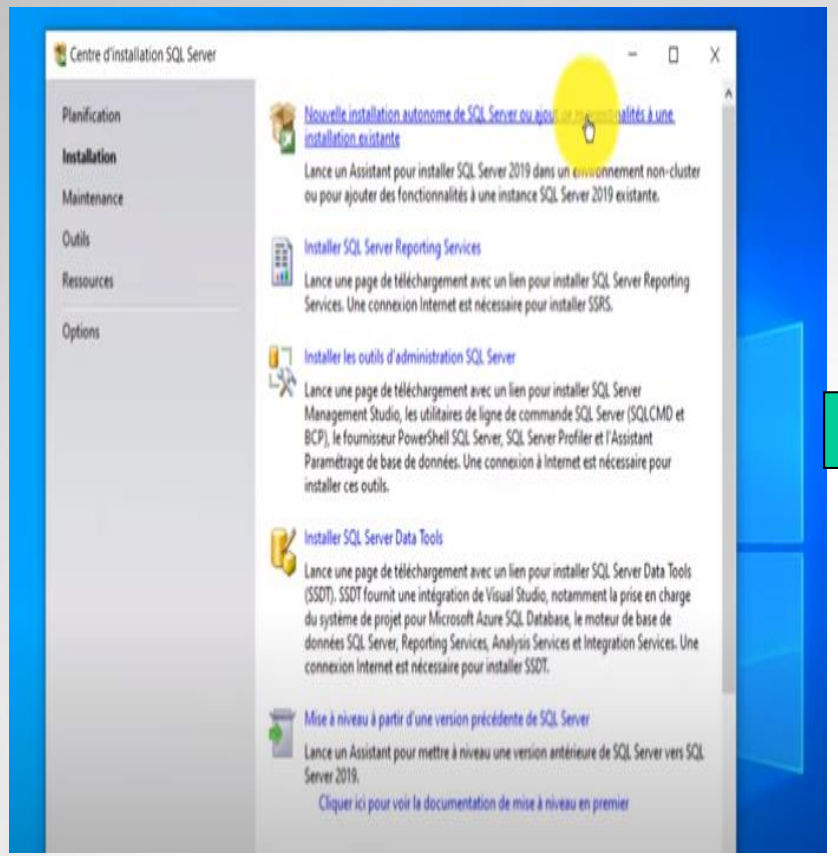
Guide d'installation SQL Server 2019

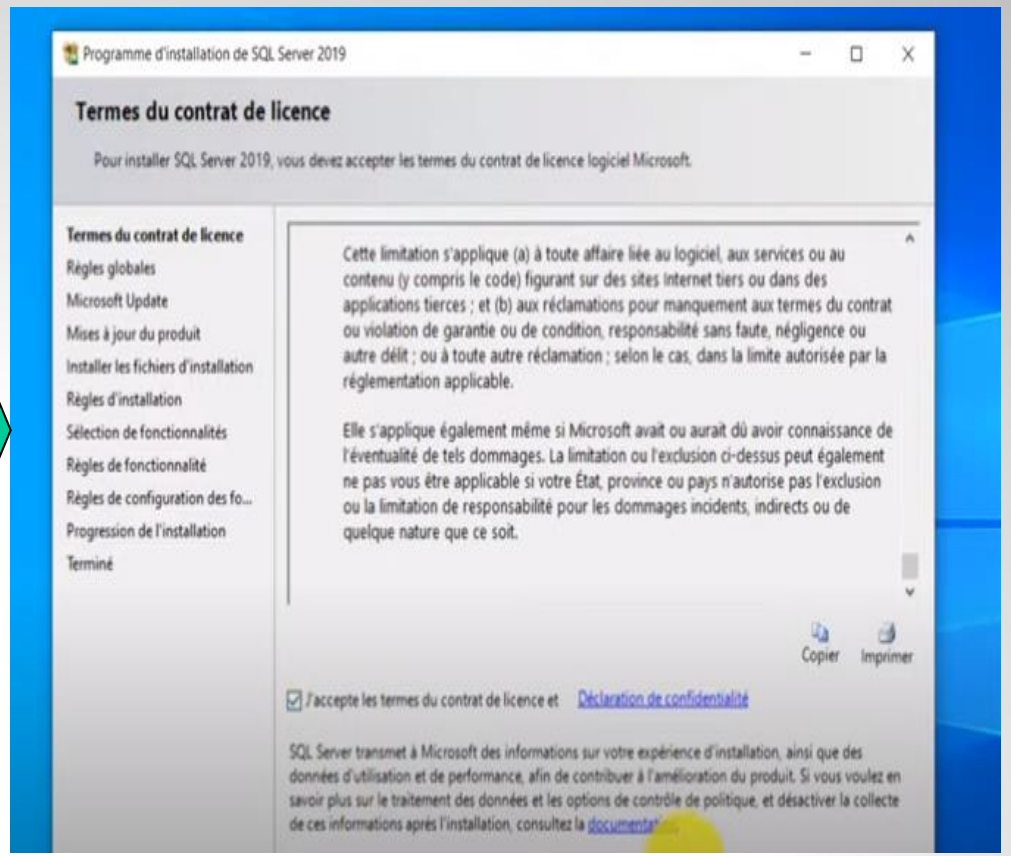


Guide d'installation SQL Server 2019

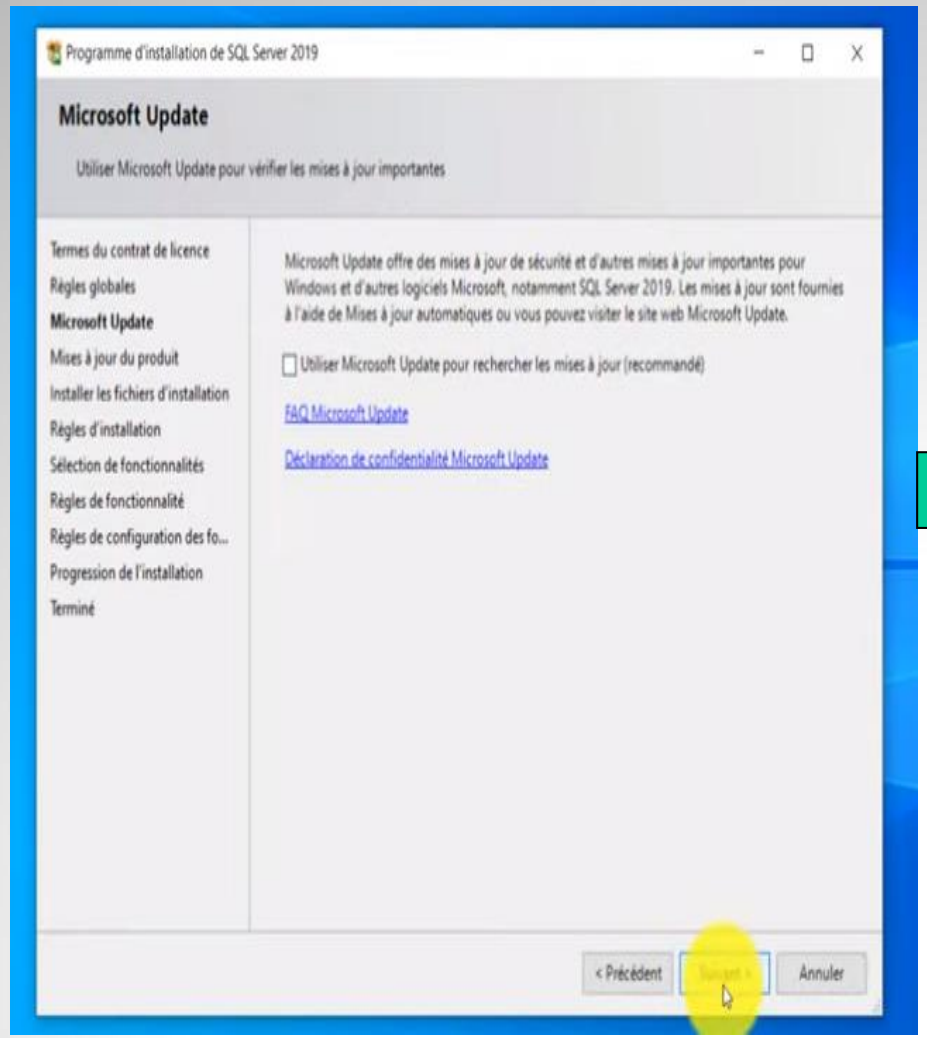


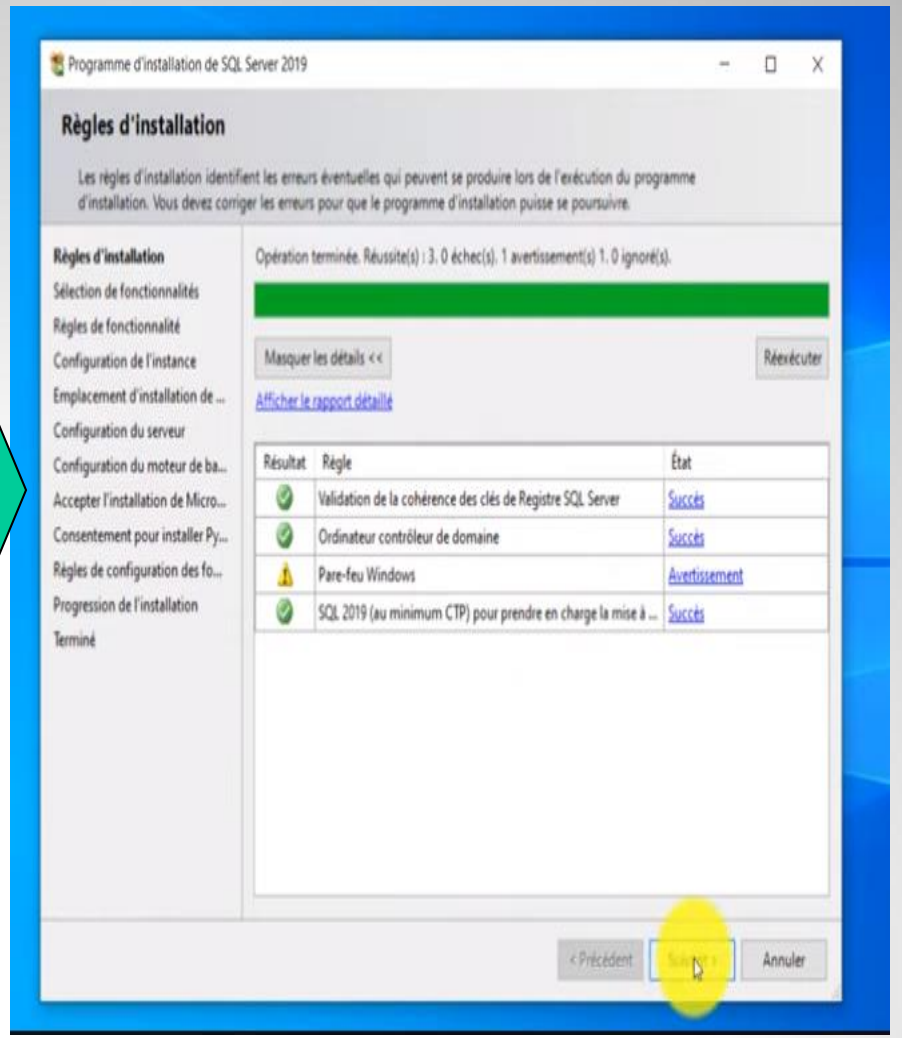
Guide d'installation SQL Server 2019



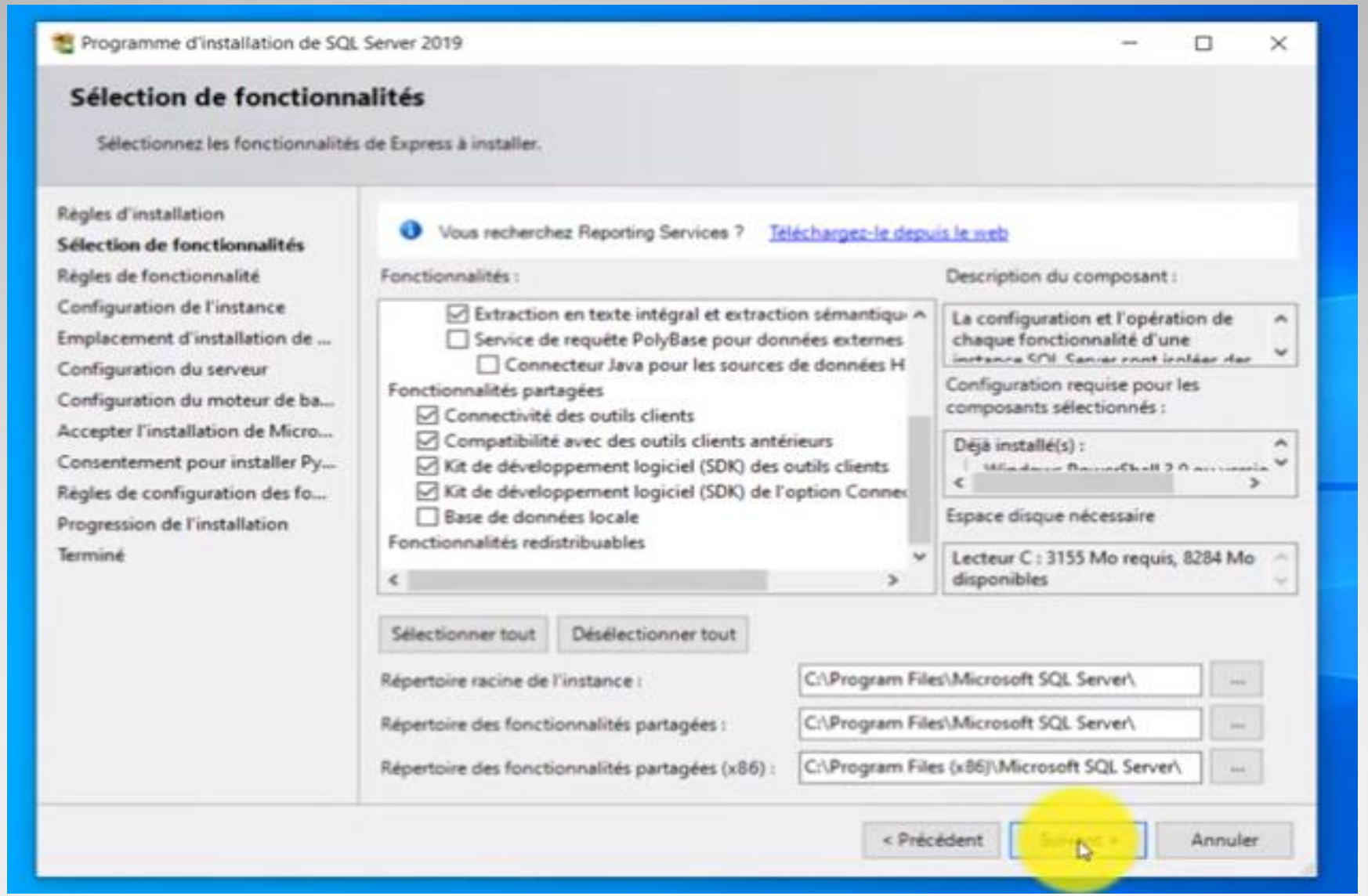


Guide d'installation SQL Server 2019

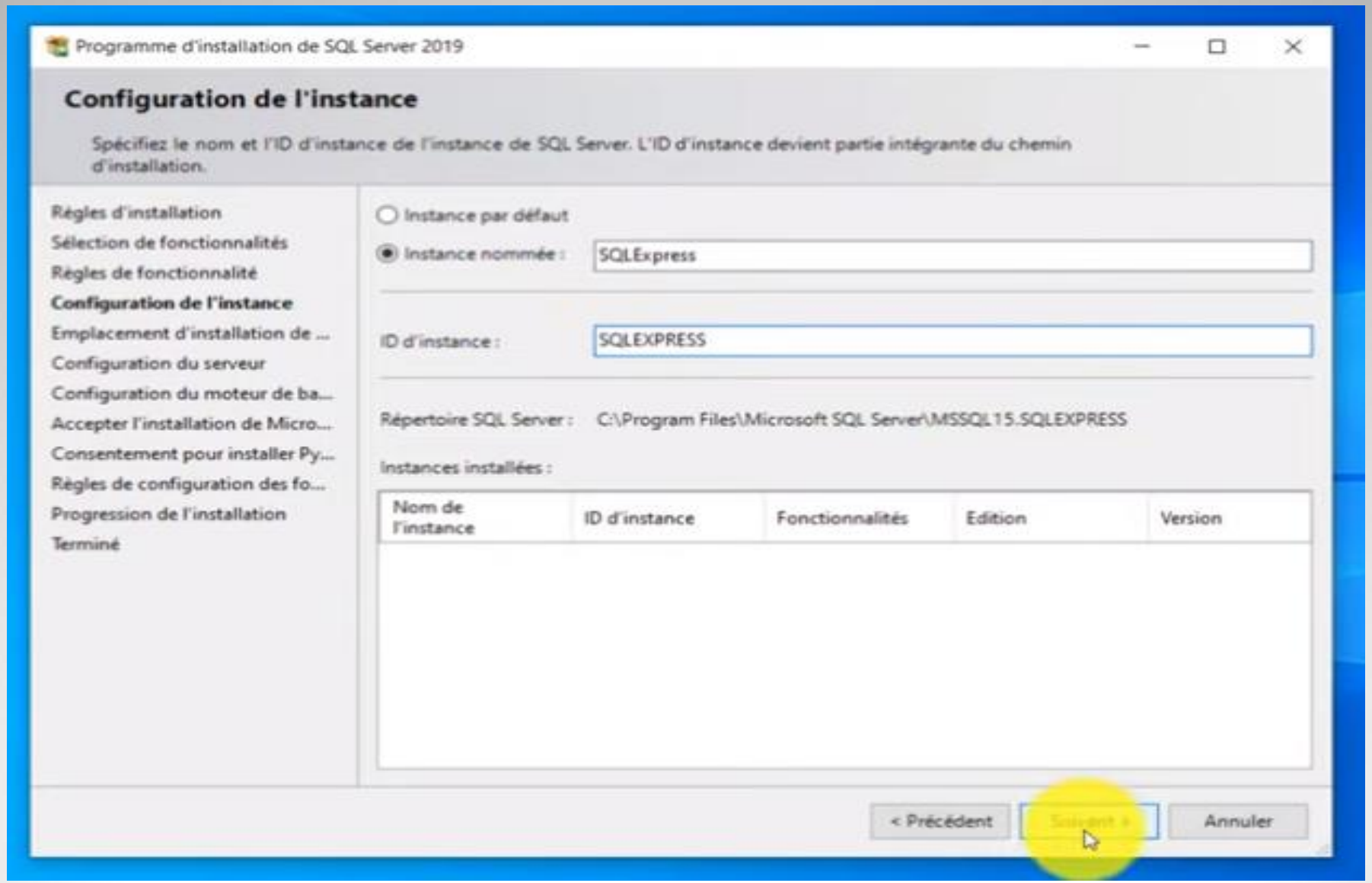




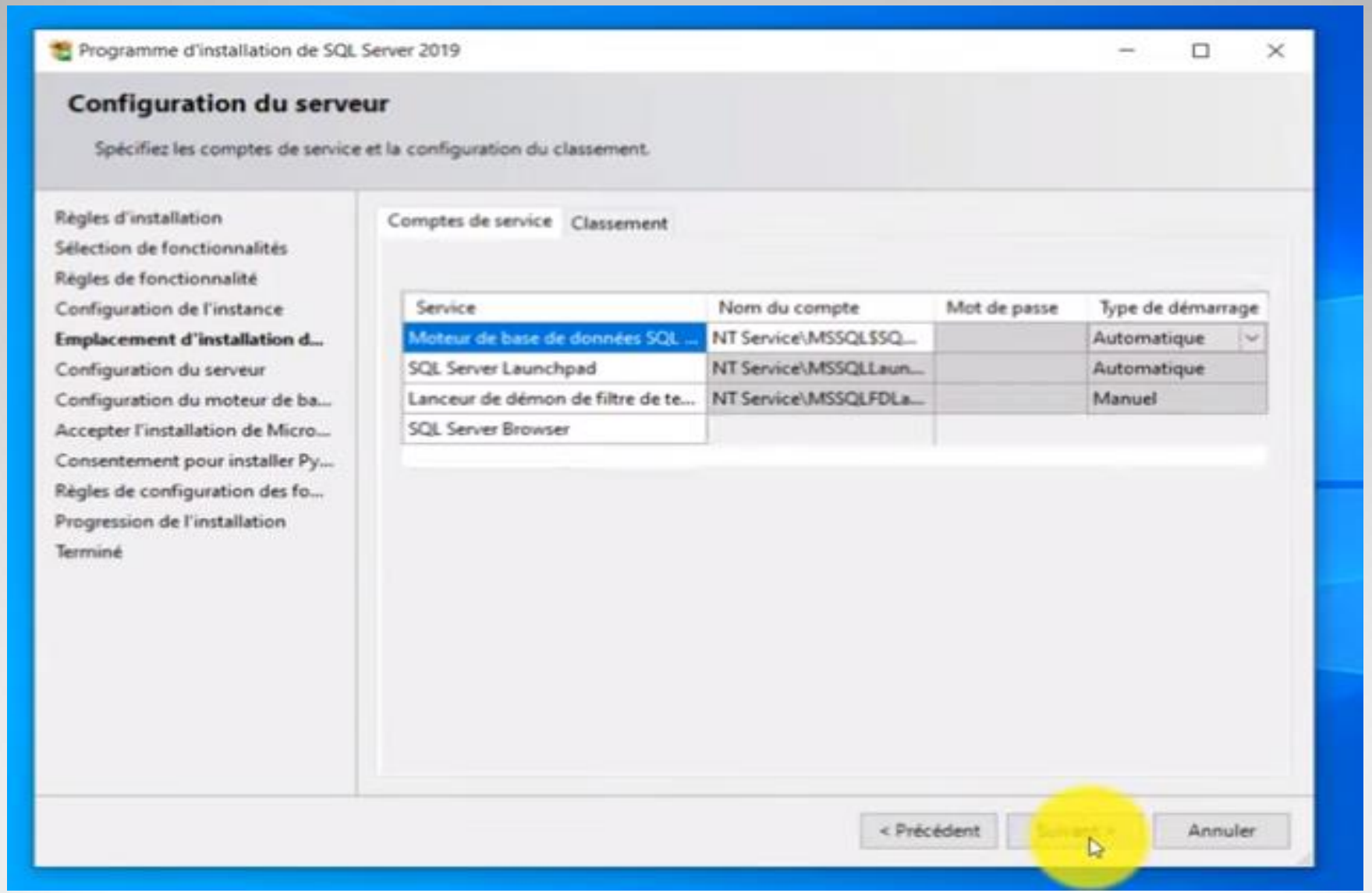
Guide d'installation SQL Server 2019



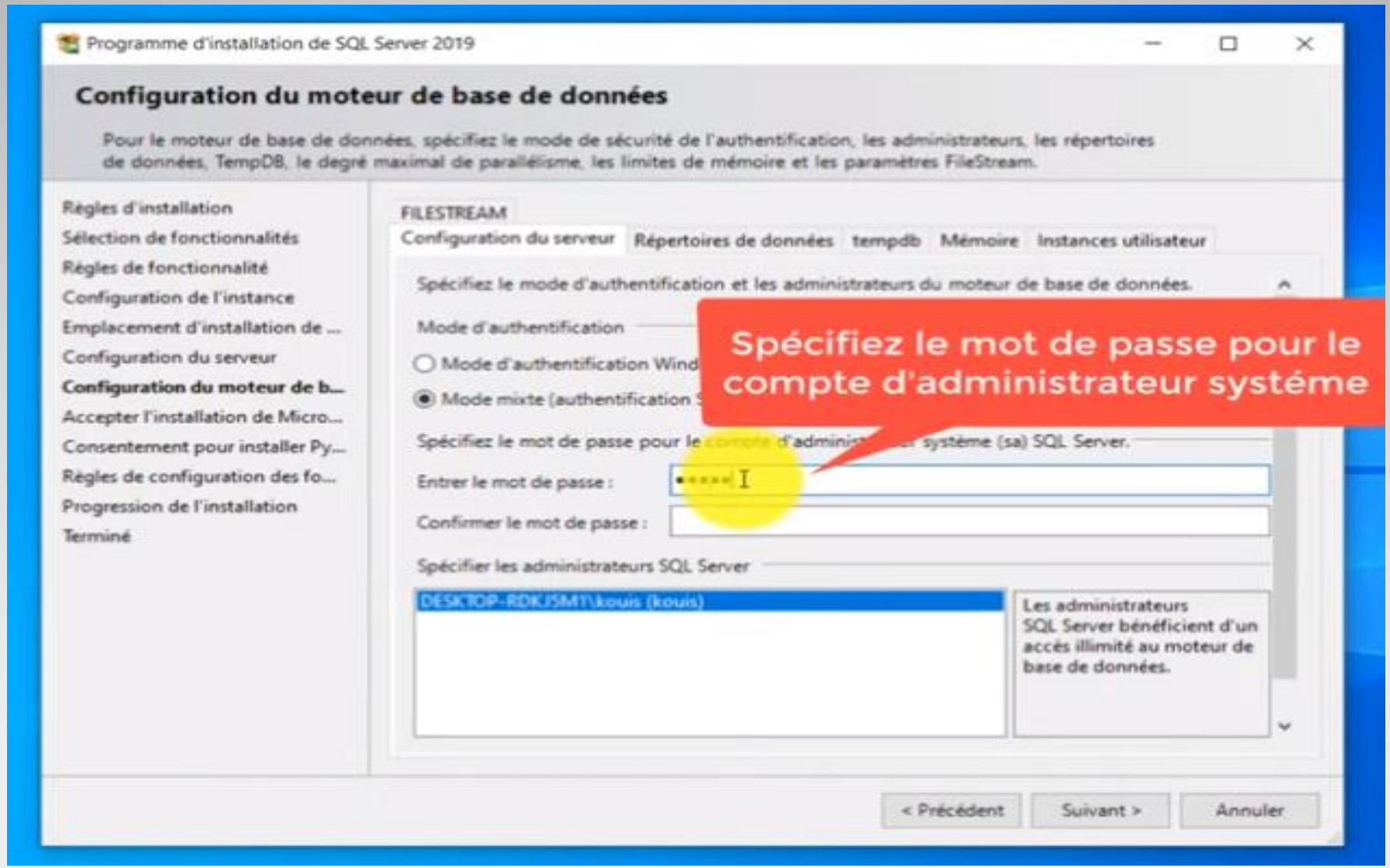
Guide d'installation SQL Server 2019



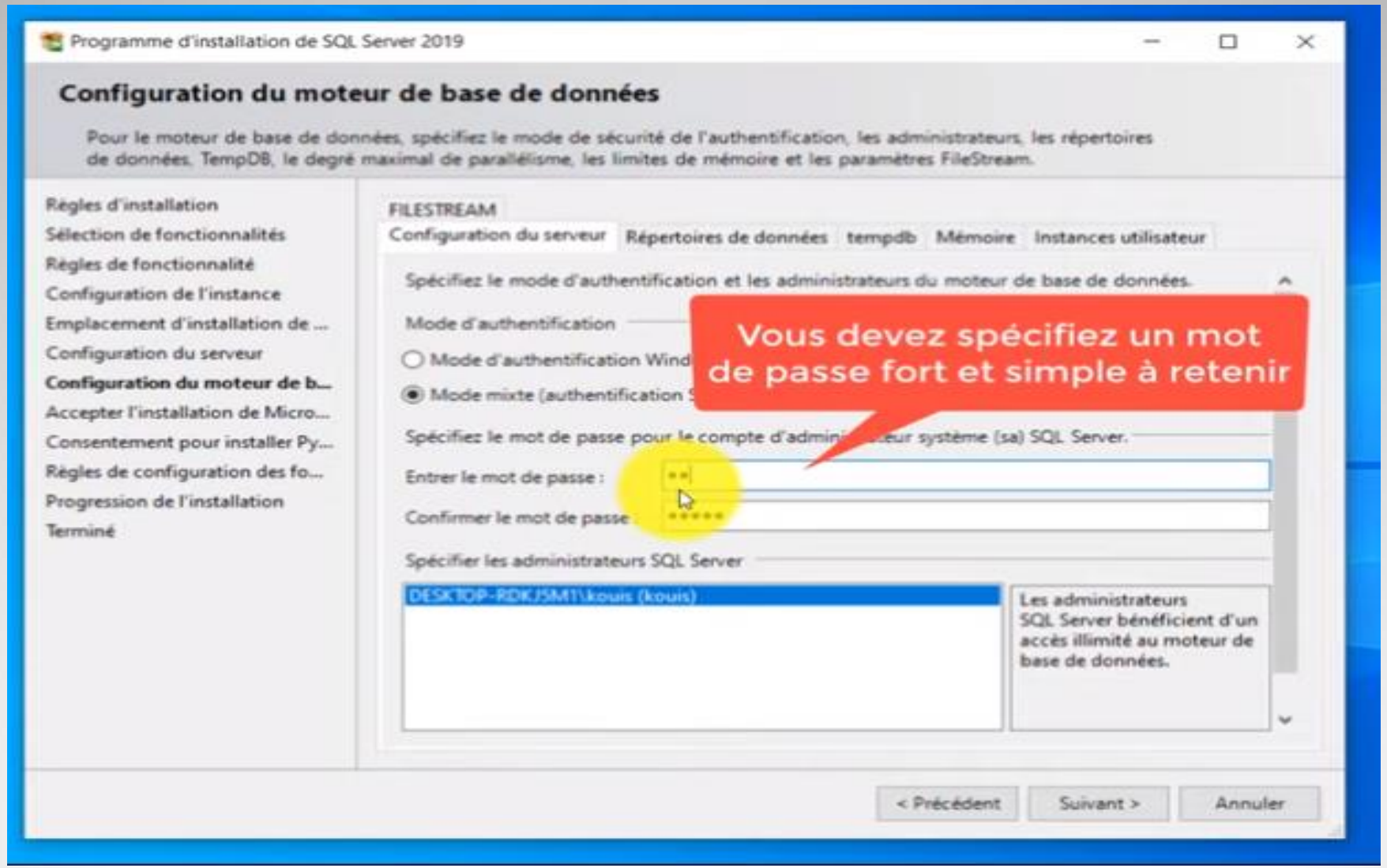
Guide d'installation SQL Server 2019



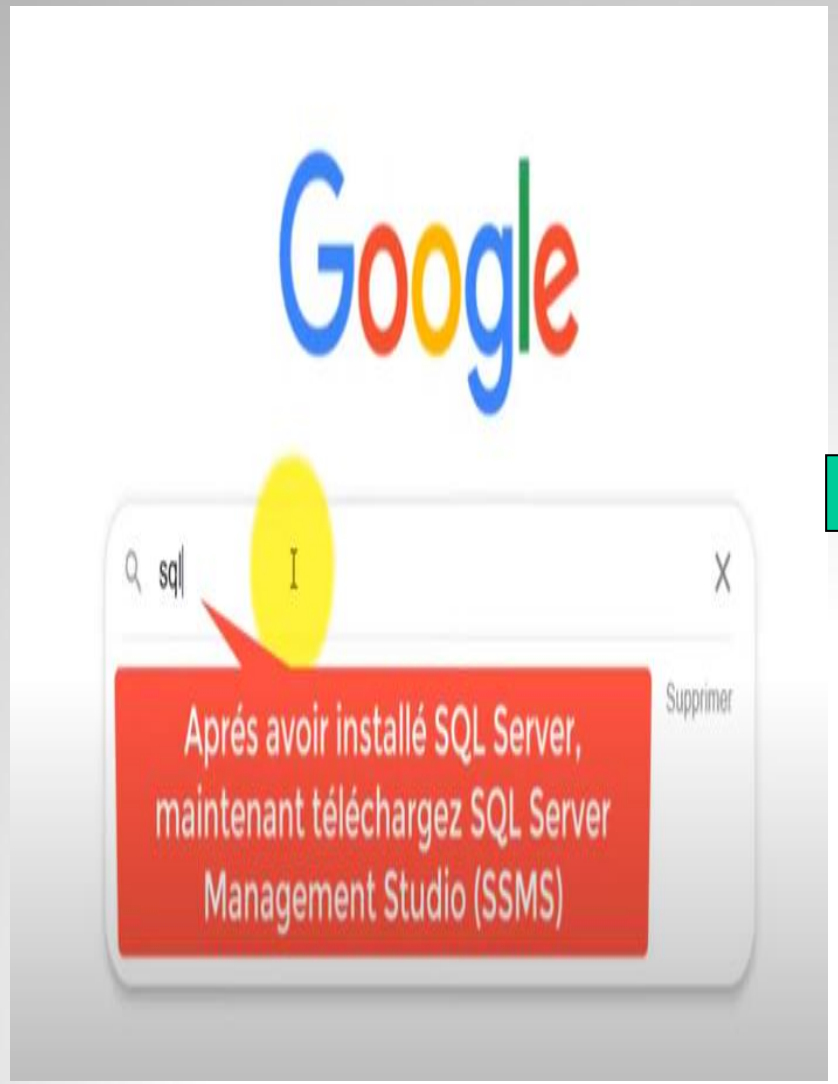
Guide d'installation SQL Server 2019

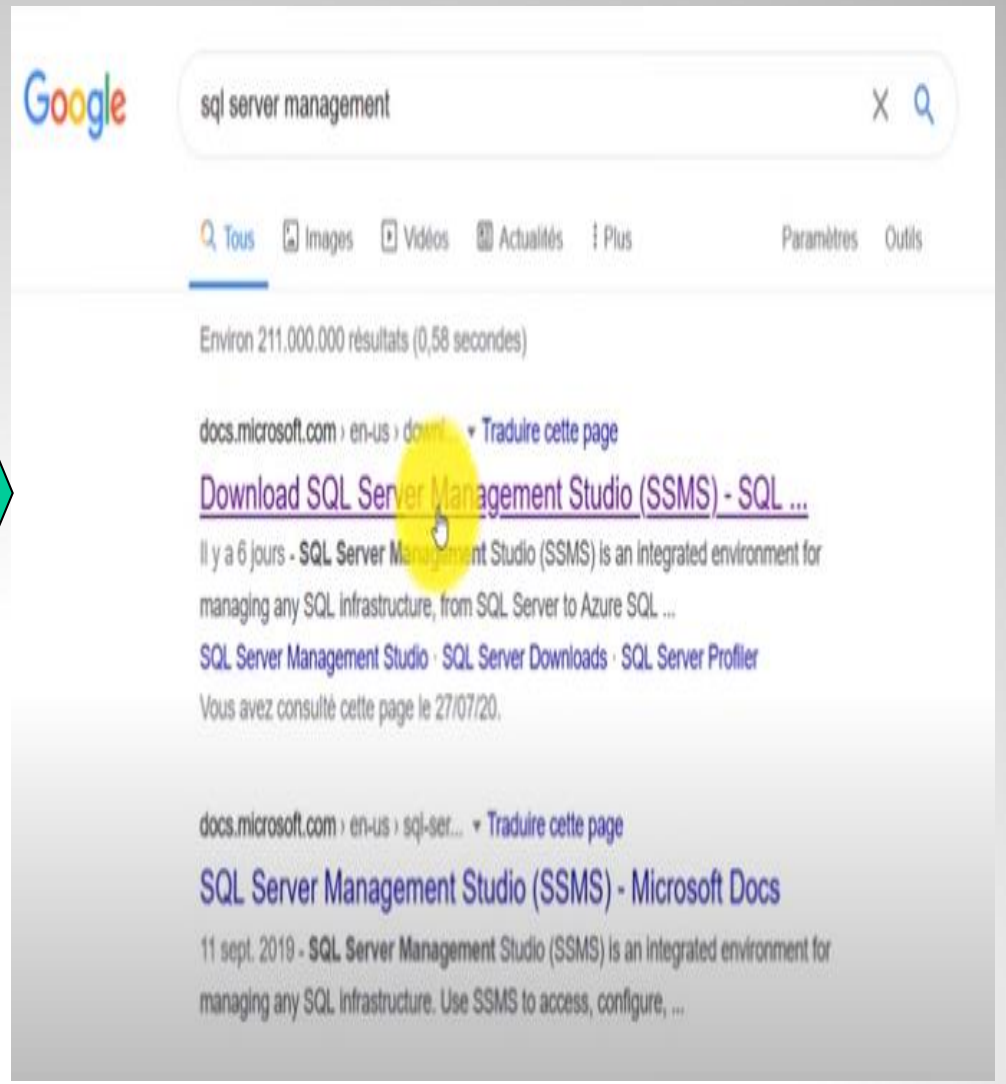


Guide d'installation SQL Server 2019

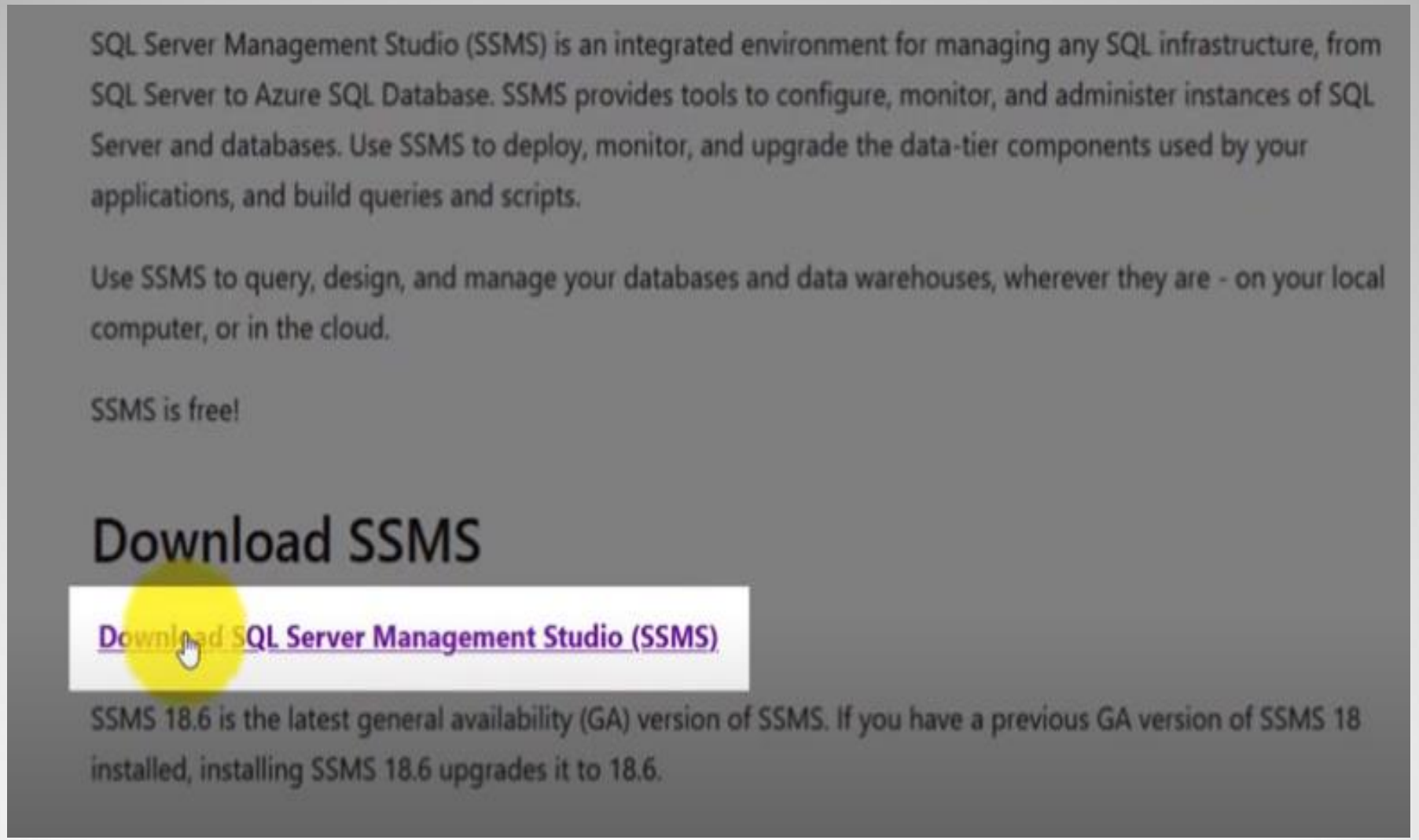


Guide d'installation SQL Server 2019



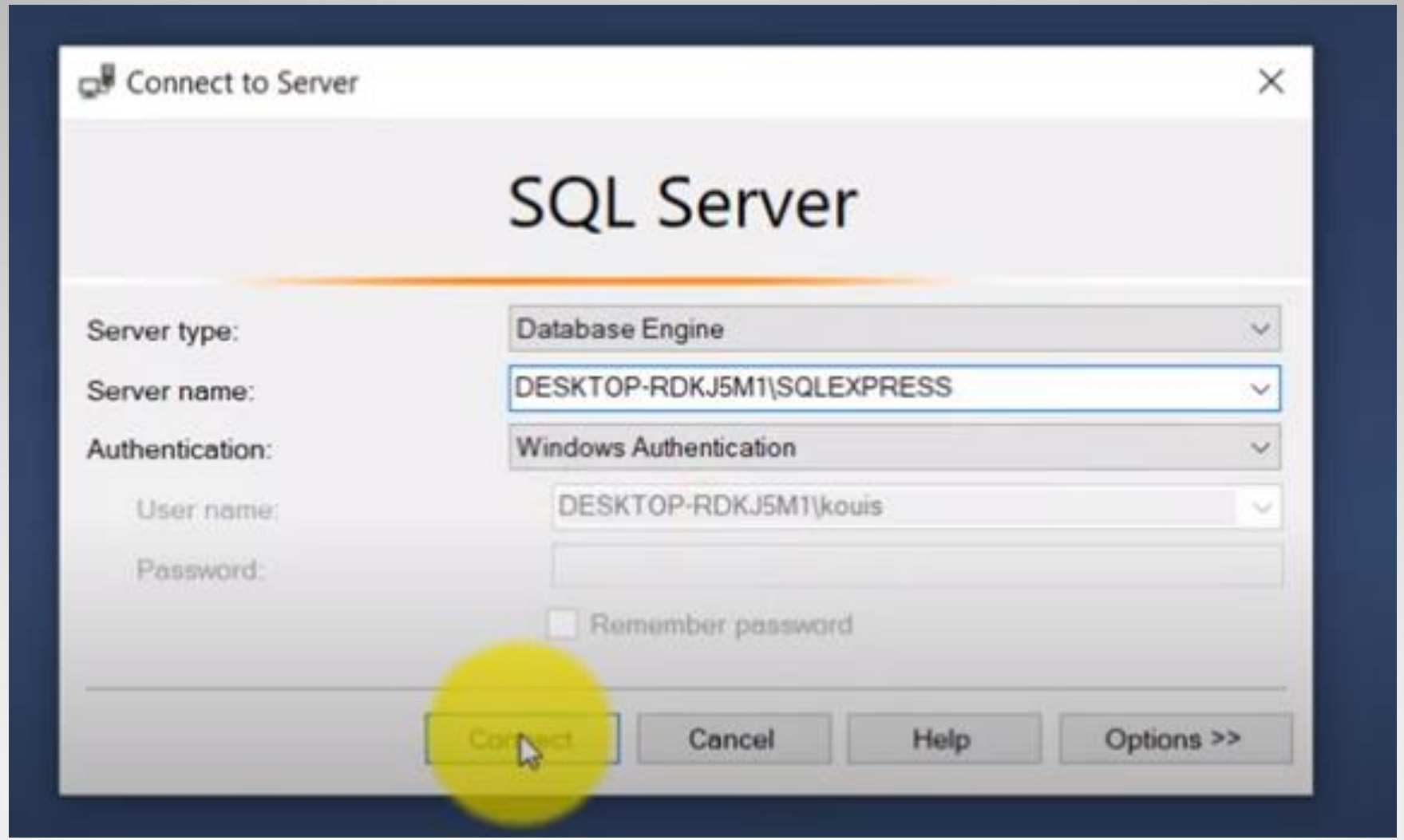


Guide d'installation SQL Server 2019



Guide d'installation SQL Server 2019

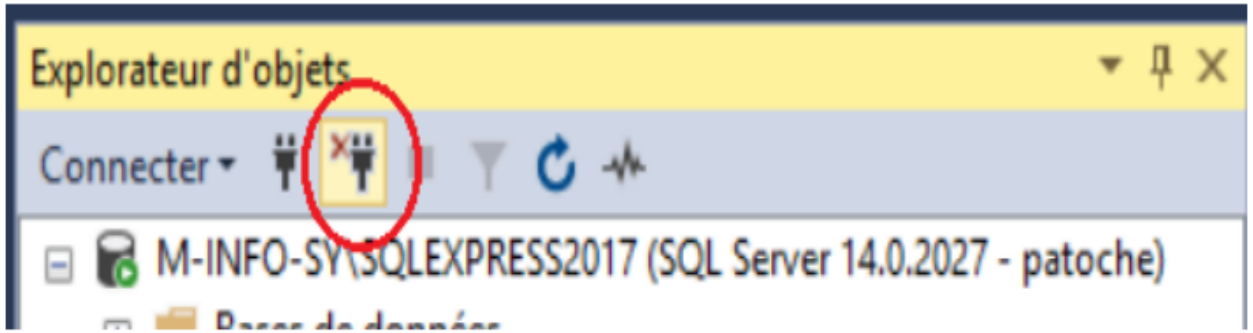
Guide d'installation SQL Server 2019



Connexion avec l'authentification SQL Server

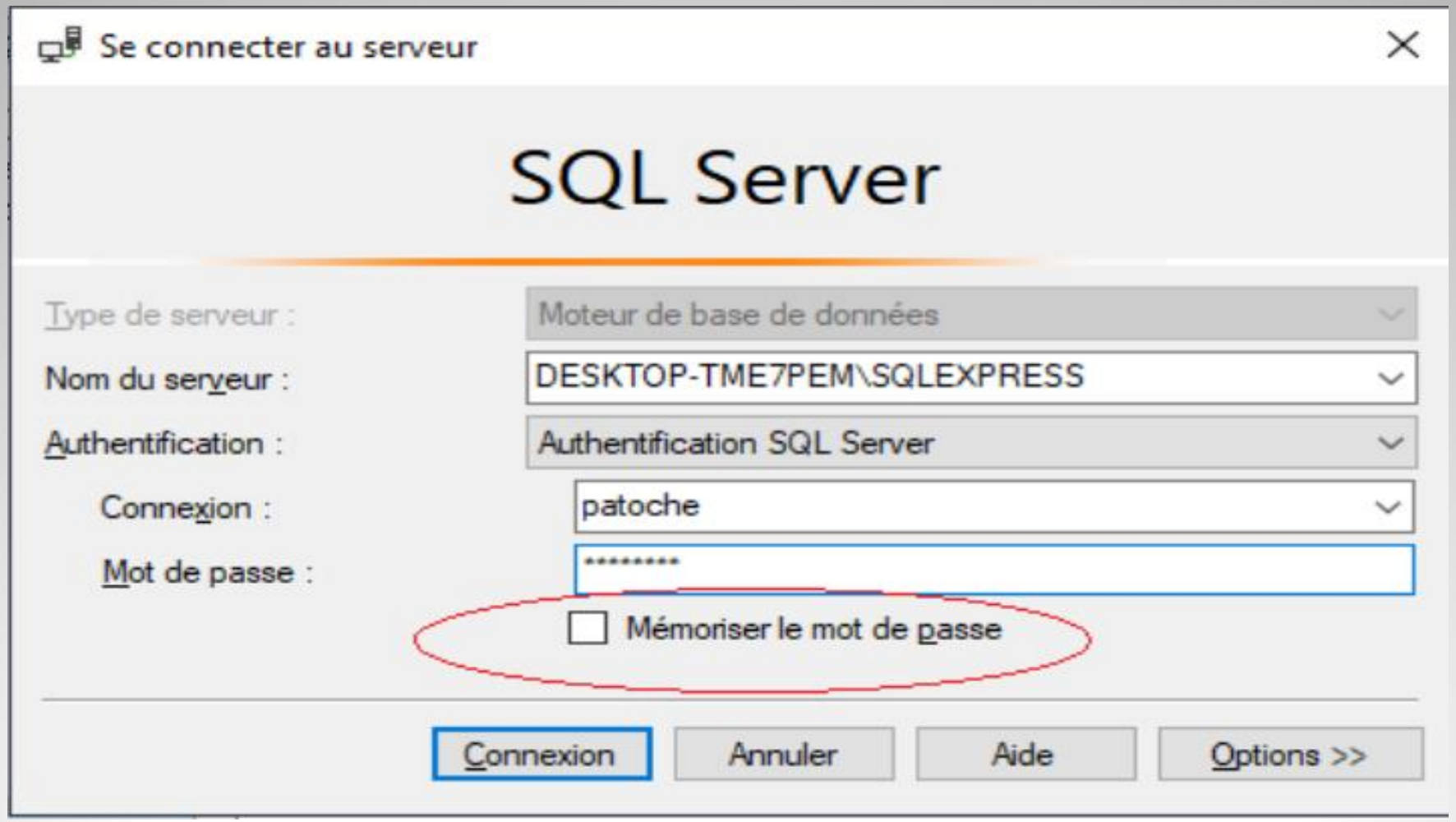
Vous pouvez vous déconnecter du serveur et vous reconnecter avec votre nouvelle connexion (SQL Server) comme suit.

Pour vous déconnecter du serveur, utiliser le bouton Déconnecter



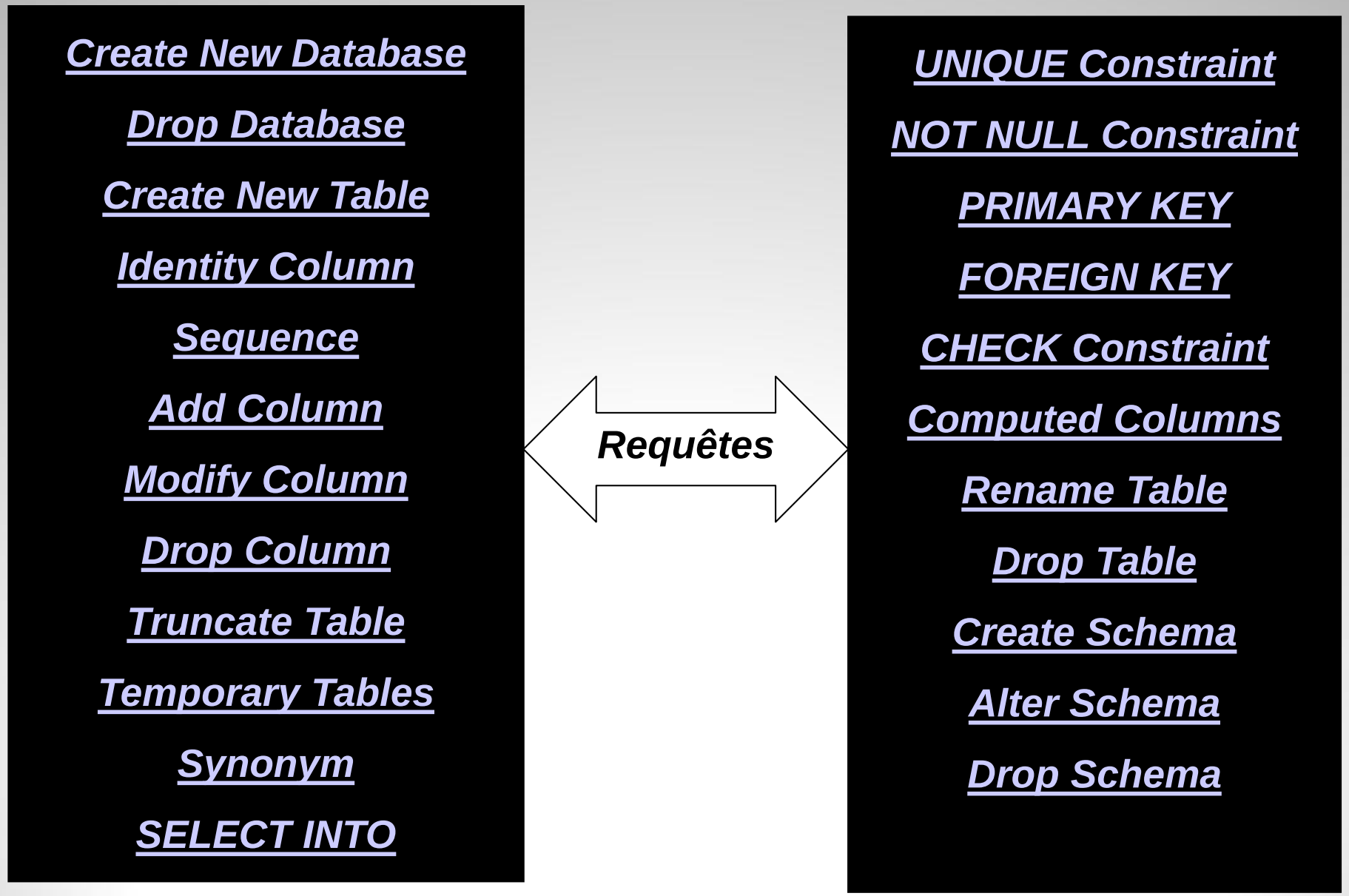
Ou bien Bouton droit sur la Votre serveur, puis **déconnecter** .

Connexion avec l'authentification SQL Server



Ne jamais mémoriser le mot de passe

Data Definition Language: DDL



Server configuration

DB Engine

SQL server Browser

SQL Server Agent

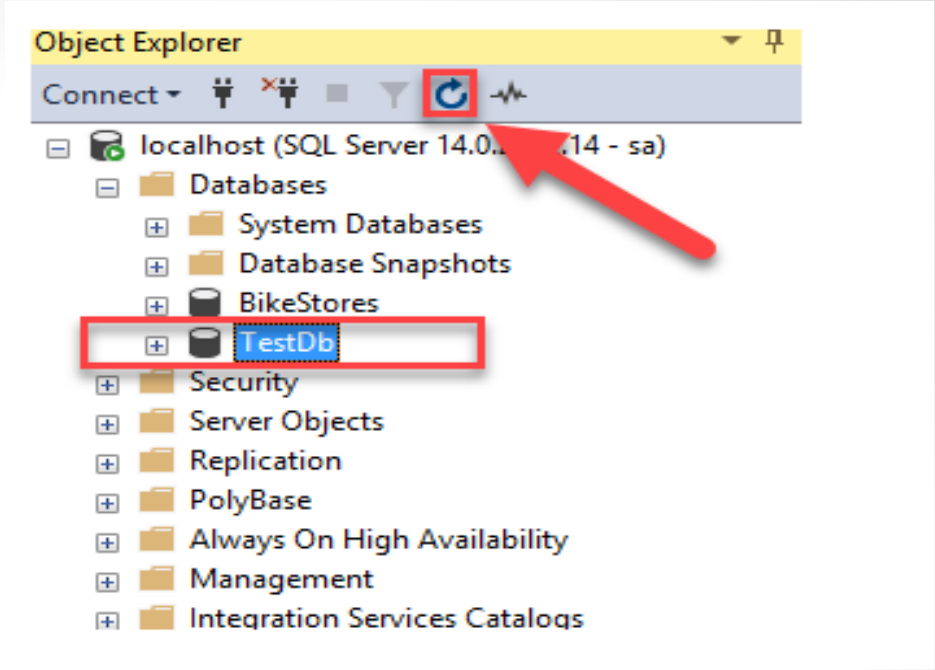
Création de la base de données

Résumé : dans cette partie, vous apprendrez à créer une nouvelle base de données dans SQL Server en utilisant l'instruction **CREATE DATABASE** ou **SQL Server Management Studio.**

CREATE DATABASE database_name;

Création de la base de données

Une fois l'instruction exécutée avec succès, vous pouvez visualiser la base de données nouvellement créée dans **l'Explorateur d'objets**. Si la nouvelle base de données n'apparaît pas, vous pouvez cliquer sur le bouton **Actualiser** ou appuyer sur la touche **F5** du clavier pour mettre à jour la liste des objets.



Création de la base de données

Cette déclaration répertorie toutes les bases de données du serveur SQL :

Vous pouvez également exécuter la procédure stockée sp_databases :

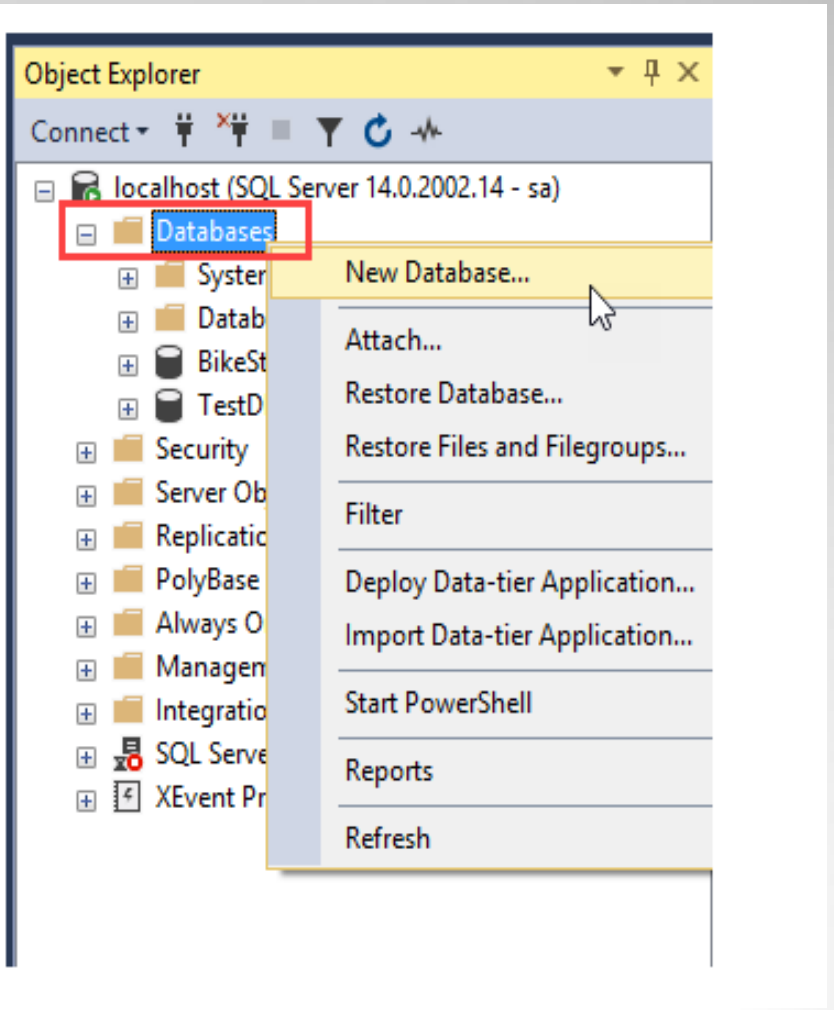
EXEC sp_databases;

```
SELECT
    name
FROM
    master.sys.databases
ORDER BY
    name;
```

name
BikeStores
master
model
msdb
tempdb
TestDb

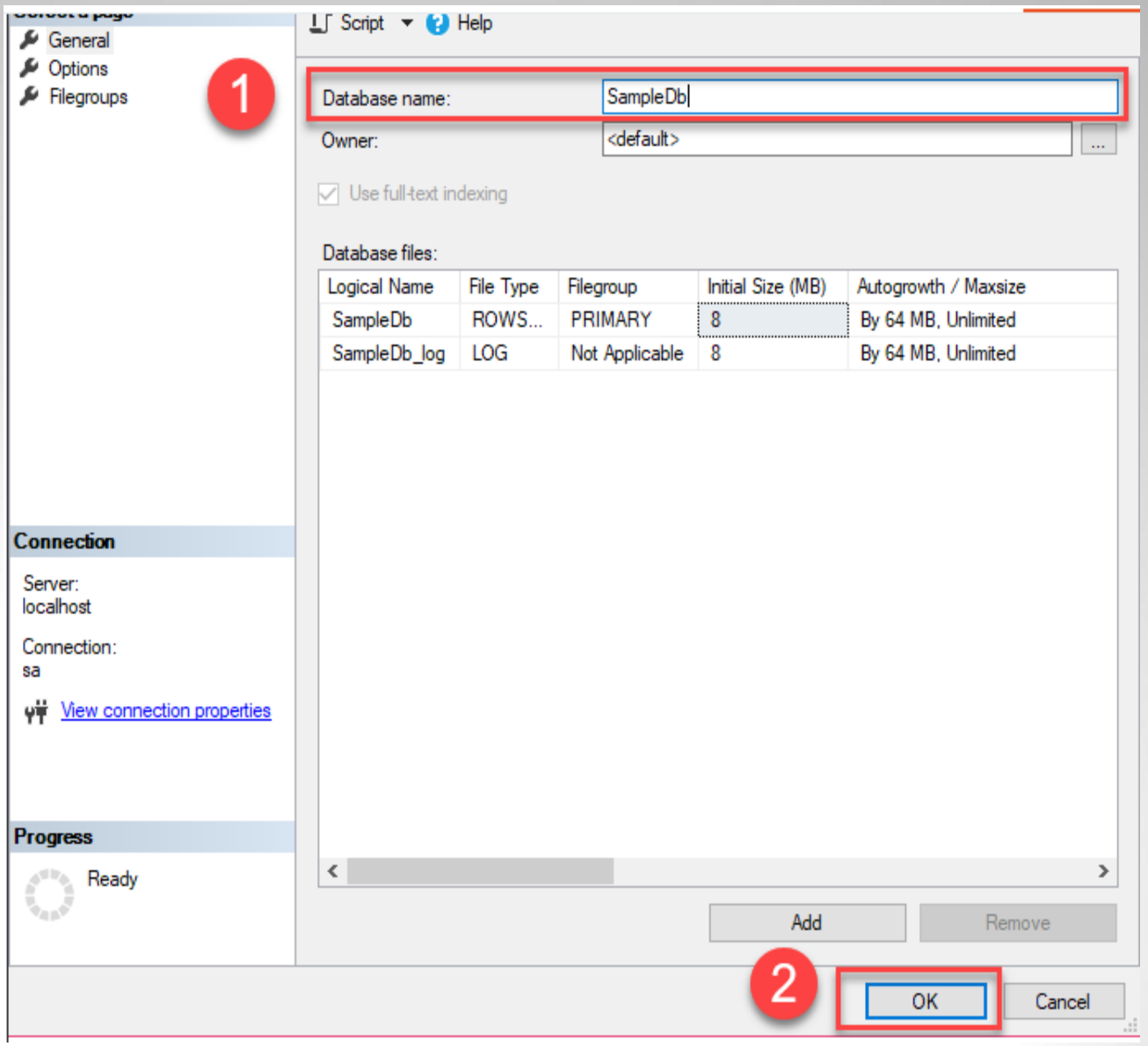
Création de la base de données: SQL Server Management Studio

Tout d'abord, cliquez sur la **base de données** avec le bouton droit de la souris et choisissez l'option de menu **Nouvelle base de données**....

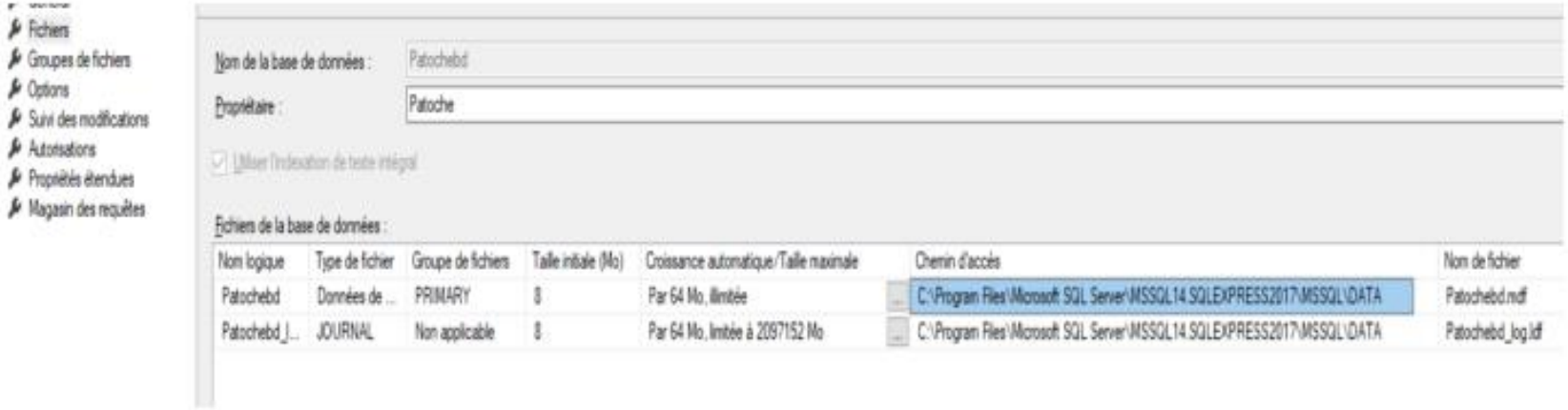


Création de la base de données

Ensuite, entrez le nom de la base de données, par exemple **SampleDb**, et cliquez sur le bouton **OK**.



Création de la base de données



Ces fichiers sont dans :

C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS2017\MSSQL\DATA

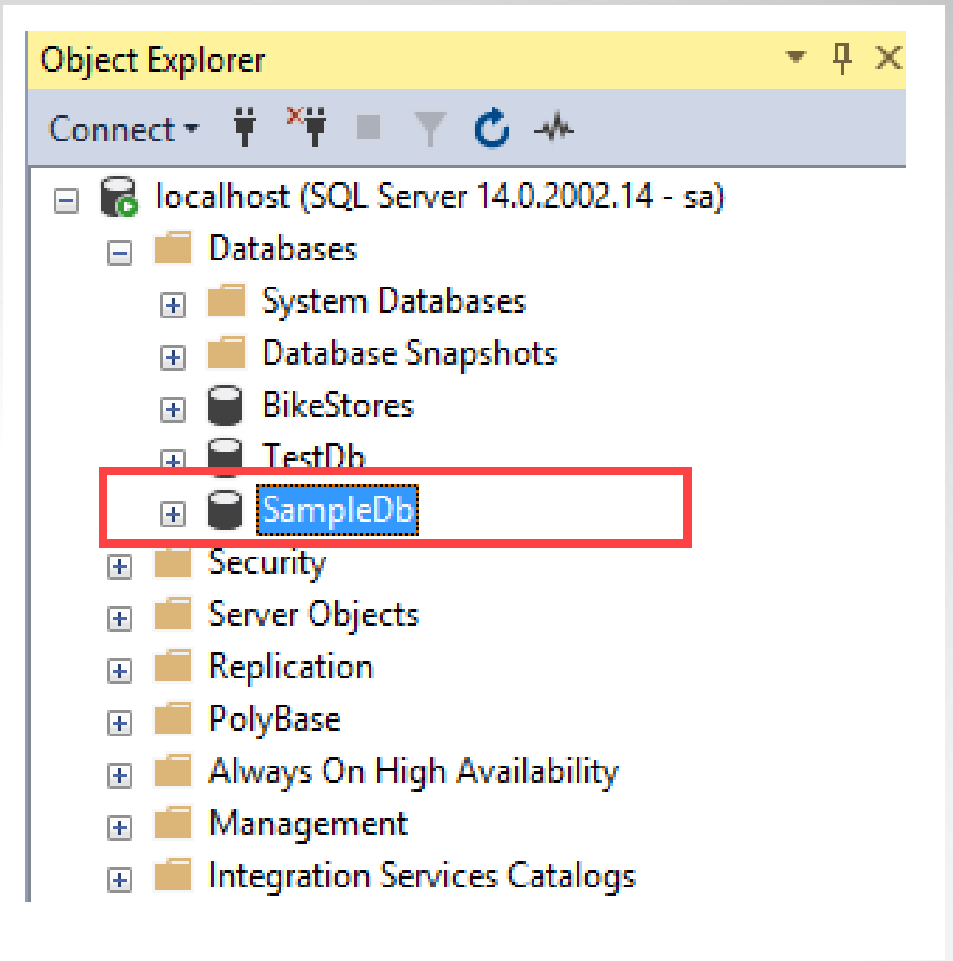
Vous y trouverez deux types de fichiers pour chaque BD : L'un **.mdf** et l'autre. **ldf**

Les données sont stockées dans un fichier MDF, toutes les transactions, les modifications de la base de données SQL Server effectuées par chaque transaction sont stockées dans un fichier LDF

Patochebd.mdf et Patochebd_log.ldf

Création de la base de données

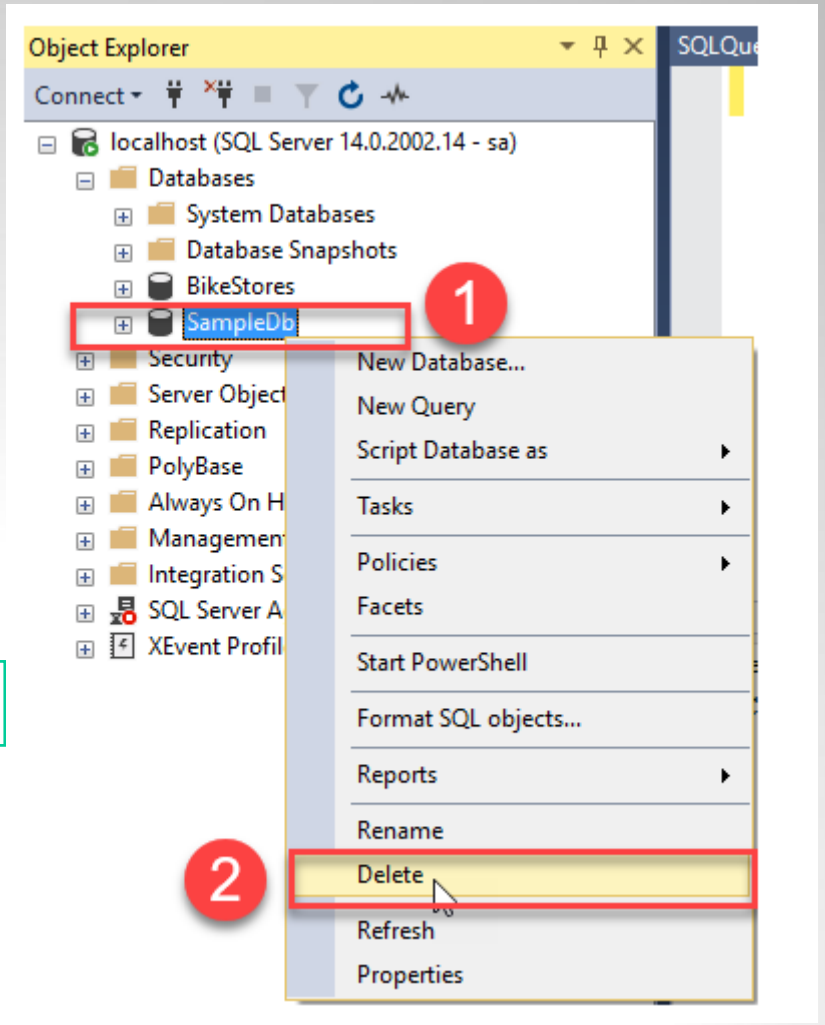
Troisièmement, affichez la base de données nouvellement créée dans **l'Explorateur d'objets** :



Supprimer une base de données

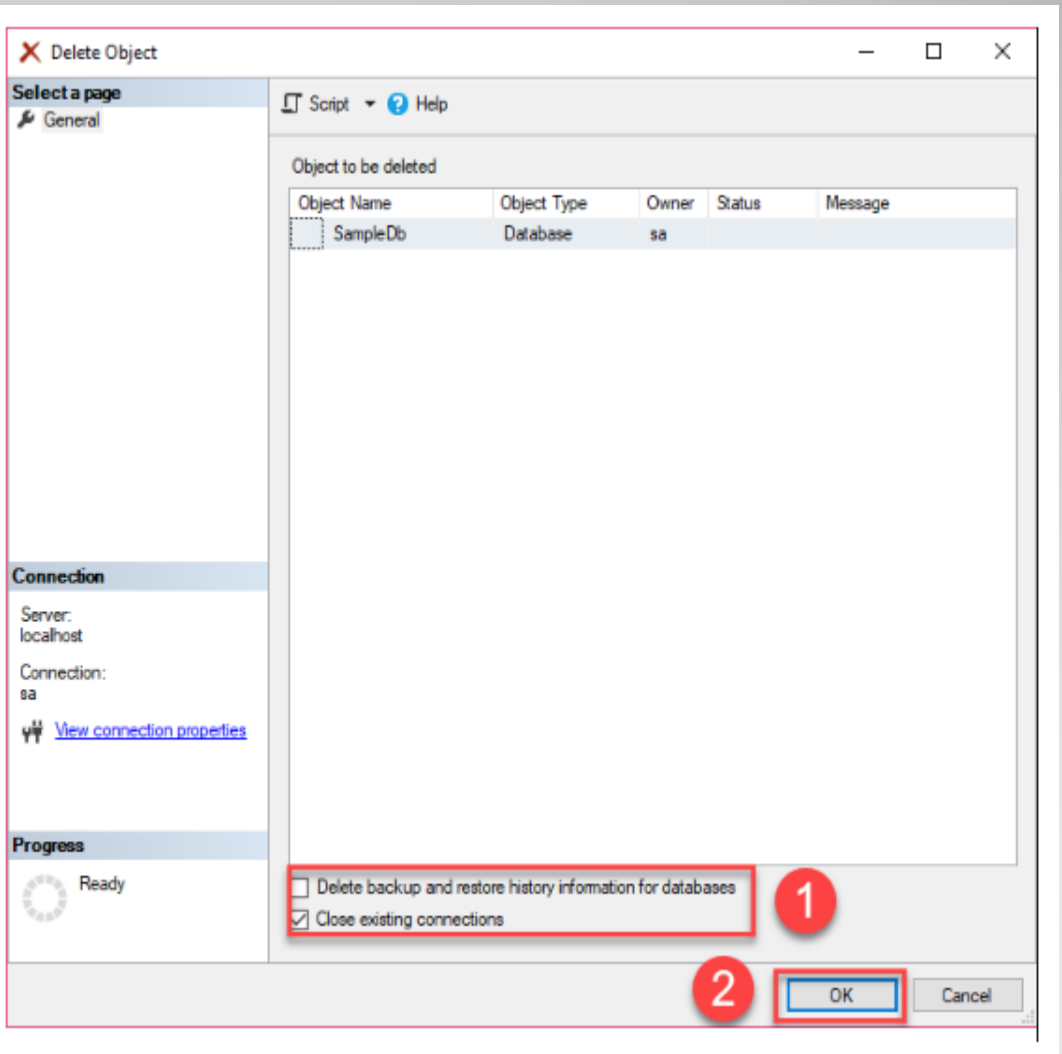
Résumé : dans cette partie, vous apprendrez à supprimer une base de données dans une instance de SQL Server en utilisant l'instruction **DROP DATABASE** et **SQL Server Management Studio**.

```
DROP DATABASE IF EXISTS TestDb;
```



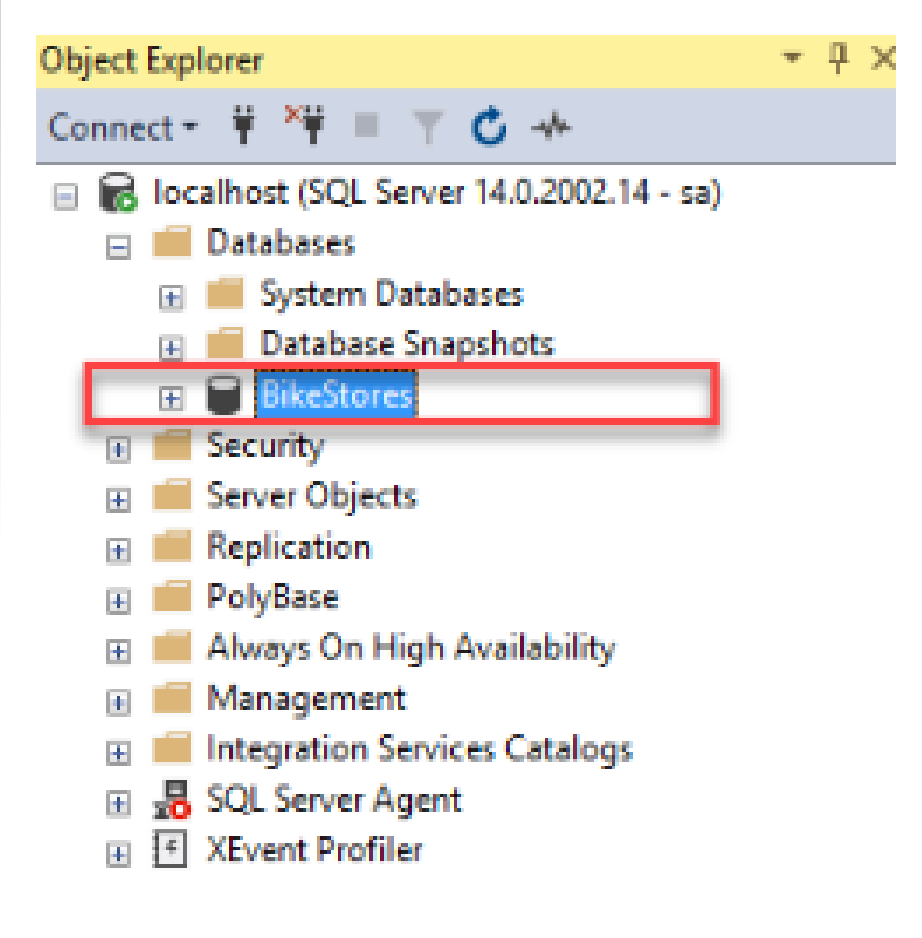
Supprimer une base de données

Ensuite, décochez la case ***Supprimer les informations de l'historique de sauvegarde et de restauration des bases de données****, cochez la case* ***Fermer les connexions existantes*** *et cliquez sur le bouton* ***OK*** *pour supprimer la base de données.*



Supprimer une base de données

Troisièmement, vérifiez que la base de données a été supprimée de ***l'Explorateur d'objets.***



Types des données SQL Server

Types de données SQL Server

Types numériques exacts

Type	À partir de	À
bigint	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
int	-2147483648	2147483647
smallint	-32768	32767
tinyint	0	255
bit	0	1
Decimal	-10 ^ 38 1	10 ^ 38 -1
numeric	-10 ^ 38 1	10 ^ 38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Types des données SQL Server

Numerics approximatif

Type	À partir de	À
float	1,79 E + 308-	1,79 E + 308
reel	-3.40E + 38	3.40E + 38

datetime et smalldatetime

Type	À partir de	À
datetime (3,33 exactitude millisecondes)	1 janvier 1753	31 déc 9999
smalldatetime (précision de 1 minute)	1 janvier 1900	6 juin 2079

Chaînes de caractères

Type	Description
char	De longueur fixe de caractères Unicode avec une longueur maximum de 8000 caractères.
varchar	Texte unicode de longueur variable allant jusqu'à 2 Go.
text	Texte non unicode de longueur maximale 2Go

Types des données SQL Server

Les chaînes de caractères Unicode

Type	Description
nchar	la longueur de données Unicode-fixe avec une longueur maximale de 4000 caractères.
nvarchar	la longueur de données Unicode et variable, avec une longueur maximum de 4000 caractères.
nvarchar (max)	longueur Unicode données variables avec une longueur maximale de 230 caractères (**SQL Server 2005 uniquement**).
ntext	la longueur de données Unicode et variable, avec une longueur maximale de 1073741823 caractères.

Types des données SQL Server (Constraints)

- ✓ ***Unique constraint ➔***
- ✓ ***NOT NULL constraint ➔***
- ✓ ***Primary key ➔***
- ✓ ***CHECK constraint ➔ StudentAge >= 15 AND StudentAge <= 20***
- ✓ ***DEFAULT ➔ 0 or 1 …… True or False***
- ✓ ***FOREING Key ➔***
- ✓ ***XML Data Type……..***

Création des tables

L'instruction suivante ***crée une nouvelle table*** nommée ***Table_Name*** pour suivre les visites des clients en magasin :

```
CREATE TABLE Abdelali_Store (
    visit_id INT,
    first_name VARCHAR (50) NOT NULL,
    last_name VARCHAR (50) NOT NULL,
    visited_at DATETIME,
    phone VARCHAR(20),
    store_id INT NOT NULL,
);
```

Création des tables

La propriété « IDENTITY » d'une table

Vous pouvez mettre en œuvre des colonnes d'identification à l'aide de la propriété **IDENTITY.** Ce qui permet de réaliser un auto incrément sur une colonne. En général, la propriété **IDENTITY** se définie sur la **clé primaire**. **SI** la propriété **IDENTITY** est définie sur une colonne, alors c'est le système qui insères des données dans cette colonne (pas l'utilisateur).

Vous ne pouvez pas modifier une colonne existante pour y ajouter la propriété IDENTITY.

Création des tables

- ✓ Une table ne peut comprendre qu'une colonne définie à l'aide de la propriété **IDENTITY**, et cette colonne doit être définie à l'aide d'un type de données ***decimal, int, numeric,smallint, bigint ou tinyint***.

- ✓ Vous pouvez spécifier la ***valeur de départ*** et ***l'incrément***. La valeur par défaut est ***1*** dans les deux cas.

- ✓ La colonne d'identification ne doit ni accepter les valeurs ***NULL***, ni contenir une définition ou un objet **DEFAULT**. ***En général c'est la clé primaire***.

Création des tables: PRIMARY KEY

L'instruction suivante crée une nouvelle table en utilisant la propriété **IDENTITY** pour la colonne du numéro d'identification personnel :

```
CREATE TABLE Abdelali (
    visit_id INT PRIMARY KEY IDENTITY (1, 1),
    first_name VARCHAR (50) NOT NULL,
    last_name VARCHAR (50) NOT NULL,
    visited_at DATETIME,
    phone VARCHAR(20),
    store_id INT NOT NULL,
);
```

ADD and DROP Tables

ADD Table

```
ALTER TABLE table_name
ADD column_name data_type column_constraint;
```

Si vous souhaitez ajouter plusieurs colonnes à une table en une seule fois à l'aide d'une seule instruction **ALTER TABLE**, vous devez utiliser la syntaxe suivante :

```
ALTER TABLE table_name
ADD
    column_name_1 data_type_1 column_constraint_1,
    column_name_2 data_type_2 column_constraint_2,
    ...,
    column_name_n data_type_n column_constraint_n;
```

DROP Table

```
ALTER TABLE Table_1
    DROP COLUMN column_2;
```

Modifier des tables: ADD

L'instruction suivante ajoute deux nouvelles colonnes nommées **montant** et **nom_client** à la table **Abdelali** :

```
ALTER TABLE Abdelali
   ADD
      amount DECIMAL (10, 2) NOT NULL,
      customer_name VARCHAR (50) NOT NULL;
```

```
ALTER TABLE Table_1
   DROP COLUMN column_2;
```

Modifier des tables: constraint

```
ALTER TABLE table_name DROP PRIMARY KEY;
```

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name
UNIQUE(column1, column2...);
```

```
ALTER TABLE EMPLOYEES ADD CONSTRAINT CONST
UNIQUE(NAME);
```

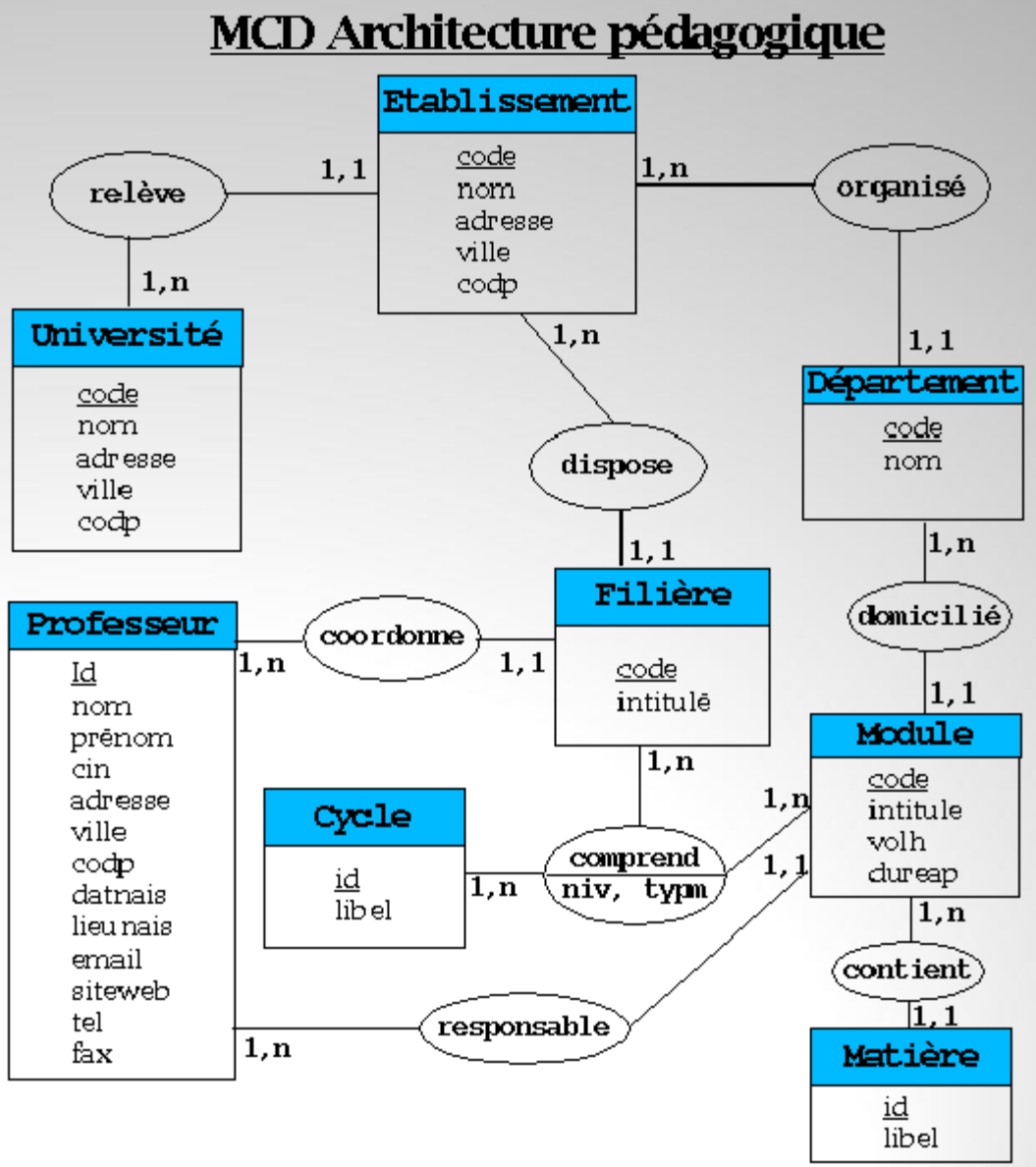
```
ALTER TABLE table_name DROP CONSTRAINT
constraint_name;
```

```
ALTER TABLE table_name RENAME COLUMN old_column_name
to new_column_name;
```

Types des données SQL Server (Foreing Key)

Types of relations

- ✓ ***One to One*** ➔ ***1 - 1***
- ✓ ***One to Many*** ➔ ***1 - N***
- ✓ ***Many to Many*** ➔ ***N - N***



Schéma

Un **schéma** est une collection d'objets de base de données comprenant des tables, des vues, des déclencheurs, des procédures stockées, des index, etc. Un schéma est associé à un nom d'utilisateur, connu sous le nom de propriétaire du schéma, qui est le propriétaire des objets de base de données logiquement liés.

Un schéma appartient toujours à une base de données. En revanche, une base de données peut avoir un ou plusieurs schémas. Par exemple, dans notre exemple de base de données **BikeStores**, nous avons deux schémas : ventes et production. Un objet au sein d'un schéma est qualifié à l'aide du format « **nom_schéma.nom_objet »**, par exemple **ventes.commandes**. Deux tables dans deux schémas peuvent partager le même nom, vous pouvez donc avoir **hr.employees** et **sales.employees**.

✓ *Schémas intégrés dans SQL Server*

SQL Server nous fournit quelques schémas prédéfinis qui ont les mêmes noms que les utilisateurs et les rôles de la base de données intégrée, par exemple : **dbo**, **guest**, **sys**, et **INFORMATION_SCHEMA**.

Notez que SQL Server réserve les schémas **sys** et **INFORMATION_SCHEMA** aux objets du système. Par conséquent, vous ne pouvez pas créer ou supprimer des objets dans ces schémas.

Le **schéma** par défaut d'une base de données nouvellement créée est **dbo**, qui appartient au compte d'utilisateur **dbo**. Par défaut, lorsque vous créez un nouvel utilisateur à l'aide de la commande **CREATE USER**, l'utilisateur prend **dbo** comme schéma par défaut.

✓ *Présentation de l'instruction SQL Server CREATE SCHEMA*

L'instruction **CREATE SCHEMA** permet de créer un nouveau schéma dans la base de données actuelle.

La version simplifiée de l'instruction **CREATE SCHEMA** est illustrée ci-dessous :

```
CREATE SCHEMA schema_name
        [AUTHORIZATION owner_name]
```

Dans cette syntaxe,

Premièrement, indiquez le nom du schéma que vous souhaitez créer dans la clause **CREATE SCHEMA**.

Deuxièmement, spécifiez le propriétaire du schéma après le mot-clé **AUTHORIZATION**.

✓ *Exemple d'instruction SQL Server CREATE SCHEMA*

L'exemple suivant montre comment utiliser l'instruction CREATE SCHEMA pour créer le schéma **customer_services** :

```
CREATE SCHEMA customer_services;
        GO
```

Schéma

Notez que la commande **GO** demande à SQL Server Management Studio d'envoyer les instructions SQL jusqu'à l'instruction GO au serveur pour qu'elles soient exécutées.
Une fois l'instruction exécutée, vous pouvez trouver le schéma nouvellement créé sous **Sécurité > Schémas** du nom de la base de données.

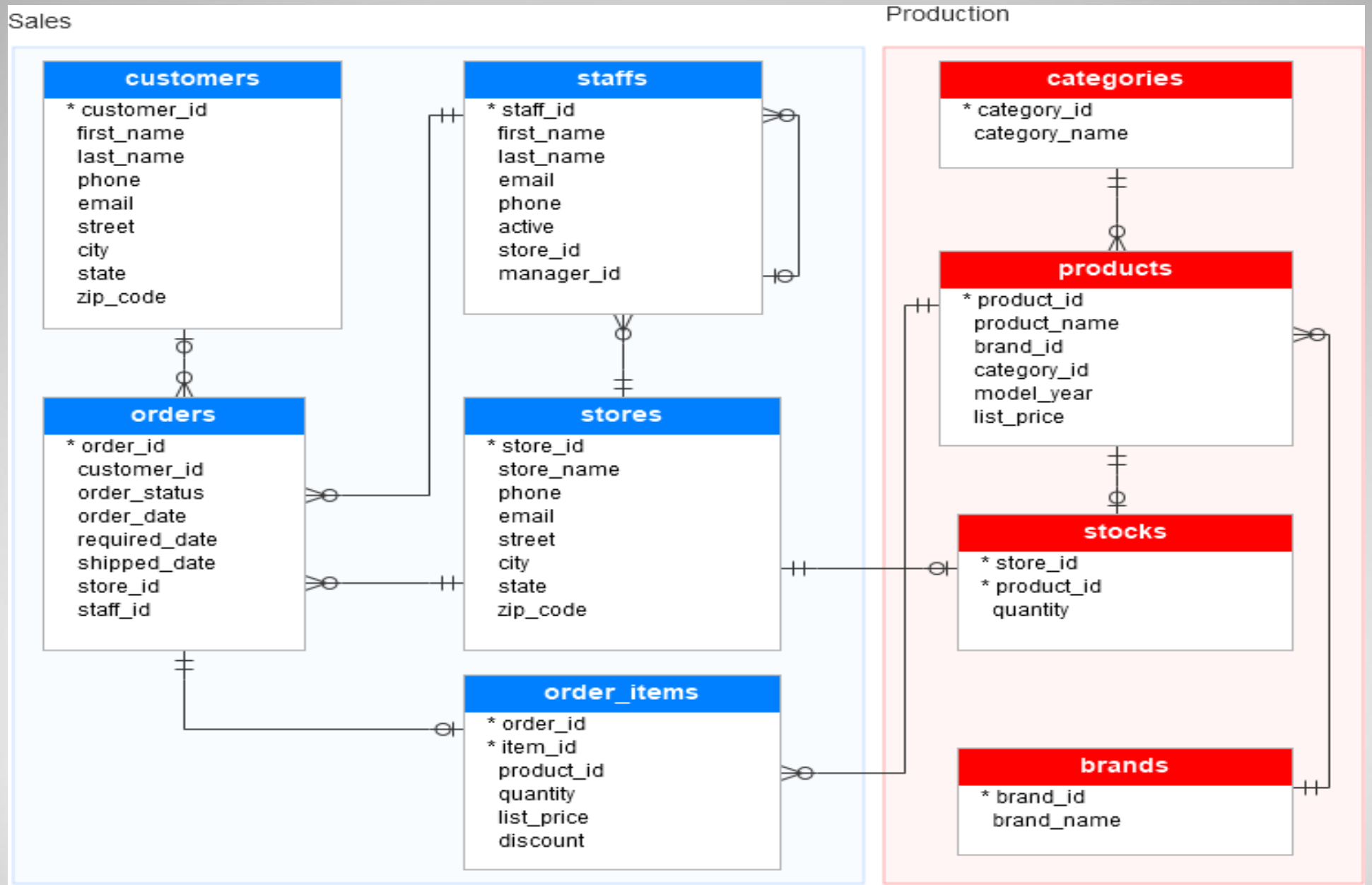
Après avoir créé le schéma **services_clients**, vous pouvez créer des objets pour ce schéma. Par exemple, l'instruction suivante crée une nouvelle table nommée **jobs** dans le schéma **services_clients** :

Schéma

```
CREATE TABLE customer_services.jobs(
    job_id INT PRIMARY KEY IDENTITY,
    customer_id INT NOT NULL,
    description VARCHAR(200),
    created_at DATETIME2 NOT NULL
);
```

Types des données SQL Server

Le schéma suivant illustre le diagramme de la base de données **BikeStores** :



Sample Database: BikeStores

Comme vous pouvez le voir dans le diagramme, la base de données d'exemple **BikeStores** comporte deux schémas, **ventes** et **production**, et ces schémas comportent neuf tables.

1. Tables de la base de données

✓ Table sales.stores

La table ***sales.stores*** contient les informations relatives aux magasins. Chaque magasin possède un nom, des informations de contact telles que le téléphone et l'adresse électronique, ainsi qu'une adresse comprenant la rue, la ville, l'État et le code postal.

Table: sales.staffs

```
CREATE TABLE sales.stores (
        store_id INT IDENTITY (1, 1) PRIMARY KEY,
        store_name VARCHAR (255) NOT NULL,
        phone VARCHAR (25),
        email VARCHAR (255),
        street VARCHAR (255),
        city VARCHAR (255),
        state VARCHAR (10),
        zip_code VARCHAR (5)
);
```

Table: sales.staffs

✓ *Table sales.staffs*

La table **sales.staffs** stocke les informations essentielles du personnel, notamment le prénom et le nom. Elle contient également les informations de communication telles que l'adresse électronique et le numéro de téléphone.

Un employé travaille dans un magasin spécifié par la valeur de la colonne **store_id**. Un magasin peut avoir un ou plusieurs employés. Un employé dépend d'un directeur de magasin spécifié par la valeur de la colonne **manager_id**. Si la valeur de la colonne manager_id est nulle, l'employé est le responsable principal.

Si un employé ne travaille plus pour aucun point de vente, la valeur de la colonne active est fixée à zéro.

Table: sales.staffs

```
CREATE TABLE sales.staffs (
        staff_id INT IDENTITY (1, 1) PRIMARY KEY,
        first_name VARCHAR (50) NOT NULL,
        last_name VARCHAR (50) NOT NULL,
        email VARCHAR (255) NOT NULL UNIQUE,
        phone VARCHAR (25),
        active tinyint NOT NULL,
        store_id INT NOT NULL,
        manager_id INT,
    FOREIGN KEY (store_id)
    REFERENCES sales.stores (store_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (manager_id)
    REFERENCES sales.staffs (staff_id)
    ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

Table: production.categories

✓ *Table production.categories*

La table **production.categories** stocke les catégories du vélo, telles que les vélos pour enfants, les vélos de confort et les vélos électriques.

```
CREATE TABLE production.categories (
        category_id INT IDENTITY (1, 1) PRIMARY KEY,
        category_name VARCHAR (255) NOT NULL
);
```

✓ *Table production.brands*

La table **production.brands** stocke les informations relatives à la marque des vélos, par exemple **Electra**, **Haro** et **Heller**.

```
CREATE TABLE production.brands (
        brand_id INT IDENTITY (1, 1) PRIMARY KEY,
        brand_name VARCHAR (255) NOT NULL
);
```

✓ *Table production.products*

La table **production.products** stocke les informations relatives au produit, telles que le nom, la marque, la catégorie, l'année du modèle et le prix-catalogue.

Chaque produit appartient à une marque spécifiée par la colonne **brand_id**. Une marque peut donc avoir zéro ou plusieurs produits.

Chaque produit appartient également à une catégorie spécifiée par la colonne **category_id**. De même, chaque catégorie peut avoir zéro ou plusieurs produits.

production.products

```
CREATE TABLE production.products (
        product_id INT IDENTITY (1, 1) PRIMARY KEY,
        product_name VARCHAR (255) NOT NULL,
        brand_id INT NOT NULL,
        category_id INT NOT NULL,
        model_year SMALLINT NOT NULL,
        list_price DECIMAL (10, 2) NOT NULL,
    FOREIGN KEY (category_id)
    REFERENCES production.categories (category_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (brand_id)
    REFERENCES production.brands (brand_id)
    ON DELETE CASCADE ON UPDATE CASCADE );
```

✓ *Table sales.customers*

La table **sales.customers** stocke les informations relatives aux clients, notamment le prénom, le nom, le téléphone, l'adresse électronique, la rue, la ville, l'État et le code postal.

sales.customers

```
CREATE TABLE sales.customers (
        customer_id INT IDENTITY (1, 1) PRIMARY KEY,
        first_name VARCHAR (255) NOT NULL,
        last_name VARCHAR (255) NOT NULL,
        phone VARCHAR (25),
        email VARCHAR (255) NOT NULL,
        street VARCHAR (255),
        city VARCHAR (50),
        state VARCHAR (25),
        zip_code VARCHAR (5)
);
```

✓ *Table sales.orders*

La table **sales.orders** stocke les informations d'en-tête de la commande client, notamment le client, le statut de la commande, la date de la commande, la date requise et la date d'expédition.

Elle contient également des informations sur l'endroit où la transaction commerciale a été créée (magasin) et sur la personne qui l'a créée (personnel).

Chaque commande client possède une ligne dans la table **sales_orders**. Une commande client comporte un ou plusieurs postes enregistrés dans la table **sales.order_items**.

Table sales.orders

```
CREATE TABLE sales.orders (
        order_id INT IDENTITY (1, 1) PRIMARY KEY,
        customer_id INT,
        order_status tinyint NOT NULL,
-- Order status: 1 = Pending; 2 = Processing; 3 = Rejected; 4 = Completed
        order_date DATE NOT NULL,
        required_date DATE NOT NULL,
```

Table: sales.order_items

```
        shipped_date DATE,
        store_id INT NOT NULL,
        staff_id INT NOT NULL,
        FOREIGN KEY (customer_id)
        REFERENCES sales.customers (customer_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (store_id)
        REFERENCES sales.stores (store_id)
        ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (staff_id)
        REFERENCES sales.staffs (staff_id)
        ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

Table: sales.order_items

La table **sales.order_items** contient les postes d'une commande client.

Chaque poste appartient à une commande client spécifiée par la colonne **order_id**.

Un poste de commande client comprend le produit, la quantité commandée, le prix catalogue et la remise.

```
CREATE TABLE sales.order_items(
    order_id INT,
    item_id INT,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    list_price DECIMAL (10, 2) NOT NULL,
    discount DECIMAL (4, 2) NOT NULL DEFAULT 0,
```

Table: sales.order_items

```
PRIMARY KEY (order_id, item_id),
        FOREIGN KEY (order_id)
         REFERENCES sales.orders (order_id)
         ON DELETE CASCADE ON UPDATE CASCADE,
         FOREIGN KEY (product_id)
         REFERENCES production.products (product_id)
         ON DELETE CASCADE ON UPDATE CASCADE
);
```

Table: production.stocks

✓ *Table production.stocks*

La table **production.stocks** stocke les informations relatives aux stocks, c'est-à-dire la quantité d'un produit particulier dans un magasin spécifique.

Table: production.stocks

```
CREATE TABLE production.stocks (
        store_id INT,
        product_id INT,
        quantity INT,
        PRIMARY KEY (store_id, product_id),
        FOREIGN KEY (store_id)
         REFERENCES sales.stores (store_id)
         ON DELETE CASCADE ON UPDATE CASCADE,
         FOREIGN KEY (product_id)
          REFERENCES production.products (product_id)
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

Contrainte de **clé étrangère**

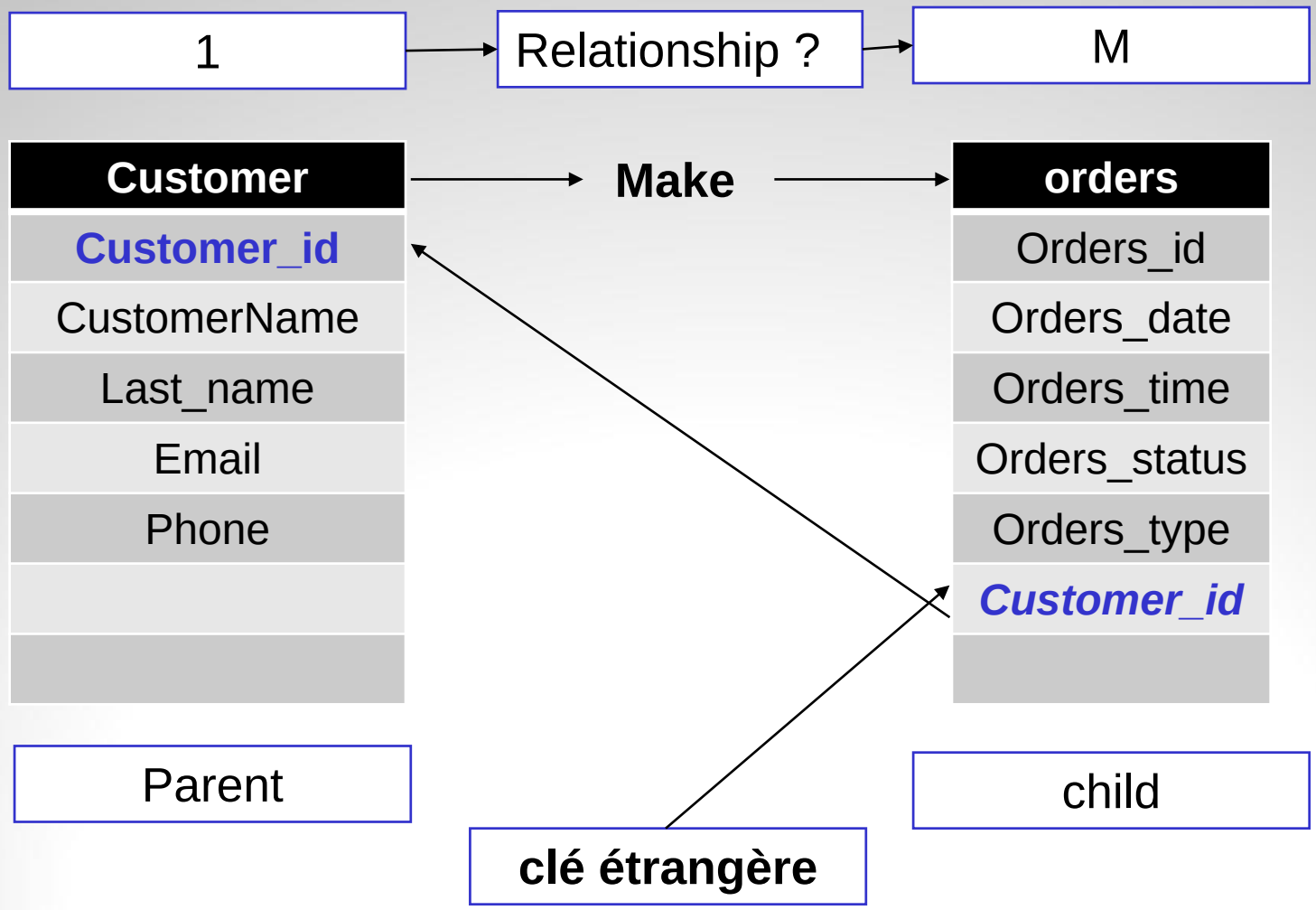
Considérons les tables **vendor_groups** et **vendors** suivantes :

```
CREATE TABLE procurement.vendor_groups (
    group_id INT IDENTITY PRIMARY KEY,
    group_name VARCHAR (100) NOT NULL
);


CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
);
```

Contrainte de **clé étrangère**

Considérons les tables **vendor_groups** et **vendors** suivantes :



Syntaxe de la contrainte **FOREIGN KEY**

La syntaxe générale pour créer une contrainte **FOREIGN KEY** est la suivante :

```
CONSTRAINT fk_constraint_name  FOREIGN KEY (column_1, column2,...)
REFERENCES parent_table_name(column1, column2,..)
```

Tout d'abord, indiquez le nom de la contrainte **FOREIGN KEY** après le mot-clé **CONSTRAINT**. Le nom de la contrainte est facultatif. Il est donc possible de définir une contrainte de clé étrangère comme suit :

```
FOREIGN KEY (column_1, column2,...)
REFERENCES parent_table_name(column1,column2,..)
```

Contrainte de **clé étrangère**

Chaque fournisseur appartient à un groupe de fournisseurs et chaque groupe de fournisseurs peut avoir zéro ou plusieurs fournisseurs. La relation entre les tables **vendor_groups** et **vendors** est de type un à plusieurs (**one-to-many)**.

Pour chaque ligne de la table **vendeurs**, vous pouvez toujours trouver une ligne correspondante dans la table **vendor_groups**.

Contrainte de **clé étrangère**

Toutefois, avec la configuration actuelle des tables, vous pouvez insérer une ligne dans la table **vendeurs** sans qu'il y ait de ligne correspondante dans la table **vendor_groups**.

De même, vous pouvez également supprimer une ligne dans la table **vendor_groups** sans mettre à jour ou supprimer les lignes correspondantes dans la table **vendeurs**, ce qui se traduit par des lignes orphelines dans la table **vendeurs**.

Pour renforcer le lien entre les données des tables **vendor_groups** et **vendors**, vous devez établir une **clé étrangère** dans la table **vendors**.

Contrainte de **clé étrangère**

Une **clé étrangère** est une colonne ou un groupe de colonnes d'une table qui identifie de manière unique une ligne d'une autre table (ou de la même table en cas d'autoréférence).

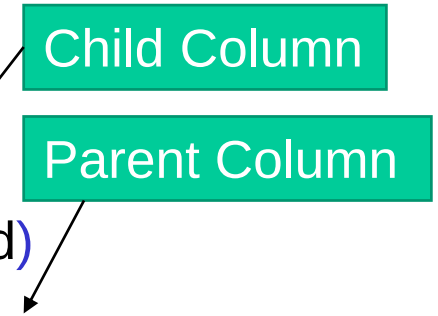
*Pour créer une clé étrangère, vous utilisez la contrainte **FOREIGN KEY**.*

Les instructions suivantes suppriment la table **vendeurs** et la recréent avec une contrainte **FOREIGN KEY** :

Contrainte de **clé étrangère**

```
DROP TABLE vendors;


CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
        CONSTRAINT fk_group FOREIGN KEY (group_id)
        REFERENCES procurement.vendor_groups(group_id)
);
```



Contrainte de **clé étrangère**

La table **vendor_groups** est désormais appelée table **parent**, c'est-à-dire la table à laquelle la contrainte de **clé étrangère** fait référence. La table **vendeurs** est appelée table **enfant**, c'est-à-dire la table à laquelle la contrainte de **clé étrangère** est appliquée.

Dans l'instruction ci-dessus, la clause suivante crée une contrainte de **clé étrangère** nommée **fk_group** qui lie le **group_id** de la table **vendeurs** au **group_id** de la table **vendor_groups** :

```
CONSTRAINT fk_group FOREIGN KEY (group_id)
REFERENCES procurement.vendor_groups(group_id)
```

Syntaxe de la contrainte **FOREIGN KEY**

Dans ce cas, SQL Server génère automatiquement un nom pour la contrainte **FOREIGN KEY**.

Deuxièmement, indiquez une liste de colonnes de **clé étrangère** séparées par des virgules et placées entre parenthèses après le mot-clé **FOREIGN KEY.**

Troisièmement, indiquez le nom de la table parent à laquelle la **clé étrangère** fait référence et une liste de colonnes séparées par des virgules ayant un lien avec la colonne de la table enfant.

Exemple de contrainte **FOREIGN KEY**

Tout d'abord, insérez quelques lignes dans la table **vendor_groups** :

```
INSERT INTO procurement.vendor_groups(group_name)
VALUES('Abdelali'),
      ('Smail'),
      ('Fatima');
```

Deuxièmement, insérez un nouveau fournisseur avec un groupe de fournisseurs dans la table des fournisseurs :

```
INSERT INTO procurement.vendors(vendor_name, group_id)
VALUES('Ali',1);
```

Exemple de contrainte **FOREIGN KEY**

Troisièmement, essayez d'insérer un nouveau fournisseur dont le groupe de fournisseurs n'existe pas dans la table **vendor_groups** :

```
INSERT INTO procurement.vendors(vendor_name, group_id)
VALUES('XYZ',4);
```

SQL Server a émis l'erreur suivante :

```
The INSERT statement conflicted with the FOREIGN KEY constraint
"fk_group". The conflict occurred in database "BikeStores", table
"procurement.vendor_groups", column 'group_id'.
```

Dans cet exemple, en raison de la contrainte **FOREIGN KEY**, SQL Server a rejeté l'insertion et a émis une erreur.

Actions référentielles

La contrainte de **clé étrangère** garantit l'intégrité référentielle. Cela signifie que vous ne pouvez insérer une ligne dans la table **enfant** que s'il existe une ligne correspondante dans la table **parent**.

En outre, la contrainte de **clé étrangère** vous permet de définir les actions référentielles lorsque la ligne de la table **parent** est mise à jour ou supprimée, comme suit :

```
FOREIGN KEY (foreign_key_columns)
    REFERENCES parent_table(parent_key_columns)
    ON UPDATE action
    ON DELETE action;
```

Actions référentielles

Les options **ON UPDATE** et **ON DELETE** spécifient l'action qui sera exécutée lorsqu'une ligne de la table **parent** est **mise à jour** ou **supprimée**.

Les actions autorisées sont les suivantes : **NO ACTION**, **CASCADE**, **SET NULL** et **SET DEFAULT**

➔ Actions de suppression des lignes de la table **parente**

Si vous supprimez une ou plusieurs lignes de la table **parente**, vous pouvez définir l'une des actions suivantes :

Actions référentielles

ON DELETE NO ACTION : SQL Server affiche une erreur et **annule** l'action de suppression de la ligne dans la table **parent.**

ON DELETE CASCADE : SQL Server supprime les lignes de la table **enfant** correspondant à la ligne supprimée de la table **parent**.

ON DELETE SET NULL : SQL Server attribue la valeur **NULL** aux lignes de la table **enfant** si les lignes correspondantes de la table **parent** sont supprimées. Pour exécuter cette action, les colonnes de la clé étrangère doivent être nulles.

Actions référentielles

ON DELETE SET DEFAULT SQL Server définit les lignes de la table **enfant** à leurs valeurs par défaut si les lignes correspondantes de la table **parent** sont supprimées. Pour exécuter cette action, les colonnes de clés étrangères doivent avoir des définitions par défaut. Notez qu'une colonne nullable a une valeur par défaut de **NULL** si aucune valeur par défaut n'est spécifiée.

Par défaut, SQL Server applique **DELETE NO ACTION** si vous ne spécifiez pas explicitement une action.

Actions référentielles

*Action de **mise à jour** des lignes de la table parent*

Si vous mettez à jour une ou plusieurs lignes de la table **parent**, vous pouvez définir l'une des actions suivantes :

ON UPDATE NO ACTION : SQL Server affiche une erreur et annule l'action de **mise à jour** sur la ligne de la table **parent.**

ON UPDATE CASCADE : SQL Server met à jour les lignes correspondantes dans la table **enfant** lorsque les lignes de la table **parent** sont mises à jour.

Actions référentielles

ON UPDATE SET NULL : SQL Server met les lignes de la table **enfant** à **NULL** lorsque la ligne correspondante de la table **parent** est mise à jour.

*Notez que les colonnes de la clé étrangère doivent être **nulles** pour que cette action soit exécutée.*

ON UPDATE SET DEFAULT : SQL Server définit les valeurs par défaut pour les lignes de la table **enfant** dont les lignes correspondantes de la table **parent** ont été mises à jour.

SELECT INTO

L'instruction **SELECT INTO** crée une nouvelle table et y insère les lignes de la requête.

L'instruction **SELECT INTO** suivante crée la table de destination et copie les lignes, qui satisfont à la condition **WHERE**, de la table source vers la table de destination :

```
SELECT
    select_list
INTO
    destination
FROM
    source
[WHERE condition]
```

SELECT INTO

Si vous souhaitez copier les données partielles de la table source, utilisez la clause **WHERE** pour spécifier les lignes à copier. De même, vous pouvez spécifier les colonnes de la table source à copier dans la table de destination en les spécifiant dans la liste de sélection.

Notez que l'instruction **SELECT INTO** ne copie pas les contraintes telles que la *clé primaire* et les index de la table source vers la table de destination.

Exemples

A) Exemple d'utilisation de l'instruction **SELECT INTO** pour copier une table dans la même base de données
Tout d'abord, créez un nouveau schéma pour stocker la nouvelle table.

```
CREATE SCHEMA marketing;
GO
```

Ensuite, créez la table **marketing.customers** comme la table **sales.customers** et copiez toutes les lignes de la table **sales.customers** dans la table **marketing.customers** :

```
SELECT
    *
INTO
    marketing.customers
FROM
    sales.customers;
```

SELECT INTO

Troisièmement, interrogez les données de la table **marketing.customers** pour vérifier la copie :

```
SELECT
    *
FROM
    marketing.customers;
```

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950
11	Deshawn	Mendoza	NULL	deshawn.mendoza@yahoo.com	8790 Cobblestone Street	Monsey	NY	10952
12	Robby	Sykes	(516) 583-7761	robby.sykes@hotmail.com	486 Rock Maple Street	Hempstead	NY	11550
13	Lashawn	Ortiz	NULL	lashawn.ortiz@msn.com	27 Washington Rd.	Longview	TX	75604
14	Garry	Espinoza	NULL	garry.espinoza@hotmail.com	7858 Rockaway Court	Forney	TX	75126

SELECT INTO

B) Utilisation de l'instruction **SELECT INTO** pour copier une table dans plusieurs bases de données.
Tout d'abord, créez une nouvelle base de données nommée **TestDb** pour les tests :

```
CREATE DATABASE TestDb;
GO
```

Deuxièmement, copiez le fichier **sales.customers** de la base de données actuelle (**BikeStores**) vers la table **TestDb.dbo.customers**. Cette fois, nous nous contentons de copier l'identification du client, le prénom, le nom et l'adresse électronique des clients établis en Californie :

SELECT INTO

```
SELECT
    customer_id,
    first_name,
    last_name,
    email
INTO
    TestDb.dbo.customers

FROM
    sales.customers

WHERE
    state = 'CA';
```

Troisièmement, interrogez les données de **TestDb.dbo.customers** pour vérifier la copie :

SELECT INTO

Troisièmement, interrogez les données de **TestDb.dbo.customers** pour vérifier la copie :

```
SELECT
    *
FROM
    TestDb.dbo.customers;
```

customer_id	first_name	last_name	email
2	Kasha	Todd	kasha.todd@yahoo.com
3	Tameka	Fisher	tameka.fisher@aol.com
5	Charolette	Rice	charolette.rice@msn.com
24	Corene	Wall	corene.wall@msn.com
30	Jamaal	Albert	jamaal.albert@gmail.com
31	Williemae	Holloway	williemae.holloway@msn.com
32	Araceli	Golden	araceli.golden@msn.com
33	Deloris	Burke	deloris.burke@hotmail.com
40	Ronna	Butler	ronna.butler@gmail.com
46	Monika	Berg	monika.berg@gmail.com
47	Bridgette	Guerra	bridgette.guerra@hotmail.com
53	Saturnina	Gamer	saturnina.gamer@gmail.com
60	Neil	Mccall	neil.mccall@gmail.com
67	Tommie	Melton	tommie.melton@gmail.com

DML: SELECT

Les tables sont des objets qui stockent toutes les données d'une base de données. Dans une table, les données sont organisées logiquement sous forme de lignes et de colonnes, comme dans un tableur.

Chaque ligne représente un enregistrement unique dans une table et chaque colonne représente un champ de l'enregistrement. Par exemple, la table Clients contient des données sur les clients telles que le numéro d'identification du client, le prénom, le nom, le téléphone, l'adresse électronique et l'adresse postale, comme indiqué ci-dessous :

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950

DML: SELECT

SQL Server utilise des schémas pour regrouper logiquement les tables et autres objets de la base de données. Dans notre exemple de base de données, nous avons deux schémas : **ventes** et **production**. Le schéma des **ventes** regroupe toutes les tables liées aux ventes, tandis que le schéma de **production** regroupe toutes les tables liées à la production.

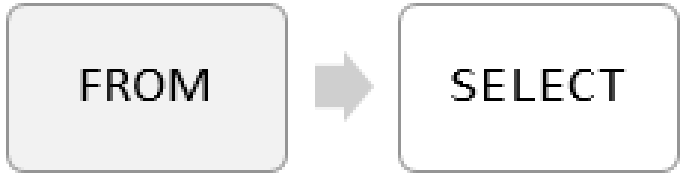
Pour interroger les données d'une table, vous utilisez l'instruction **SELECT**. La figure suivante illustre la forme la plus élémentaire de l'instruction **SELECT** :

```
SELECT
    select_list
FROM
    schema_name.table_name;
```

DML: SELECT

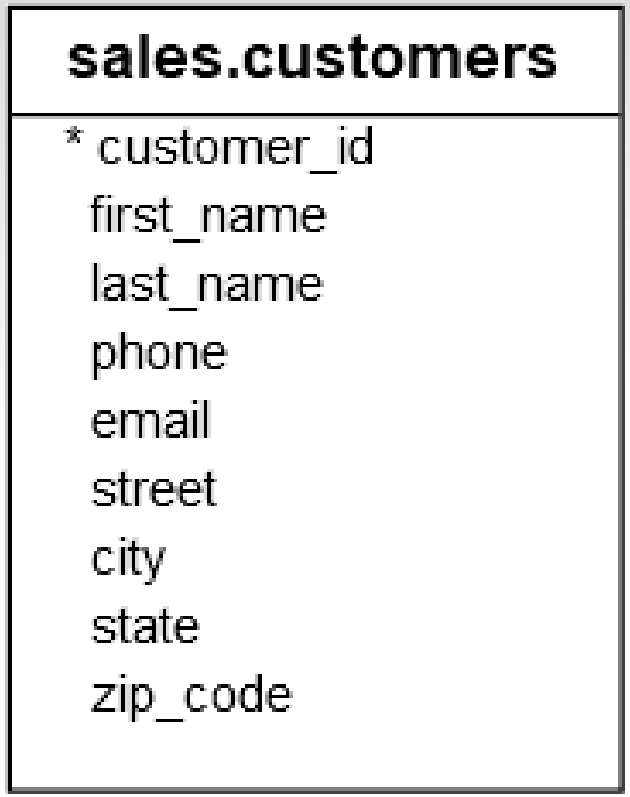
Dans cette syntaxe :

Premièrement, spécifiez une liste de colonnes séparées par des virgules à partir desquelles vous souhaitez interroger les données dans la clause **SELECT**.
Deuxièmement, indiquez la table source et son nom de schéma dans la clause FROM.
Lors du traitement de l'instruction **SELECT**, SQL Server traite d'abord la clause FROM, puis la clause SELECT, même si cette dernière apparaît en premier dans la requête.



DML: SELECT ➔Exemple

Pour la démonstration, nous utiliserons la table "**clients**" de la base de données d'exemple.



DML: SELECT

A) Récupérer certaines colonnes d'une table.

La requête suivante permet de trouver le prénom et le nom de famille de tous les clients :

```
SELECT
    first_name,
    last_name
FROM
    sales.customers;
```

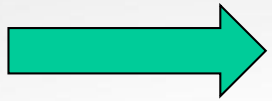
first_name	last_name
Debra	Burks
Kasha	Todd
Tameka	Fisher
Daryl	Spence
Charolette	Rice
Lyndsey	Bean
Latasha	Hays
Jacquline	Duncan
Genoveva	Baldwin
Pamelia	Newman
Deshawn	Mendoza

DML: SELECT

Le résultat d'une requête est appelé ensemble de résultats.

L'instruction suivante renvoie les prénoms, les noms et les adresses électroniques de tous les clients :

```
SELECT
    first_name,
    last_name,
    email
FROM
    sales.customers;
```



first_name	last_name	email
Debra	Burks	debra.burks@yahoo.com
Kasha	Todd	kasha.todd@yahoo.com
Tameka	Fisher	tameka.fisher@aol.com
Daryl	Spence	daryl.spence@aol.com
Charolette	Rice	charolette.rice@msn.com
Lyndsey	Bean	lyndsey.bean@hotmail.com
Latasha	Hays	latasha.hays@hotmail.com
Jacquline	Duncan	jacquline.duncan@yahoo.com
Genoveva	Baldwin	genoveva.baldwin@msn.com
Pamelia	Newman	pamelia.newman@gmail.com
Deshawn	Mendoza	deshawn.mendoza@yahoo.com
Robby	Sykes	robby.sykes@hotmail.com

DML: SELECT

B) Récupérer toutes les colonnes d'une table.

Pour obtenir les données de toutes les colonnes d'un tableau, vous pouvez spécifier toutes les colonnes dans la liste de sélection. Vous pouvez également utiliser **SELECT** * comme raccourci pour économiser de la frappe :

```
SELECT
    *
FROM
    sales.customers;
```

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950
11	Deshawn	Mendoza	NULL	deshawn.mendoza@yahoo.com	8790 Cobblestone Street	Monsey	NY	10952
12	Robby	Sykes	(516) 583-7761	robby.sykes@hotmail.com	486 Rock Maple Street	Hempstead	NY	11550

DML: SELECT

C) Trier l'ensemble des résultats:

Pour filtrer les lignes en fonction d'une ou de plusieurs conditions, vous utilisez une clause **WHERE**, comme le montre l'exemple suivant :

```
SELECT
    *
FROM
    sales.customers
WHERE
    state = 'CA';
```

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
24	Corene	Wall	NULL	corene.wall@msn.com	9601 Ocean Rd.	Atwater	CA	95301
30	Jamaal	Albert	NULL	jamaal.albert@gmail.com	853 Stonybrook Street	Torrance	CA	90505
31	Williemae	Holloway	(510) 246-8375	williemae.holloway@msn.com	69 Cypress St.	Oakland	CA	94603
32	Araceli	Golden	NULL	araceli.golden@msn.com	12 Ridgeview Ave.	Fullerton	CA	92831
33	Deloris	Burke	NULL	deloris.burke@hotmail.com	895 Edgemont Drive	Palos Verdes Peninsula	CA	90274
40	Ronna	Butler	NULL	ronna.butler@gmail.com	9438 Plymouth Court	Encino	CA	91316
46	Monika	Berg	NULL	monika.berg@gmail.com	369 Vernon Dr.	Encino	CA	91316
47	Bridgette	Guerra	NULL	bridgette.guerra@hotmail.com	9982 Manor Drive	San Lorenzo	CA	94580

DML: SELECT

L'instruction suivante **trie** la liste des clients par ordre **croissant** de prénom :

```
SELECT
    *
FROM
    sales.customers
WHERE
    state = 'CA'
ORDER BY
    first_name (ASC);
```

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
673	Adam	Henderson	NULL	adam.henderson@hotmail.com	167 James St.	Los Banos	CA	93635
1261	Adelaida	Hancock	NULL	adelaida.hancock@aol.com	669 S. Gartner Street	San Pablo	CA	94806
574	Adriene	Rivera	NULL	adriene.rivera@hotmail.com	973 Yukon Avenue	Encino	CA	91316
1353	Agatha	Daniels	NULL	agatha.daniels@yahoo.com	125 Canal St.	South El Monte	CA	91733
735	Aide	Franco	NULL	aide.franco@msn.com	8017 Lake Forest St.	Atwater	CA	95301
952	Aileen	Marquez	NULL	aileen.marquez@msn.com	8899 Newbridge Street	Torrance	CA	90505
697	Alane	Mccarty	(619) 377-8586	alane.mccarty@hotmail.com	8254 Hilldale Street	San Diego	CA	92111
562	Alejandro	Norman	NULL	alejandro.norman@yahoo.com	8918 Marsh Lane	Upland	CA	91784
1288	Allie	Conley	NULL	allie.conley@msn.com	96 High Point Road	Lawndale	CA	90260
701	Alysia	Nicholson	(805) 493-7311	alysia.nicholson@hotmail.com	868 Trusel St.	Oxnard	CA	93035
619	Ana	Palmer	(657) 323-8684	ana.palmer@yahoo.com	7 Buckingham St.	Anaheim	CA	92806
947	Angele	Castro	NULL	angele.castro@yahoo.com	15 Acacia Drive	Palos Verdes Peninsula	CA	90274

DML: SELECT

D) regrouper les lignes

Pour regrouper les lignes en groupes, vous utilisez la clause **GROUP BY**. Par exemple, l'instruction suivante renvoie toutes les villes des clients situés en Californie et le nombre de clients dans chaque ville.

```
SELECT
    city,
    COUNT (*)
FROM
    sales.customers
WHERE
    state = 'CA'
GROUP BY
    city
ORDER BY
    city;
```

city	(No column name)
Anaheim	11
Apple Valley	11
Atwater	5
Bakersfield	5
Banning	7
Campbell	10
Canyon Country	12
Coachella	6
Duarte	9
Encino	8
Fresno	5
Fullerton	6
Glendora	8

E) Groupes de filtrage

```
SELECT
    city,
    COUNT (*)
FROM
    sales.customers
WHERE
    state = 'CA'
GROUP BY
    city
HAVING
    COUNT (*) > 10
ORDER BY
    city;
```

Pour filtrer des groupes en fonction d'une ou plusieurs conditions, vous utilisez la clause **HAVING**. L'exemple suivant renvoie la ville de Californie qui compte plus de dix clients :

city	(No column name)
Anaheim	11
Apple Valley	11
Canyon Country	12
South El Monte	11
Upland	11

E) Groupes de filtrage

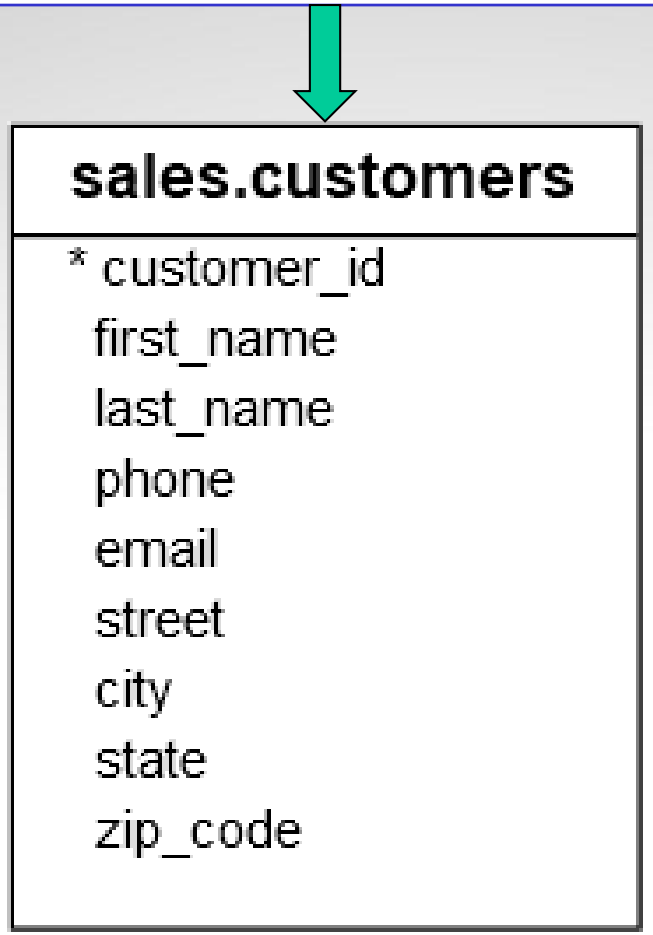
```
SELECT
    city,
    COUNT (*)
FROM
    sales.customers
WHERE
    state = 'CA'
GROUP BY
    city
HAVING
    COUNT (*) > 10
ORDER BY
    city;
```

Pour filtrer des groupes en fonction d'une ou plusieurs conditions, vous utilisez la clause **HAVING**. L'exemple suivant renvoie la ville de Californie qui compte plus de dix clients :

city	(No column name)
Anaheim	11
Apple Valley	11
Canyon Country	12
South El Monte	11
Upland	11

DML: SELECT ➔ ORDER BY

Nous utiliserons la table clients de la base de données d'exemple de la démonstration.



SELECT ➔ ORDER BY

A) Trier un ensemble de résultats en fonction d'une colonne par ordre croissant

L'instruction suivante **trie** la liste des clients par ordre **croissant** de prénom :

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    first_name (ASC);
```

first_name	last_name
Aaron	Knapp
Abbey	Pugh
Abby	Gamble
Abram	Copeland
Adam	Henderson
Adam	Thornton
Addie	Hahn

SELECT ➔ ORDER BY

B) Trier un ensemble de résultats en fonction d'une colonne dans l'ordre décroissant.

L'instruction suivante **trie** la liste des clients par ordre **décroissant** de prénom :

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    first_name DESC;
```

first_name	last_name
Zulema	Browning
Zulema	Clemons
Zoraida	Patton
Zora	Ford
Zona	Cameron
Zina	Bonner
Zenia	Bruce
Zelma	Browning

C) Trier un ensemble de résultats en fonction de plusieurs colonnes

L'instruction suivante récupère le prénom, le nom et la ville des clients. Elle trie la liste des clients d'abord par la ville, puis par le prénom.

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    city,
    first_name;
```

city	first_name	last_name
Albany	Douglass	Blankenship
Albany	Mi	Gray
Albany	Priscilla	Wilkins
Amarillo	Andria	Rivers
Amarillo	Delaine	Estes
Amarillo	Jonell	Rivas
Amarillo	Luis	Tyler
Amarillo	Narcisa	Knapp

SELECT ➔ ORDER BY

D) Trier un ensemble de résultats en fonction de plusieurs colonnes et dans des ordres différents.

L'instruction suivante trie les clients par ville dans l'ordre décroissant, puis trie l'ensemble des résultats triés par prénom dans l'ordre croissant.

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    city DESC,
    first_name ASC;
```

city	first_name	last_name
Yuba City	Louanne	Martin
Yorktown Heights	Demarcus	Reese
Yorktown Heights	Jenna	Saunders
Yorktown Heights	Latricia	Lindsey
Yorktown Heights	Shasta	Combs
Yorktown Heights	Shauna	Edwards
Yonkers	Aaron	Knapp
Yonkers	Alane	Munoz

SELECT ➔ ORDER BY

E) Trier un ensemble de résultats par une colonne qui n'est pas dans la liste de sélection

Il est possible de **trier** l'ensemble des résultats en fonction d'une colonne qui **ne figure pas** dans la liste de sélection. Par exemple, l'instruction suivante trie le client en fonction de l'État, bien que la colonne État n'apparaisse pas dans la liste de sélection.

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    state;
```

city	first_name	last_name
Sacramento	Charolette	Rice
Campbell	Kasha	Todd
Redondo Beach	Tameka	Fisher
Torrance	Jamaal	Albert
Oakland	Williemae	Holloway
Fullerton	Araceli	Golden
Palos Verdes Peninsula	Deloris	Burke

F) Trier un ensemble de résultats par une expression

La fonction **LEN()** renvoie le nombre de caractères d'une chaîne. L'instruction suivante utilise la fonction **LEN()** dans la clause **ORDER BY** pour récupérer une liste de clients triée en fonction de la longueur du prénom :

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    LEN(first_name) DESC;
```

first_name	last_name
Guillermina	Noble
Christopher	Richardson
Alejandrina	Hodges
Charlesetta	Soto
Hildegarde	Christensen
Margaretta	Clayton
Marguerite	Berger
Christoper	Gould

G) Tri par position ordinale des colonnes

SQL Server vous permet de trier l'ensemble de résultats en fonction des **positions ordinales** des colonnes qui apparaissent dans la liste de sélection.

L'instruction suivante trie les clients par prénom et par nom. Mais au lieu de spécifier explicitement les noms des colonnes, elle utilise les positions ordinales des colonnes :

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    1,
    2;
```

G) Tri par position ordinale des colonnes

Dans cet exemple, **1** correspond à la colonne f**irst_name** et **2** à la colonne **last_name**.

L'utilisation des positions ordinales des colonnes dans la clause **ORDER BY** est considérée comme une mauvaise pratique de programmation pour plusieurs raisons.

Premièrement, les colonnes d'une table n'ont pas de position ordinale et doivent être référencées par leur nom.

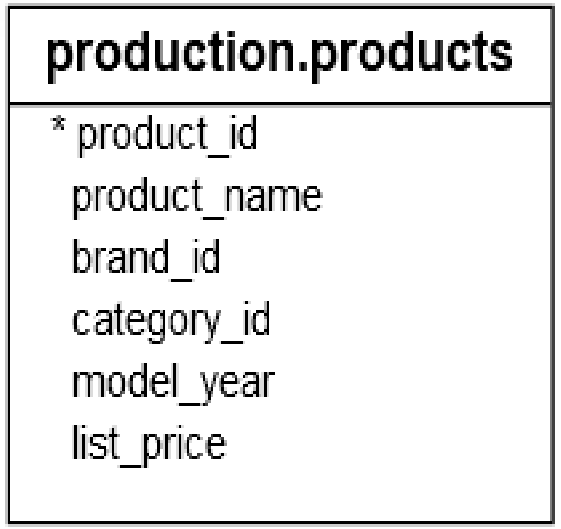
Deuxièmement, lorsque vous modifiez la liste de sélection, vous risquez d'oublier d'apporter les modifications correspondantes dans la clause **ORDER BY**.

Par conséquent, il est conseillé de toujours spécifier explicitement le nom des colonnes dans la clause **ORDER BY**.

OFFSET FETCH

Dans cette section, vous apprendrez à utiliser les clauses **OFFSET FETCH** du serveur SQL pour **limiter le nombre de lignes** renvoyées par une requête.

Les clauses **OFFSET et FETCH** sont les options de la clause **ORDER BY**. Elles permettent de limiter le nombre de lignes renvoyées par une requête.
Nous utiliserons la table des produits de la base de données d'exemple pour la démonstration.



OFFSET FETCH

Pour **ignorer** les **10** premiers produits et renvoyer les autres, vous utilisez la clause **OFFSET** comme indiqué dans l'instruction suivante :

```
SELECT
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price,
    product_name
OFFSET 10 ROWS;
```

product_name	list_price
Haro Shredder 20 Girls - 2017	209.99
Trek Precaliber 16 Boy's - 2018	209.99
Trek Precaliber 16 Boys - 2017	209.99
Trek Precaliber 16 Girl's - 2018	209.99
Trek Precaliber 16 Girls - 2017	209.99
Trek Precaliber 20 Boy's - 2018	229.99
Trek Precaliber 20 Girl's - 2018	229.99
Haro Shredder Pro 20 - 2017	249.99
Strider Sport 16 - 2018	249.99
Trek MT 201 - 2018	249.99
Sun Bicycles Revolutions 24 - 2017	250.99
Sun Bicycles Revolutions 24 - Girl's - 2017	250.99
Electra Cruiser 1 (24-Inch) - 2016	269.99
Electra Cruiser 1 (24-Inch) - 2016	269.99
Electra Cruiser 1 - 2016/2017/2018	269.99

OFFSET FETCH

Pour **ignorer** les 10 premiers produits et sélectionner les 10 suivants, vous utilisez les clauses **OFFSET** et **FETCH** comme suit :

```
SELECT
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price,
    product_name
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY;
```

product_name	list_price
Haro Shredder 20 Girls - 2017	209.99
Trek Precaliber 16 Boy's - 2018	209.99
Trek Precaliber 16 Boys - 2017	209.99
Trek Precaliber 16 Girl's - 2018	209.99
Trek Precaliber 16 Girls - 2017	209.99
Trek Precaliber 20 Boy's - 2018	229.99
Trek Precaliber 20 Girl's - 2018	229.99
Haro Shredder Pro 20 - 2017	249.99
Strider Sport 16 - 2018	249.99
Trek MT 201 - 2018	249.99

OFFSET FETCH

Pour obtenir les 10 produits les plus chers, vous utilisez les clauses OFFSET et FETCH :

```
SELECT
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC,
    product_name
OFFSET 0 ROWS
FETCH FIRST 10 ROWS ONLY;
```

product_name	list_price
Trek Domane SLR 9 Disc - 2018	11999.99
Trek Domane SLR 8 Disc - 2018	7499.99
Trek Domane SL Frameset - 2018	6499.99
Trek Domane SL Frameset Women's - 2018	6499.99
Trek Emonda SLR 8 - 2018	6499.99
Trek Silque SLR 8 Women's - 2017	6499.99
Trek Silque SLR 7 Women's - 2017	5999.99
Trek Domane SL 8 Disc - 2018	5499.99
Trek Domane SLR 6 Disc - 2017	5499.99
Trek Domane SLR 6 Disc - 2018	5499.99

Dans cet exemple, la clause ORDER BY trie les produits par ordre décroissant de leur prix. Ensuite, la clause OFFSET saute une ligne et la clause FETCH récupère les 10 premiers produits de la liste.

INSÉRER

Dans cette section, vous apprendrez à utiliser l'instruction **INSERT** pour ajouter une nouvelle ligne à un tableau.

Pour ajouter une ou plusieurs lignes dans un tableau, vous utilisez l'instruction **INSERT**.

```
INSERT INTO table_name (column_list)
VALUES (value_list);
```

Tout d'abord, vous spécifiez le nom de la table que vous souhaitez insérer. En règle générale, vous référencez le nom de la table par le nom du schéma, par exemple **production.products** où **production** est le nom du *schéma* et **products** est le nom de la *table.*

INSÉRER

Deuxièmement, vous spécifiez une liste d'une ou plusieurs colonnes dans lesquelles vous souhaitez insérer des données. Vous devez mettre la liste des colonnes entre parenthèses et séparer les colonnes par des virgules.

Si une colonne d'une table n'apparaît pas dans la liste des colonnes, SQL Server doit être en mesure de fournir une valeur à insérer, sinon la ligne ne peut pas être insérée.

INSÉRER

SQL Server utilise automatiquement la valeur suivante pour la colonne disponible dans la table mais n'apparaissant pas dans la liste des colonnes de l'instruction **INSERT** :

- La valeur incrémentielle suivante si la colonne possède une propriété **IDENTITY**.
- La valeur par défaut si la colonne a une valeur par défaut spécifiée.
- Le **NULL** si la colonne est nullable.
- La valeur **calculée** si la colonne est une colonne calculée.

INSÉRER

Troisièmement, vous fournissez une liste de valeurs à insérer dans la clause **VALUES**. Chaque colonne de la liste de colonnes doit avoir une valeur correspondante dans la liste de valeurs. De plus, vous devez mettre la liste de valeurs entre parenthèses.

INSÉRER

Pour **ajouter** une ou plusieurs **lignes** dans un tableau, vous utilisez l'instruction **INSERT**. Ce qui suit illustre la forme la plus élémentaire de l'instruction **INSERT** :

```
INSERT INTO table_name (column_list)
VALUES (value_list);
```

Exemple

```
INSERT INTO sales.promotions (
    promotion_name,
    discount,
    start_date,
    expired_date
)
VALUES
    (
        '2018 Summer Promotion',
        0.15,
        '20180601',
        '20180901'
    );
```

promotion_id	promotion_name	discount	start_date	expired_date
1	2018 Summer Promotion	0.15	2018-06-01	2018-09-01

Insérer et renvoyer les valeurs insérées

Pour capturer les valeurs insérées, vous utilisez la clause **OUTPUT**. Par exemple, l'instruction suivante insère une nouvelle ligne dans la table promotions et renvoie la valeur insérée de la colonne **promotion_id** :

```
INSERT INTO sales.promotions (
    promotion_name,
    discount,
    start_date,
    expired_date
) OUTPUT inserted.promotion_id
VALUES
    (
        '2018 Fall Promotion',
        0.15,
        '20181001',
        '20181101'
    );
```

INSÉRER

Pour capturer les valeurs insérées à partir de plusieurs colonnes, vous spécifiez les colonnes dans la sortie comme indiqué dans l'instruction suivante :

promotion_id	promotion_name	discount	start_date	expired_date
3	2018 Winter Promotion	0.15	2018-12-01	2019-01-01

```
INSERT INTO sales.promotions (
    promotion_name,
    discount,
    start_date,
    expired_date
) OUTPUT inserted.promotion_id,
 inserted.promotion_name,
 inserted.discount,
 inserted.start_date,
 inserted.expired_date
VALUES
    (
        '2018 Winter Promotion',
        0.2,
        '20181201',
        '20190101'
    );
```

INSÉRER

SQL Server a généré l'erreur suivante :

Cannot insert explicit value for identity column in table 'promotions' when IDENTITY_INSERT is set to OFF.

Pour insérer une valeur explicite pour la colonne d'identité, vous devez d'abord exécuter l'instruction suivante :

```
SET IDENTITY_INSERT table_name ON;
```

Pour désactiver l'insertion d'identité, vous utilisez l'instruction similaire :

```
SET IDENTITY_INSERT table_name OFF;
```

Alias de colonne

dans cette section, vous apprendrez à utiliser l'alias SQL Server, y compris l'alias de colonne et l'alias de table.

Lorsque vous utilisez l'instruction **SELECT** pour interroger les données d'une table, **SQL Server** utilise les noms de colonne comme en-têtes de colonne pour la sortie. Voir l'exemple suivant :

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    first_name;
```

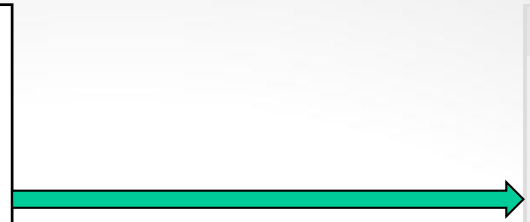
first_name	last_name
Aaron	Knapp
Abbey	Pugh
Abby	Gamble
Abram	Copeland
Adam	Henderson
Adam	Thornton
Addie	Hahn
Adelaida	Hancock

Alias de colonne

Comme le montre clairement le résultat, les noms de colonnes **first_name** et **last_name** ont été utilisés respectivement pour les en-têtes de colonnes.

Pour obtenir les noms complets des clients, vous pouvez concaténer le prénom, l'espace et le nom de famille à l'aide de l'opérateur de concaténation **+** comme indiqué dans la requête suivante :

```
SELECT
    first_name + ' ' + last_name
FROM
    sales.customers
ORDER BY
    first_name;
```



(No column name)
Aaron Knapp
Abbey Pugh
Abby Gamble
Abram Copeland
Adam Henderson
Adam Thornton
Addie Hahn
Adelaida Hancock

SQL Server a renvoyé la colonne de nom complet sous la forme (**No column name**), ce qui n'a pas de sens dans ce cas.

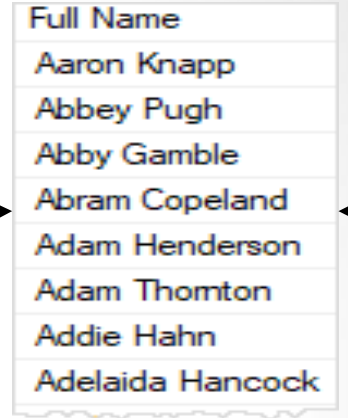
Alias de colonne

Revenant à l'exemple ci-dessus, vous pouvez réécrire la requête en utilisant un alias de colonne :

```
SELECT
    first_name + ' ' + last_name AS full_name
FROM
    sales.customers
ORDER BY
    first_name;
```

Notez que si l'alias de colonne contient des espaces, vous devez le mettre entre guillemets, comme indiqué dans l'exemple suivant :

```
SELECT
    first_name + ' ' + last_name AS 'Full Name'
FROM
    sales.customers
ORDER BY
    first_name;
```



Full Name
Aaron Knapp
Abbey Pugh
Abby Gamble
Abram Copeland
Adam Henderson
Adam Thornton
Addie Hahn
Adelaida Hancock

Jointures

Dans cette section, vous découvrirez différentes **jointures** SQL Server qui vous permettent de **combiner** les données de **deux tables**.

Dans une base de données relationnelle, les données sont distribuées dans plusieurs tables logiques. Pour obtenir un ensemble complet et significatif de données, vous devez interroger les données de ces tables à l'aide de jointures. SQL Server prend en charge de nombreux types de jointures, notamment la **jointure interne** et la **jointure gauche**. , **jointure à droite**, **jointure externe** complète et < a i=9> **jointure croisée**. Chaque type de jointure spécifie comment SQL Server utilise les données d'une table pour sélectionner des lignes dans une autre table.

Jointures

Tout d'abord, créez un nouveau schéma nommé hr :

```
CREATE SCHEMA hr;
GO
```

Deuxièmement, créez deux nouvelles tables nommées **candidates** et **employees** dans le hr schéma :

```
CREATE TABLE hr.candidates(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);

CREATE TABLE hr.employees(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);
```

Jointures

Troisièmement, insérer quelques lignes dans les tableaux candidates et employees :

Appelons la table **candidates** la table de **gauche** et la table **employees** la table de **droite**.

```
INSERT INTO
    hr.candidates(fullname)
VALUES
    ('John Doe'),
    ('Lily Bush'),
    ('Peter Drucker'),
    ('Jane Doe');


INSERT INTO
    hr.employees(fullname)
VALUES
    ('John Doe'),
    ('Jane Doe'),
    ('Michael Scott'),
    ('Jack Sparrow');
```

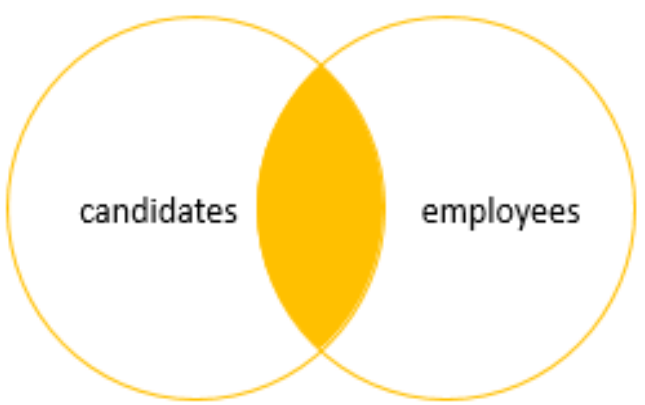
Jointure interne

La **Jointure interne** produit un ensemble de données qui **inclut** des lignes de **la table de gauche**, correspondant aux lignes de **la table de droite**.

L'exemple suivant utilise la clause de jointure interne pour obtenir les lignes de la table **candidates** qui contient les lignes correspondantes avec les mêmes valeurs dans la colonne **fullname** du **employees** tableau :

```
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    INNER JOIN hr.employees e
        ON e.fullname = c.fullname;
```

candidate_id	candidate_name	employee_id	employee_name
1	John Doe	1	John Doe
4	Jane Doe	2	Jane Doe



Rejoindre gauche

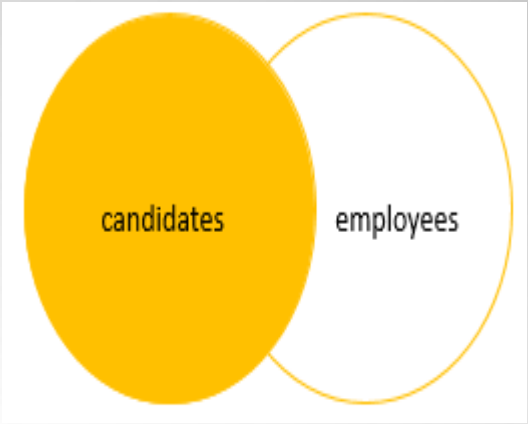
La **Jointure gauche** sélectionne les données à partir du tableau de gauche et les lignes correspondantes du tableau de droite. La jointure gauche renvoie toutes les lignes de la table de gauche et les lignes correspondantes de la table de droite. Si une ligne du tableau de gauche n'a pas de ligne correspondante dans le tableau de droite, les colonnes du tableau de droite auront des valeurs nulles.

La jointure gauche est également connue sous le nom de jointure externe gauche. Le mot-clé externe est facultatif.

Rejoindre gauche

L'instruction suivante joint la table **candidates** à la table **employees** en utilisant la **jointure à gauche** :

```
SELECT
        c.id candidate_id,
        c.fullname candidate_name,
        e.id employee_id,
        e.fullname employee_name
FROM
        hr.candidates c
        LEFT JOIN hr.employees e
                ON e.fullname = c.fullname;
```



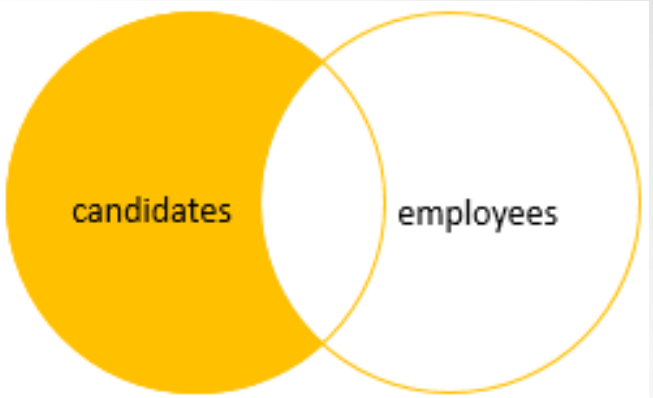
candidate_id	candidate_name	employee_id	employee_name
1	John Doe	1	John Doe
2	Lily Bush	NULL	NULL
3	Peter Drucker	NULL	NULL
4	Jane Doe	2	Jane Doe

Rejoindre gauche

Pour obtenir les lignes disponibles uniquement dans le tableau de gauche mais pas dans le tableau de droite, vous ajoutez une clause **WHERE** à la requête ci-dessus :

```
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    LEFT JOIN hr.employees e
        ON e.fullname = c.fullname
WHERE
    e.id IS NULL;
```

candidate_id	candidate_name	employee_id	employee_name
2	Lily Bush	NULL	NULL
3	Peter Drucker	NULL	NULL



Rejoindre à droite

La **jointure droite** ou **jointure externe droite** sélectionne les données en commençant par la bonne table. Il s'agit d'une version inversée de la **jointure gauche**.

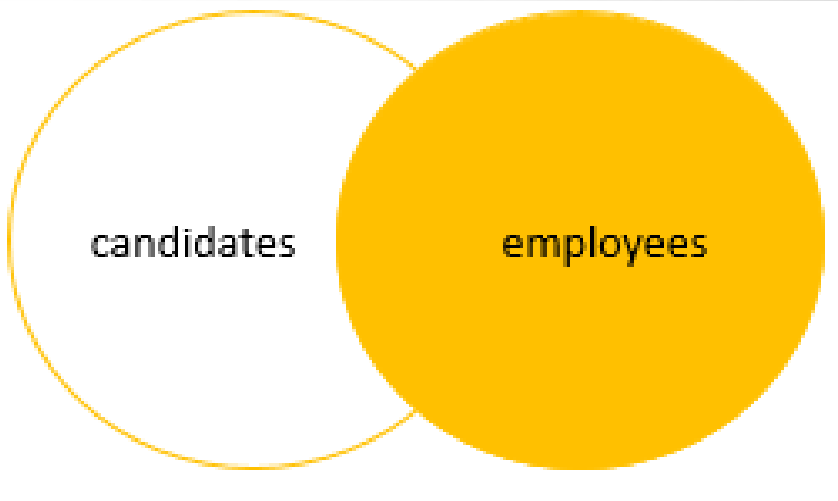
La jointure de droite renvoie un jeu de résultats contenant toutes les lignes de la table de droite et les lignes correspondantes de la table de gauche. Si une ligne du tableau de droite n’a pas de ligne correspondante dans le tableau de gauche, toutes les colonnes du tableau de gauche contiendront des valeurs **NULL**.

Rejoindre à droite

L'exemple suivant utilise la jointure droite pour interroger les lignes des tables **candidates** et **employees** :

```
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    RIGHT JOIN hr.employees e
        ON e.fullname = c.fullname;
```

candidate_id	candidate_name	employee_id	employee_name
1	John Doe	1	John Doe
4	Jane Doe	2	Jane Doe
NULL	NULL	3	Michael Scott
NULL	NULL	4	Jack Sparrow

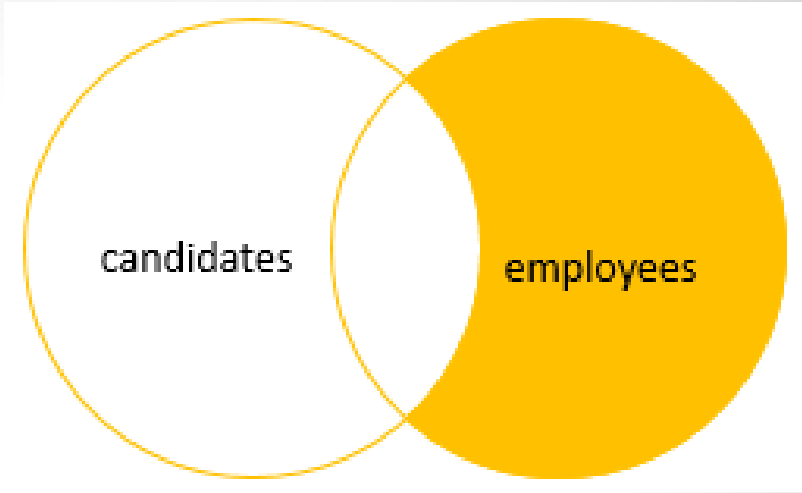


Rejoindre à droite

De même, vous pouvez obtenir des lignes disponibles uniquement dans la table de droite en ajoutant une clause **WHERE** à la requête ci-dessus comme suit :

```
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    RIGHT JOIN hr.employees e
        ON e.fullname = c.fullname
WHERE
    c.id IS NULL;
```

candidate_id	candidate_name	employee_id	employee_name
NULL	NULL	3	Michael Scott
NULL	NULL	4	Jack Sparrow



Jointure complète

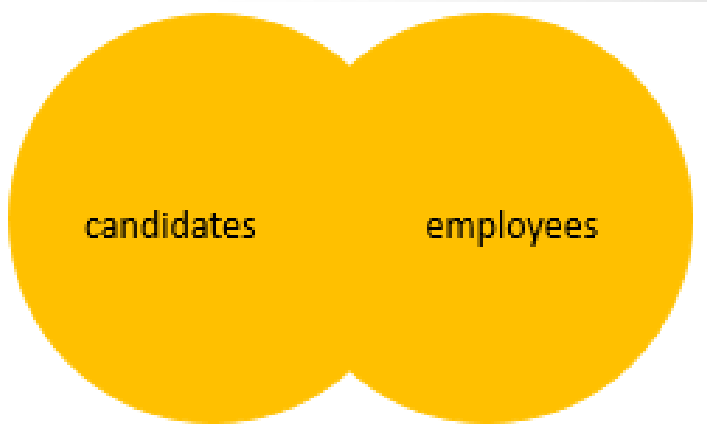
La **jointure externe complète ou jointure complète** renvoie un ensemble de résultats qui contient toutes les lignes des tables de gauche et de droite, avec les lignes correspondantes des deux côtés lorsqu'elles sont disponibles. S'il n'y a pas de correspondance, le côté manquant aura des valeurs **NULL**.

Rejoindre à droite

L'exemple suivant montre comment effectuer une jointure complète entre les tables **candidates** et **employees** :

```
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    FULL JOIN hr.employees e
        ON e.fullname = c.fullname;
```

candidate_id	candidate_name	employee_id	employee_name
1	John Doe	1	John Doe
2	Lily Bush	NULL	NULL
3	Peter Drucker	NULL	NULL
4	Jane Doe	2	Jane Doe
NULL	NULL	3	Michael Scott
NULL	NULL	4	Jack Sparrow

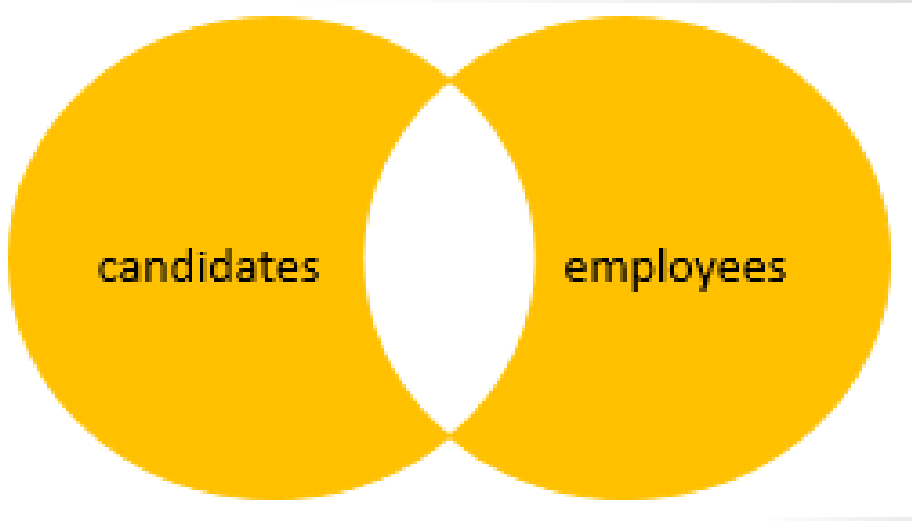


Rejoindre à droite

Pour sélectionner des lignes qui existent dans une table de gauche ou de droite, vous excluez les lignes communes aux deux tables en ajoutant une clause WHERE comme indiqué dans la requête suivante :

```
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    FULL JOIN hr.employees e
        ON e.fullname = c.fullname
WHERE
    c.id IS NULL OR
    e.id IS NULL;
```

candidate_id	candidate_name	employee_id	employee_name
2	Lily Bush	NULL	NULL
3	Peter Drucker	NULL	NULL
NULL	NULL	3	Michael Scott
NULL	NULL	4	Jack Sparrow



Exercice

Exercice 1:

Soit la base de données relationnelle des vols quotidiens d'une compagnie aérienne qui contient les tables **Avion, Pilote et Vol**.

Table **Avion** (

NA : numéro avion de type entier (clé primaire),
Nom : nom avion de type texte (12),
Capacite : capacité avion de type entier,
Localite : ville de localité de l'avion de type texte (10)

)

Table **Pilote** (

NP : numéro pilote de type entier,
Nom : nom du pilote de type texte (25),
Adresse : adresse du pilote de type texte (40)

)

Exercice

Table **Vol** (
NV : numéro de vol de type texte (6),
NP : numéro de pilote de type entier,
NA : numéro avion de type entier,
VD : ville de départ de type texte (10),
VA : ville d'arrivée de type texte (10),
HD : heure de départ de type entier,
HA : heure d'arrivée de type entier
)

1) Insérer les avions suivants dans la table Avion :
(100, AIRBUS, 300, RABAT),
(101,B737,250,CASA),
(101, B737,220,RABAT)
2) Afficher tous les avions
3) Afficher tous les avions par ordre croissant sur le nom
4) Afficher les noms et les capacités des avions

Exercice

5) Afficher les localités des avions sans redondance

6) Afficher les avions dans la localité et Rabat ou Casa

7) Modifier la capacité de l'avion numéro 101, la nouvelle capacité et 220

8) Supprimer les avions dans la capacité et inférieure à 200

9) Afficher la capacité maximale, minimale, moyenne des avions

10) Afficher les données des avions dont la capacité et la plus basse

11) Afficher les données des avions dont la capacité et supérieure à la capacité moyenne