

GESTION DES DONNEES ET DES TRANSACTIONS

Du 29/09/2025 au 01/10/2025

Fondamentaux de la Gestion des Données

Principes de la Modélisation des Données

1- Compréhension des relations entre les données, normalisation, et formes normales.

En 1970, Edgar F. Codd (Directeur de recherche du centre IBM de San Jose, CA, USA) publie un article de référence posant les bases du modèle relationnel, appelé aussi COD 70.

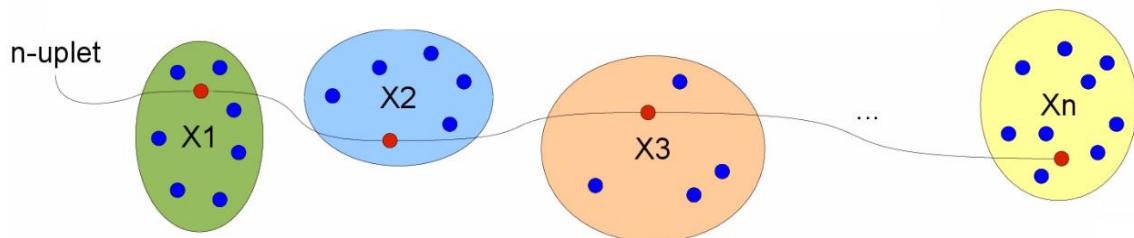
Par ce fait, toutes les limitations des précédents modèles sont résolues.

Le but initial de ce modèle était d'améliorer l'indépendance entre les données et les traitements. De ce point de vue, c'est une réussite le tout en ajoutant de nouvelles fonctionnalités :

- ➡ Normalisation (dépendances fonctionnelles) et théorie des ensembles (algèbre relationnelle).
- ➡ Cohérence des données (non-redondance et intégrité référentielle).
- ➡ Langage SQL (déclaratif et normalisé).
- ➡ Accès aux données optimisé (choix du chemin par le SGBD).
- ➡ Indexation, etc.

Les liens entre les enregistrements de la base de données sont réalisés non pas à l'aide de pointeurs physiques, mais à l'aide des valeurs des clés étrangères et des clés primaires. Pour cette raison, le modèle relationnel est dit « modèle à valeurs ».

Visuellement, on pourrait le représenter comme ceci :



Une relation au niveau du modèle relationnel se traduira en pratique, lors de la manipulation d'un SGBD relationnel par une table contenant des colonnes. Nous parlerons donc de n-uplet lorsqu'il s'agira de caractériser le contenu d'une relation, mais nous parlerons de lignes lorsqu'il s'agira de caractériser le contenu d'une table présente dans la BDD relationnelle.

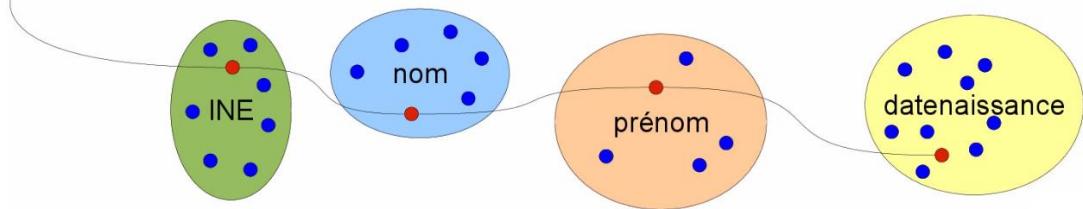
Exemple

Reprenant l'exemple graphique précédent et améliorons-le.

R pourrait être par exemple une table listant les étudiants.

Table : Etudiants(INE, nom, prénom, datenaissance). En image cela donne ceci (**voir ci-dessous**).

Le 4-uplet (1234567890A, Dupont, Jean, 01/03/1994) peut en faire partie



Pas forcément prise en compte lors de la mise en place d'un projet dynamique, la normalisation d'une base de données devient rapidement une nécessité une fois que le projet a pris de l'ampleur, et qu'il ne peut plus se suffire de deux ou trois tables seulement.

Il est courant dans de telles situations de scinder une table en plusieurs, de réorganiser les données et de tenter de reprendre en main la logique de stockage. Malheureusement, c'est lors de ces réorganisations que des problèmes peuvent se créer : requêtes SQL à rallonge, anomalies de mise à jour, redondances... En optimisant les tables, la normalisation tente de parer ces difficultés.

La normalisation correspond au processus d'organiser ses données afin de limiter les redondances, divisant une table en plusieurs, et en les reliant entre elles par des clefs primaires et étrangères. L'objectif est d'isoler les données afin que l'ajout, l'effacement ou la modification d'un champ puisse se faire sur une seule table, et se propager au reste de la base par le biais des relations. Pour ce faire, la normalisation introduit en tout 8 formes normales.

Les formes normales	
Forme	Implication
1NF First Normal Form	Chaque table doit avoir une clef primaire. Il faut éliminer les colonnes en doublon. Chaque ligne doit contenir une seule valeur.
2NF Second Normal Form	Si une table dispose d'une clef, toutes les propriétés doivent en dépendre : les données que l'on retrouve dans plusieurs lignes doivent être sorties dans une table séparée.
3NF Third Normal Form	Les données d'une table ne dépendent que de la clef, et pas d'une autre colonne de la table : toute colonne dépend non seulement de la clef, mais également d'une autre colonne qui doit être sortie dans sa propre table.
BCNF Boyce-Cod Normal Form	Aucune propriété faisant partie de la clef d'une relation 3NF, ne doit dépendre d'une propriété ne faisant pas partie de la clef primaire. Il faut donc créer une nouvelle table dont la clef primaire sera la propriété dont provient la relation.
4NF Fourth Normal Form	Il ne doit y avoir qu'une et une seule dépendance multivaluée élémentaire.
5NF Fifth Normal Form	Toute dépendance de jointure est impliquée par des clefs candidates de la relation.
Domaine/key Normal Form	Il ne doit exister aucune contrainte sinon celles de domaine ou de clef.
6NF Sixth Normal Form	La sixième forme, encore récente, demande à prendre en compte la dimension temporelle.

1. Forme normale 1 (1NF) :

👉 Chaque boîte contient une seule info par case.

C'est comme si, sur une étagère, tu avais une case pour chaque type de pain (baguette, pain de campagne, etc.), sans mélanger plusieurs types dans la même case.

Exemple **pas bien** :

Une fiche de commande avec un client et **toutes ses commandes dans une seule ligne** :

Client	Commande
Paul	Baguette, Croissant

Exemple **bien** (1NF) :

Tu fais une ligne par commande :

Client	Commande
Paul	Baguette
Paul	Croissant

2. Forme normale 2 (2NF) :

👉 Ne pas répéter les mêmes infos inutiles.

C'est comme si tu écrivais une fiche pour chaque pain et chaque ingrédient **au lieu de répéter les détails à chaque commande**.

Exemple **pas bien** :

Commande	Pain	Prix	Fournisseur	Ville Fournisseur
001	Baguette	1€	Dupont	Paris
001	Pain de mie	2€	Dupont	Paris

Tu répètes "Dupont - Paris" plusieurs fois. Ça ne sert à rien.

En 2NF, tu crées deux boîtes (tables) :

1. Une table pour les **fournisseurs** :

Fournisseur	Ville
Dupont	Paris

2. Une table pour les commandes :

Commande	Pain	Prix	Fournisseur
001	Baguette	1€	Dupont
001	Pain de mie	2€	Dupont

3. Forme normale 3 (3NF) :

👉 Ne garde que les infos qui ont un lien direct.

Si une info peut être déduite d'une autre, tu la mets ailleurs.

C'est comme si tu ne répétais pas "Une baguette = 1€" partout, mais tu fais une fiche des prix.

Exemple **pas bien** :

Pain	Prix	Fournisseur	Ville Fournisseur
Baguette	1€	Dupont	Paris
Pain de mie	2€	Dupont	Paris

En 3NF, tu sépares **les prix** dans une table :

1. Table des pains :

Pain	Prix
Baguette	1€
Pain de mie	2€

2. Table des fournisseurs :

Fournisseur	Ville
Dupont	Paris

BCNF (Boyce-Codd Normal Form) :

👉 Évite les dépendances bizarres entre les boîtes.

C'est comme si tu ranges tes pains et viennoiseries dans des boîtes, mais certaines boîtes commencent à "dépendre" des autres d'une façon illogique.

Exemple **pas bien** :

Imagine une table qui mélange des rôles étranges :

Employé	Poste	Supérieur
Paul	Boulanger	Dupont
Jean	Livreur	Dupont
Paul	Boulanger	Martin

Ici, "Supérieur" ne dépend pas toujours directement du "Poste". Ça rend la table confuse.

Solution BCNF :

On réorganise en deux tables pour clarifier :

1. Table des employés et leurs postes :

Employé	Poste
Paul	Boulanger
Jean	Livreur

2. Table des Supérieur associés aux postes :

Poste	Supérieur
Boulanger	Dupont
Boulanger	Martin
Livreur	Dupont

4NF (Quatrième Forme Normale) :

👉 Pas de doublons inutiles quand il y a plusieurs types d'infos indépendantes.
C'est comme si tu ranges **deux infos indépendantes** (par ex., pains et boissons vendues) dans une seule boîte, ce qui crée des doublons.

Exemple pas bien :

Commande	Pain	Boisson
001	Baguette	Café
001	Pain de mie	Café
001	Baguette	Thé

Les pains et boissons n'ont rien à voir, mais ici tu les mélanges.

Solution 4NF : Sépare les tables pour chaque info indépendante :

1. Table des pains commandés :

Commande	Pain
001	Baguette
001	Pain de mie

2. Table des boissons commandées :

Commande	Boisson
001	Café
001	Thé

5NF (Cinquième Forme Normale) :

👉 Évite les liens compliqués entre plusieurs boîtes.

C'est comme si plusieurs relations "à trois" (ou plus) faisaient un gros bazar. Le but est de tout découper correctement.

Exemple pas bien :

Imagine une table comme ça :

Pain	Fournisseur	Client
Baguette	Dupont	Paul
Pain de mie	Martin	Paul
Baguette	Dupont	Marie

Ici, tu mélanges trois infos en même temps ("Pain", "Fournisseur", "Client"). Ça devient dur à gérer.

Solution 5NF : On divise encore plus :

1. Table pains et fournisseurs :

Pain	Fournisseur
Baguette	Dupont
Pain de mie	Martin

2. Table clients et pains :

Client	Pain
Paul	Baguette
Paul	Pain de mie
Marie	Baguette

Domaine/Key Normal Form (DKNF) :

👉 Fais tes règles comme un pro !

Tu définis des **règles strictes** pour dire ce qui est possible ou pas dans tes boîtes.

Exemple de règle :

- Un "Boulanger" ne peut avoir qu'un "Fournisseur".
- Un prix doit être un nombre positif.

Pourquoi ?

Pour éviter des trucs qui n'ont aucun sens dans ta base, comme :

Paul	Boulanger	Dupont
Paul	Livreur	-5€

DKNF, c'est comme un chef très strict qui vérifie **tout** avant de valider.

6NF (Sixième Forme Normale) :

👉 Rentre dans les détails au max.

C'est comme si tu séparais **chaque micro-détail** dans une boîte à part. On utilise ça surtout pour des bases super complexes, avec beaucoup de changements (comme des historiques).

Exemple pas bien :

Employé	Poste	Date Début	Date Fin
Paul	Boulanger	2023-01-01	2023-06-30
Paul	Livreur	2023-07-01	2023-12-31

Si un employé change souvent de poste, cette table devient un enfer.

Solution 6NF : Tu fais une boîte pour chaque info changeante :

1. Table des employés :

Employé	Poste	Date
Paul	Boulanger	2023-01-01
Paul	Livreur	2023-07-01

Résumé des formes avancées :

1. **BCNF** : Clarifie les relations entre les boîtes.
2. **4NF** : Pas de mélanges d'infos indépendantes.
3. **5NF** : Découpe encore plus les relations complexes.
4. **DKNF** : Mets des règles strictes pour éviter les erreurs.
5. **6NF** : Organise chaque petit détail dans sa boîte.

Conception de Schémas de Base de Données

2- Atelier pratique sur la création de modèles entité-association pour différents scénarios d'application.

Le modèle conceptuel de données (MCD) est un des principaux modèles de la méthode Merise, qui permet de représenter les données du système d'information de manière conceptuelle et indépendante des aspects techniques de la mise en œuvre. Le MCD permet de décrire les entités principales du système, leurs attributs et leurs relations.

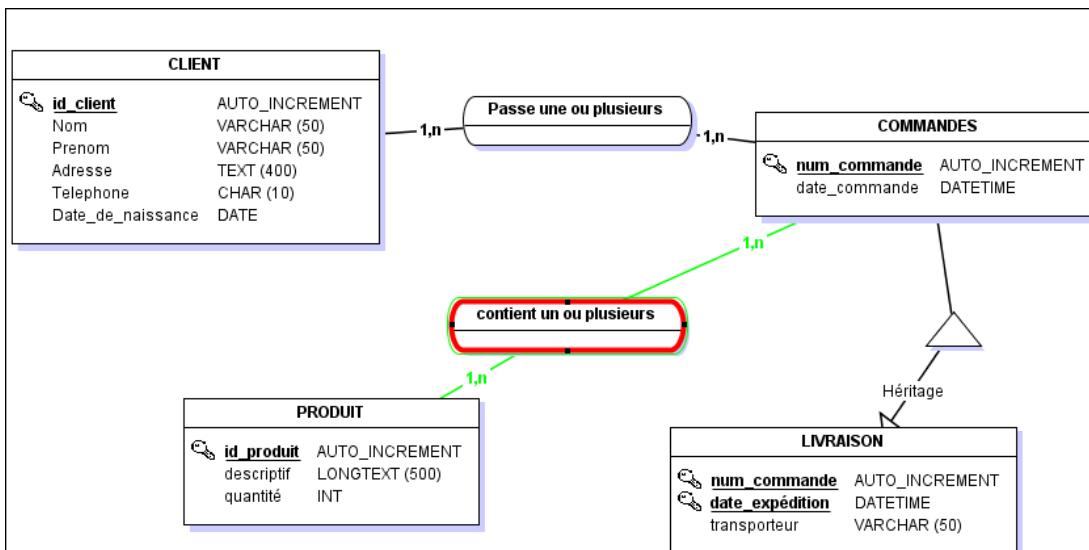
Le MCD est souvent considéré comme la première étape de la modélisation d'un système d'information dans la méthode Merise. Il est généralement élaboré à partir d'informations recueillies lors de l'analyse des besoins du système, en collaboration avec les différents acteurs impliqués.

Le MCD peut être représenté sous forme de diagrammes entité-association (ER), qui utilisent des symboles graphiques pour représenter les entités, les attributs et les relations. Les entités sont des objets concrets ou abstraits qui représentent des éléments du monde réel, tels que les clients, les produits ou les commandes. Les attributs sont les caractéristiques des entités, tels que les noms, les adresses ou les quantités. Les relations décrivent les associations entre les entités, tels que les commandes passées par les clients ou les produits fournis par les fournisseurs.

Le MCD est utilisé comme base pour les étapes suivantes de la méthode Merise, notamment la conception de la base de données et la mise en place du système d'information. En utilisant le MCD, les concepteurs peuvent s'assurer que les données sont cohérentes, structurées et répondent aux besoins du système.

M.C.D

Voici un exemple concret de MCD :



1- Entité

Terme	Définition	Exemple
Entité	Ensemble d'éléments informatifs relatif à un même concept dans un modèle. Correspond généralement à une table dans l'univers des bases de données.	Entité "Personne"

2- Entité : Propriété

Terme	Définition	Exemple
Entité : Propriété ou plus simplement Attribut	Propriété d'une entité dans un modèle correspondant généralement à une colonne dans une table de la base.	Attribut nom de famille d'une personne

3- Entité : Identifiant

Terme	Définition	Exemple
Identifiant	Autre nom du concept de clé.	Identifiant N° de commande de la table commande

4- Entité : Occurrence

Terme	Définition	Exemple
Occurrence	Une occurrence est une instance particulière d'une entité, c'est-à-dire un exemple concret de l'entité avec des valeurs spécifiques pour chaque attribut.	Par exemple, si nous avons une entité "Employé" avec des attributs tels que "Nom", "Prénom", "Numéro d'employé", "Adresse", etc., une occurrence spécifique de cette entité serait un employé particulier avec un nom, un

		prénom, un numéro d'employé et une adresse spécifiques.
--	--	---

5- Association

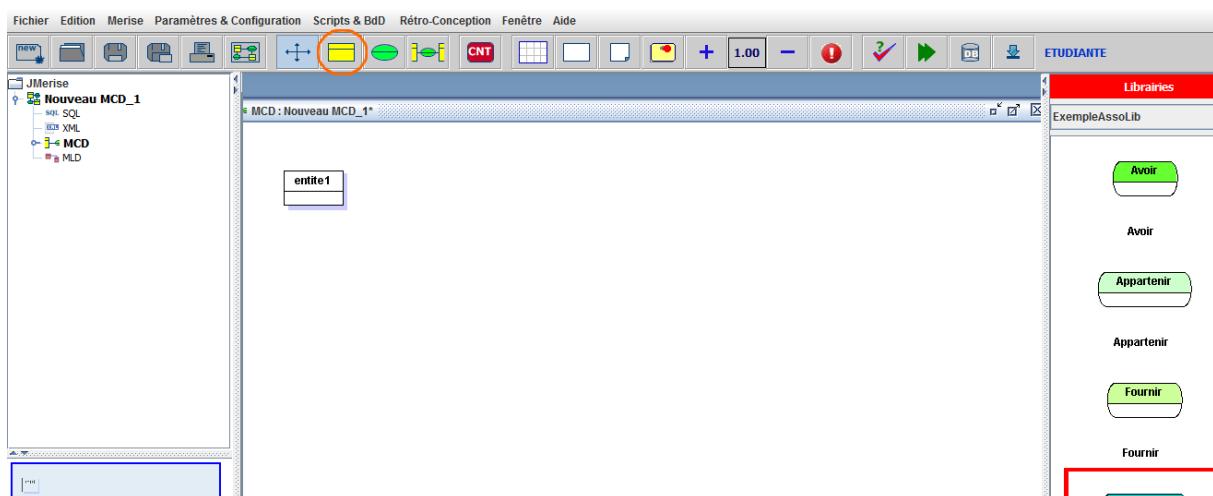
Terme	Définition	Exemple
Association ou Relation	Lien logique entre deux entités, représenté dans l'univers des SGBDR par l'insertion d'une clef étrangère ou par l'utilisation d'une table de jointure.	La relation entre les tables "client" et "commande" se fait par une table de jointure possédant comme clé, l'ensemble des colonnes relatives aux deux clés des deux tables.

Travail dirigé (1) :

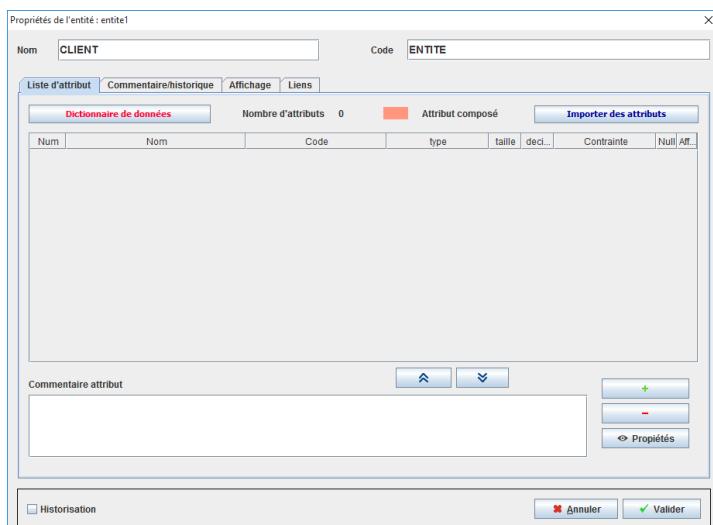
Lancer le fichier JMerise.jar fourni par le formateur.

Aide :

Pour créer une entité, cliquez sur ce bouton entouré en orange ci-dessous :



Double cliquer sur votre entité vous pouvez lui donner un nom (**CLIENT** en l'occurrence) comme ci-dessous :



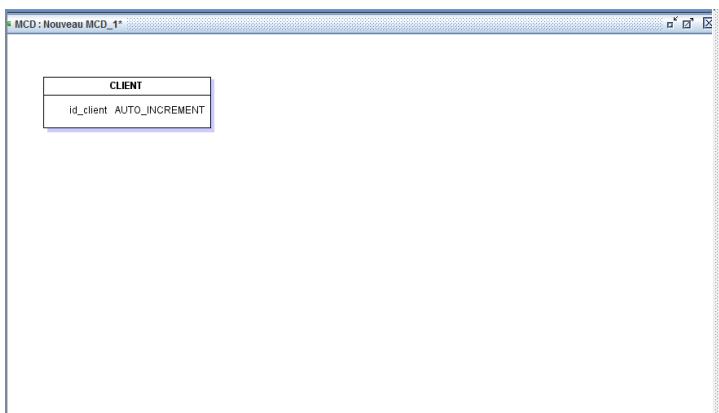
Pour ajouter un attribut cliquez sur le bouton + comme entouré en orange ci-dessous :



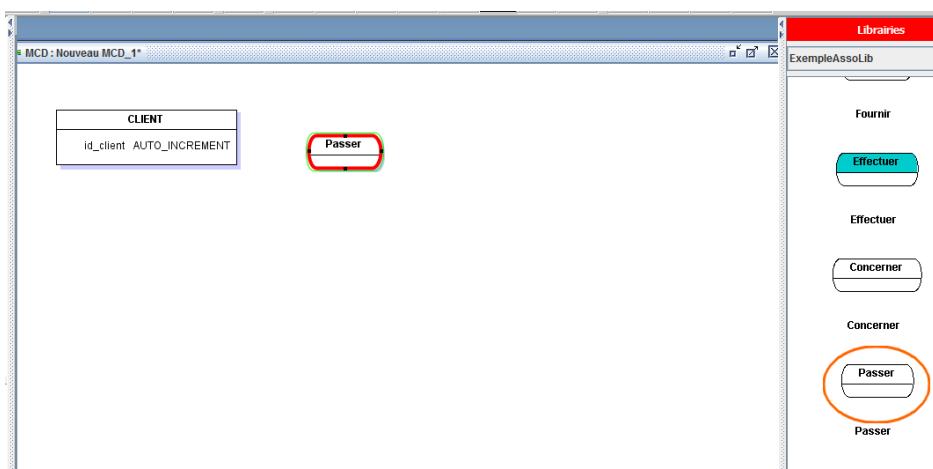
Nous allons créer notre premier attribut qui sera **id_client** qui sera une clé, auto-incrémentée. Renseignez la ligne comme ci-dessous :

Num	Nom	Code	type	taille	deci...	Contrainte	Null	Aff...
1	id_client	ID_CLIENT	Auto_increment					

Cliquez sur Valider et vous obtenez ceci :



Nous allons créer une nouvelle relation, la relation **passer**. Cliquez sur **passer** entouré en orange comme ci-dessous.

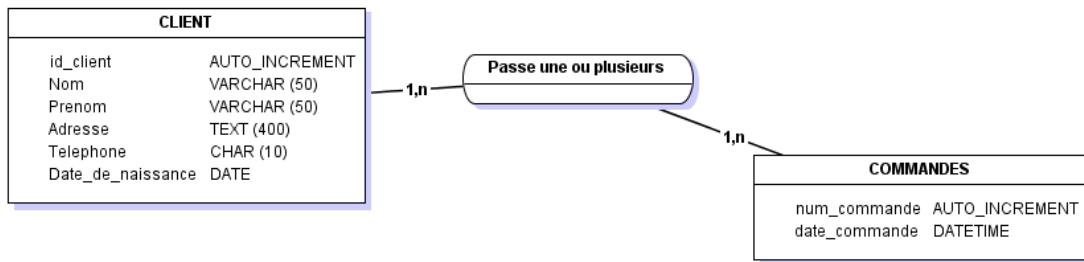


Renommez la relation en **passer une ou plusieurs** comme ci-dessous :



Travail dirigé (2) :

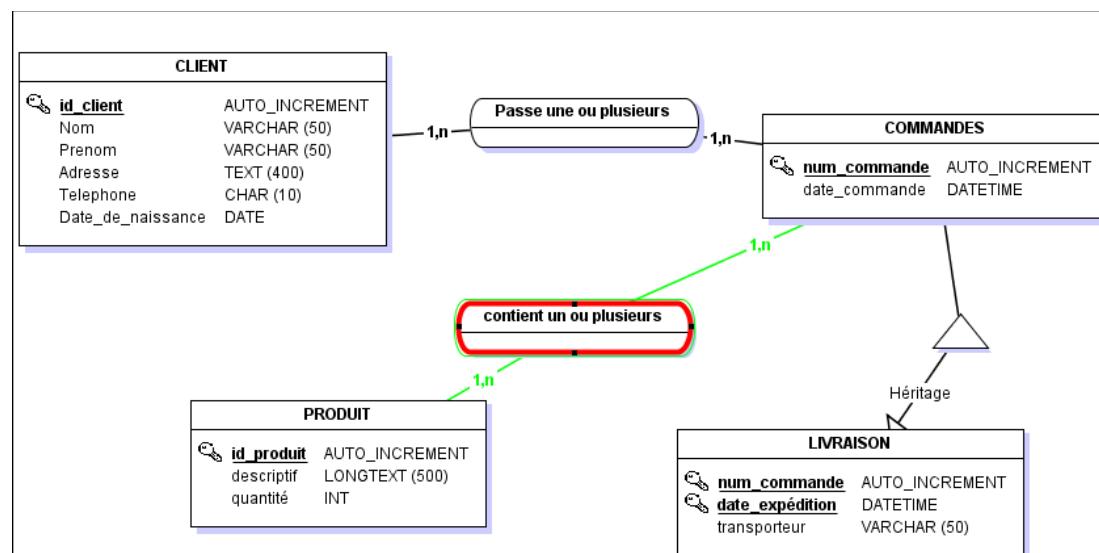
A partir de ce point de départ fait dans le TP 1, finir le petit MCD. Voici le résultat :



Travail dirigé (3) :

Une fois cela fait, rajouter 2 entités : **produit** et **livraison**.

Voici ce que vous devez obtenir finalement avec la mise en place des clés (identifiants).



Atelier pratique 1 : Création de modèles Entité–Association (E/A)

🎯 Objectifs pédagogiques

À l'issue de cet atelier, les étudiants seront capables de :

- Identifier les **entités** et leurs **attributs** dans un scénario donné.
- Déterminer les **relations** entre entités et leurs cardinalités.
- Concevoir un **modèle E/A clair et cohérent**.
- Justifier leurs choix de modélisation.

⌚ Durée

Environ 1h30 – 2h (selon le niveau des étudiants).

📍 **Déroulement de l'atelier**

1. Introduction (10 min)

- Rappel rapide des concepts clés :
 - **Entités** (objets du monde réel ou abstrait).
 - **Attributs** (propriétés d'une entité).
 - **Clé primaire**.
 - **Relations** (associations entre entités).
 - **Cardinalités** (1-1, 1-N, N-N).
- Exemple simple illustré (ex : étudiants – cours – enseignants).

2. Mise en pratique en groupe (60 min)

Les étudiants travaillent en binômes ou petits groupes.

Chaque groupe reçoit un **scénario d'application** et doit produire un **modèle E/A**.

◆ **Scénario 1 : Gestion des employés en formation**

- Nous avons une entité EMPLOYE.
- Les employés travaillent (ou non) sur un projet.
- Les employés sont inscrits (ou non) à un séminaire.
- Un séminaire correspond à un cours.
- Les employés participent (ou non) à un séminaire.

Concernant les **attributs**, vous êtes libres, mais essayez d'être **le plus complet possible**.

◆ **Scénario 2 : Système d'analyse des ventes pour une entreprise internationale**

- L'entreprise vend des produits en ligne via son site et dans des points de vente physiques (magasins) répartis dans différents pays.
- Chaque vente est enregistrée avec le client, le produit, le lieu (pays), la date et le moyen de paiement.
- Un produit appartient à une catégorie et une sous-catégorie.
- Un client peut effectuer plusieurs achats, et chaque achat concerne un seul produit à la fois.

◆ **Scénario 3 : Tableau de bord performance pour le groupe AUCHAN**

- AUCHAN suit les performances de ses **régions** (ex.: Nord, Ile-de-France, Espagne) qui regroupent plusieurs magasins.
- Chaque région lance des **campagnes marketing** (promotions flyers, publicités TV) pour dynamiser les ventes de ses **produits** (marque propre AUCHAN, produits nationaux).
- Les **promotions** (soldes, offres spéciales) sont appliquées sur des produits spécifiques et leur impact sur les stocks et le chiffre d'affaires est mesuré.
- Le service financier consolide les **données financières** (CA, dépenses marketing, bénéfices) par région et par campagne pour évaluer la rentabilité de chaque initiative.
- Faire ressortir les dimensions retail (magasin), drive et livraison.

👉 Les étudiants doivent :

1. Identifier les entités et leurs attributs.
2. Déterminer les relations et cardinalités.
3. Dessiner le schéma E/A (papier ou outil de modélisation).

3. Mise en commun et discussion (20 min)

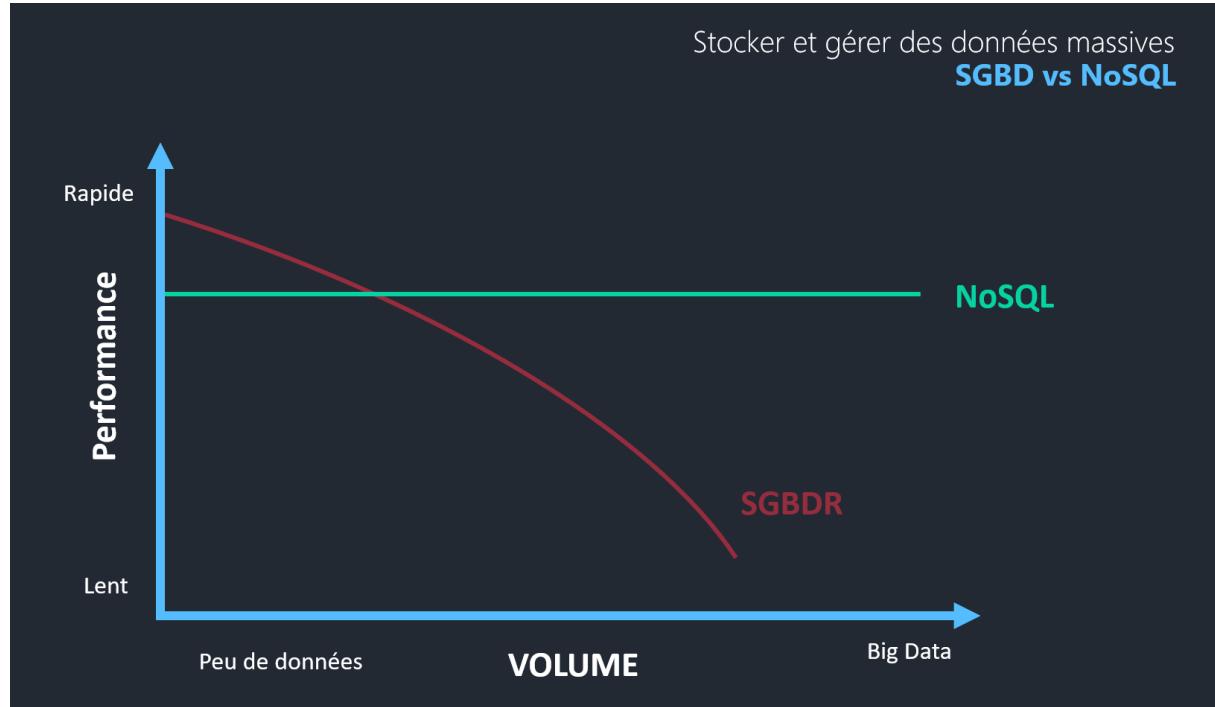
- Chaque groupe présente son modèle (5 min max).
- Discussion : points forts, erreurs fréquentes, alternatives possibles.
- L'enseignant apporte des corrections ou propose une version "référence".

Résultats attendus

- Modèles E/A propres et justifiés.
- Capacité à argumenter les choix de modélisation.
- Compréhension renforcée de la transition entre un **problème métier** et un **modèle conceptuel de données**.

Introduction aux SGBD

3- Comparaison entre SGBD relationnels et non relationnels. Avantages et inconvénients.



SGBD vs NoSQL

- **Explication :** Ce graphique résume tout. Les SGBDR excellent sur des volumes de données modestes à moyens, avec des besoins transactionnels forts (ACID). Les bases NoSQL prennent le relais quand le volume devient très important ("Big Data"), quitte à offrir moins de fonctionnalités avancées (comme les jointures) mais en étant beaucoup plus performantes à cette échelle.

Résumé :

Comparaison :

- Pour petits volumes, SQL reste performant.

- Pour gros volumes, NoSQL est mieux adapté (scalable et rapide).

Questions possibles :

1. *Donc NoSQL est toujours plus rapide ?*
2. *Quand garder SQL alors ?*

Réponses pédagogiques :

1. Pas forcément → sur de petits volumes, SQL peut être aussi rapide voire plus.
2. SQL est pertinent quand il faut des **transactions fiables** et une forte cohérence (banques, ERP...).

Comparaison SGBD Relationnels (SQL) vs Non Relationnels (NoSQL)

SGBD Relationnel (SQL)

- **Modèle** : Données structurées en tables (lignes/colonnes) avec un schéma prédéfini et strict.
- **Langage** : SQL (Structured Query Language), puissant et standardisé.
- **Propriété ACID** : Transactions complexes garantissant l'intégrité des données (Atomicité, Cohérence, Isolation, Durabilité).

Avantages :

- **Intégrité et cohérence des données** : Idéal pour les applications critiques (comptabilité, systèmes bancaires).
- **Puissance des jointures** : Requêtes complexes facilitées entre plusieurs tables.
- **Schéma rigide** : Structure claire qui impose une discipline et réduit les erreurs.
- **Standardisation et maturité** : Large communauté, excellente documentation.

Inconvénients :

- **Difficulté de mise à l'échelle (Scaling)** : Le scaling vertical (ajouter de la puissance à une seule machine) est coûteux. Le scaling horizontal (ajouter des machines) est complexe.
- **Rigidité du schéma** : Les modifications de structure (ajout de colonnes) peuvent être lourdes.
- **Performance sur gros volumes** : Peut devenir lent pour des données massives ou non structurées.

SGBD Non Relationnel (NoSQL)

- **Modèle** : Diversifié (document, clé-valeur, graphe, colonne). Schéma flexible et dynamique.
- **Langage** : Pas de langage standardisé, souvent des API spécifiques.
- **Propriété BASE** : Priorité à la disponibilité et la tolérance aux pannes (Basically Available, Soft state, Eventual consistency).

Avantages :

- **Scalabilité horizontale facile** : Conçu pour répartir la charge sur de nombreuses machines (idéal pour le Big Data).
- **Flexibilité du schéma** : Permet d'ajouter facilement de nouveaux champs, adapté aux données changeantes.
- **Haute performance** : Optimisé pour des opérations simples et rapides sur des volumes massifs.

- **Adapté aux données non structurées** (images, logs, données IoT).

Inconvénients :

- **Cohérence éventuelle (Eventual Consistency)** : Les données peuvent ne pas être immédiatement cohérentes sur tous les nœuds.
- **Pas de jointures natives** : La gestion des relations entre les données est plus complexe et doit être gérée par l'application.
- **Manque de standardisation** : Chaque base a son propre fonctionnement, courbe d'apprentissage spécifique.

Quand choisir l'un ou l'autre ?

Critère de choix	Préférer un SGBD Relationnel (SQL)	Préférer un SGBD Non Relationnel (NoSQL)
Structure des données	Données structurées, relations complexes	Données non structurées ou schéma flexible
Intégrité	Transactions ACID critiques (ex: virement bancaire)	Cohérence éventuelle acceptable (ex: panier d'achat)
Volume et échelle	Volume modéré, scaling prévisible	Volume massif (Big Data), besoin de scalabilité horizontale
Type d'opérations	Requêtes complexes avec nombreuses jointures	Accès simples et rapides, lecture/écriture massive

Conclusion : Le choix n'est pas une question de "meilleur" mais de "plus adapté". Les architectures modernes utilisent souvent les deux (base SQL pour le cœur de métier et base NoSQL pour des fonctionnalités spécifiques), c'est ce qu'on appelle l'approche **polyglotte**.

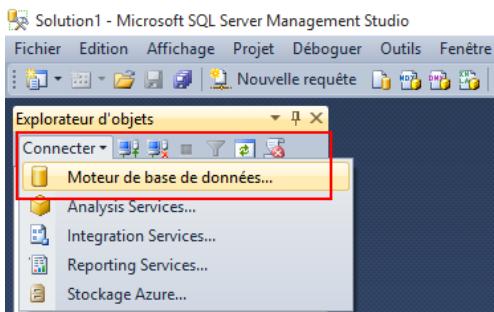
4- Pratique avec SQL pour les bases de données relationnelles et introduction à MongoDB pour les bases de données non relationnelles

➤ Installation de SQL Server 2016

(Voir l'annexe 1, installation de SQL Server 2016).

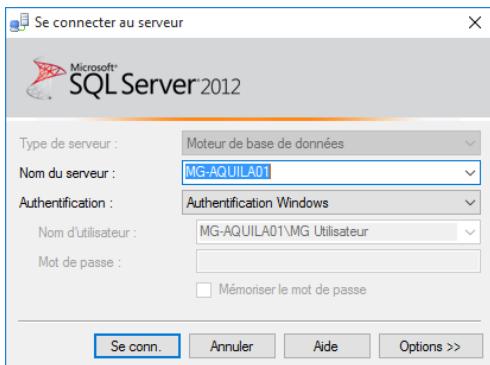
Pour se connecter à une instance de SQL Server

Démarrez SQL Server Management Studio. La première fois que vous exécutez SSMS, la fenêtre **Se connecter** au serveur s'ouvre. Si elle ne s'ouvre pas, vous pouvez l'ouvrir manuellement en sélectionnant **Explorateur d'objets > Se connecter > Moteur de base de données**.



Dans la fenêtre **Se connecter au serveur**, suivez les instructions de la liste ci-dessous :

- ▶ Pour **Type de serveur**, sélectionnez **Moteur de base de données** (généralement l'option par défaut).
- ▶ Pour **Nom du serveur**, entrez le nom de votre instance SQL Server. (Le poste formateur utilise le nom d'instance MG Utilisateur sur le nom d'hôte MG-AQUILA01 [MG-AQUILA01\ MG Utilisateur].)
- ▶ Pour **Authentification**, sélectionnez **Authentification Windows**. Pour cette formation, nous utilisons l'authentification Windows, mais la connexion SQL Server est également prise en charge. Si vous sélectionnez **Connexion SQL**, vous êtes invité à fournir un nom d'utilisateur et un mot de passe.



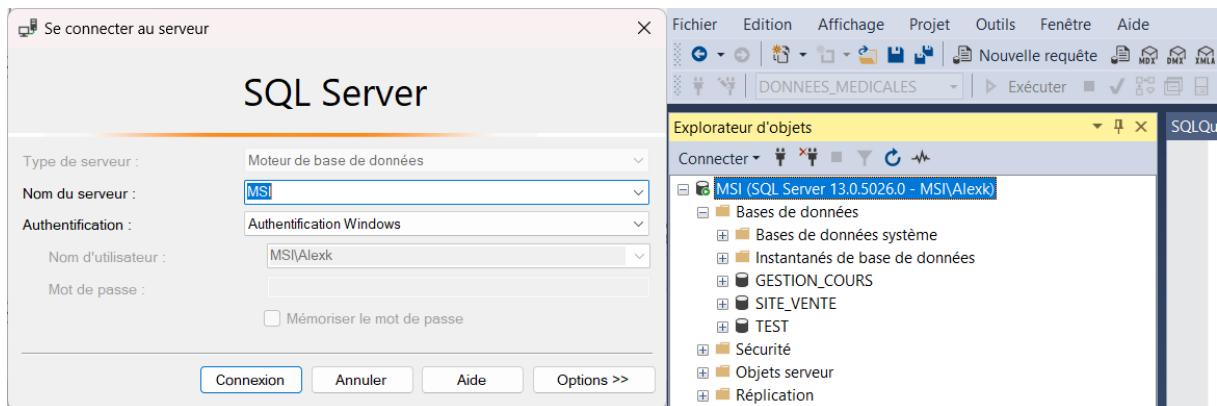
Vous pouvez également modifier d'autres options de connexion en sélectionnant **Options**. Exemples d'options de connexion : la base de données à laquelle vous êtes connecté, la valeur du délai d'expiration de la connexion et le protocole réseau. Cet article utilise les valeurs par défaut pour toutes les options.

Une fois que vous avez renseigné tous les champs, sélectionnez **Se connecter**.

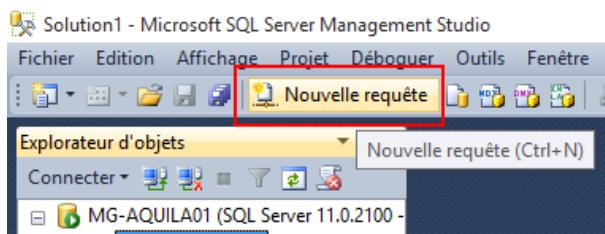
Exemples de connexions réussies

Pour vérifier que votre connexion au serveur SQL Server a réussi, développez et explorez les objets dans l'**Explorateur d'objets**. Ces objets varient en fonction du type de serveur auquel vous choisissez de vous connecter.

Connexion à un serveur SQL Server local ; dans le cas présent, MG-AQUILA01\ MG Utilisateur :

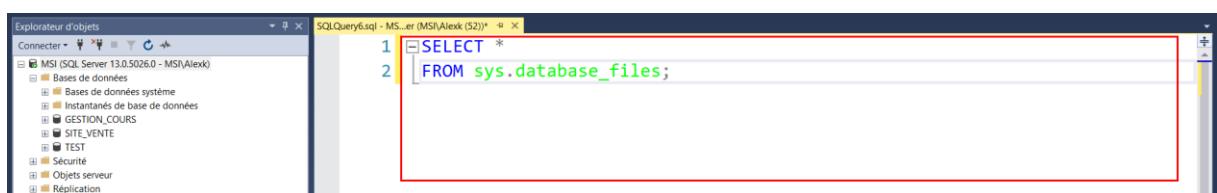


Votre base de données est déjà installée. Pour pouvoir réaliser des requêtes dessus, vous devez ouvrir une nouvelle session. Cliquez sur « Nouvelle requête » ou avec le raccourci clavier Ctrl+N.



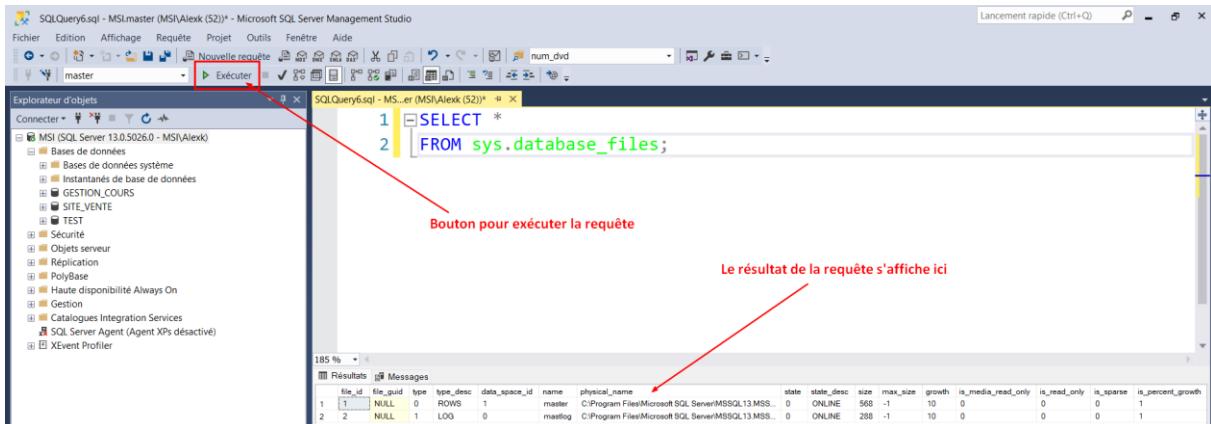
Une nouvelle fenêtre s'ouvre. Le numéro au bout à droite dans cette fenêtre (dans ma capture c'est le 52) n'est pas un simple numéro de fichier ou de requêtes mais un numéro de session (SPID).

Vous saisissez votre requête dans cet espace de cette fenêtre encadrée en rouge :



Pour exécuter votre requête, vous pouvez comme moi simplement appuyer sur la touche F5 ou sinon cliquer sur ce bouton encadré en rouge.

Voici le résultat ci-dessous :



Liens utiles :

<https://learnsql.com/blog/sql-basics-cheat-sheet/>

La clause SELECT permet de récupérer des lignes de la base de données et permet de sélectionner une ou plusieurs lignes ou colonnes d'une ou de plusieurs tables dans SQL Server 2016. La syntaxe complète de l'instruction SELECT est complexe mais en voici les principales clauses :

SELECT	Spécifie des colonnes de résultats
ALL	Renvoie toutes les lignes correspondantes (peu utilisé).
DISTINCT	Permet de retourner une seule fois les lignes dupliquées.
TOP	Limite les lignes renvoyées dans un jeu de résultats.
PERCENT	Renvoie un pourcentage sur les lignes limitées par TOP dans un jeu de résultats.
WITH TIES	Permet de récupérer toutes les lignes similaires à un ensemble de résultats de base.
expression	Colonnes ou calculs que l'on souhaite récupérer. En tapant * on sélectionne toutes les colonnes.
FROM	Spécifie les tables sur lesquelles porte l'ordre.
WHERE	Filtre portant sur les données. Spécifié les conditions pour que les lignes soient présentes au résultat.
GROUP BY	Regroupe les enregistrements en une ou plusieurs colonnes.
HAVING	Filtre portant sur les résultats (conditions de regroupement des lignes).
ORDER BY	Permet de faire des tris sur les résultats. Notamment par ordre croissant (ASC) ou décroissant (DESC).

Pour récapituler :

- ▶ Seule la clause SELECT est vraiment obligatoire.
- ▶ Dans SQL Server 2016, la clause FROM est obligatoire sauf lorsque la liste de sélection ne contient que des constantes, des variables et des expressions arithmétiques et donc aucun nom de colonne.
- ▶ L'ordre des clauses doit être respecté.
- ▶ La clause HAVING n'existe qu'avec la clause GROUP BY.

Exemple (1) :

Nous allons mettre en application ce cours, en appliquant un opérateur logique à notre requête SQL.
Nous allons chercher dans notre GESTION_COURS les employés occupant le poste Commercial.

```
SELECT NOMEMP, PRENOMEMP, POSTE  
FROM EMPLOYE  
WHERE POSTE = 'Commercial';
```

Résultat

	NOMEMP	PRENOMEMP	POSTE
1	DUPONT	Pierre	Commercial
2	BERNARDI	Patrick	Commercial
3	ESTIVAL	Sophie	Commercial
4	JOUINI	Noura	Commercial
5	GOMES	Manuel	Commercial

- ▶ Vous pouvez rajouter des commentaires en dessous de votre ou de vos requêtes. Cela est même recommandé, je vous incite à insérer un maximum de commentaires. Cela est important pour vous-mêmes dans un souci de compréhension de votre requête à postériori mais aussi pour vos collègues qui peuvent être amenés à reprendre votre travail. Pour insérer un commentaire voici la syntaxe à employer.

```
SELECT NOMEMP, PRENOMEMP, POSTE  
FROM EMPLOYE  
WHERE POSTE = 'Commercial';  
/* Nous allons chercher les salariés occupant le poste Commercial */
```

D'autres opérateurs sont aussi utilisés dans les conditions de recherche de Transact-SQL :

▶ [NOT] LIKE

L'opérateur LIKE est utilisé dans la clause WHERE dans votre requête SQL. Cet opérateur permet d'effectuer une recherche sur un modèle particulier. Dans notre exemple qui suit, nous allons l'utiliser pour chercher des enregistrements dont la valeur d'une colonne commence par une lettre en particulier.

Exemple (2) :

Nous allons chercher les employés dont le nom de famille commence par la lettre A.

```
SELECT *
```

```

FROM EMPLOYE
WHERE NOMEMP LIKE 'A%';
/* Nous allons chercher les employés dont le nom de famille commence par 'A' */

```

Résultats

	NUMEMP	NOMEMP	PRENOMEMP	POSTE	SALAIRE	PRIME	CODEPROJET	SUPERIEUR
1	11	ANTHONY	Henri	Responsable technique	39500.00	0.00	PR2	10
2	33	ANDERSON	Paul	Designer	21000.00	1500.00	PR4	14

Le caractère % est un caractère joker qui remplace tous les autres caractères. Dans notre exemple explicite, il remplace tous les caractères venant après la lettre A.

➡ [NOT] BETWEEN

L'opérateur BETWEEN est utilisé dans votre requête SQL pour sélectionner un intervalle de données à l'intérieur de la clause WHERE. Cet intervalle peut être constitué de plusieurs types comme des nombres, des chaînes de caractère mais aussi des dates.

Exemple (3) :

Dans notre exemple, nous allons sélectionner des enregistrements entre 2 salaires différents. Nous allons chercher les employés (noms et prénoms) qui gagnent entre 13.000 € et 17.000 €.

```

SELECT *
FROM EMPLOYE
WHERE SALAIRE BETWEEN '13000' AND '17000';
/* Nous allons chercher qui touchent un salaire en 13.000 et 17.000 Euros */

```

Résultats

	NUMEMP	NOMEMP	PRENOMEMP	POSTE	SALAIRE	PRIME	CODEPROJET	SUPERIEUR
1	4	DUCHATEL	Mireille	Assistant de direction	16500.00	3000.00	PR2	7
2	5	MARTIN	Robert	Secrétaire-comptable	13000.00	0.00	PR3	7
3	9	GALLI	Jean Daniel	Technicien Support	17000.00	2000.00	PR2	5
4	22	DUBOIS	Claire	Secrétaire-comptable	13500.00	0.00	PR4	5
5	25	BOUDIAF	Nadia	Assistant de direction	16500.00	2000.00	PR3	7
6	35	LEGRAND	Isabelle	Secrétaire-comptable	14500.00	500.00	PR5	10
7	36	DIOUF	Mamadou	Technicien Support	17000.00	1000.00	PR2	6

Le résultat comprend 5 entrées, en l'occurrence 5 employés.

➡ IS [NOT] NULL

L'opérateur IS permet de filtrer les valeurs qui sont NULL. Couplé à NOT, nous faisons l'opération inverse en cherchant cette fois les valeurs qui ne sont pas NULL.

Exemple (4) :

Cet exemple va nous permettre de trouver quel salarié a pour prime une valeur NULL. Donc qui ne reçoit aucune prime de quelque sorte qu'elle soit.

```

SELECT NOMEMP, PRENOMEMP
FROM EMPLOYE
WHERE PRIME IS NULL;
/* Nous cherchons le(s) employé(s) (Nom et prénom) dont la prime est NULL */

```

Résultats

	NOMEMP	PRENOMEMP
1	GIMOND	Antoine
2	SAULT	Jean
3	RIVIERE	Maurice
4	RUSSOT	Eric
5	LUNEAU	Henri
6	BRESSON	Pierre
7	LECLERC	Sophie

➡ [NOT] IN

L'opérateur logique IN s'utilise avec la clause WHERE dans une requête SQL. Cet opérateur sert à vérifier une égalité dans une colonne par rapport à une liste de valeurs que l'on a déterminées dans notre requête. Il s'agit dans les faits d'une méthode qui permet de vérifier si une colonne est égale à telle valeur, ou telle valeur, ou encore telle valeur. Cela économise l'utilisation de l'opérateur OR.

Exemple (5) :

Pour cet exercice, nous allons chercher les employés dont le prénom est Rolland, Mireille et Francis dans la table EMPLOYE.

```

SELECT *
FROM EMPLOYE
WHERE PRENOMEMP IN ('Rolland', 'Mireille', 'Francis');
/* Nous cherchons les employés dont les prénoms sont Rolland, Mireille ou Francis */

```

Résultats

	NUMEMP	NOMEMP	PRENOMEMP	POSTE	SALAIRE	PRIME	CODEPROJET	SUPERIEUR
1	2	JOLIBOIS	Rolland	Agent de maîtrise	20500.00	1500.00	PR2	5
2	4	DUCHATEL	Mireille	Assistant de direction	16500.00	3000.00	PR2	7

2 résultats sont retournés dans la mesure où aucun employé ne porte le prénom Francis.

Prédicats composés :

Comme on a pu l'observer dans le TP3, on peut combiner les prédictats à l'aide des opérateurs AND, OR et NOT.

➡ NOT

Inverse l'expression booléenne spécifiée par le prédictat.

Exemple (6) :

Cet Exemple va nous permettre de trouver quels employés (noms et prénoms) ont pour prime une valeur NOT NULL. C'est-à-dire l'inverse du TP4.

```
SELECT NOMEMP, PRENOMEMP  
FROM EMPLOYE  
WHERE PRIME IS NOT NULL;  
/* Nous cherchons les employés (noms et prénoms) dont la prime n'est pas NULL */
```

Résultats

	NOMEMP	PRENOMEMP
14	FARNY	Daniel
15	ESTIVAL	Sophie
16	DUBOIS	Claire
17	HAKIMI	Karim
18	BOUDIAF	Nadia
19	MENDY	Vincent
20	JOUINI	Noura
21	MACHADO	Ricardo
22	BENAMARA	Amina
23	GOMES	Manuel
24	LAROCHE	Jean-Pierre
25	BERTRAND	Marie
26	ANDERSON	Paul
27	KHAN	Ahmed
28	LEGRAND	Isabelle
29	DIOUF	Mamadou

➔ AND

L'opérateur AND permet de s'assurer que plusieurs conditions sont TRUE (vraies) dans une requête. Dans les faits, il permet de vérifier qu'une condition 1 et une condition 2 d'une requête soient vraies. L'opérateur logique AND peut être utilisé dans la condition WHERE. Nous avons précédemment vu cet opérateur lors du TP 3.

Exemple (7) :

Cette requête va nous permettre de trouver les employés (noms et prénoms) qui travaillent sur le projet PR1 et qui touchent une prime supérieure à 1400 €.

```
SELECT NOMEMP, PRENOMEMP  
FROM EMPLOYE  
WHERE (CODEPROJET = 'PR1' AND PRIME > 1400)  
/* Nous cherchons les employés qui travaillent sur le projet PR1 et dont la prime est  
supérieure à 1400 € */
```

Résultats

	NOMEMP	PRENOMEMP
1	BEAUMONT	Jean
2	BEUGNIES	Maurice
3	LAROCHE	Jean-Pierre

➡ OR

L'opérateur OR fonctionne à peu près comme l'opérateur AND. A la seule différence qu'il permet de vérifier qu'une condition 1 ou une condition 2 d'une requête soient vraies. L'opérateur logique OR peut aussi être utilisé dans la condition WHERE.

Exemple (8) :

Nous allons écrire la même requête que celle du TP 9, à la seule différence que l'on va changer le montant de la prime en l'élevant à 5000 €.

```
SELECT NOMEMP, PRENOMEMP
FROM EMPLOYE
WHERE (CODEPROJET = 'PR1' OR PRIME > 5000)
/* Nous cherchons les employés qui travaillent sur le projet PR1 ou qui touchent une
prime supérieure à 5000 € */
```

Résultats

	NOMEMP	PRENOMEMP
1	DUPONT	Pierre
2	BEAUMONT	Jean
3	CANE	Michel
4	BEUGNIES	Maurice
5	LAROCHE	Jean-Pierre
6	KHAN	Ahmed

En utilisant l'opérateur logique OR, le résultat affiche une des 2 conditions, à savoir les salariés travaillant sur le projet PR1. En écrivant la même requête avec l'opérateur AND, le résultat est vide car une des 2 conditions n'est pas respectée

Savoir écrire des requêtes sur plusieurs tables

Les jointures permettent d'associer deux ou plusieurs tables dans une même requête afin d'extraire des données de ces tables en fonction des relations logiques existant entre ces tables. Les jointures indiquent à SQL Server comment il doit utiliser les données d'une table pour sélectionner les lignes d'une autre table.

La jointure permet d'obtenir dans une même ligne des informations provenant de plusieurs tables. On peut faire des jointures sur toutes les colonnes du moment qu'elles sont compatibles.

Une condition de jointure définit la manière dont deux tables sont liées dans une requête :

- ▶ en spécifiant la colonne de chaque table à utiliser pour la jointure. Une condition de jointure qui revient souvent spécifie la clé primaire de la première table et la clé étrangère de la seconde table.
- ▶ en spécifiant un opérateur logique que l'on a vu dans le second chapitre **Introduction à Transact-SQL** (comme par Exemple = ou <>) que l'on utilise pour comparer les valeurs des colonnes si là-encore elles sont compatibles.

Il est possible de spécifier des jointures internes dans les clauses FROM ou WHERE. Les jointures externes ne pourront être spécifiées que dans la clause FROM. Les conditions de jointure peuvent être combinées en cumulant les conditions de recherche WHERE et HAVING dans le but de contrôler les lignes qui sont sélectionnées parmi les tables de base mentionnées dans la clause FROM.

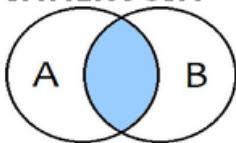
Toutefois, il est préférable de spécifier les conditions de jointure dans la clause FROM car cela vous permet de les séparer des autres conditions de recherche que l'on pourrait spécifier dans une clause WHERE. Là-aussi, cela fait partie des bonnes pratiques de SQL Server 2016. Vous retrouvez ci-dessous une syntaxe de jointure simplifiée d'une clause FROM :

```
SELECT [nom_colonne]
FROM <table-1> join_type <table-2>
ON (condition de jointure)
```

join_type spécifie le type de jointure sachant qu'il en existe plusieurs notamment interne, externe ou croisée. La **condition de jointure** définit le prédicat à évaluer pour chaque paire de lignes jointes. Concernant les types de jointures, je vous propose le schéma ci-dessous qui vous montre de manière explicite les principaux types de jointures possibles dans Transact-SQL.

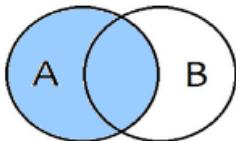
JOINTURES SQL

INNER JOIN



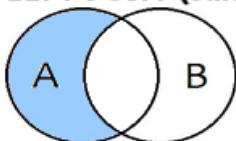
```
SELECT *  
FROM A  
INNER JOIN B ON A.Key = B.Key
```

LEFT JOIN



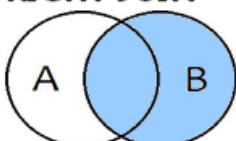
```
SELECT *  
FROM A  
LEFT JOIN B ON A.Key = B.Key
```

LEFT JOIN (sans l'intersection de B)



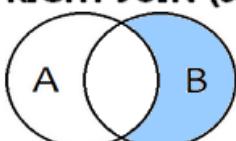
```
SELECT *  
FROM A  
LEFT JOIN B ON A.Key = B.Key  
WHERE B.key IS NULL
```

RIGHT JOIN



```
SELECT *  
FROM A  
RIGHT JOIN B ON A.key = B.key
```

RIGHT JOIN (sans l'intersection de A)



```
SELECT *  
FROM A  
RIGHT JOIN B ON A.key = B.key  
WHERE B.key IS NULL
```

FULL JOIN



```
SELECT *  
FROM A  
FULL JOIN B ON A.key = B.key
```

FULL JOIN (sans intersection)



```
SELECT *  
FROM A  
FULL JOIN B ON A.key = B.key  
WHERE A.key IS NULL  
OR B.key IS NULL
```

CROSS JOIN (permet d'obtenir un produit cartésien)



```
SELECT *  
FROM A  
CROSS JOIN B
```

Écrire des requêtes avec une jointure interne

Le principe d'une jointure interne est de comparer des valeurs des colonnes jointes à l'aide d'un opérateur de comparaison.

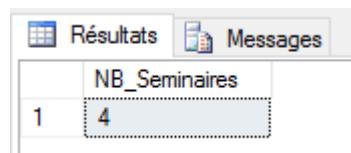
On peut spécifier les jointures internes soit dans une clause FROM ou soit dans une clause WHERE, au choix. Il s'agit du seul type de jointure pris en charge par la norme ISO dans la clause WHERE. Nous allons toutefois privilégier la clause FROM dans cette formation.

Exemple (9) :

Dans cet exercice pratique, nous allons simplement chercher et calculer le nombre de séminaires auxquels l'employé JOLIBOIS va participer. Pour cela nous allons faire une jointure interne entre la table INSCRIT et la table EMPLOYE.

```
SELECT COUNT(*) AS NB_Seminaires
FROM INSCRIT INNER JOIN EMPLOYE
ON EMPLOYE.NUMEMP = INSCRIT.NUMEMP
WHERE NOMEMP = 'JOLIBOIS';
/* Cette requête donne comme résultat le nombre de séminaires auxquels JOLIBOIS va
prendre part.
Pour cela, nous avons joint les tables INSCRIT ET EMPLOYE. */
```

Résultats



	NB_Seminaires
1	4

Vous pouvez vérifier en affichant la table INSCRIT (SELECT * FROM INSCRIT), le NUMEMP de JOLIBOIS est 2, et il participe bien à 4 séminaires.

Exemple (10) :

Dans cette requête nous allons chercher la liste des employés inscrits au cours « *Administration SQL Server* ».

```
SELECT NOMEMP, PRENOMEMP, LIBELLECOURS
FROM EMPLOYE
INNER JOIN INSCRIT
ON EMPLOYE.NUMEMP = INSCRIT.NUMEMP
INNER JOIN SEMINAIRE
ON INSCRIT.CODESEMI = SEMINAIRE.CODESEMI
INNER JOIN COURS
ON SEMINAIRE.CODECOURS = COURS.CODECOURS
WHERE LIBELLECOURS = 'Administration SQL Server';
/* Listing des employés inscrits au cours Administration SQL Server */
```

Résultats

	NOMEMP	PRENOMEMP	LIBELLECOURS
1	DUPONT	Pierre	Administration SQL Server
2	JOLIBOIS	Rolland	Administration SQL Server
3	DUCHATEL	Mireille	Administration SQL Server
4	SAULT	Jean	Administration SQL Server
5	CANE	Michel	Administration SQL Server
6	RUSSOT	Eric	Administration SQL Server
7	BERNARDI	Patrick	Administration SQL Server
8	BEUGNIES	Maurice	Administration SQL Server
9	FARNY	Daniel	Administration SQL Server

Jointures multiples

Exemple (11) :

Dans cet exemple de jointure multiple, nous allons afficher la liste des employés inscrits au séminaire **BR0350216** ainsi que les projets sur lesquels ils travaillent.

```
SELECT EMPLOYE.NOMEMP, EMPLOYE.PRENOMEMP, PROJET.NOMPROJET, SEMINAIRE.CODESEMI,
SEMINAIRE.DATEDEBUTSEM
FROM EMPLOYE
INNER JOIN PROJET ON EMPLOYE.CODEPROJET = PROJET.CODEPROJET
INNER JOIN INSCRIT ON EMPLOYE.NUMEMP = INSCRIT.NUMEMP
INNER JOIN SEMINAIRE ON INSCRIT.CODESEMI = SEMINAIRE.CODESEMI
WHERE SEMINAIRE.CODESEMI = 'BR0350216';
```

Résultat

	NOMEMP	PRENOMEMP	NOMPROJET	CODESEMI	DATEDEBUTSEM
1	DUPONT	Pierre	Le retour de Mouss au Flo	BR0350216	2022-02-13 00:00:00.000
2	JOLIBOIS	Rolland	Cash Pneus	BR0350216	2022-02-13 00:00:00.000
3	BEAUMONT	Jean	Le retour de Mouss au Flo	BR0350216	2022-02-13 00:00:00.000

Corréler des sous-requêtes

Exemple (12) :

Considérons cette requête corrélée :

```
SELECT EMPLOYE.NOMEMP, EMPLOYE.PRENOMEMP
FROM EMPLOYE
WHERE EMPLOYE.NUMEMP IN (
    SELECT INSCRIT.NUMEMP
    FROM INSCRIT
    WHERE INSCRIT.CODESEMI IN (
        SELECT SEMINAIRE.CODESEMI
        FROM SEMINAIRE
        WHERE SEMINAIRE.CODECOURS = 'BR034'
    )
);
```

Résultats

	NOMEMP	PRENOMEMP
1	DUPONT	Pierre
2	JOLIBOIS	Rolland
3	BEAUMONT	Jean
4	DUCHATEL	Mireille
5	MARTIN	Robert
6	MAZAUD	Patricia
7	GIMOND	Antoine
8	SAULT	Jean
9	GALLI	Jean Daniel
10	JACONO	Marie
11	ANTHONY	Henri
12	CANE	Michel
13	GOMEZ	Joseph
14	RIVIERE	Maurice
15	BERNARDI	Patrick
16	BEUGNIES	Maurice

Cette requête récupère les noms et prénoms des employés qui sont inscrits à au moins un séminaire ayant le code de cours '**BR034**'. La première sous-requête corrélée sélectionne les numéros d'employé des inscriptions correspondant aux séminaires ayant le code de cours spécifié. La requête principale utilise ensuite ces numéros d'employé pour récupérer les informations des employés correspondants.

Exemple (13) :

Considérons cette autre requête corrélée :

```
SELECT PROJET.NOMPROJET, PROJET.NOMCONTACT
FROM PROJET WHERE PROJET.CODEPROJET IN
(SELECT EMPLOYE.CODEPROJET FROM EMPLOYE WHERE EMPLOYE.SALAIRE >
10000) ;
```

Résultats

	NOMPROJET	NOMCONTACT
1	Le retour de Mouss au Flo	Monsieur DUROI
2	Cash Pneus	Monsieur MEYZANDIER
3	Les 3 Dauphins	Madame MOULINIE
4	Comptoir pièces auto	Monsieur BERTIN
5	La Chrysalide	Monsieur BONNET

Cette requête récupère les noms de projet et les contacts des projets dont au moins un employé a un salaire supérieur à 10000. La sous-requête corrélée compare le salaire de chaque employé avec la valeur 10000 extraite de la sous-requête imbriquée. Si le salaire de l'employé est supérieur à 10000, le code de projet de cet employé est inclus dans les résultats de la requête principale.

Ces exemples illustrent l'utilisation de requêtes corrélées pour effectuer des comparaisons ou des vérifications entre les enregistrements de différentes tables.

La commande MERGE

La commande MERGE est une commande SQL utilisée pour combiner les opérations INSERT, UPDATE et DELETE dans une seule instruction. Elle permet de synchroniser les données entre une table source et une table cible en fonction de certaines conditions.

Exemple (14) :

Supposons que vous ayez une table "CLIENT_TEMP" contenant des enregistrements que vous souhaitez fusionner avec la table "CLIENT" de votre base de données. Vous voulez insérer les nouveaux clients et mettre à jour les informations des clients existants en utilisant l'adresse e-mail comme critère de correspondance.

Créons ces 2 tables dans **tempdb** :

```
USE tempdb;
-- Création de la table Client
CREATE TABLE Client (
    ID INT PRIMARY KEY,
    Nom VARCHAR(50),
    Prenom VARCHAR(50),
    Adresse VARCHAR(100),
    Email VARCHAR(100)
);
GO
-- Création de la table Client_TMP
CREATE TABLE Client_TMP (
    ID INT PRIMARY KEY,
    Nom VARCHAR(50),
    Prenom VARCHAR(50),
    Adresse VARCHAR(100),
    Email VARCHAR(100)
);
```

Insérons quelques données :

```
INSERT INTO Client (ID, Nom, Prenom, Adresse, Email)
VALUES
    (1, 'Durand', 'Jean', '123 Rue du Commerce', 'jean.durand@example.com'),
    (2, 'Martin', 'Sophie', '456 Avenue des Fleurs', 'sophie.martin@example.com'),
    (3, 'Dubois', 'Pierre', '789 Boulevard du Soleil', 'pierre.dubois@example.com');

-- Génération des données pour la table Client_TMP
INSERT INTO Client_TMP (ID, Nom, Prenom, Adresse, Email)
VALUES
    (1, 'Dupont', 'Alice', '111 Rue de la Liberté', 'alice.dupont@example.com'),
    (4, 'Lefebvre', 'Thomas', '222 Avenue des Champs', 'thomas.lefebvre@example.com'),
    (5, 'Moreau', 'Julie', '333 Boulevard des Roses', 'julie.moreau@example.com');
```

Voici comment vous pouvez le faire en utilisant la commande MERGE :

```
MERGE INTO Client AS C
USING Client_TMP AS CT
ON (C.ID = CT.ID)
WHEN MATCHED THEN
    UPDATE SET
```

```

C.Nom = CT.Nom,
C.Prenom = CT.Prenom,
C.Adresse = CT.Adresse,
C.Email = CT.Email
WHEN NOT MATCHED THEN
    INSERT (ID, Nom, Prenom, Adresse, Email)
    VALUES (CT.ID, CT.Nom, CT.Prenom, CT.Adresse, CT.Email);

```

Résultats

(3 lignes affectées)

Travaux pratiques SQL : Création de requête avec des jointures multiples INNER JOIN.

A PARTIR DE LA BDD GESTION COURS :

Requête (1) :

Afficher le code du séminaire, la date de début et le libellé du cours correspondant.

Requête (2) :

Lister tous les employés (nom, prénom) avec le nom de leur projet affecté (même ceux sans projet).

Requête (3) :

Trouver les séminaires qui n'ont aucun employé inscrit.

Requête (4) :

Compter le nombre total d'inscriptions pour chaque cours (via les séminaires).

Requête (5) :

Afficher pour chaque employé le nom et prénom de son supérieur direct.

Requête (6) :

Lister toutes les participations avec le nom de l'employé, le libellé du cours et la date.

Requête (7) :

Trouver les employés qui se sont inscrits à un séminaire mais n'ont participé à aucune journée.

Requête (8) :

Classer les cours par nombre total de participants distincts.

Requête (9) :

Afficher toutes les inscriptions avec le nom de l'employé, le cours et la date d'inscription.

Requête (10) :

Pour chaque projet, lister les employés et les cours qu'ils ont suivis.

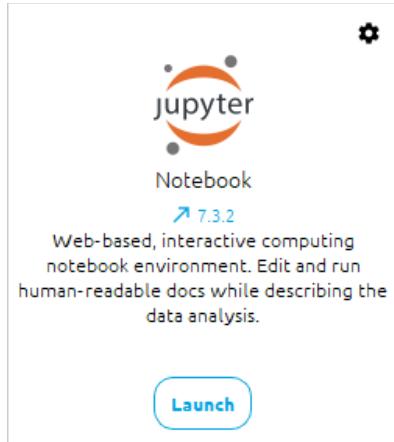
Installation d'Anaconda et de MongoDB



Installer la distribution Python proposée par Anaconda :
<https://www.anaconda.com/products/distribution>.

Exécuter le fichier .exe, et garder les options par défaut.

- Ouvrir Anaconda Navigator
- Lancer jupyter notebook



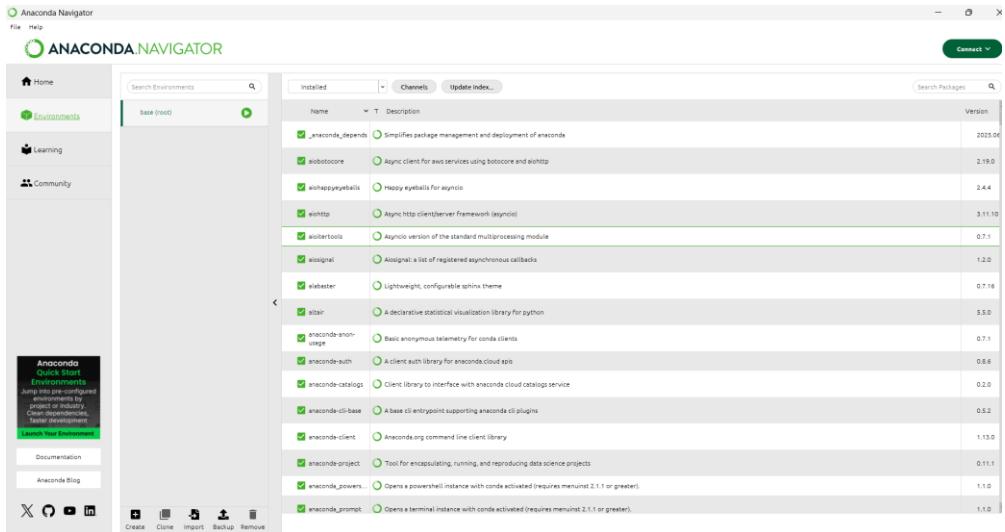
- Ouvrir le notebook **initiation_mongodb.ipynb**



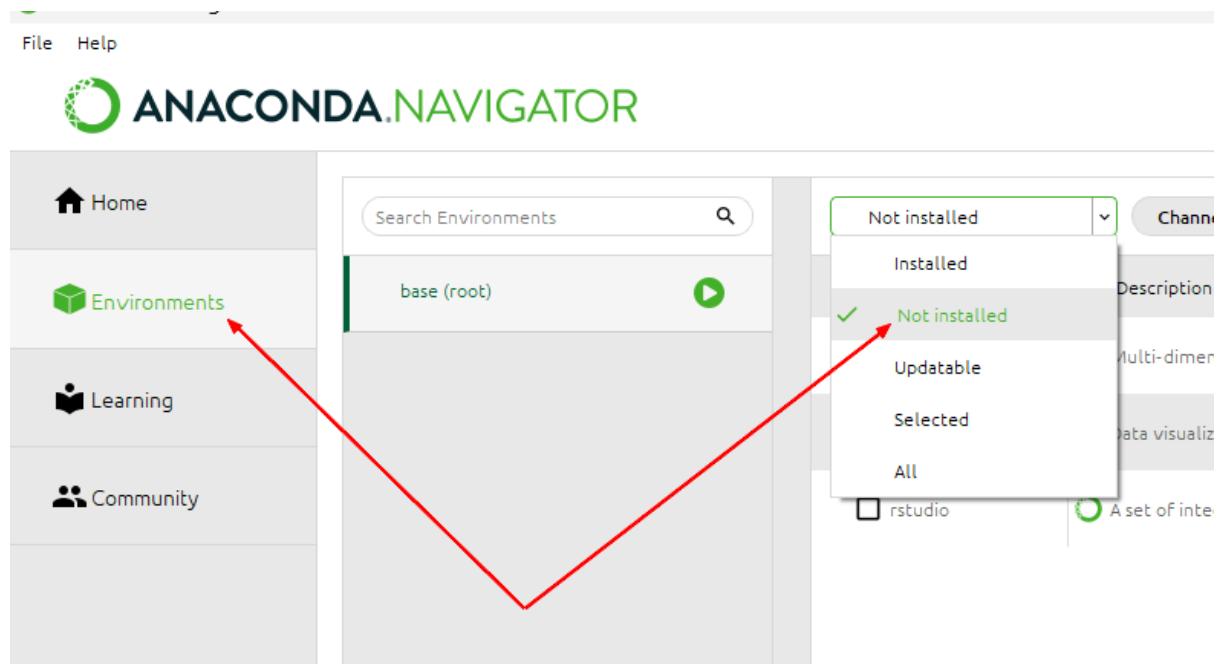
Par la suite...

- ✓ Nous allons installer MongoDB (fourni). Laissez tout par défaut et lancer l'installateur. Cela peut prendre un certain temps.
- ✓ En attendant la fin de l'installation de MongoDB, nous allons installer les librairies mongoDB sur Anaconda, pour faire communiquer notre notebook avec la base MongoDB.

Ouvrez le navigateur Anaconda :



Laissez l'onglet « Environments » à gauche coché, et faites dérouler sur Not Installed :



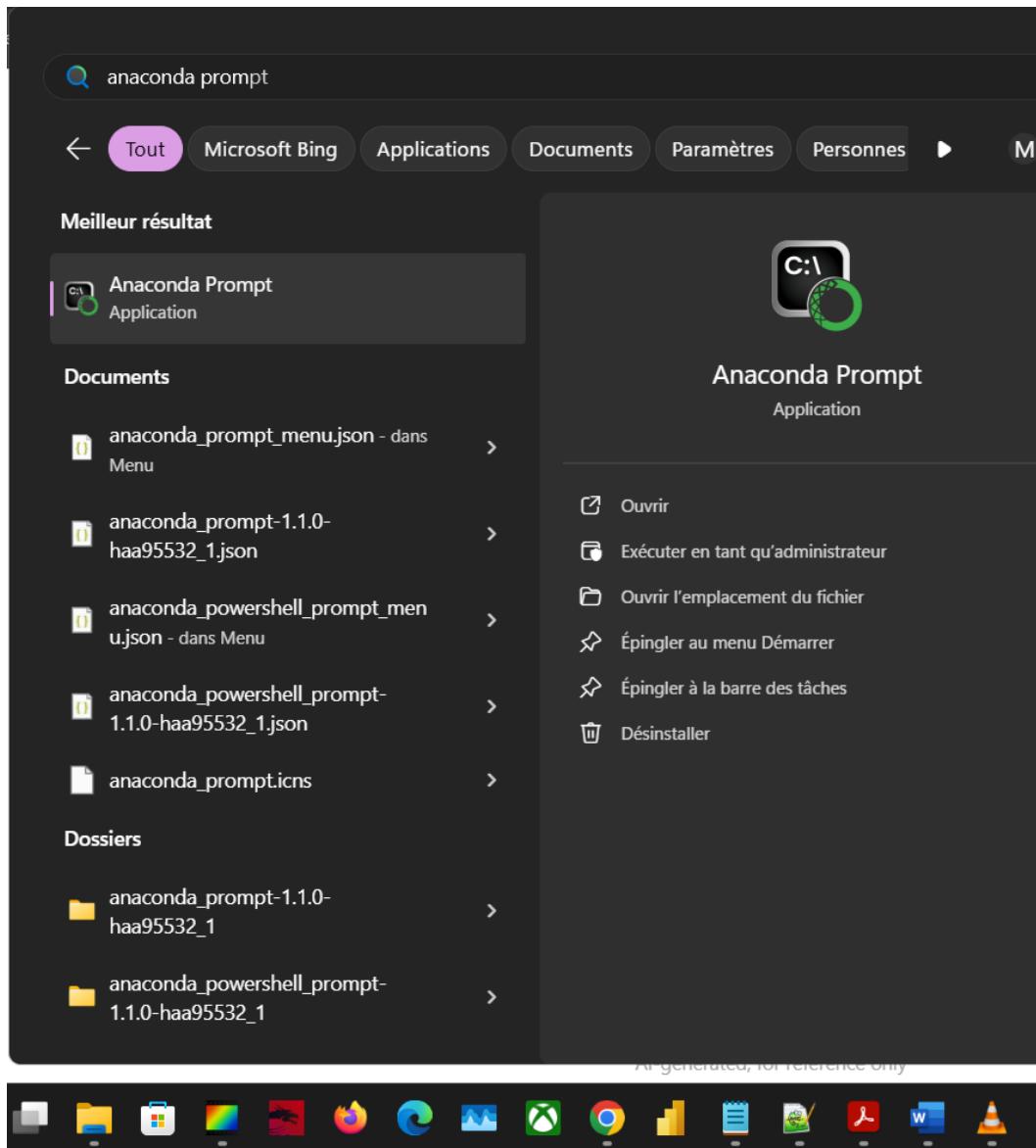
En tapant dans la barre de recherche à droite « pymongo » vous obtenez (normalement, enfin si tout se passe bien) ceci :



Dans le cas où « pymongo » n'apparait pas il va falloir l'installer à la main.

Pour ce faire :

1-Lancer prompt d'Anaconda



2-Taper la commande suivante

```
conda install -c conda-forge pymongo
```

3-Accepter l'installation des paquets

```
(base) C:\Users\Alexk>conda install -c conda-forge pymongo
Do you accept the Terms of Service (ToS) for https://repo.anaconda.com/pkgs/r? [(a)ccept/(r)eject/(v)iew]: a
Do you accept the Terms of Service (ToS) for https://repo.anaconda.com/pkgs/msys2? [(a)ccept/(r)eject/(v)iew]: a
```

4-Patientez jusqu'à obtenir ce résultat

```
The following packages will be downloaded:
package          build
ca-certificates-2025.8.3      h4c7d964_0      151 KB  conda-forge
certifi-2025.8.3              pyhd8ed1ab_0      155 KB  conda-forge
conda-25.7.0                  py313hfa70ccb_0    1.2 MB  conda-forge
dnspython-2.7.0                pyhff2d567_1      168 KB  conda-forge
openssl-3.1.0                 hcfcfb64_3       7.1 MB  conda-forge
pymongo-4.13.0                py313h827c3e9_0    1.6 MB  conda-forge
ucrt-10.0.26100.0              h57928b3_0       678 KB  conda-forge
Total:                      11.0 MB

The following NEW packages will be INSTALLED:

dnspython      conda-forge/noarch::dnspython-2.7.0-pyhff2d567_1
pymongo        pkgs/main/win-64::pymongo-4.13.0-py313h827c3e9_0
ucrt          conda-forge/win-64::ucrt-10.0.26100.0-h57928b3_0

The following packages will be UPDATED:

ca-certificates  pkgs/main/win-64::ca-certificates-202~ --> conda-forge/noarch::ca-certificates-2025.8.3-h4c7d964_0
certifi         pkgs/main/win-64::certifi-2025.4.26-p~ --> conda-forge/noarch::certifi-2025.8.3-pyhd8ed1ab_0
conda           pkgs/main::conda-25.5.1-py313haa95532~ --> conda-forge::conda-25.7.0-py313hfa70ccb_0
openssl         pkgs/main::openssl-3.0.16-h3f729d1_0 --> conda-forge::openssl-3.1.0-hcfcfb64_3

Proceed ([y]/n)?
```

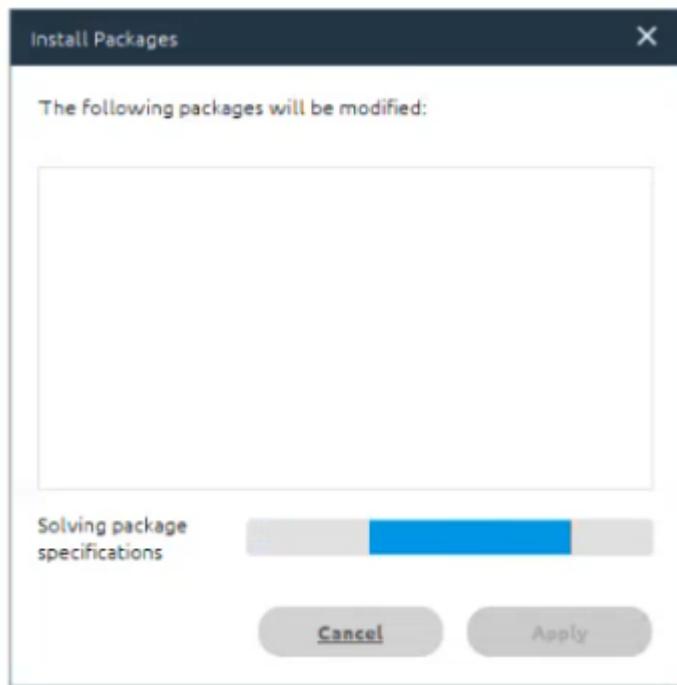
```
Downloading and Extracting Packages:
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\Alexk>
```

5-Revenez dans Anaconda et désormais il apparaît dans les NOT INSTALLED



6-Laissez-vous guider jusqu'à l'installation complète de la librairie (cela peut prendre 15 min voire plus)



7-Très important : pensez à placer le json dans le même dossier que votre notebook.

Nom	Modifié le	Type	Taille
.ipynb_checkpoints	25/09/2025 18:51	Dossier de fichiers	
Atelier1	25/09/2025 17:29	Dossier de fichiers	
JMeriseEtudiant	25/09/2025 16:07	Dossier de fichiers	
Cours_AJC_Mourad_FILALI-09-2025.docx	25/09/2025 17:27	Document Micros...	2 261 Ko
installation_python.ipynb	02/09/2025 11:19	Fichier source Jupy...	168 Ko
JMeriseEtudiant.zip	25/09/2025 16:07	Dossier compressé	42 871 Ko
programme.docx	25/09/2025 14:42	Document Micros...	17 Ko
restaurants_NY.json	04/08/2025 15:09	Fichier source JSON	12 419 Ko
initiation_mongodb.ipynb	25/09/2025 18:53	Fichier source Jupy...	24 Ko
Travaux_pratiques_SQL.txt	25/09/2025 18:24	Document texte	4 Ko

```
[9]: client.list_database_names()
[9]: ['Restaurants', 'admin', 'config', 'local']

Nous allons à présent nous connecter à une base de données spécifique. Si nous essayons de nous connecter à une base qui n'existe pas, elle sera alors automatiquement créée, une fois que nous y aurons ajouté des collections et des données. Nous allons travailler sur les données de restaurants de la ville de New York. Nous vous conseillons de commencer par explorer le fichier restaurants_NY.json manuellement, par exemple à l'aide de NotePad++.

Nous allons ensuite créer une base de données et une collection pour stocker ces données. Dedans nous allons y charger les données qui se trouvent pour l'instant dans le fichier

[10]: db = client['Restaurants']
collection = db['NY']

## Une fois ce code exécuté avec succès, ne le réexécutez pas, afin de ne pas rajouter les données en plusieurs exemplaires

# Code à commenter/décommenter si vous voulez vider votre collection avant de la remplir :
collection.drop()

requesting = []

with open(r"restaurants_NY.json") as f:
    for jsonObj in f:
        myDict = json.loads(jsonObj)
        requesting.append(InsertOne(myDict))

result = collection.bulk_write(requesting)
```

On vérifie que la collection a bien été créée. Si tout va bien, votre collection `NY` doit apparaître lorsque vous exécutez la ligne de code suivante :

```
[12]: db.list_collection_names()
[12]: ['NY']
```

6- Atelier Pratique 2 : Conception d'un schéma de base de données pour un système de gestion de contenu (CMS)

Titre : "Conception d'une base de données pour un CMS"

Consigne unique aux étudiants :

"Une entreprise veut créer un système de gestion d'articles de blog. Elle a besoin de stocker :

- Des articles (*titre, contenu, date, auteur*)
- Des utilisateurs (*nom, email*)
- Des commentaires (*texte, date, auteur*)

Votre mission : Proposez une solution de base de données."

DÉROULEMENT CLAIR

Partie 1 : SQL (2 heures)

Travail à rendre :

1. **Schéma des tables** (sur papier)
2. **Code SQL** pour créer les 3 tables
3. **3 requêtes simples** (ex: "lister les articles")

Partie 2 : NoSQL (1 heure)

Travail à rendre :

1. **Exemple de document MongoDB** pour un article avec ses commentaires
2. **2 avantages/2 inconvénients** par rapport au SQL

EXEMPLE DE RENDU ATTENDU

Partie 1 - SQL :

```
CREATE TABLE articles (id INT, titre VARCHAR, contenu TEXT);
CREATE TABLE commentaires (id INT, article_id INT, texte TEXT);
-- + 3 requêtes
```

Partie 2 - NoSQL :

```
{  
    "article": "Titre",  
    "commentaires": [  
        {"texte": "Super article"}  
    ]  
}
```

Avantage : Plus rapide pour lire un article

Inconvénient : Si on change un commentaire, faut modifier tout l'article

Transactions et Intégrité des Données

1- Principes des Transactions : Définition, propriétés ACID, et leur importance pour l'intégrité des données.

La transaction sous SQL Server est dite ACID.

Cet acronyme résume très bien les caractéristiques principales de la transaction :



Une transaction se découpe de cette manière en utilisant les commandes suivantes : **BEGIN**, **COMMIT**, **ROLLBACK TRAN**.

```
-- Début de notre transaction
BEGIN TRANSACTION
-- nos instructions Sql
-- Par exemple une requête UPDATE et/ou DELETE
-- validation de nos instructions
COMMIT TRANSACTION
-- annulation de nos instructions
ROLLBACK TRANSACTION
-- on utilise soit un commit soit un rollback en fin de transaction
```

Exemple (14) :

Dans ce premier exemple illustrant les transactions, nous allons supprimer de la table EMPLOYE, les employés gagnant plus de 20.000.

```
BEGIN TRAN
DELETE FROM EMPLOYE
WHERE SALAIRE > 20000
COMMIT;
/* les salariés gagnant + de 20.000 ont été supprimés de la table */
```

Résultats

Commande réussi(e)

Exemple (15) :

Dans ce second exemple, nous allons nous placer sur la table PARTICIPER et supprimer les CODEJOUR correspondant à Lundi.

```
BEGIN TRAN
DELETE FROM PARTICIPER
WHERE CODEJOUR = 'LU';
/* Nous allons nous placer sur la table PARTICIPER.
```

Nous allons supprimer les CODEJOUR correspondant au Lundi, soit 'LU' dans notre table.
*/

Résultats

Commande réussie(e)

Exemple (16) :

Ce TP fait directement suite au précédent puisque nous allons poser un rollback pour revenir plus en arrière dans notre transaction.

```
BEGIN TRAN  
DELETE FROM PARTICIPER  
WHERE CODEJOUR = 'LU'  
ROLLBACK TRAN JOUR;  
/* Nous allons revenir en arrière en plaçant un ROLLBACK. */
```

Résultats

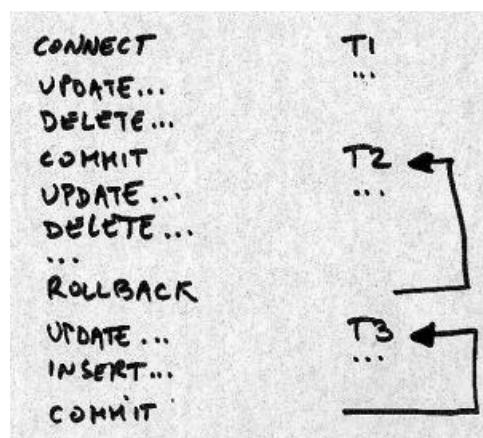
Commande réussie(e)

Ce qu'il faut retenir, c'est qu'une transaction est un ensemble cohérent de modifications faites sur les données.

Une transaction est au choix, soit entièrement annulée (ordre SQL ROLLBACK), soit entièrement validée (ordre SQL COMMIT).

Tant qu'il n'y a pas eu de COMMIT, seul l'utilisateur courant voit ses mises à jour.

Une transaction débute lorsque l'on se connecte à la base (en début de session donc) ou lorsque la transaction précédente se termine.



À signaler que la structure permettant de faire des retours en arrière (ROLLBACK) s'appelle un ROLLBACK SEGMENT et est gérée par le DBA. Les COMMIT et ROLLBACK peuvent être implicites.

- **COMMIT** : Lorsqu'on envoie un ordre SQL de création ou modification de table par exemple, ou plus généralement tout ordre du langage de définition de données.
- **ROLLBACK** : en cas d'interruption anormale du traitement ou d'erreur.

La transaction sous SQL Server est dite **ACID**.

Cet acronyme résume très bien les caractéristiques principales de la transaction :



Une transaction se découpe de cette manière en utilisant les commandes suivantes : **BEGIN**, **COMMIT**, **ROLLBACK TRAN**.

```
-- Début de notre transaction
BEGIN TRANSACTION
-- nos instructions Sql
-- Par exemple une requête UPDATE et/ou DELETE
-- validation de nos instructions
COMMIT TRANSACTION
-- annulation de nos instructions
ROLLBACK TRANSACTION
-- on utilise soit un commit soit un rollback en fin de transaction
```

Exemple (17) :

Dans ce premier exemple illustrant les transactions, nous allons supprimer de la table EMPLOYE, les employés gagnant plus de 20.000.

```
BEGIN TRAN
DELETE FROM EMPLOYE
WHERE SALAIRE > 20000
COMMIT;
/* les salariés gagnant + de 20.000 ont été supprimés de la table */
```

Résultats

Commande réussi(e)

Exemple (18) :

Dans ce second exemple, nous allons nous placer sur la table PARTICIPER et supprimer les CODEJOUR correspondant à Lundi.

```
BEGIN TRAN
DELETE FROM PARTICIPER
WHERE CODEJOUR = 'LU';
/* Nous allons nous placer sur la table PARTICIPER.
Nous allons supprimer les CODEJOUR correspondant au Lundi, soit 'LU' dans notre table.
*/
```

Résultats

Commande réussi(e)

Exemple (19) :

Ce TP fait directement suite au précédent puisque nous allons poser un rollback pour revenir plus en arrière dans notre transaction.

```

BEGIN TRAN
DELETE FROM PARTICIPER
WHERE CODEJOUR = 'LU'
ROLLBACK TRAN JOUR;
/* Nous allons revenir en arrière en plaçant un ROLLBACK. */

```

Résultats

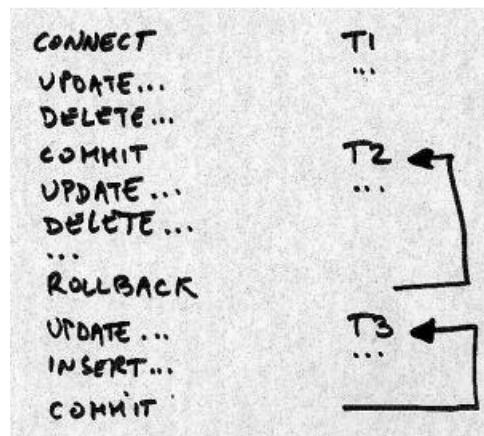
Commande réussi(e)

Ce qu'il faut retenir, c'est qu'une transaction est un ensemble cohérent de modifications faites sur les données.

Une transaction est au choix, soit entièrement annulée (ordre SQL ROLLBACK), soit entièrement validée (ordre SQL COMMIT).

Tant qu'il n'y a pas eu de COMMIT, seul l'utilisateur courant voit ses mises à jour.

Une transaction débute lorsque l'on se connecte à la base (en début de session donc) ou lorsque la transaction précédente se termine.



À signaler que la structure permettant de faire des retours en arrière (ROLLBACK) s'appelle un ROLLBACK SEGMENT et est gérée par le DBA. Les COMMIT et ROLLBACK peuvent être implicites.

- **COMMIT** : Lorsqu'on envoie un ordre SQL de création ou modification de table par exemple, ou plus généralement tout ordre du langage de définition de données.
- **ROLLBACK** : en cas d'interruption anormale du traitement ou d'erreur.

2- Gestion des Transactions : Techniques de programmation pour la gestion des transactions en SQL et dans les SGBD NoSQL.

Définition

Une **transaction** est une séquence d'opérations qui doit être exécutée comme une unité indivisible (principe ACID).

Principes ACID

- **Atomicité** : Tout ou rien
- **Cohérence** : Respect des contraintes
- **Isolation** : Exécution indépendante
- **Durabilité** : Persistance des résultats

2. Gestion des Transactions en SQL (GESTION_COURS)

2.1 Transactions Basiques avec COMMIT/ROLLBACK

Exemple (20) :

```
-- Exemple 1 : Transaction simple d'inscription
BEGIN TRANSACTION;

INSERT INTO INSCRIT (NUMEMP, CODESEMI, DATEINSCRIT)
VALUES (25, 'BR0350216', GETDATE());

INSERT INTO PARTICIPER (NUMEMP, CODESEMI, CODEJOUR)
VALUES (25, 'BR0350216', 'LU');

COMMIT TRANSACTION;
```

2.2 Gestion d'Erreurs avec TRY...CATCH

Exemple (21) :

```
-- Exemple 2 : Transaction avec gestion d'erreurs
BEGIN TRY
    BEGIN TRANSACTION;

    -- Mise à jour du salaire avec prime
    UPDATE EMPLOYE
    SET SALAIRE = SALAIRE + 1000,
        PRIME = ISNULL(PRIME, 0) + 500
    WHERE NUMEMP = 7;

    -- Vérification de la contrainte de salaire
    IF (SELECT SALAIRE FROM EMPLOYE WHERE NUMEMP = 7) > 50000
    BEGIN
        RAISERROR('Salaire trop élevé', 16, 1);
    END

    COMMIT TRANSACTION;
    PRINT 'Transaction réussie';
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT 'Erreur: ' + ERROR_MESSAGE();
END CATCH
```

2.3 Transactions Imbriquées et Points de Sauvegarde

Exemple (22) :

```
-- Exemple 3 : Transaction complexe avec savepoints
BEGIN TRANSACTION MainTransaction;

SAVE TRANSACTION Point1;

-- Insertion d'un nouveau cours
INSERT INTO COURS (CODECOURS, LIBELLECOURS, NBJOURS)
VALUES ('BR080', 'Blockchain Fundamentals', 4);

SAVE TRANSACTION Point2;

-- Création d'un séminaire associé
INSERT INTO SEMINAIRE (CODESEMI, CODECOURS, DATEDEBUTSEM)
```

```

VALUES ('BR0800124', 'BR080', '2024-03-01');

-- Si erreur, retour au point2
IF @@ERROR <> 0
BEGIN
    ROLLBACK TRANSACTION Point2;
    PRINT 'Erreur création séminaire';
END
ELSE
BEGIN
    COMMIT TRANSACTION;
    PRINT 'Cours et séminaire créés';
END

```

3. Transactions Avancées en SQL

3.1 Niveaux d'Isolation

Exemple (23) :

```

-- Exemple 4 : Gestion des niveaux d'isolation
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;

-- Lecture consistante des inscriptions
SELECT COUNT(*) AS NbInscriptions
FROM INSCRIT i
JOIN SEMINAIRE s ON i.CODESEMI = s.CODESEMI
WHERE s.CODECOURS = 'BR035';

-- Mise à jour sécurisée
UPDATE EMPLOYE
SET PRIME = ISNULL(PRIME, 0) + 200
WHERE NUMEMP IN (
    SELECT NUMEMP FROM INSCRIT
    WHERE CODESEMI IN (
        SELECT CODESEMI FROM SEMINAIRE
        WHERE CODECOURS = 'BR035'
    )
);

COMMIT TRANSACTION;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

```

3.2 Transactions Distribuées

Exemple (24) :

```

-- Exemple 5 : Transaction distribuée (si bases multiples)
BEGIN DISTRIBUTED TRANSACTION;

-- Opération sur la base courante
UPDATE GESTION_COURS.dbo.EMPLOYE
SET SALAIRE = SALAIRE * 1.05
WHERE POSTE LIKE '%Commercial%';

-- Opération sur une autre base (exemple)
UPDATE AUTRE_BASE.dbo.SALAIRES
SET MONTANT = MONTANT * 1.05
WHERE EMPLOYEE_ID IN (SELECT NUMEMP FROM EMPLOYE WHERE POSTE LIKE '%Commercial%');

COMMIT TRANSACTION;

```

4. Gestion des Transactions en NoSQL (JSON)

4.1 Approche BASE vs ACID

BASE (Basically Available, Soft state, Eventual consistency) :

- Disponibilité immédiate
- État potentiellement incohérent temporairement
- Cohérence à terme

Gestion des Transactions en MongoDB avec Python

4.2 Configuration de la connexion MongoDB

Exemple (25) :

```
# Configuration MongoDB
from pymongo import MongoClient, WriteConcern
from pymongo.read_concern import ReadConcern
from bson.objectid import ObjectId
import datetime

# Connexion à MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['gestion_contenu']

# Collections basées sur votre JSON
sites = db['sites']
utilisateurs = db['utilisateurs']
articles = db['articles']
```

4.3 Transaction Simple MongoDB

Exemple (26) :

```
# Exemple 3 : Transaction MongoDB de base
def publier_article_transaction(titre, contenu, auteur_id, categorie, tags):
    """Transaction pour publier un nouvel article"""

    # Récupération des informations de l'auteur
    auteur = utilisateurs.find_one({"_id": auteur_id})
    if not auteur:
        raise Exception("Auteur non trouvé")

    # Début de la session transactionnelle
    with client.start_session() as session:
        session.start_transaction(
            read_concern=ReadConcern('snapshot'),
            write_concern=WriteConcern('majority')
        )

        try:
            # 1. Insertion du nouvel article
            nouvel_article = {
                "titre": titre,
                "contenu": contenu,
                "auteur": {
                    "id": auteur_id,
                    "nom": f'{auteur["prenom"]} {auteur["nom"]}',
                    "email": auteur['email']
                },
                "categorie": categorie,
            }
            db.articles.insert_one(nouvel_article)
        except Exception as e:
            session.abort_transaction()
            raise e
        finally:
            session.commit_transaction()
```

```

        "date_publication": datetime.datetime.utcnow(),
        "statut": "publie",
        "tags": tags,
        "commentaires": []
    }

    result_article = articles.insert_one(nouvel_article, session=session)
    print(f"☑ Article créé avec ID: {result_article.inserted_id}")

    # 2. Mise à jour des statistiques du site
    sites.update_one(
        {"_id": 1}, # Tech News France
        {"$inc": {"nombre_articles": 1}},
        session=session
    )

    # 3. Mise à jour du compteur de l'auteur
    utilisateurs.update_one(
        {"_id": auteur_id},
        {"$inc": {"nombre_articles_publies": 1}},
        session=session
    )

    # Validation de la transaction
    session.commit_transaction()
    print("☑ Transaction réussie")

    return result_article.inserted_id

except Exception as e:
    # Annulation en cas d'erreur
    session.abort_transaction()
    print(f"☒ Transaction annulée: {e}")
    raise

# Test
try:
    article_id = publier_article_transaction(
        titre="L'avenir de l'IA générative",
        contenu="Les modèles d'IA générative révolutionnent la création de contenu...",
        auteur_id=2,
        categorie="Intelligence Artificielle",
        tags=["ia", "génération", "innovation"]
    )
except Exception as e:
    print(f"Erreur: {e}")

```

5. Bonnes Pratiques et Considerations

☑ Pour SQL :

- Utiliser des transactions courtes
- Choisir le bon niveau d'isolation
- Toujours gérer les erreurs avec ROLLBACK
- Éviter les transactions longues qui bloquent les ressources

☑ Pour NoSQL :

- Préférer les transactions courtes et simples
- Utiliser des patterns comme Saga pour les workflows complexes

- Implémenter des mécanismes de compensation
- Surveiller la cohérence à terme

✗ À éviter :

- Transactions trop longues
- Niveaux d'isolation trop restrictifs inutilement
- Oublier la gestion d'erreurs
- Transactions dans les boucles

6. Exercices Pratiques pour Étudiants

Exercice 1 : Transaction SQL

Créez une transaction qui transfère un employé d'un projet à un autre en mettant à jour toutes les références.

Exercice 2 : Transaction NoSQL

Implémentez un système de publication d'article avec gestion des commentaires en une transaction atomique.

Exercice 3 : Gestion d'Erreurs

Créez une procédure stockée robuste qui gère les inscriptions aux séminaires avec contraintes de capacité.

3- Contrôle de Conflits et Verrouillage : Mécanismes de verrouillage optimiste et pessimiste, isolation des transactions, et gestion des deadlocks.

1. ⚡ ANALOGIE COMPRÉHENSIBLE

Le problème de base :

Imaginez une bibliothèque avec un seul exemplaire d'un livre populaire

- Sans contrôle : Deux personnes prennent le livre en même temps → Conflit !
- Avec verrouillage : Une personne prend le livre, met un cadenas, le lit, puis le rend

2. ⚡ EXEMPLE 1 : VERROUILLAGE PESSIMISTE (SQL Server)

→ "Je verrouille dès le début pour être sûr"

Exemple (27) :

```
-- ⚡ EXEMPLE SIMPLE : Réservation de place de séminaire
-- Scénario : 2 personnes veulent s'inscrire en même temps

-- 🧑 UTILISATEUR 1 (Premier arrivé)
BEGIN TRANSACTION;

-- 🔒 Je verrouille la place IMMÉDIATEMENT
UPDATE SEMINAIRE
SET places_restantes = places_restantes - 1
WHERE CODESEMI = 'BR0350216';

-- 😊 Je réfléchis 10 secondes avant de valider
WAITFOR DELAY '00:00:10';
```

```

--  Je valide
COMMIT TRANSACTION;
PRINT 'Utilisateur 1 : Incription réussie!';

--  UTILISATEUR 2 (Arrive juste après)
BEGIN TRANSACTION;

--  BLOQUÉ ! Doit attendre que User 1 termine
UPDATE SEMINAIRE
SET places_restantes = places_restantes - 1
WHERE CODESEMI = 'BR0350216';

COMMIT TRANSACTION;
PRINT 'Utilisateur 2 : Incription réussie!';

```

Explication :

- **User 1** verrouille la place immédiatement
- **User 2** doit attendre comme dans une file d'attente
- **Garantie** : Pas de double réservation

3. EXEMPLE 2 : VERROUILLAGE OPTIMISTE (MongoDB)

→ "Je vérifie seulement à la fin si ça a changé"

Exemple (28) :

```

# EXEMPLE 2 CORRIGÉ : Verrouillage Optimiste MongoDB
# Scenario : 2 rédacteurs modifient le même article

import pymongo
from bson.objectid import ObjectId
import time

#  CONNEXION À MONGODB (CE QUI MANQUAIT !)
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["gestion_contenu"] # Utilise le nom de votre base

#  CRÉATION DE L'ARTICLE DE TEST (si il n'existe pas)
if db.articles.count_documents({"_id": ObjectId("507f1f77bcf86cd799439011")}) == 0:
    db.articles.insert_one({
        "_id": ObjectId("507f1f77bcf86cd799439011"),
        "titre": "Titre Original",
        "contenu": "Contenu de l'article original...",
        "auteur": "Admin",
        "date_creation": time.time()
    })
    print("Article de test créé !")

def modifier_article_simple(article_id, nouveau_titre, redacteur_name):
    """Version optimiste : On vérifie seulement à la fin si ça a changé"""

    print(f"\n== {redacteur_name} COMMENCE À TRAVAILLER ==")

    # 1. Je lis l'article (sans verrouiller)
    article = db.articles.find_one({"_id": ObjectId(article_id)})

    if not article:
        print("X Article non trouvé !")

```

```

        return

titre_original = article['titre']
print(f"👤 {redacteur_name} lit l'article : '{titre_original}'")

# 2. Je travaille sur ma modification (simulation)
print(f"✍️ {redacteur_name} écrit pendant 3 secondes...")
time.sleep(3) # Je prends mon temps pour écrire

# 3. AU MOMENT DE SAUVEGARDER, je vérifie si ça a changé
print(f"💾 {redacteur_name} essaie de sauvegarder...")

resultat = db.articles.update_one(
    {
        "_id": ObjectId(article_id),
        "titre": titre_original # ✖ VÉRIFICATION OPTIMISTE !
    },
    {"$set": {"titre": nouveau_titre}}
)

if resultat.modified_count > 0:
    print(f"✅ {redacteur_name} : Modification sauvegardée !")
    print(f"    Nouveau titre : '{nouveau_titre}'")
else:
    print(f"✖ {redacteur_name} : Oups ! Quelqu'un a modifié entre-temps.")
    print(f"    Je dois recommencer avec la nouvelle version...")

# 🧐 TEST SIMULTANÉ (à exécuter dans deux fenêtres différentes)
print("⚠ DÉMONSTRATION VERROUILLAGE OPTIMISTE")
print("=". * 50)

# Redacteur 1 :
modifier_article_simple("507f1f77bcf86cd799439011", "Titre de Redacteur 1",
"RÉDACTEUR 1")

print("\n" + "=" * 50)

# Redacteur 2 (en même temps) :
modifier_article_simple("507f1f77bcf86cd799439011", "Titre de Redacteur 2",
"RÉDACTEUR 2")

# ✅ VÉRIFICATION FINALE
article_final = db.articles.find_one({"_id": ObjectId("507f1f77bcf86cd799439011")})
print(f"\TITRE FINAL : '{article_final['titre']}'")

```

💻 Explication :

- Chacun travaille librement sans bloquer les autres
- Au moment de sauvegarder : "Est-ce que c'est toujours comme quand j'ai commencé ?"
- Si oui → Sauvegarde, Si non → "Recommencez s'il vous plaît"

4. 🔔 EXEMPLE 3 : NIVEAUX D'ISOLATION (SQL Server)

→ "Combien de confidentialité je veux ?"

Exemple (29) :

-- ⚡ EXEMPLE SIMPLE : Consultation des salaires

```

-- 🏠 NIVEAU 1 : "READ COMMITTED" (Défaut) → Je vois seulement ce qui est validé
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN TRANSACTION;
SELECT * FROM EMPLOYE; -- Je vois les salaires actuels validés
COMMIT TRANSACTION;

-- 🔎 NIVEAU 2 : "REPEATABLE READ" → Ce que je lis ne change pas
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRANSACTION;
SELECT SALAIRE FROM EMPLOYE WHERE NUMEMP = 1; -- Je vois 30000

-- 🕒 Pendant ce temps, quelqu'un d'autre change le salaire à 35000...
SELECT SALAIRE FROM EMPLOYE WHERE NUMEMP = 1; -- Je vois TOUJOURS 30000 !
COMMIT TRANSACTION;

-- 📊 NIVEAU 3 : "SERIALIZABLE" → Personne ne peut modifier ce que je regarde
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;
SELECT * FROM EMPLOYE WHERE SALAIRE > 25000;

-- 🕒 Pendant ce temps, quelqu'un veut ajouter un employé à 30000 → BLOQUÉ !
-- Il doit attendre que je finisse ma lecture
COMMIT TRANSACTION;

```

💡 Explication :

- **READ COMMITTED** : "Je vois les résultats officiels"
- **REPEATABLE READ** : "Ce que je lis reste figé pendant ma lecture"
- **SERIALIZABLE** : "Personne ne peut modifier ce que je suis en train de regarder"

5. ⚡ EXEMPLE 4 : DEADLOCK (Étreinte mortelle)

→ "Chacun attend que l'autre lâche"

Exemple (30) :

On va déclencher un interblocage ici à l'aide de transactions. Nous allons créer deux tables dans tempdb : NOMHOMME et NOMFEMME.

```

Create table NOMHOMME (
Id int identity primary key, Nom nvarchar(50)
)

Insert into NOMHOMME values ('Eric')
-- création de la table NOMHOMME

Create table NOMFEMME (
Id int identity primary key, Nom nvarchar(50)
)
-- création de la table NOMFEMME

Insert into NOMFEMME values ('Erika')

```

Nous définissons deux transactions dans 2 sessions différentes. Dans mon exemple, la **transaction 1** occupe la **session 53** alors que la **transaction 2** occupe la **session 52** comme dans l'illustration ci-dessous. **Nous exécuterons les blocs en suivant un certain ordre !!**

```

SQLQuery2.sql - MG-AQUILA01\tempdb (MG-AQUILA01\MG Utilisateur (53))*
***** Transaction 1 *****/
Begin Tran
Update NOMHOMME Set Nom = 'Eric Transaction 1' where Id = 1
-- Dans la session Transaction 2 executer la première requête
Update NOMFEMME Set Nom = 'Erika Transaction 1' where Id = 1
-- Dans la session Transaction 2 executer la seconde requête
Commit Transaction
***** Fin de la Transaction 1 *****

100 % < Messages
(1 ligne(s) affectée(s))

100 % < Messages
Msg 1205, Niveau 13, État 51, Ligne 1
La transaction (ID de processus 52) a été bloquée sur les ressources verrou par un autre procédu

SQLQuery1.sql - MG-AQUILA01\tempdb (MG-AQUILA01\MG Utilisateur (52))
***** Transaction 2 *****/
Begin Tran
Update NOMHOMME Set Nom = 'Eric Transaction 2' where Id = 1
-- Dans la session Transaction 1 execute la seconde requête
Update NOMFEMME Set Nom = 'Erika Transaction 2' where Id = 1
-- Après quelques secondes il fait sa victime
Commit Transaction
***** Fin de la Transaction 2 *****

100 % < Messages
Requête terminée avec des erreurs. MG-AQUILA01 (11.0 RTM) MG-AQUILA01\MG Utilisa... tempdb 00:00:09 0 lignes

100 % < Messages
Requête terminée avec des erreurs. MG-AQUILA01 (11.0 RTM) MG-AQUILA01\MG Utilisa... tempdb 00:00:02 0 lignes

```

Voici donc la transaction 1 :

```

***** Transaction 1 *****/
BEGIN TRAN
UPDATE NOMHOMME SET Nom = 'Jean Transaction 1' WHERE Id = 1

-- Dans la session Transaction 2 executer la première requête

UPDATE NOMFEMME SET Nom = 'Denise Transaction 1' WHERE Id = 1
-- Dans la session Transaction 2 executer la seconde requête Commit Transaction

***** Fin de la Transaction 1 *****/

```

Voici enfin la transaction 2 :

```

***** Transaction 2 *****/
Begin Tran
Update NOMFEMME Set Nom = 'Julie Transaction 2' where Id = 1

-- Dans la session Transaction 1 execute la seconde requête

Update NOMHOMME Set Nom = 'Jeremy Transaction 2' where Id = 1
-- Après quelques secondes il fait sa victime Commit Transaction

***** Fin de la Transaction 2 *****/

```

Nous allons d'abord exécuter la première requête de la transaction 1 comme ci-dessous :

```

Begin Tran Update NOMHOMME
Set Nom = 'Jean Transaction 1' where Id = 1

```

Allez dans l'autre session où se trouve la transaction 2 pour effectuer la MAJ de NOMFEMME :

```

Begin Tran Update NOMFEMME
Set Nom = 'Julie Transaction 2' where Id = 1

```

Revenons dans la session de la transaction 1, nous allons essayer de mettre à jour à nouveau le NOMFEMME :

```

Update NOMFEMME
Set Nom = 'Denise Transaction 1' where Id = 1

```

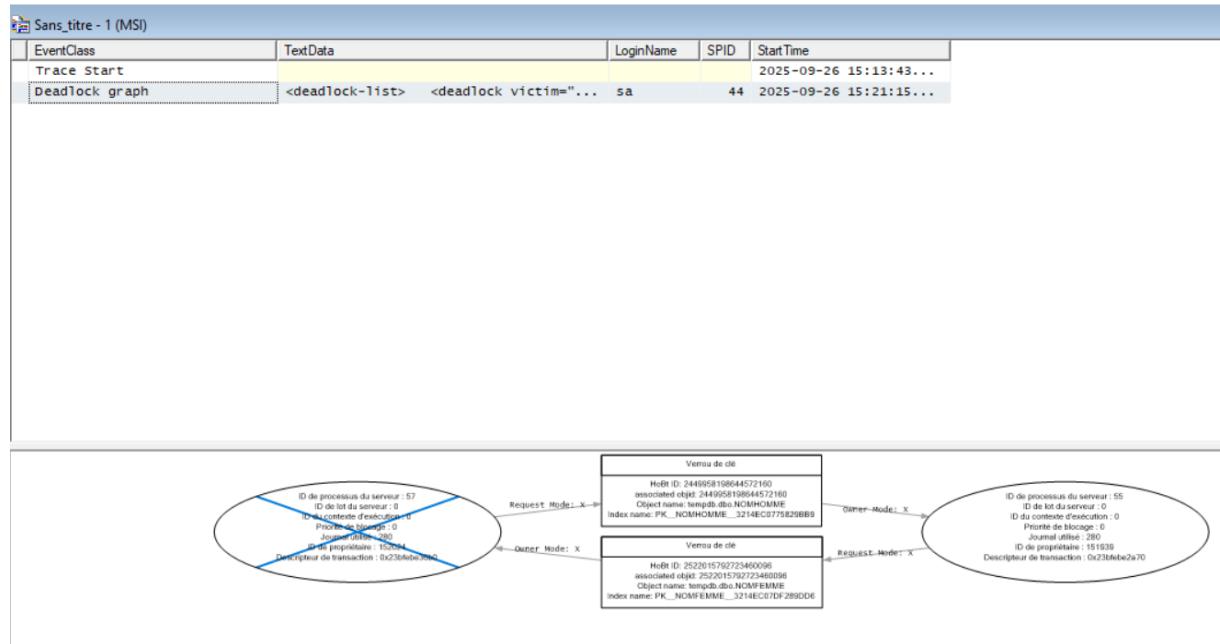
Cette fois-ci, SQL Server ne répond pas. L'exécution est en cours auprès du moteur de BDD, mais ce moteur refuse de répondre tout simplement parce que NOMFEMME est verrouillée par cette première transaction qui n'est pas encore fini et qui pour l'instant portait sur le prénom.

Donc si j'ai un autre utilisateur qui veut mettre à jour le nom pour Erika, il doit attendre que cette première transaction soit finie. Et là le temps d'attente peut être long !

Si on retourne dans la section où il y a notre transaction 2 et qu'on met à jour Éric :

```
Update NOMHOMME
Set Nom = 'Jeremy Transaction 2' where Id = 1
```

Le moteur de BDD indique très rapidement cette fois un message d'erreur. On libère la transaction à l'aide du ROLLBACK. Ce mécanisme est donc le **verrou mortel (deadlock)**.



6. 🔥 EXEMPLE 5 : GESTION SIMPLE DES CONFLITS (Python/MongoDB)

Exemple (31) :

```
# EXEMPLE SIMPLE : Système de likes sur un article
```

```
def ajouter_like_simple(article_id, utilisateur):
    """Version simple avec réessay automatique"""

    tentatives = 0
    while tentatives < 3: # On réessaie max 3 fois
        try:
            # Trouver l'article actuel
            article = db.articles.find_one({"_id": ObjectId(article_id)})
            likes_actuels = article.get('likes', [])

            # Vérifier si l'utilisateur a déjà liké
            if utilisateur in likes_actuels:
                print("X Vous avez déjà liké cet article!")
                return False
```

```

# Ajouter le like
nouveaux_likes = likes_actuels + [utilisateur]
resultat = db.articles.update_one(
    {"_id": ObjectId(article_id)},
    {"$set": {"likes": nouveaux_likes}}
)

if resultat.modified_count > 0:
    print("☑ Like ajouté avec succès!")
    return True
else:
    print("✗ Conflit détecté, réessai...")
    tentatives += 1
    time.sleep(1) # Attendre 1 seconde

except Exception as e:
    print(f"✗ Erreur: {e}")
    tentatives += 1

print("✗ Échec après 3 tentatives")
return False

# TEST
ajouter_like_simple("507f1f77bcf86cd799439011", "Pierre")
ajouter_like_simple("507f1f77bcf86cd799439011", "Marie") # En même temps

```

7. 📋 RÉSUMÉ SIMPLIFIÉ

Type	Comment ça marche	Quand l'utiliser
Verrouillage Pessimiste	"Je verrouille dès le début"	Réservations, transactions bancaires
Verrouillage Optimiste	"Je vérifie seulement à la fin"	Wikis, éditeurs collaboratifs
Read Committed	"Je vois seulement ce qui est validé"	Lecture normale des données
Repeatable Read	"Mes données ne changent pas pendant ma lecture"	Rapports financiers
Serializable	"Personne ne modifie ce que je regarde"	Calculs critiques

8. ⚙ EXERCICES (optionnels)

Exercice 1 : Comprendre le blocage

```

-- Session 1 :
BEGIN TRANSACTION;
UPDATE EMPLOYE SET SALAIRE = 35000 WHERE NUMEMP = 1;
-- Ne pas committer tout de suite

-- Session 2 (essayer en même temps) :
UPDATE EMPLOYE SET SALAIRE = 40000 WHERE NUMEMP = 1;
-- Que se passe-t-il ? Pourquoi ?

```

Exercice 2 : Conflit optimiste

Sur python

```
# Ouvrir deux fenêtres Python en même temps et exécuter :
modifier_article_simple("507f1f77bcf86cd799439011", "Mon Titre")
```

```
# Observer ce qui se passe
```

Exercice 3 : Solution deadlock

```
-- Comment modifier l'exemple du deadlock pour qu'il n'arrive plus ?  
-- Indice : Toujours prendre les livres dans le même ordre (par ID)
```

4- Atelier Pratique 3 : Implémentation de transactions sécurisées dans un scénario de commerce électronique.

🎯 Objectif

Mettre en place une mini-boutique en ligne **TechShop** dans SQL Server et apprendre à gérer :

- le passage de commandes de manière atomique,
- les annulations/remboursements,
- la gestion concurrente des stocks,
- la production de rapports et vues analytiques.

L'accent est mis sur les **transactions ACID**, la **sécurité des données** et la **gestion des erreurs**.

📋 Tâches à effectuer

1. Création et initialisation de la base

- Créer la base TechShop.
- Créer les tables : Clients, Produits, Commandes, DetailsCommande, Inventaire.
- Insérer des **données de test** (clients, produits, inventaire initial).

2. Mise en place de procédures sécurisées

- **sp_PasserCommande**

Implémenter une transaction qui :

1. Vérifie l'existence du client.
2. Vérifie la validité du panier (JSON) et des produits.
3. Vérifie les stocks disponibles.
4. Calcule le total de la commande.
5. Vérifie le solde du client.
6. Réserve les stocks et débite le client.
7. Crée la commande et ses détails.
8. Gère les erreurs avec rollback en cas de problème.

- **sp_AnnulerCommande**

Implémenter une transaction qui :

1. Vérifie que la commande peut être annulée.
2. Libère les stocks réservés.
3. Rembourse le client.
4. Met à jour le statut de la commande.

- **sp_VerifierStock**

Mettre en place une vérification avec **verrouillage pessimiste** (UPDLOCK + REPEATABLE READ) pour tester la concurrence.

3. Scénarios de test

Exécuter et observer les résultats des tests :

- Passage d'une commande valide.
- Commande avec stock insuffisant.
- Commande avec solde insuffisant.
- Annulation d'une commande.
- Vérification concurrente de stock avec deux sessions.

4. Vues et rapports

- Créer une vue **vw_StatistiquesVentes** (commandes, dépenses, panier moyen, dernière commande par client).
- Créer une vue **vw_EtatInventaire** (état du stock disponible/réservé).
- Créer une procédure **sp_RapportVentes** pour obtenir un **rapport journalier** avec : nombre de commandes, chiffre d'affaires, panier moyen, clients uniques.

5. Exécution finale

- Vérifier l'état initial des tables.
- Exécuter les tests et rapports.
- Observer les **messages de confirmation et d'erreur**.

Compétences travaillées

- Transactions **ACID** (atomicité, cohérence, isolation, durabilité).
- Gestion des **verrous et niveaux d'isolation**.
- Sécurité des données avec contraintes et validations métier.
- Gestion robuste des erreurs (TRY...CATCH et rollback).
- Mise en place de vues et rapports pour l'analyse.

Performances et Optimisation SQL

1- Optimisation des Requêtes : Techniques d'indexation, partitionnement des données, et tuning des requêtes.

▪ Optimisation des index :

Vous pouvez examiner les requêtes que vous prévoyez d'exécuter fréquemment sur vos tables et créer des index adaptés. Par exemple, si vous effectuez souvent des recherches sur le nom d'un projet dans la table "PROJET", vous pouvez créer un index non cluster sur la colonne "NOMPROJET" pour accélérer ces recherches.

Exemple de création d'un index non cluster sur la table "PROJET" :

Relancer le serveur SQL Server et rouvrez SSMS.

En exécutant à nouveau cette requête, vous obtenez une erreur, car le nom logique n'est plus lié à son fichier physique.

Exemple (32) :

```
CREATE NONCLUSTERED INDEX IX_NOMPROJET ON dbo.PROJET (NOMPROJET);
```

Résultat

Commande réussie.

▪ Optimisation des types de données :

Assurez-vous que les types de données utilisés sont appropriés pour les valeurs stockées. Si une colonne ne nécessite pas de décimales, utilisez un type de données entier plutôt qu'un type décimal. Par exemple, si la colonne "NBJOURS" dans la table "COURS" est un nombre entier, vous pouvez la définir comme "INT" au lieu de "DECIMAL(38, 0)".

▪ Maintenance de la base de données :

Planifiez des opérations de maintenance régulières, telles que la réorganisation d'index et la mise à jour des statistiques.

Exemple (33) :

Exemple de réorganisation d'index :

```
ALTER INDEX ALL ON dbo.PROJET REORGANIZE;
```

Résultat

Commande réussie.

Exemple (34) :

Exemple de mise à jour des statistiques :

```
UPDATE STATISTICS dbo.PROJET;
```

Résultat

Commande réussie.

■ Analyse des performances des requêtes :

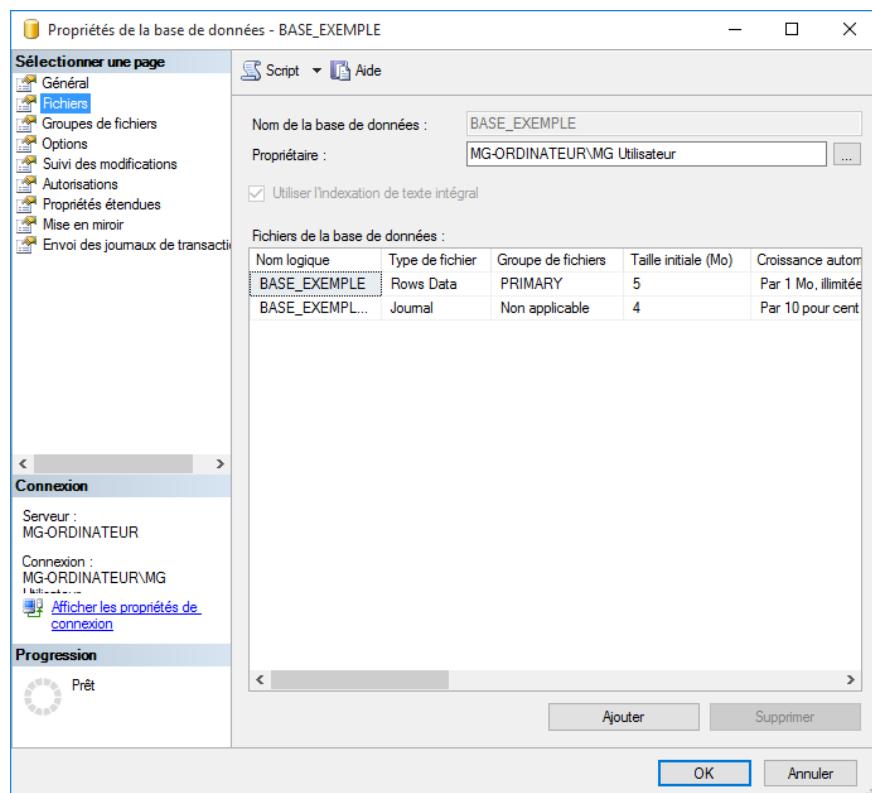
Utilisez l'Explorateur de plans d'exécution (Query Execution Plan) pour analyser les performances de vos requêtes. Examinez les plans d'exécution et recherchez des avertissements ou des opérations coûteuses.

Vue d'ensemble des fichiers et des bases

Repérer les bases de données système préinstallés sur votre SGBDR. Explorez-les par la suite.



En consultant les propriétés de notre BASE_EXEMPLE, nous trouvons les options « Fichiers » et « Groupes de fichiers ».



Les fichiers sont tout simplement les fichiers de notre base de données d'un point de vue physique. On trouve donc un fichier de notre BDD au format .mdf ainsi que le journal de notre BDD au format .ldf.

Atelier 2 :

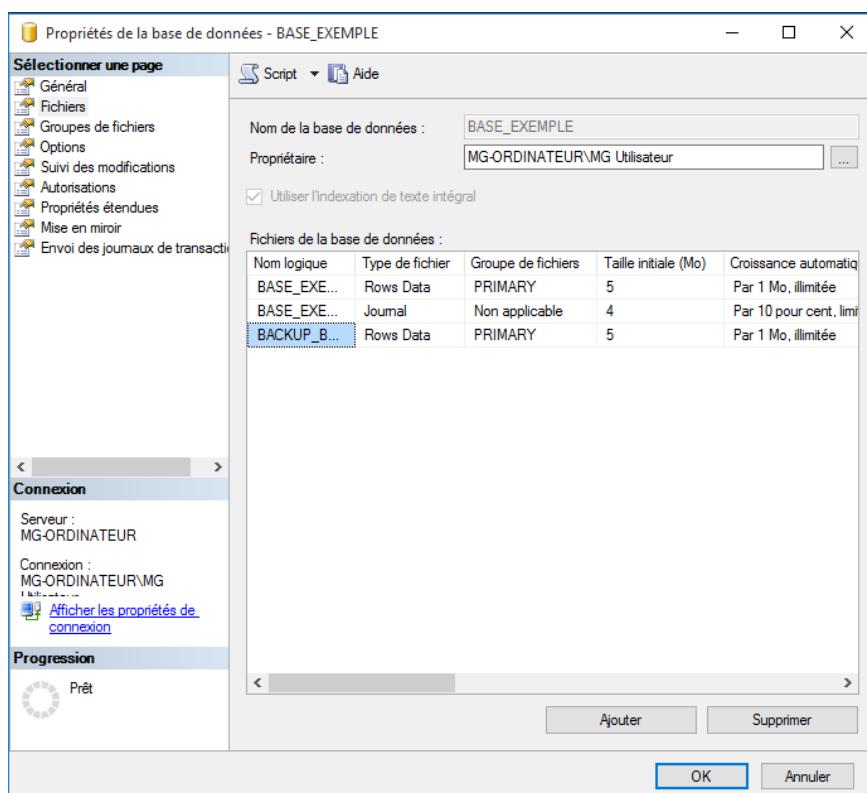
Nous allons créer un nouveau fichier dans notre BDD.

- Aller sur notre **BASE_EXEMPLE** > clic droit > propriétés
- Sélectionnez ensuite « fichiers »
- Pour ajouter un fichier de données ou un fichier de journal des transactions, cliquez sur Ajouter.

- Dans la grille Fichiers de la base de données, tapez le nom logique du fichier. Ce nom doit être unique dans la base de données.
- Sélectionnez le type de fichier : données ou journal.
- Pour un fichier de données, sélectionnez le groupe de fichiers dans lequel le fichier doit être inclus dans la liste, ou sélectionnez <nouveau_groupe_de_fichiers> pour créer un groupe de fichiers. Les journaux des transactions ne peuvent pas être placés dans des groupes de fichiers.
- Spécifiez la taille initiale du fichier. Attribuez aux fichiers de données un maximum d'espace en tenant compte du volume maximal de données qu'est censée contenir la base de données.

Par défaut, les fichiers de données et les journaux des transactions sont placés au même endroit sur le même lecteur pour des raisons de compatibilité avec les systèmes à disque unique, ce qui n'est parfois pas idéal pour les environnements de production.

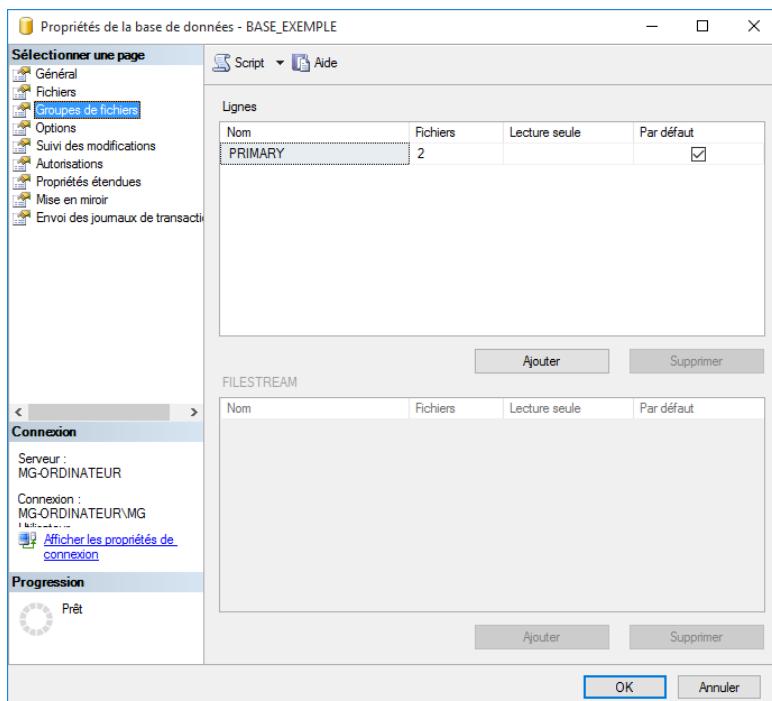
- Cliquez sur OK.



Les groupes de fichiers permettent :

- **Une structure logique.** Cela regroupe des fichiers de données et ce regroupement permet de les gérer comme des unités logiques.
- On distingue **2 types de groupes de fichiers** : Primaire et ceux définis par l'utilisateur.

Pour résumer un groupe de fichiers n'est ni plus ni moins qu'une unité logique auxquels sont rattachés des fichiers physiques de types MDF ou LDF. Quand on crée une BDD, nous avons à la création au moins un groupe de fichiers, le PRIMARY (ou primaire).



Comme vous le remarquez seuls les fichiers DATA sont rattachés à un groupe de fichiers. Très bien, mais à quoi cela sert en pratique de créer ses groupes de fichiers ?

Et bien tout simplement à faire des répartitions de données par type, par exemple Table et Indexes. En général, on sépare donc les tables et les indexes à la fois pour une organisation logique mais aussi physique. Par exemple, à chaque fois que l'on effectue un INSERT dans une table, il s'effectue aussi dans l'index, donc une double écriture sur la table et l'index. On peut aussi créer ces 2 groupes dans 2 différents disques.

Nous pouvons créer un groupe de fichier via l'interface, mais il est tout à fait possible de le faire en T-SQL. Voici la syntaxe :

```
ALTER DATABASE <nom_BDD>
ADD FILEGROUP <nom_groupe_fichiers>;
```

Atelier 3

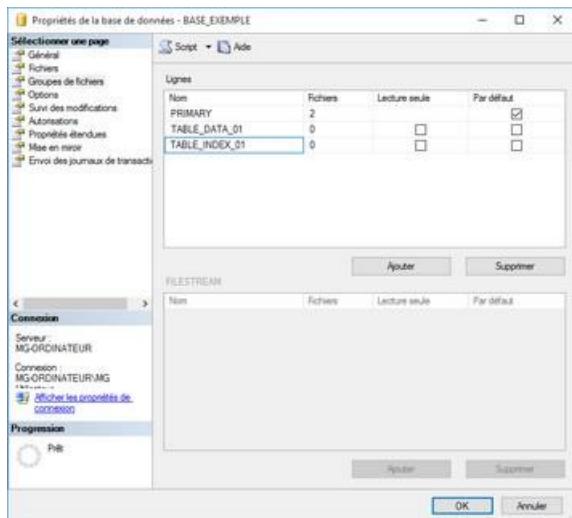
Nous allons créer 2 groupes de fichiers. Un qui va permettre de stocker les objets de type Table, et un qui va permettre de stocker les objets de type Index. Le premier groupe se nommera TABLE_DATA_01

```
ALTER DATABASE
BASE_EXEMPLE ADD FILEGROUP
TABLE_DATA_01;

/* Crédit à la création du groupe de fichiers TABLE_DATA_01 */
```

Pour la création du groupe de fichiers qui stockera les indexes, nous allons passer cette fois par

L'interface graphique.



Pour pouvoir utiliser ces groupes de fichiers, il va falloir rajouter des fichiers physiques. Donc rattacher des fichiers physiques, à ces groupes de fichiers logiques. Voici la syntaxe :

```
ALTER DATABASE <nom_BDD>
ADD FILE <specification_fichier>
TO FILEGROUP <nom_groupe_fichiers>;
```

Atelier 4

Nous allons créer un fichier qui ira sur le groupe de fichier TABLE_DATA_01 en commande SQL.

```
ALTER DATABASE BASE_EXEMPLE ADD FILE ( NAME=BASE_EXEMPLE_DATA_01,
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\BASE_EXEMPLE_DATA_01.NDF', SIZE=10MB,
MAXSIZE=100MB, FILEGROWTH=10MB
) TO FILEGROUP TABLE_DATA_01;
/* Création du fichier BASE_EXEMPLE_DATE_01 dans le groupe de fichiers TABLE_DATA_01
*/
```

Nous pouvons ne pas mentionner le TO FILEGROUP si l'on veut, mais notre fichier ira directement se créer dans le PRIMARY.

Atelier 5

Maintenant nous allons créer une table qui ira se stocker dans le groupe de fichiers TABLE_DATA_01.

```
CREATE TABLE FOURNISSEURS (ID int, nom_fournisseur varchar(50))
ON TABLE_DATA_01
/* Création de la table FOURNISSEURS dans le groupe de fichiers
TABLE_DATA_01 */
```

Partitionnement de table

Dans le cas de tables de grandes dimensions, SQL Server donne la possibilité de stocker la table sur plusieurs groupes de fichiers. On parle alors de partitionnement de table.

Une même table va donc avoir de données qui vont réparti sur plusieurs groupes de fichiers

différents et donc plusieurs fichiers. Pour décider de ce partitionnement, SQL Server va utiliser une fonction de partition. Cette fonction de partition va renvoyer un numéro compris entre 1,n de fichiers de partitions, puis le schéma de partitionnement va affecter en fonction de son numéro un groupe de fichiers destination pour le stockage de la valeur.

Il est bien évident que cette fonction de partitionnement doit prendre en compte une donnée qui est stable dans la table.

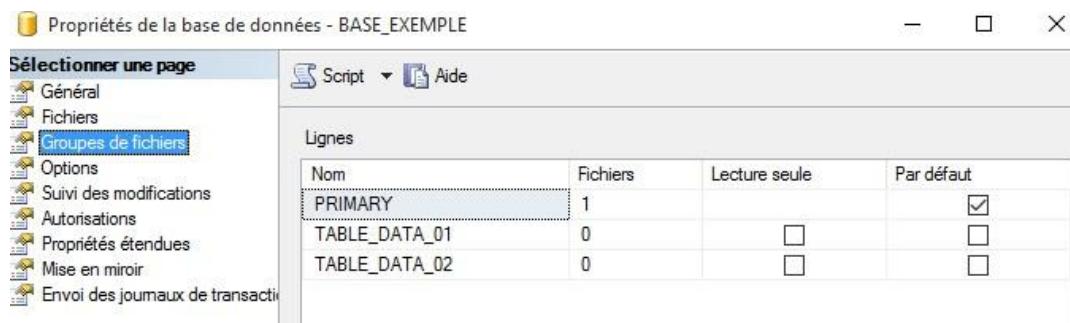
En effet, la mise à jour d'une donnée qui est liée à une fonction de partitionnement pourrait entraîner le déplacement d'une ligne, d'un groupe de fichiers à un autre, et dans ce cas on perdrait fortement en performances. Il est donc important de partitionner par rapport à des valeurs stables et la clé primaire est en général un bon exemple de partitionnement.

Nous allons voir comment mettre en place ce partitionnement :

- La fonction de partitionnement.
- Le schéma de partitionnement.

Atelier 9

Tout d'abord, créez le groupe de fichiers **TABLE_DATA_02** dans notre **BASE_EXEMPLE**.



Nous allons ensuite ajouter un fichier à notre nouveau groupe de fichiers.

```
ALTER DATABASE BASE_EXEMPLE
ADD FILE (
NAME = BASE1,
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\BASE1.ndf',
SIZE = 5MB, MAXSIZE = 50MB, FILEGROWTH = 2MB) TO FILEGROUP TABLE_DATA_02;
/* Création d'un fichier dans un nouveau groupe de fichiers dans la BASE_EXEMPLE */
```

Résultat

Commande(s) réussie(s).

Nous allons ensuite créer une fonction de partitionnement et un schéma de partitionnement dans la foulée. En voici le code T-SQL.

```
CREATE PARTITION FUNCTION etudiantFonction1(int) AS RANGE LEFT FOR
VALUES(1) GO

CREATE PARTITION SCHEME etudiantSchemaPartition1 AS PARTITION etudiantFonction1 TO
(TABLE_DATA_02, TABLE_DATA_01);

GO
```

```
/* Création de la fonction de partition puis création du schéma de partition qui ira sur les 2 groupes de fichiers mentionnés */
```

Résultat

Commande(s) réussie(s).

*Nous allons maintenant créer la table partitionnée **Etudiant**. La **partition** se fait toujours (peu importe le SGBDR) sur la **clé primaire**, et de manière horizontale.*

```
CREATE TABLE
ETUDIANT (
    NOM varchar(50),
    CIVILITE
    varchar(3), FAC
    varchar(50),
    EMAIL
    varchar(10),
    ID int primary key NOT NULL
) ON etudiantSchemaPartition1(ID);
/* création de la table ETUDIANT dans notre schéma de partition (donc 2 FILEGROUPS) */
```

Résultat

Commande(s) réussie(s).

*Affichons maintenant notre table **Etudiant**. La **partition** a bien été prise en compte.*

```
exec sp_help 'ETUDIANT';
/* Affichage DDL de la table ETUDIANT */
```

Résultat

Résultats				Messages		
Identity	Seed	Increment	Not For Replication			
1	No identity column defined.	NULL	NULL	NULL		
RowGuidCol						
1	No rowguidcol column defined.					
Data_located_on_filegroup						
1	commercialSchemaPartition1					
index_name	index_description		index_keys			
1	PK__ETUDIANT__3214EC27B652A610	clustered, unique, primary key located on commercialSchemaPartition1	ID			
constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	PRIMARY KEY (clustered)	PK__ETUDIANT__3214EC27B652A610	(n/a)	(n/a)	(n/a)	ID

2- Analyse des Performances : Outils et méthodes pour le monitoring et l'analyse des performances des bases de données.

Vue d'ensemble des statistiques, de leur calcul et de leur intérêt

Les statistiques dans SQL Server sont des objets de la base de données qui contiennent des informations sur la distribution des données dans une ou plusieurs colonnes de table. Elles sont utilisées par l'optimiseur de requêtes pour générer des plans d'exécution efficaces. Voici une vue d'ensemble des statistiques, de leur calcul et de leur intérêt, avec des exemples :

Création de Statistiques :

Les statistiques sont créées automatiquement pour les colonnes de table lorsque des index (généralement des index non cluster) sont créés. Vous pouvez également les créer manuellement à l'aide de l'instruction CREATE STATISTICS.

Atelier 25

Exemple de Création de Statistiques :

```
-- Créer des statistiques sur la colonne SALAIRE de la table EMPLOYE  
CREATE STATISTICS Stats_Salaire ON EMPLOYE(SALAIRE);
```

Résultats

Commandes réussies.

Utilité des Statistiques :

Les statistiques permettent à l'optimiseur de requêtes de prendre des décisions éclairées sur les plans d'exécution en estimant le nombre de lignes affectées par une opération. Cela aide à optimiser la sélection des opérations de recherche, de tri, de jointure, etc.

Exemple d'Utilité des Statistiques :

Supposons que vous ayez une requête de recherche d'employés dont le salaire est supérieur à 15 000. Les statistiques permettent à l'optimiseur de déterminer si une recherche séquentielle de la table est nécessaire ou si une recherche sélective utilisant un index serait plus efficace.

Mise à Jour des Statistiques :

Les statistiques sont automatiquement mises à jour lorsque des modifications importantes sont apportées aux données (par exemple, insertion, mise à jour ou suppression de données). Cependant, vous pouvez également les mettre à jour manuellement à l'aide de la commande UPDATE STATISTICS.

Atelier 26

Exemple de Mise à Jour des Statistiques

```
-- Mettre à jour les statistiques pour la table EMPLOYE  
UPDATE STATISTICS EMPLOYE;
```

Résultats

Commandes réussies.

Analyse des Statistiques :

Vous pouvez afficher des informations sur les statistiques à l'aide de l'instruction DBCC SHOW_STATISTICS pour voir la distribution des valeurs, les histogrammes et d'autres informations liées aux statistiques.

Atelier 27

Exemple d'Analyse des Statistiques :

```
-- Afficher les statistiques pour la colonne SALAIRE de la table  
EMPLOYE DBCC SHOW_STATISTICS ('EMPLOYE', 'Stats_Salaire');
```

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	Stats_Salaire	oct 14 2023 4:43PM	22	22	15	1	5	NO	NULL	22	0

	All density	Average Length	Columns
1	0.0625	5	SALAIRE

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	7500.00	0	1	0	1
2	8000.00	0	1	0	1
3	9500.00	0	1	0	1
4	9800.00	0	1	0	1
5	10000.00	0	2	0	1
6	10500.00	0	1	0	1
7	10800.00	0	2	0	1
8	11500.00	0	3	0	1
9	12000.00	0	2	0	1

Suppression de Statistiques :

Vous pouvez supprimer des statistiques à l'aide de l'instruction DROP STATISTICS.

Atelier 28

Exemple de Suppression de Statistiques :

```
-- Supprimer les statistiques pour la colonne SALAIRE de la table EMPLOYE  
DROP STATISTICS EMPLOYE.Stats_Salaire;
```

Résultats

Commandes réussies.

En résumé, les statistiques dans SQL Server jouent un rôle essentiel dans l'optimisation des performances des requêtes. Elles permettent à l'optimiseur de requêtes de prendre des décisions éclairées sur la sélection des plans d'exécution, en estimant le nombre de lignes affectées par les opérations. La mise à jour régulière des statistiques est importante pour garantir que les plans d'exécution restent efficaces à mesure que les données évoluent.

Les statistiques et les index intelligents jouent un rôle clé dans l'optimisation des performances des requêtes dans SQL Server. Voici des exemples concrets sur la base de données "BASE_EXEMPLE" pour illustrer comment les statistiques et les index intelligents peuvent améliorer les performances :

Création d'un Index Intelligent :

Supposons que vous ayez besoin de rechercher fréquemment des employés en fonction de leur poste. Vous pouvez créer un index intelligent sur la colonne "POSTE" de la table "EMPLOYE" pour accélérer les opérations de recherche.

Atelier 29

Exemple de Suppression de Statistiques :

```
-- Supprimer les statistiques pour la colonne SALAIRE de la table EMPLOYE
DROP STATISTICS EMPLOYE.Stats_Salaire;
```

Résultats

Commandes réussies.

Cet index intelligent permettra d'accélérer les recherches d'employés en fonction de leur poste.

Création de Statistiques sur un Index Intelligent :

Les statistiques sont automatiquement créées pour les colonnes indexées. Vous n'avez donc pas besoin de créer des statistiques spécifiques pour l'index intelligent que vous venez de créer.

Utilisation d'un Index Intelligent dans une Requête :

Supposons que vous souhaitez rechercher tous les employés occupant le poste de "Vendeur". Grâce à l'index intelligent que vous avez créé, la requête sera plus rapide.

```
-- Rechercher tous les employés qui ont le poste
"Vendeur" SELECT NOMEMP
FROM EMPLOYE
WHERE POSTE = 'DRH';
```

L'optimiseur de requêtes utilisera l'index intelligent "IX_Poste" pour accélérer la recherche.

Mise à Jour Régulière des Statistiques et de l'Index Intelligent :

Assurez-vous de mettre à jour régulièrement les statistiques de la table "EMPLOYE" pour garantir que l'optimiseur de requêtes dispose d'informations actuelles sur la distribution des données. Vous pouvez également surveiller les performances de l'index intelligent et le réorganiser ou le reconstruire si nécessaire.

Utilisation d'Index Columnstore Intelligent (SQL Server 2012 et ultérieur) :

Si vous avez de grandes quantités de données et que vous devez effectuer des opérations d'agrégation (par exemple, la somme des salaires), vous pouvez envisager d'utiliser un index columnstore intelligent pour améliorer les performances de ces opérations.

```
-- Créer un index columnstore intelligent
CREATE NONCLUSTERED COLUMNSTORE INDEX IX_Columnstore ON EMPLOYE(SALAIRE);
```

Un index columnstore intelligent peut accélérer les agrégations massives de données.

En utilisant des index intelligents et en maintenant des statistiques actualisées, vous pouvez

considérablement améliorer les performances de vos requêtes sur la base de données "BASE_EXEMPLE". N'oubliez pas que les besoins en index et en statistiques dépendent des types de requêtes que vous exécutez et des volumes de données que vous manipulez. Il est essentiel de surveiller et d'ajuster ces éléments en fonction de l'évolution de votre application.

■Différents types d'index

SQL Server prend en charge plusieurs types d'index pour optimiser la recherche et la récupération de données. Voici une explication des principaux types d'index, ainsi que des exemples sur la base de données "BASE_EXEMPLE" :

Index Clustered (Index Clusterisé) :

L'index clusterisé définit l'ordre physique des lignes dans une table. Il peut y avoir un seul index clusterisé par table.

Utile pour la recherche rapide de données spécifiques.

Exemple : La clé primaire de la table "COURS" dans la base de données est un index clusterisé sur la colonne "CODECOURS".

```
SELECT *
FROM COURS
WHERE CODECOURS = 'BR057';
```

Index Non-Clustered (Index Non Clusterisé) :

Un index non clusterisé est un index supplémentaire qui stocke une copie des données de la table avec une structure d'arbre B.

Permet d'accélérer les opérations de recherche.

Exemple : L'index non clusterisé "FK1_SEMINAIRE" sur la colonne "CODECOURS" dans la table "SEMINAIRE" pour améliorer les performances des requêtes de jointure.

```
SELECT S.*
FROM SEMINAIRE AS S
INNER JOIN COURS AS C ON S.CODECOURS =
C.CODECOURS WHERE C.LIBELLECOURS = 'UML' :
'Initiation';
```

Index Unique :

Un index unique garantit que les valeurs de la colonne indexée sont uniques et ne peuvent pas être répétées.

Peut être clusterisé ou non clusterisé.

Exemple : L'index clusterisé "PK_COURS" sur la colonne "CODECOURS" dans la table "COURS" garantit que chaque valeur de "CODECOURS" est unique.

```
INSERT INTO COURS (CODECOURS, LIBELLECOURS, NBJOURS)
```

```
VALUES ('BR013', 'Nouveau Cours', 3); -- Cette requête échouera si 'BR013' existe déjà.
```

Index Full-Text (Index Texte Intégral) :

Utilisé pour accélérer la recherche de texte intégral, notamment la recherche de texte dans de grands documents.

Exemple : Si vous aviez une colonne de texte intégral dans votre base de données, vous pourriez créer un index texte intégral pour accélérer les recherches de texte dans cette colonne.

```
-- Créer un catalogue de texte intégral nommé 'MonCatalogueTexteIntegral' dans la
base de données 'BASE_EXEMPLE'
USE BASE_EXEMPLE;
CREATE FULLTEXT CATALOG MonCatalogueTexteIntegral;

-- Créer un index texte intégral sur la colonne 'Description' de la table 'PROJET'
CREATE FULLTEXT INDEX ON PROJET(Description) KEY INDEX PK_PROJET;

SELECT *
FROM
PROJET
WHERE CONTAINS(NOMPROJET, 'Bonnet');
```

Index Spatial :

Utilisé pour la recherche spatiale, notamment pour les données géospatiales.

Exemple : Si votre base de données contenait des informations géographiques, vous pourriez utiliser un index spatial pour accélérer les requêtes liées à la géolocalisation.

```
SELECT *
FROM Lieux
WHERE Coordonnées.STDistance(@MaPosition) < 1000;
```

Index de colonne calculée :

Permet de créer un index sur une colonne calculée.

Utile pour accélérer les requêtes basées sur des calculs ou des fonctions.

Exemple : Si vous aviez une colonne calculée dans votre base de données, vous pourriez créer un index sur cette colonne pour améliorer les performances des requêtes.

La création d'un index sur une expression mathématique n'est pas directement prise en charge dans SQL Server. Cependant, vous pouvez créer une colonne calculée, puis créer un index sur cette colonne. Voici comment vous pouvez le faire :

Créez d'abord une colonne calculée pour « SALAIRE + PRIME » dans votre table « EMPLOYE » :

```
ALTER TABLE EMPLOYE
ADD SalaireAvecPrime AS (SALAIRE + ISNULL(PRIME, 0));
```

Ensuite, créez un index sur la nouvelle colonne calculée « SalaireAvecPrime » :

```

CREATE NONCLUSTERED INDEX IX_SalaireAvecPrime ON EMPLOYE
(SalaireAvecPrime); SELECT NOMEMP, SALAIRE, PRIME
FROM EMPLOYE
WHERE SALAIRE + PRIME > 30000;

```

Index filtré :

Un index filtré permet de définir des critères de filtrage pour les données incluses dans l'index.

Utile pour exclure certaines données de l'index.

Exemple : Vous pourriez créer un index filtré pour inclure uniquement les lignes actives dans une table.

Vous pourriez créer un index filtré pour inclure uniquement les lignes actives dans une table. Par exemple, si vous aviez une colonne "ACTIF" dans la table "EMPLOYE" ou comme dans notre cas ici, une colonne "SUPERIEUR":

```

CREATE NONCLUSTERED INDEX IX_EmployesActifs ON EMPLOYE
(NUMEMP) WHERE SUPERIEUR IS NULL;
SELECT *
FROM
EMPLOYE
WHERE SUPERIEUR IS NULL;

```

Ces types d'index offrent des moyens variés d'optimiser les performances des requêtes dans votre base de données en fonction des besoins spécifiques de votre application.

▪ Différents types de statistiques

SQL Server prend en charge différents types de statistiques pour aider l'optimiseur de requêtes à générer des plans d'exécution efficaces. Voici quelques types de statistiques courants, ainsi que des exemples concrets basés sur votre base de données "BASE_EXEMPLE" :

Statistiques de colonnes simples :

Ces statistiques sont créées automatiquement pour les colonnes indexées ou utilisées dans des clauses WHERE, JOIN ou GROUP BY.

Elles aident l'optimiseur à estimer la cardinalité des résultats de requête.

Exemple :

```

-- L'optimiseur utilise les statistiques de la colonne CODECOURS pour estimer le
nombre de lignes correspondant à une valeur donnée.
SELECT *
FROM
COURS
WHERE CODECOURS = 'BR070';

```

Statistiques de colonnes multicolonnes :

Ces statistiques portent sur plusieurs colonnes et sont utiles pour améliorer les estimations de cardinalité dans des requêtes impliquant plusieurs colonnes.

Exemple :

```
-- L'optimiseur utilise les statistiques multicolonnes pour optimiser les requêtes impliquant les colonnes NUMEMP et CODESEMI.  
SELECT *  
FROM  
INSCRIT  
WHERE NUMEMP = 1 AND CODESEMI = 'BR0340413';
```

Statistiques de colonnes filtrées :

Ces statistiques sont basées sur des conditions de filtrage spécifiques, ce qui peut aider à optimiser les requêtes sur des sous-ensembles de données.

Exemple :

```
-- Statistiques de colonnes filtrées sur le directeur adjoint.  
CREATE STATISTICS Stats_EmployesActifs ON EMPLOYE (SUPERIEUR)  
WHERE SUPERIEUR = 1;
```

Statistiques de colonnes de groupement :

Ces statistiques aident l'optimiseur à générer efficacement des plans de requête pour les clauses GROUP BY.

Exemple :

```
-- L'optimiseur utilise les statistiques de colonnes de groupement pour optimiser les requêtes avec GROUP BY.  
SELECT CODEPROJET, COUNT(*) AS  
NombreEmployes FROM EMPLOYE  
GROUP BY CODEPROJET;
```

Statistiques automatiques :

SQL Server crée automatiquement des statistiques pour les colonnes qui sont fréquemment utilisées dans des requêtes.

Exemple :

```
-- Les statistiques automatiques sont générées pour la colonne CODECOURS de la table COURS en fonction de son utilisation.  
SELECT *  
FROM  
COURS  
WHERE CODECOURS = 'BR013';
```

En utilisant les statistiques appropriées, vous pouvez aider l'optimiseur de requêtes à générer des plans d'exécution plus efficaces, ce qui améliorera les performances de vos requêtes dans la base de données "BASE_EXEMPLE".

1- Analyse des impacts selon les choix

Analyser les impacts des choix que vous faites dans la gestion d'une base de données SQL Server est essentiel pour garantir des performances optimales et une maintenance efficace. Voici comment vous pouvez analyser les impacts des choix liés à la gestion de votre base de données :

Indexation :

- Impact : Les index ont un impact significatif sur les performances de lecture et d'écriture. Plus vous avez d'index, plus les opérations d'écriture sont lentes, mais les lectures sont

rapides.

- **Analyse** : Utilisez des outils de surveillance de performances (comme les rapports d'utilisation de l'index) pour évaluer l'impact de chaque index sur les performances. Supprimez les index inutiles ou redondants.

Fragmentation d'index :

- **Impact** : L'indexation fragmentée peut ralentir les performances de lecture.
- **Analyse** : Utilisez la vue système sys.dm_db_index_physical_stats pour identifier les index fragmentés et planifiez une maintenance d'index régulière.

Choix du moteur de stockage (par exemple, disques SSD vs. disques HDD) :

- **Impact** : Les disques SSD offrent des performances de lecture/écriture plus rapides que les disques HDD.
- **Analyse** : Évaluez les besoins de performance de votre base de données et choisissez le type de stockage approprié en fonction de votre budget et des exigences de performance.

Taille des fichiers de base de données :

- **Impact** : Des fichiers de base de données trop petits peuvent entraîner une croissance fréquente et une fragmentation, tandis que des fichiers trop grands peuvent gaspiller de l'espace disque.
- **Analyse** : Surveillez la croissance de votre base de données et ajustez la taille des fichiers de manière proactive.

Gestion de la mémoire (taille du cache tampon, paramètres Max Server Memory) :

- **Impact** : L'allocation de mémoire affecte les performances globales du serveur SQL.
- **Analyse** : Surveillez l'utilisation de la mémoire et ajustez les paramètres en fonction de la mémoire disponible et des besoins de la base de données.

Stratégie de sauvegarde :

- **Impact** : Les sauvegardes inadéquates peuvent entraîner la perte de données.
- **Analyse** : Vérifiez que votre stratégie de sauvegarde correspond aux exigences de récupération et testez régulièrement les restaurations de sauvegarde.

Gestion des statistiques :

- **Impact** : Des statistiques obsolètes ou inexactes peuvent entraîner de mauvais choix de plans d'exécution.
- **Analyse** : Mettez en place des mises à jour automatiques des statistiques et surveillez les performances des requêtes.

Gestion de l'isolation des transactions :

- **Impact** : Un niveau d'isolation incorrect peut entraîner des verrouillages excessifs ou une mauvaise concurrence.
- **Analyse** : Choisissez le niveau d'isolation en fonction des besoins de votre application et surveillez les problèmes de verrouillage.

Configuration de la journalisation :

- **Impact** : Une journalisation excessive peut entraîner une utilisation excessive de l'espace disque.
- **Analyse** : Configurez la journalisation en fonction des exigences de récupération et surveillez l'espace disque.

Maintenance de l'index et des statistiques :

- Impact : Une maintenance inadéquate peut entraîner une dégradation des performances.
- Analyse : Planifiez régulièrement la maintenance de l'index et des statistiques pour optimiser les performances.

L'analyse des impacts doit être un processus continu pour garantir des performances optimales de votre base de données SQL Server. Surveillez les métriques de performance, effectuez des ajustements en fonction des besoins changeants et tenez compte des retours d'expérience pour prendre des décisions éclairées.

Les statistiques et les index intelligents jouent un rôle clé dans l'optimisation des performances des requêtes dans SQL Server. Voici des exemples concrets sur la base de données "BASE_EXEMPLE" pour illustrer comment les statistiques et les index intelligents peuvent améliorer les performances :

Atelier 30

Problème : Votre base de données contient plusieurs index, et vous soupçonnez qu'il peut y avoir des index inutiles qui ralentissent les opérations d'insertion.

Étape 1 : Analyse

Identifiez les index existants dans votre base de données en exécutant la requête suivante :

```
-- Liste des index dans la base de données  
BASE_EXEMPLE USE BASE_EXEMPLE;  
SELECT OBJECT_NAME(OBJECT_ID) AS TableName, name AS IndexName  
FROM sys.indexes  
WHERE type_desc = 'NONCLUSTERED';
```

Identifiez les opérations d'insertion fréquentes dans votre application qui peuvent être ralenties par les index.

Étape 2 : Action

Utilisez l'analyse pour déterminer quels index ne sont pas essentiels pour les opérations d'insertion et pourraient être supprimés.

Supprimez un index inutile (par exemple) :

```
-- Supprimer l'index inutile  
DROP INDEX IX_poste ON  
EMPLOYEE;
```

Étape 3 : Suivi

Surveillez les performances des opérations d'insertion après la suppression de l'index. Si les performances s'améliorent sans impact négatif sur d'autres opérations, vous avez réussi à optimiser votre base de données.

Continuez à surveiller et à ajuster les index au fur et à mesure que les besoins de votre application évoluent.

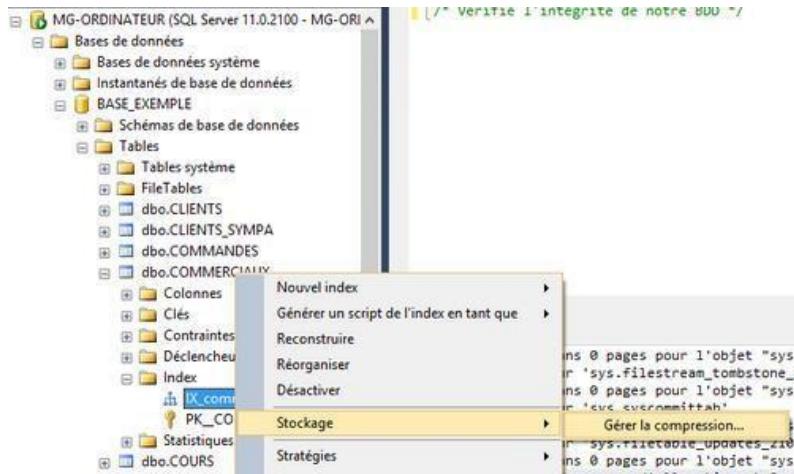
Cet exemple montre comment une analyse d'impact peut vous aider à prendre des décisions éclairées pour améliorer les performances de votre base de données en identifiant et en

supprimant des index inutiles. Il est important de suivre l'impact des actions prises pour vous assurer que les performances s'améliorent conformément à vos attentes.

1- Optimisation des index et statistiques

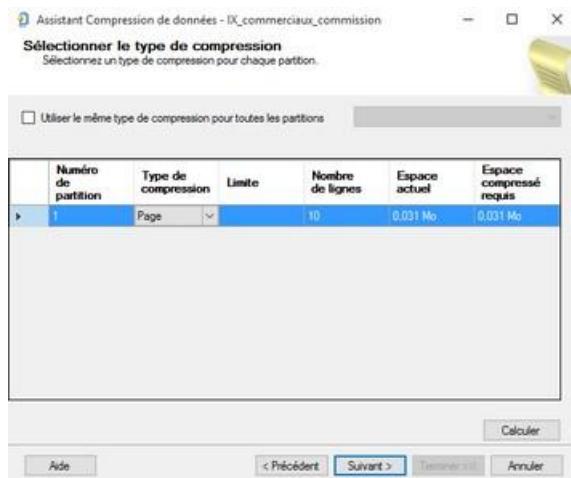
Nous pouvons dans un premier temps faire une compression.

Voir la capture pour le chemin dans Management Studio qui permet d'accéder au menu de compression des index.



Un assistant apparaît. On peut utiliser le même type de compression pour toutes les partitions. Nous disposons d'ailleurs de 3 types de partitions : ROW, PAGE et NONE.

Une option calculer nous indique qu'elle peut calculer la taille de notre index.



On clique sur suivant, s'ouvre un choix. Nous pouvons soit générer un script, exécuter en direct ou planifier. Nous exécutons immédiatement, et la compression se fait.

Atelier 31

Il est possible d'accéder aux informations contenues dans les index. La fonction sys.dm_db_index_physical_stats() renvoie les informations de taille et de fragmentation pour les données et les index de la table ou de la vue spécifiée dans SQL Server.

```
SELECT * FROM sys.dm_db_index_physical_stats(NULL, NULL, NULL, NULL, NULL);
/* Renvoie les paramètres de taille et fragmentation des indexes */
```

	database_id	object_id	index_id	partition_number	index_type_desc	alloc_unit_type_desc	index_depth	index_level	avg_fragmentation_in_perc
1	1	39671189	1	1	CLUSTERED INDEX	IN_ROW_DATA	0	0	0
2	1	87671360	1	1	CLUSTERED INDEX	IN_ROW_DATA	0	0	0
3	1	117575457	0	1	HEAP	IN_ROW_DATA	0	0	0
4	1	119671474	1	1	CLUSTERED INDEX	IN_ROW_DATA	0	0	0
5	1	133575514	0	1	HEAP	IN_ROW_DATA	0	0	0
6	1	149575571	0	1	HEAP	IN_ROW_DATA	0	0	0

Atelier 32

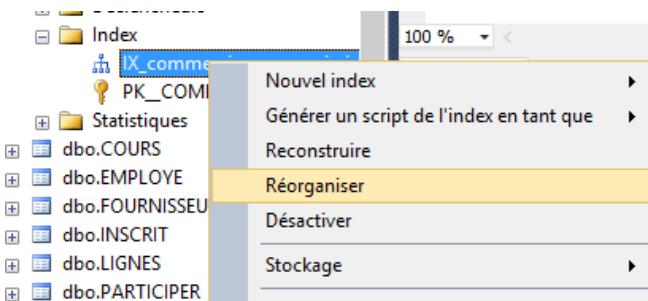
Une autre requête renvoie des résultats approchant, la voici.

```
SELECT * FROM sys.indexes;
/* Renvoie info taille et fragmentation des indexes */
```

	database_id	object_id	index_id	partition_number	index_type_desc	alloc_unit_type_desc	index_depth	index_level	avg_fragmentation_in_perc
1	1	39671189	1	1	CLUSTERED INDEX	IN_ROW_DATA	0	0	0
2	1	87671360	1	1	CLUSTERED INDEX	IN_ROW_DATA	0	0	0
3	1	117575457	0	1	HEAP	IN_ROW_DATA	0	0	0
4	1	119671474	1	1	CLUSTERED INDEX	IN_ROW_DATA	0	0	0
5	1	133575514	0	1	HEAP	IN_ROW_DATA	0	0	0
6	1	149575571	0	1	HEAP	IN_ROW_DATA	0	0	0

Atelier 33

Maintenant nous allons réorganiser sur l'index. On retourne dans Management sur l'index > Clic droit Réorganiser.



Il compacte automatiquement les index. Il n'y a qu'à faire OK pour exécuter la requête.

Pour reconstruire c'est pareil, on repart dans le menu de l'index, on clique sur Reconstruire et on clique sur OK pour exécuter la requête.

Très souvent, lorsque l'on rencontre des problèmes de performances sur SQL Server, et sur d'autre SGBD, le premier réflexe à avoir est la vérification de la fragmentation des indexes. (On peut aussi regarder qui consomme !).

Le but ici n'est pas de refaire l'explication du fonctionnement des indexes, mais juste de fournir quelques requêtes indispensables aux vérifications.

Atelier 34

```
USE BASE_EXEMPLE;
```

```

SELECT dbschemas.[name] as 'Schema',
dbtables.[name] as 'Table',
dbindexes.[name] as 'Index',
indexstats.avg_fragmentation_in_percent
, indexstats.page_count
FROM      sys.dm_db_index_physical_stats(DB_ID(),NULL,NULL,NULL)      AS
indexstats  INNER   JOIN   sys.tables    dbtables   ON   dbtables.[object_id]
=indexstats.[object_id]  INNER   JOIN   sys.schemas    dbschemas   ON
dbtables.[schema_id] =dbschemas.[schema_id]
INNER JOIN sys.indexes AS dbindexes ON dbindexes.[object_id]
=indexstats.[object_id] AND indexstats.index_id =dbindexes.index_id
WHERE indexstats.database_id =DB_ID()
--dbtables.[name] like '%%'
ORDER BY indexstats.avg_fragmentation_in_percent desc

```

	Schema	Table	Index	avg_fragmentation_in_percent	page_count
1	dbo	COURS	PK_COURS	0	1
2	dbo	EMPLOYE	pk_employe	0	1
3	dbo	INSCRIT	pk_inscrit	0	1
4	dbo	PARTICIPER	pk_participer	0	1
5	dbo	PROJET	PK_PROJET	0	1
6	dbo	SEMINAIRE	PK_SEMINAIRE	0	1
7	dbo	SEMINAIRE	FK1_SEMINAIRE	0	1
8	dbo	DEPARTEMENT	PK_DEPARTEM_3214EC272C985D81	0	1
9	dbo	VENDEUR	PK_VENDEUR_3214EC27BA5E8973	0	1
10	dbo	COMMERCIAUX	PK_COMMERCI_3214EC27127D2016	0	1
11	dbo	sysdiagrams	PK_sysdiagr__C2B05B614B7381CE	0	1

Cette requête liste tous les indexes de la base de données en cours, trier par pourcentage de fragmentation. Pour l'exécuter sur une table, dé-commenter la ligne dbtables.[name] like '%%' et ajouter votre table entre les %.

Pour réduire la fragmentation, il faudra ensuite ré-organiser ou reconstruire vos indexes selon les valeurs ci-dessous.

% Fragmentation	Action	Commande SQL
>5 et <30	Ré-organiser l'index	ALTER INDEX REORGANIZE
>30	Re-construire l'index	ALTER INDEX REBUILD

Tout ceci peut bien entendu être automatisé via un plan de maintenance.

Depuis les dernières versions de SQL, les paramètres de fragmentation, de nombre de page ou de dernière utilisation, permettent de spécifier les critères afin de gagner en efficacité et en rapidité lors de l'exécution de ces derniers. Fini les ré-indexation qui durent des week-ends entiers ou qui explosent vos journaux de transactions !

Une autre manière d'atteindre ce résultat :

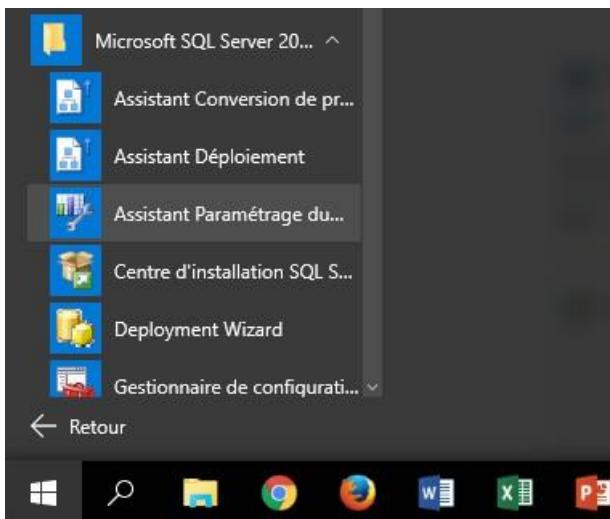
```
SELECT * FROM sys.dm_db_index_physical_stats(DB_ID(), OBJECT_ID('EMPLOYE'), NULL, NULL, 'LIMITED');
```

	Résultats	Messages
1	database_id object_id index_id partition_number index_type_desc alloc_unit_type_desc index_depth index_level avg_fragmentation_in_percent fragment_count avg_fragment_size_kb	5 277576027 1 1 CLUSTERED INDEX IN_ROW_DATA 1 0 0 1 1

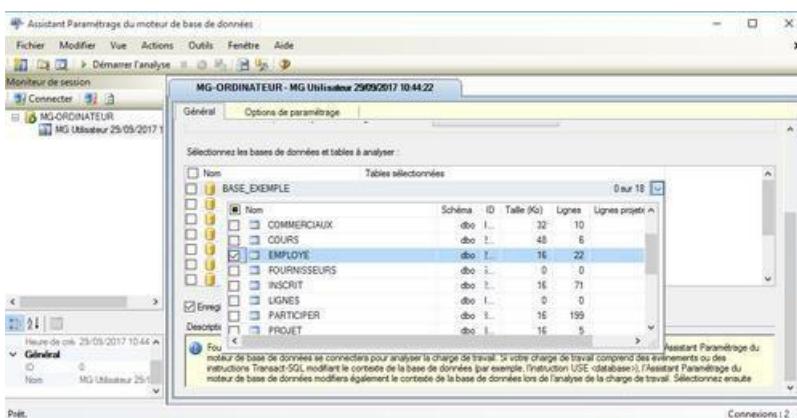
Atelier 35

Rendez-vous sur le menu « démarrer » de Windows, et vous y trouverez l'assistant qui nous intéresse.

Voir image ci-dessous :



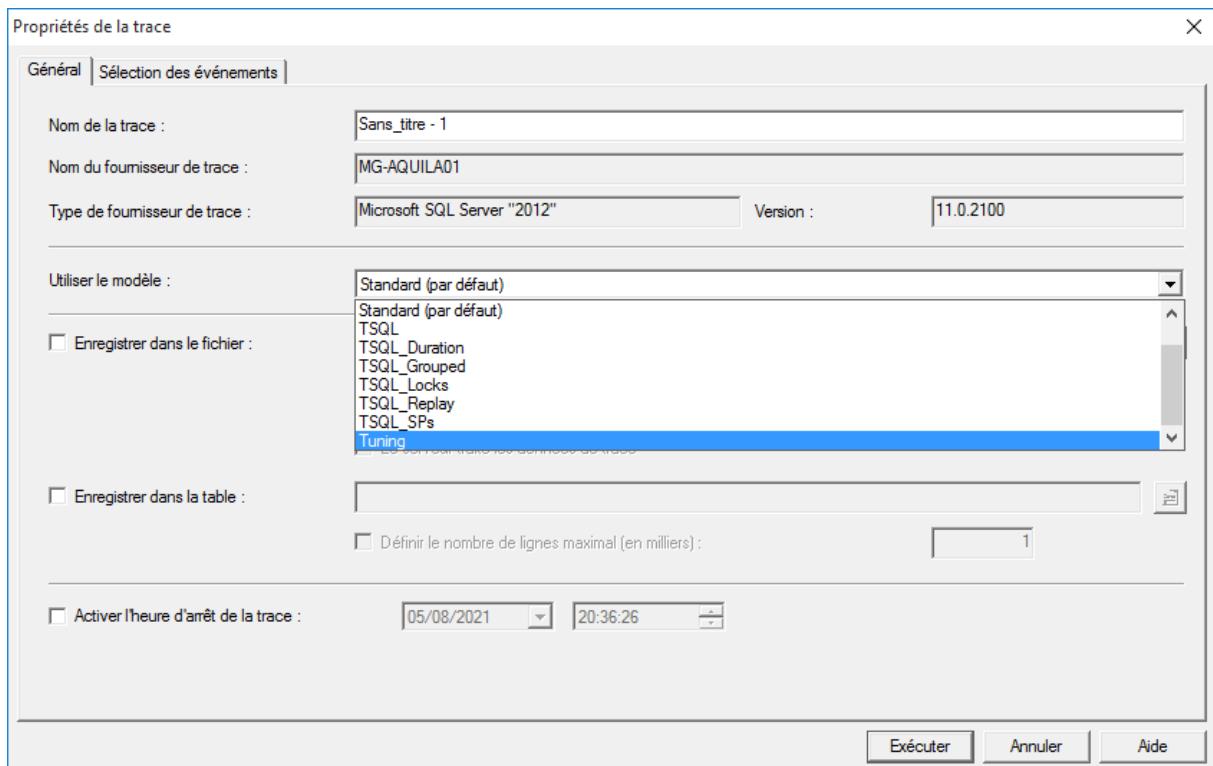
Nous nous connectons au serveur. Choisissons une BDD (BASE_EXEMPLE) et une table (table EMPLOYEE).



Au niveau de la charge de travail nous avons le choix entre un fichier, une table ou le cache du plan. Choisissons un fichier .sql et fermons.

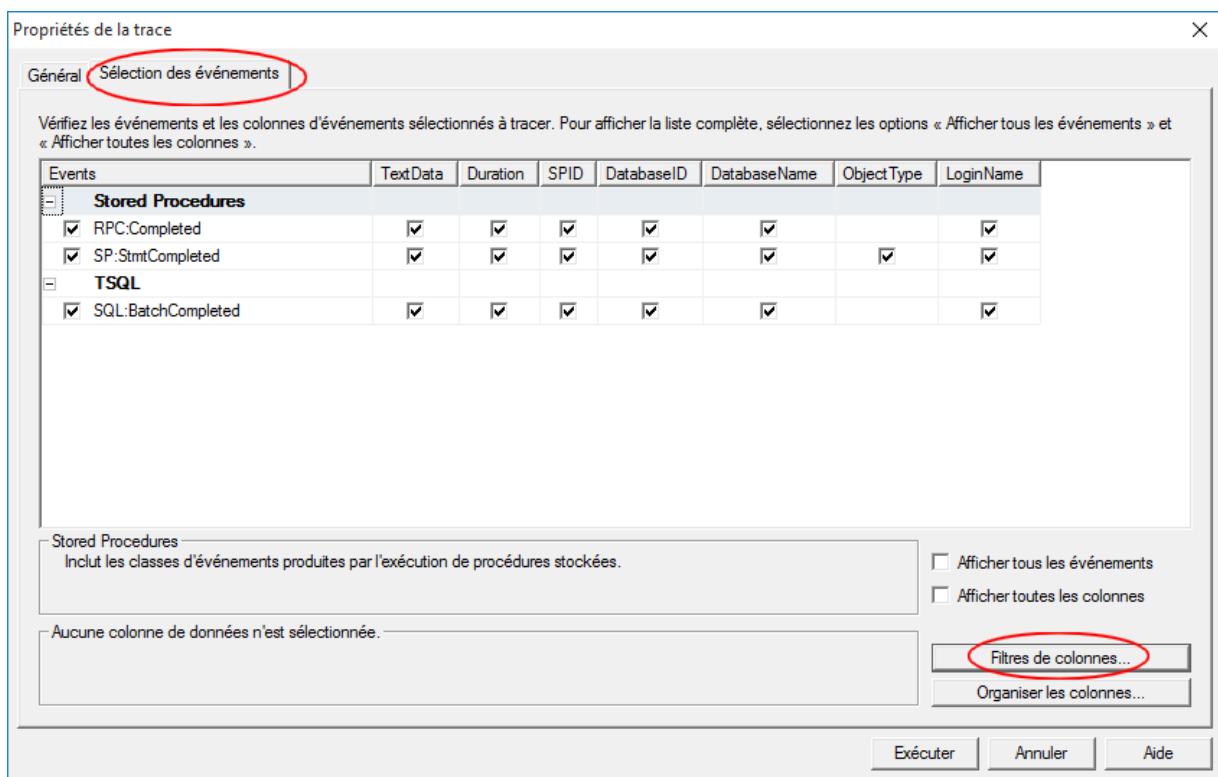
Nous allons travailler sur la table **codePostal**. Si vous n'avez pas (encore) installé cette table dans une de vos bases de données, merci de faire une insertion via SSIS du fichier fourni **codePostal.txt**. Vous obtenez donc une table disposant de 38949 lignes.

A présent, ouvrez SQL Server Profiler. L'idée ici est d'isoler une activité unique sur une seule base de données pour pouvoir mesurer l'efficacité de nos index. Ouvrez donc SQL Server Profiler et procédez aux réglages ci-dessous :

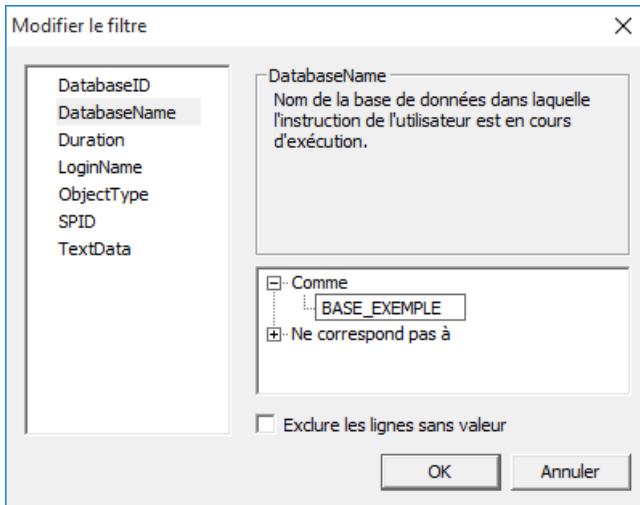


Choisissez en premier lieu le modèle **Tuning**.

Cliquez ensuite sur l'onglet **Sélection des événements** et ensuite sur le bouton **Filtres de colonnes**.



Choisissez ensuite l'option **DatabaseName** et entrez le nom de votre BDD (où se trouve la table codePostal) comme ci-dessous. Chez moi, elle se trouve dans la BASE_EXEMPLE.



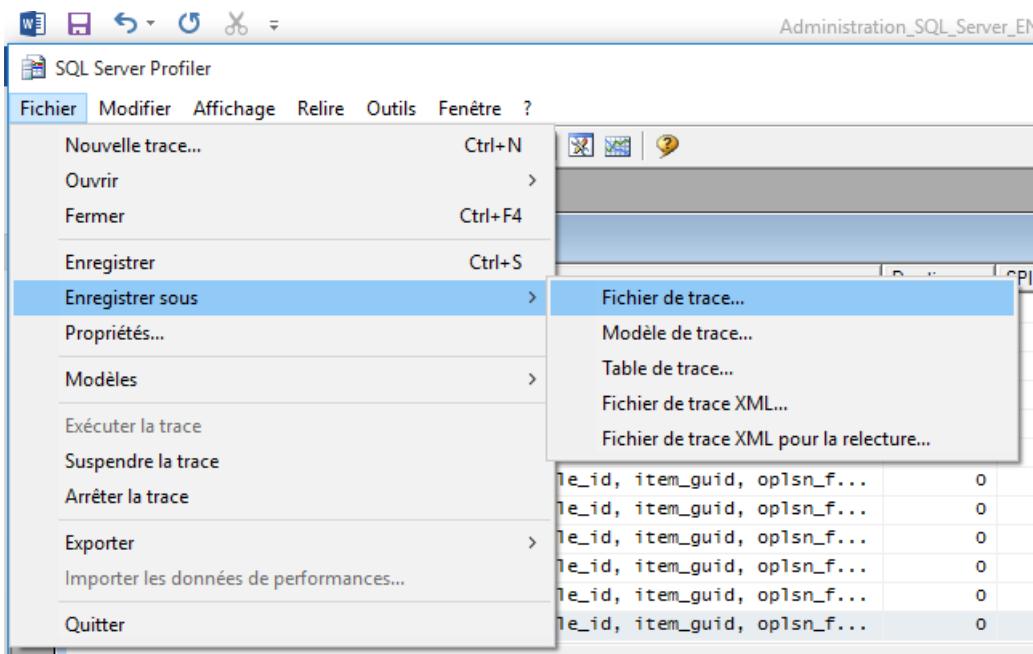
Cliquez sur OK et exécutez la trace juste derrière. A présent, créez cet index dans SSMS :

```
CREATE INDEX ix_commune ON codePostal(commune);
```

Et exécutez au moins à quatre reprises la requête suivante :

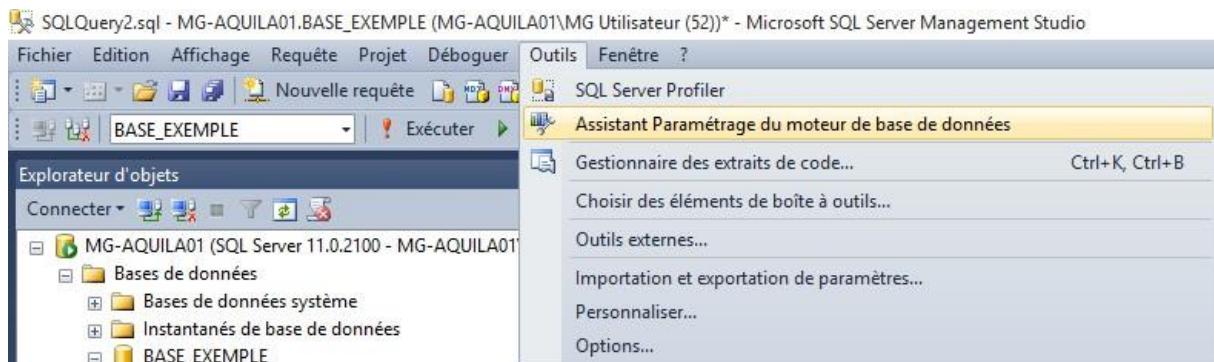
```
SELECT commune, codepos
FROM codepostal
```

Une fois que vous avez exécuté à plusieurs reprises cette requête, enregistrez la charge de travail (workload) dans un fichier de cette manière :



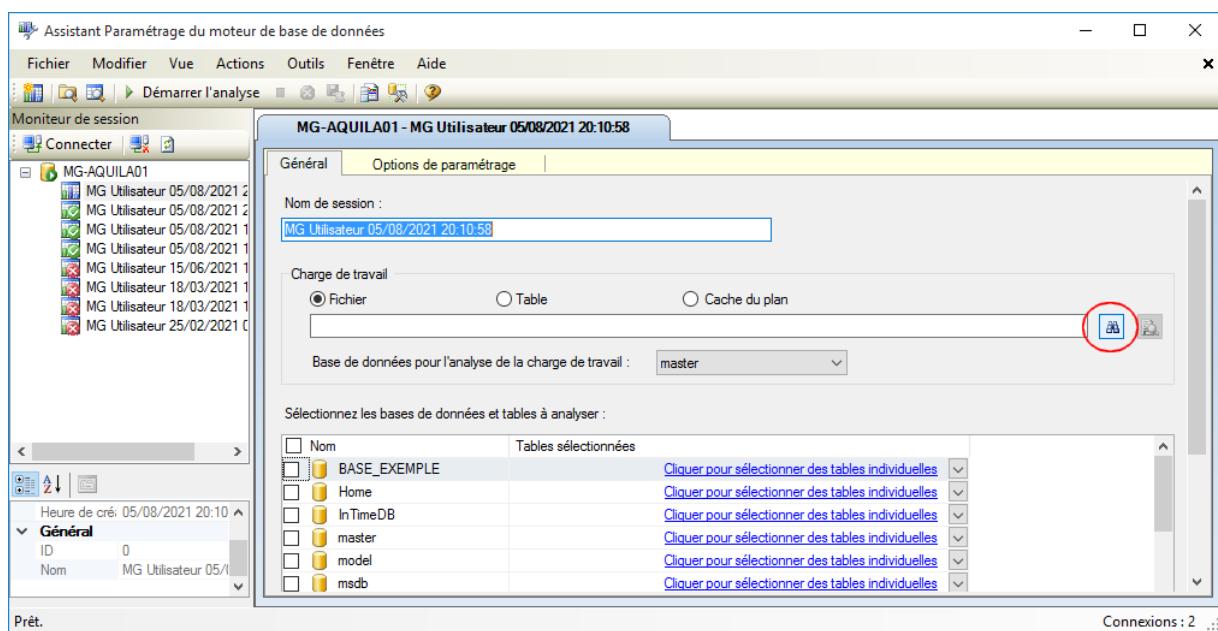
Nommez-le **optimisation.trc** et enregistrer sur votre PC.

Une fois cette chose faite, ouvrez à présent **Assistant Paramétrage du moteur de base de données**, (ou DTA en anglais) comme ci-dessous :

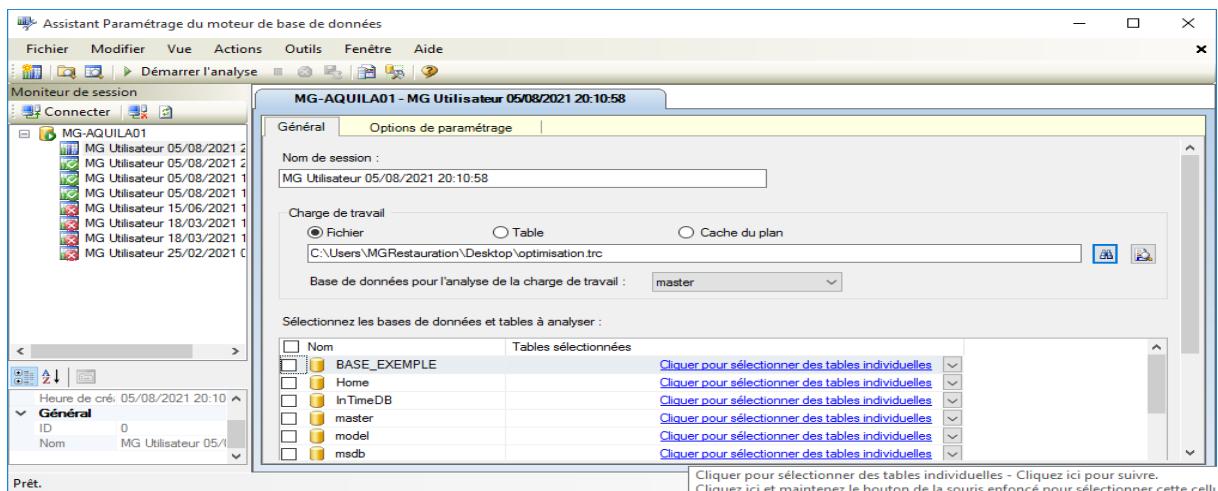


Maintenant, procédez aux réglages suivants :

Cliquez sur ce bouton pour charger votre fichier de charge de travail :

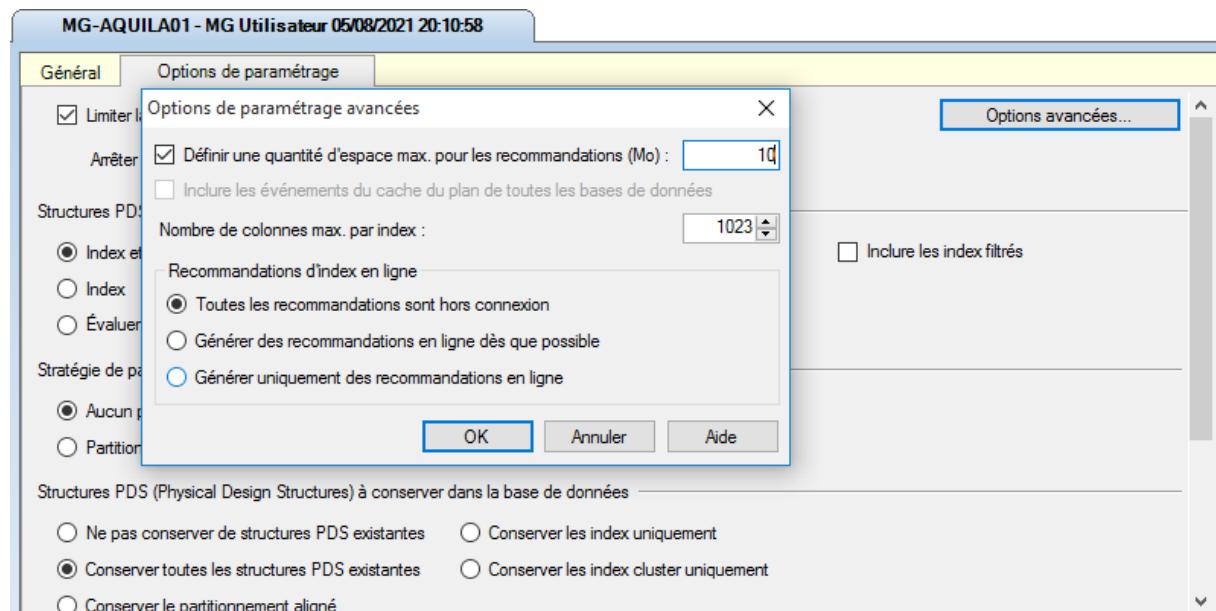


On charge donc le fichier dans l'espace réservé comme ceci :

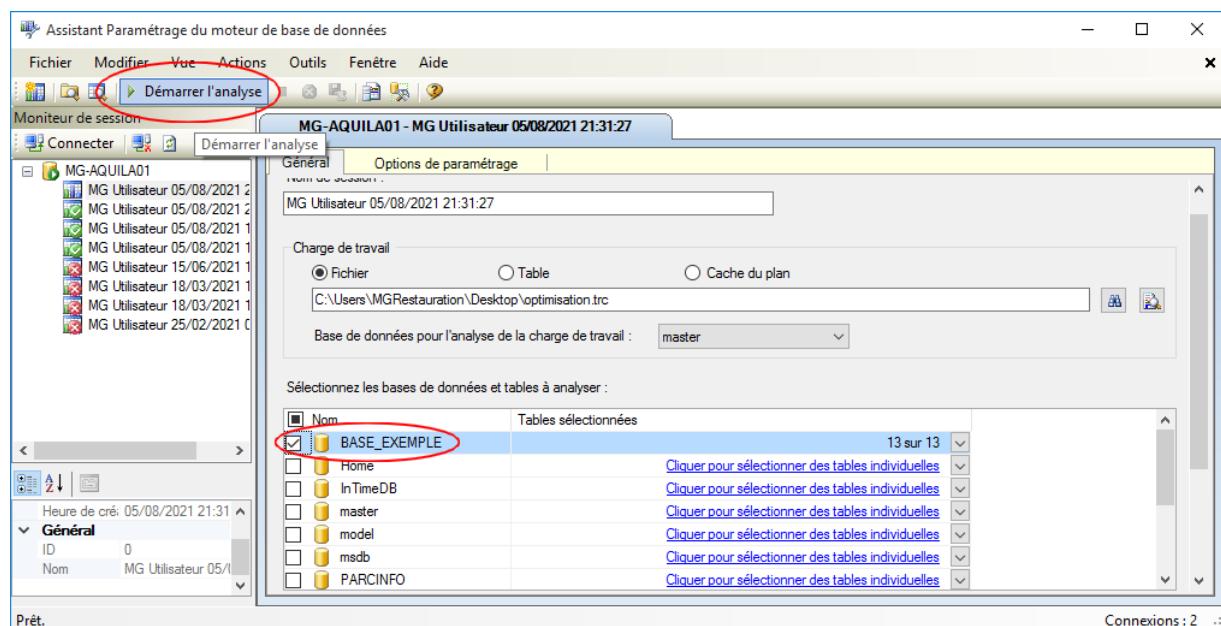


Cliquez ensuite sur l'onglet Option de paramétrage, puis cliquez sur **Option avancée...** définissez une

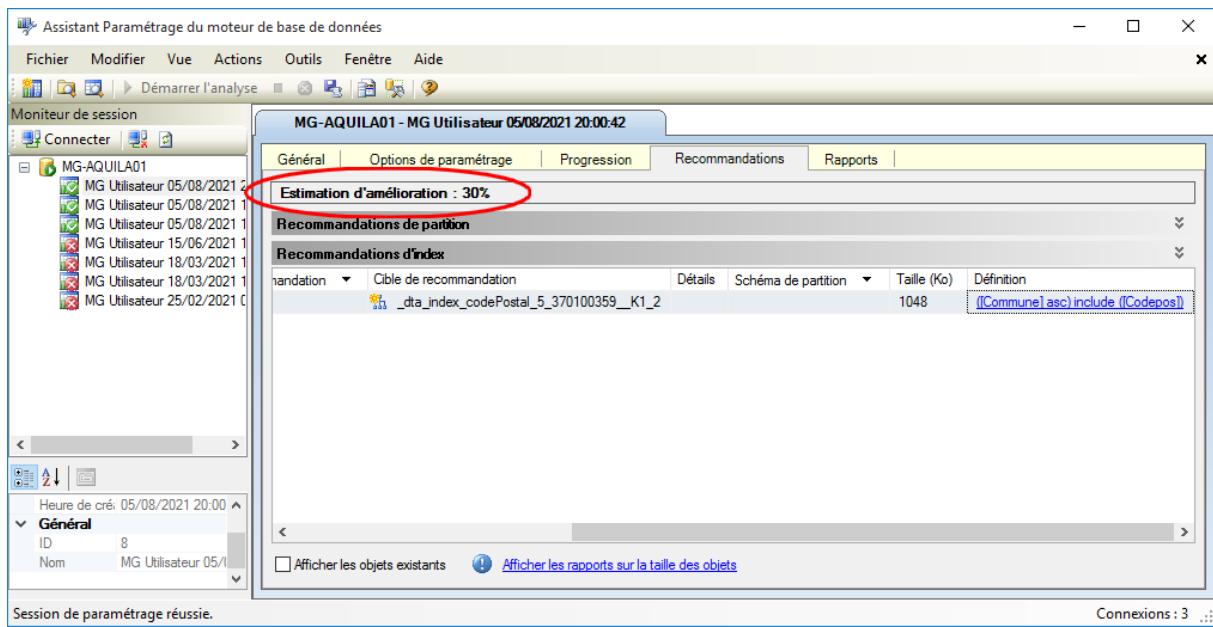
quantité d'espace max de 10 Mo (minimum).



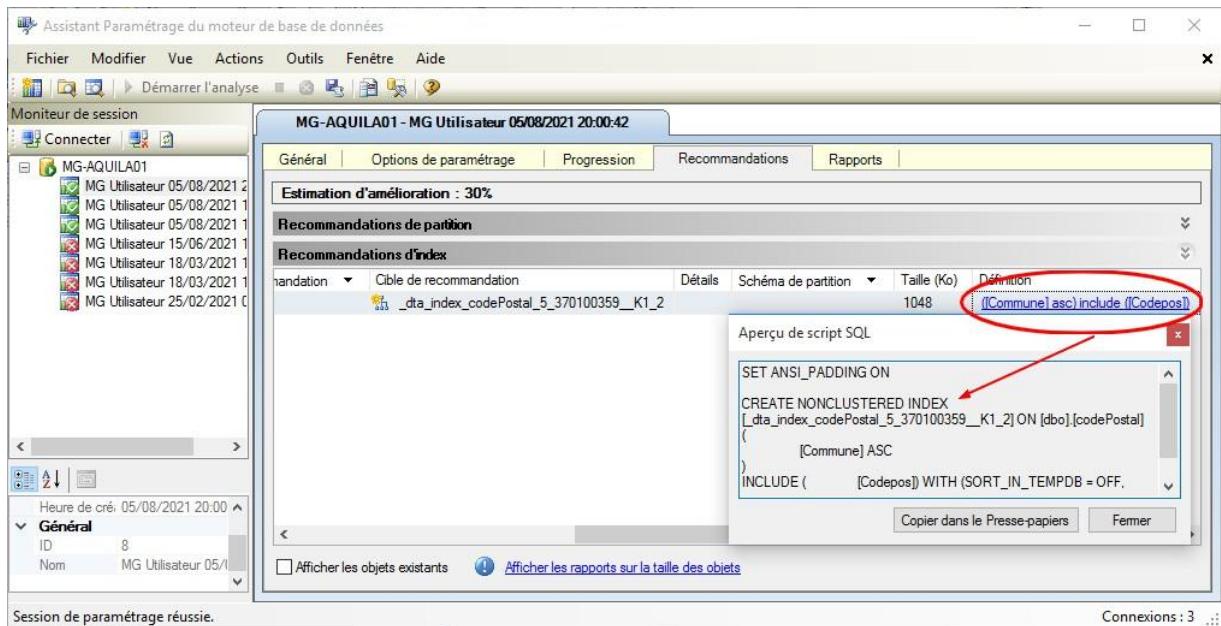
Une fois tous ces paramètres définis, choisissez la base de données voulue (BASE_EXEMPLE dans mon cas) et démarrez l'analyse :



Si tout a bien fonctionné, vous obtenez une recommandation pour l'amélioration de l'indexation de vos colonnes sujettes à votre requête.



Cliquez maintenant sur le lien sous Définition, le DTA vous suggérera un code T-SQL pour créer un index 30% plus optimisé :



Voici le code T-SQL pour créer un index 30% plus optimisé que l'actuel :

```

CREATE NONCLUSTERED INDEX [_dta_index_codePostal_5_370100359_K1_2] ON
[dbo].[codePostal]
(
    [Commune] ASC
)
INCLUDE ([Codepos]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF)
ON [PRIMARY]

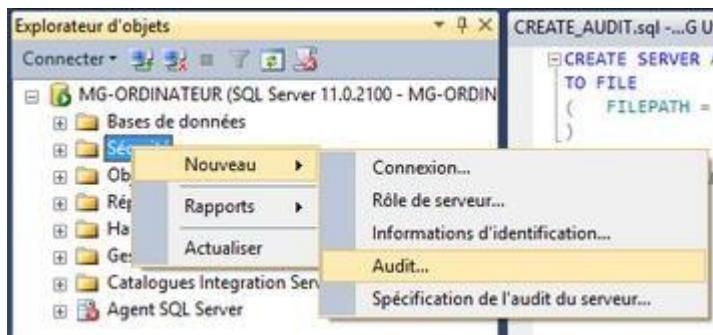
```

Le Database Engine Tuning Advisor (DTA) s'avère être un outil essentiel et des plus pertinents qui vous assistera efficacement à la création d'index optimisé.

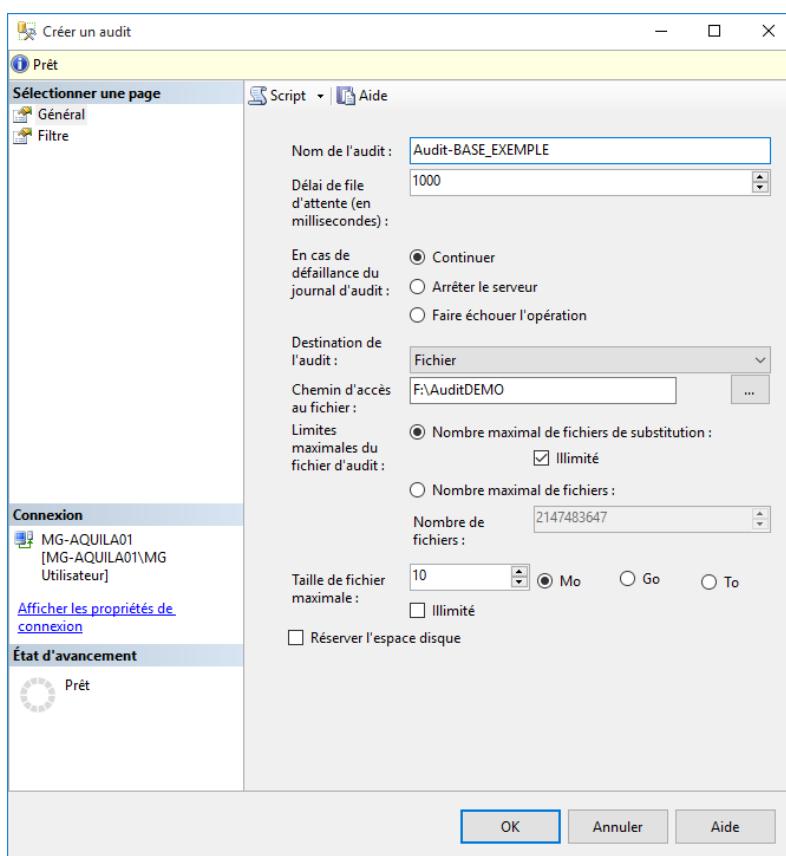
2- Mise en œuvre de l'audit SQL Server

Travaux pratiques (58) :

Dans votre instance aller sur : Sécurité > Nouveau > Audit



Ensuite vous réglez à votre guise les options de votre audit. Personnellement, j'ai choisi de l'enregistrer dans un journal de sécurité.



Voici le code T-SQL qui est généré par rapport à cette création d'audit :

```
USE [master]
GO
***** Object: Audit [Audit-BASE_EXEMPLE]      Script Date: 23/02/2020 22:13:13
*****/
CREATE SERVER AUDIT [Audit-BASE_EXEMPLE] TO FILE
```

```

(      FILEPATH = N'F:\AuditDEMO\'  

,MAXSIZE = 10 MB  

,MAX_ROLLOVER_FILES = 2147483647  

,RESERVE_DISK_SPACE = OFF  

) WITH  

(      QUEUE_DELAY = 1000  

,ON_FAILURE = CONTINUE  

,AUDIT_GUID = 'cd39b6fc-4373-423a-985a-a3a6b42c9162'  

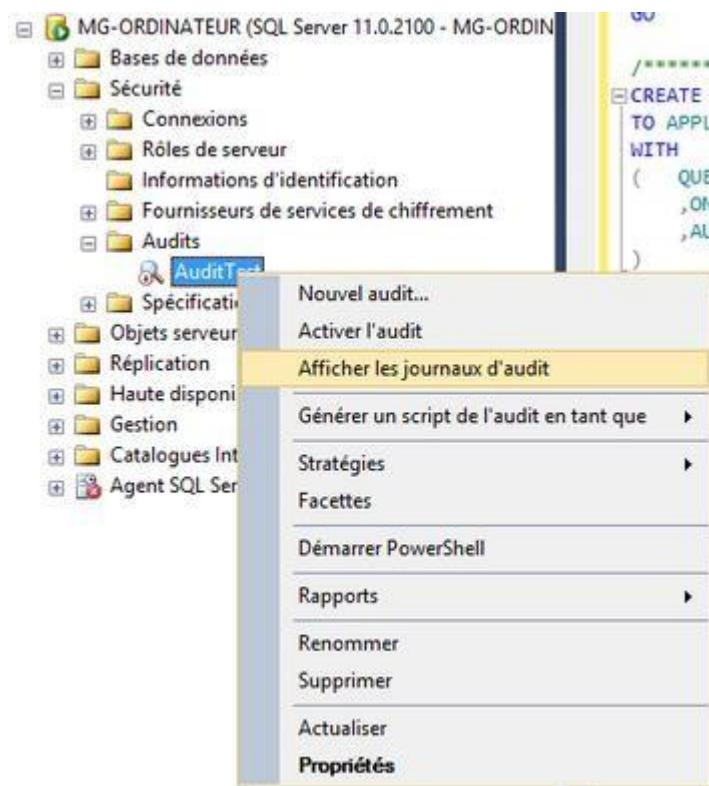
)  

ALTER SERVER AUDIT [Audit-BASE_EXEMPLE] WITH (STATE = OFF) GO

```

Pour afficher un journal d'audit dans Management Studio, rendez-vous sur : Sécurité > Audits > AuditTest.

Faites un clic droit sur AuditTest > Afficher les journaux d'audit.

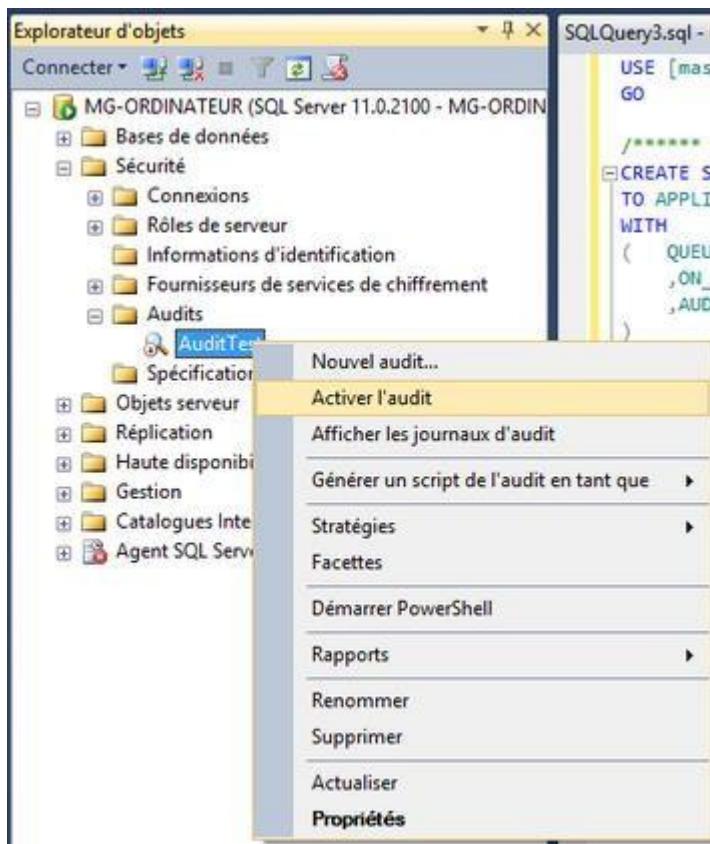


La visionneuse vous affichera les enregistrements de votre audit.

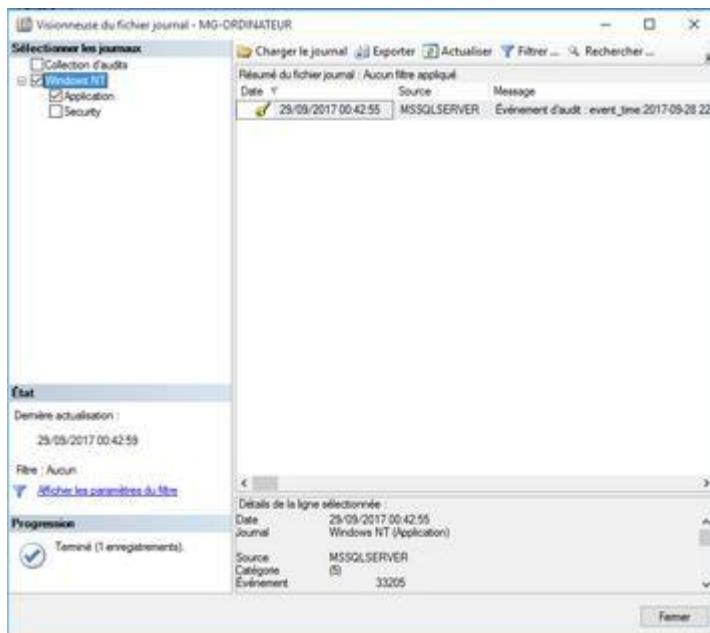
Gestion de l'audit SQL Server

Travaux pratiques (59) :

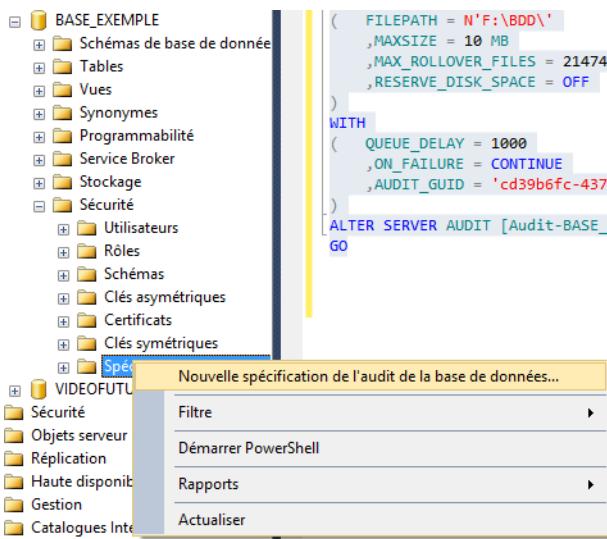
Dans votre instance aller sur : Sécurité > Audit > AuditTest > Activer l'audit



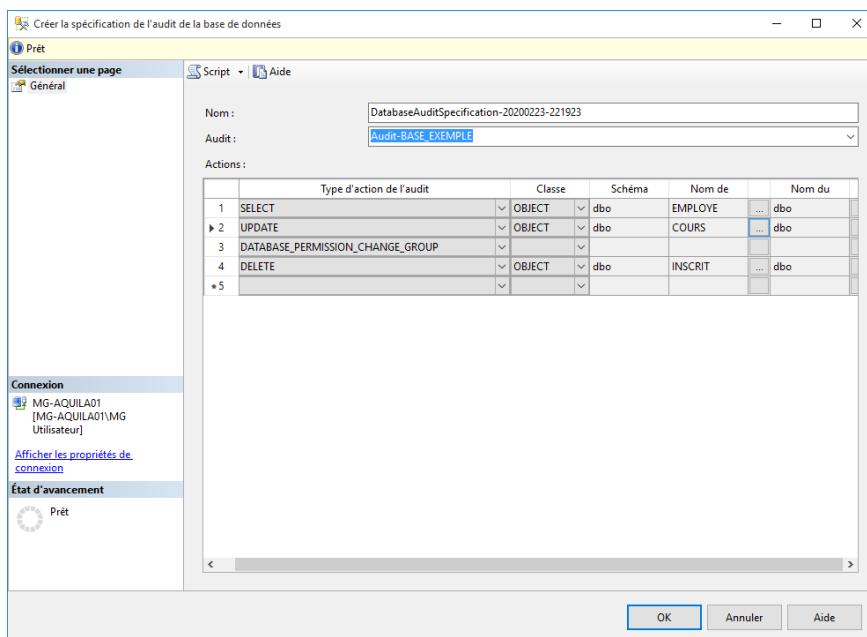
Ensuite, afficher les journaux d'audit et là vous obtenez un nouvel enregistrement.



A présent, allez dans Sécurité de la BASE_EXEMPLE, faites un clic-droit sur **Spécifications de l'audit de la base de données**, et cliquez sur **Nouvelle spécification** etc...

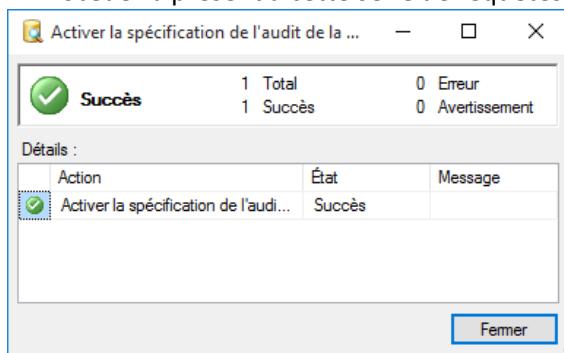


Procéder à ces réglages :



La BASE_EXEMPLE dispose désormais d'un audit prêt à l'emploi qui surveillera les objets mentionnés. Là-encore activer l'audit de la base de données.

Procédez à présent à cette série de requêtes :



Procédez à présent à cette série de requêtes :

```
USE BASE_EXEMPLE; GO
SELECT * FROM EMPLOYE;

CREATE PROCEDURE spGetCommerciaux AS
BEGIN
    SELECT NOM, GENRE FROM COMMERCIAUX
END GO

SELECT NOM, GENRE FROM COMMERCIAUX

EXEC spGetCommerciaux;

DROP TABLE codePostal;
-- cette table doit être créée précédemment

INSERT INTO COMMERCIAUX VALUES (11, 'Bernard', 'Homme', 8000, 'France'); DELETE FROM COMMERCIAUX WHERE ID=11;
```

Explorons à présent notre audit :

```
SELECT * FROM sys.fn_get_audit_file (
    'F:\BDD\*.sqlaudit', default, default
) GO
```

Résultats

	Résultats	Messages
1	database_principal_name server_instance_name database_name schema_name object_name statement additional_information file_name	<action_info xmlns="http://schemas.microsoft.com/2004/06/sqlalchemy/audit">SELECT * FROM EMPLOYE.
2	MG-AQUILA01 MG-AQUILA01 BASE_EXEMPLE dbo EMPLOYE	F:\BDD\Audit-BASE F:\BDD\Audit-BASE

Le premier SELECT a bien été « capté » par notre audit. Un fichier a d'ailleurs été créé dans le répertoire cible.

SURVEILLANCE ET BONNES PRATIQUES

1- Vue d'ensemble

Bonnes pratiques d'optimisation

Voici quelques bonnes pratiques d'optimisation pour les bases de données SQL, adaptées à votre base de données Catalogue_VOD :

1. Conception de la Base de Données :

- Concevez votre base de données en utilisant des tables normalisées pour éviter la redondance des données.
- Utilisez des types de données appropriés pour chaque colonne.
- Établissez des contraintes d'intégrité pour garantir la cohérence des données.

2. Indexation :

- Créez des index appropriés sur les colonnes fréquemment utilisées dans les clauses WHERE et JOIN.
- Évitez de sur-indexer, car cela peut ralentir les opérations de mise à jour.
- Réorganisez et reconstruisez régulièrement les index pour maintenir de bonnes performances.

3. Optimisation des Requêtes :

- Écrivez des requêtes efficaces en évitant les fonctions de texte intégral, les opérations coûteuses et les jointures excessives.
- Utilisez des opérateurs logiques tels que JOIN, WHERE et GROUP BY de manière judicieuse.
- Utilisez les indices pour accélérer la recherche de données.

4. Analyse des Plans d'Exécution :

- Utilisez les plans d'exécution pour analyser les performances de vos requêtes.
- Identifiez les opérateurs coûteux et les problèmes de fragmentation des index.

5. Optimisation des Index :

- Utilisez l'Assistant de Réglage des Performances ou l'Index Tuning Wizard pour obtenir des recommandations sur les index à créer ou à modifier.

6. Optimisation des Index Composites :

- Créez des index composites sur plusieurs colonnes fréquemment utilisées ensemble.
- Cela améliorera les performances des requêtes qui filtrent ou trient sur ces colonnes.

7. Optimisation des Procédures Stockées :

- Évitez les procédures stockées complexes et longues.
- Utilisez l'optimisation de procédure stockée pour améliorer les performances de vos procédures.

8. Maintenance de la Base de Données :

- Planifiez des tâches de maintenance régulières telles que la réorganisation des index, la mise à jour des statistiques et la sauvegarde de la base de données.

9. Surveillance des Performances :

- Utilisez des outils de surveillance des performances pour suivre l'utilisation

des ressources, les temps de réponse et les problèmes de blocage.

10. Test des Modifications :

- Avant d'appliquer des modifications majeures à la base de données, testez-les dans un environnement de développement ou de test pour évaluer leur impact sur les performances.

En suivant ces bonnes pratiques d'optimisation, vous pouvez améliorer les performances de votre base de données Catalogue_VOD et assurer son efficacité tout en minimisant les temps d'arrêt et les problèmes de performances.

Principe des évènements et des compteurs

Le moniteur système, ou Analyseur de performance dont on n'a pas parlé dans cette formation, n'est

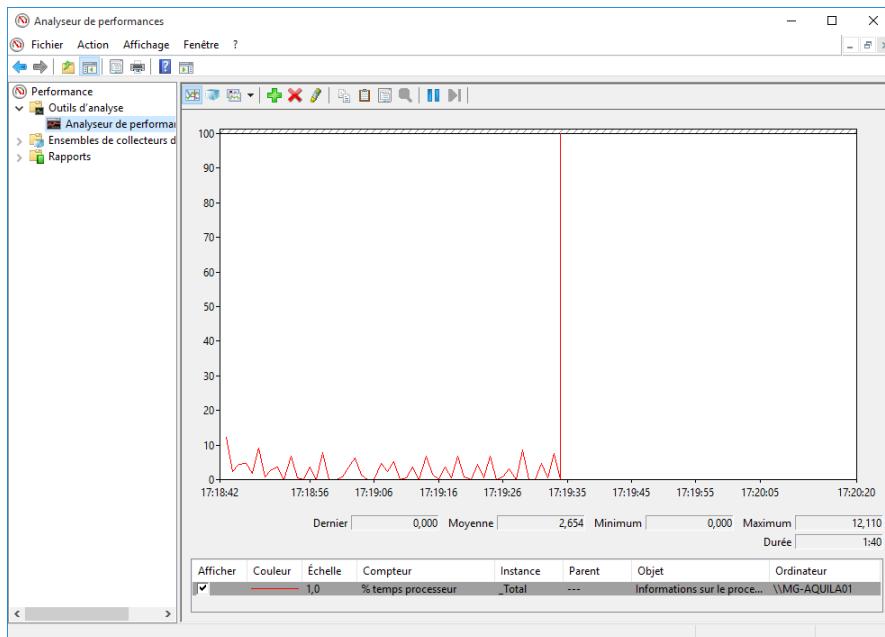
pas propre à SQL Server.

C'est un utilitaire de supervision intégré à Windows, qui agit comme le moniteur cardiaque du système. Il vous permet de suivre en temps réel tous les compteurs (des données chiffrées) indiquant le comportement de votre serveur.

Un certain nombre d'applications, dont SQL Server, ajoutent leur propre jeu de compteurs à ceux proposés par Windows. Vous pouvez ainsi, dans la même interface, suivre les signes vitaux de Windows et de SQL Server.

Atelier 47

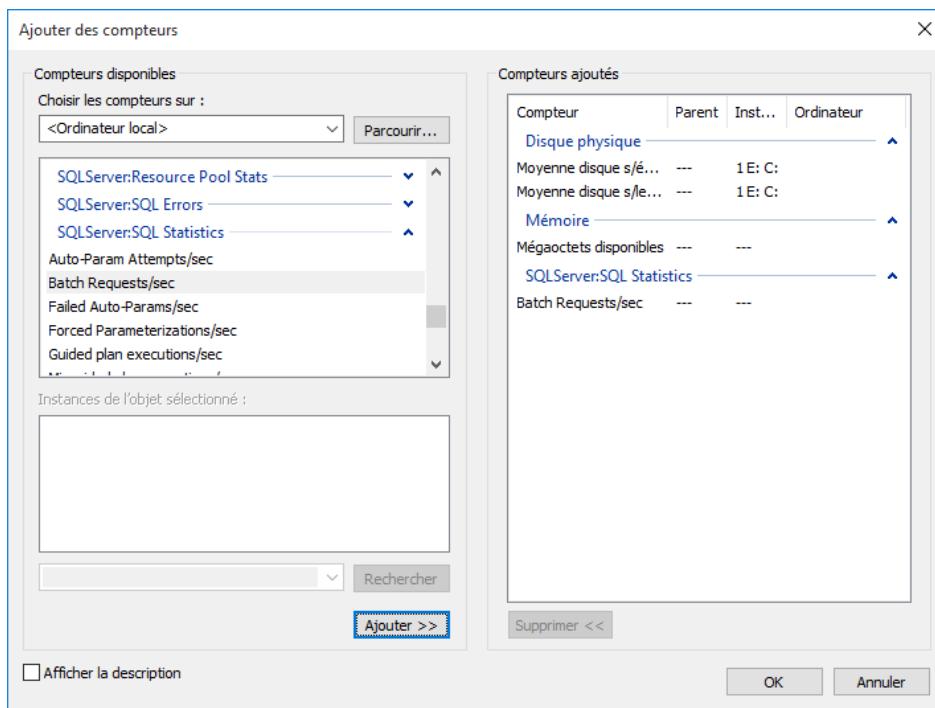
Dans l'explorateur Windows, exécutez la commande **perfmon.exe**, et le moniteur s'ouvre comme dans la capture ci-dessous.



Vous pouvez soit afficher l'évolution des compteurs en temps réel, soit enregistrer les mesures dans un fichier journal.

Cette dernière solution est idéale pour construire une baseline. Pour ajouter des compteurs, cliquez sur le bouton affichant le signe + (couleur verte) dans la barre d'outils. Par défaut, le moniteur système affiche le % du temps processeur.

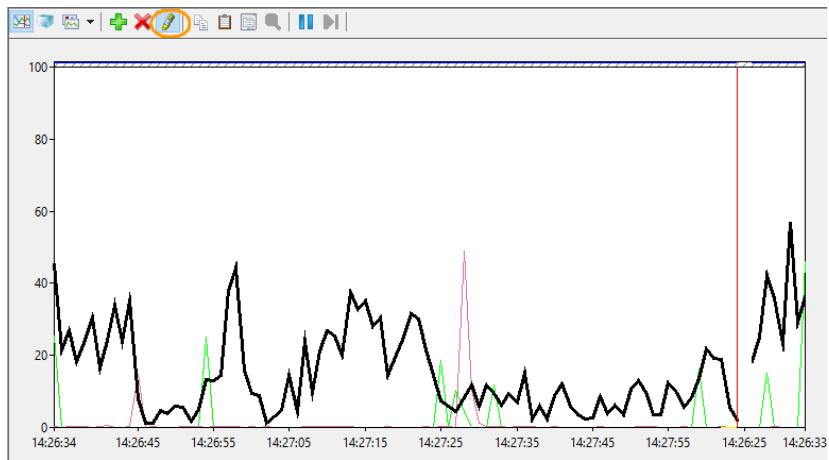
Vous obtenez ainsi la fenêtre de sélection de compteurs. Nous allons nous positionner sur notre machine et ajouter des compteurs, notamment concernant les statistiques SQL, les batches de requêtes par secondes, mais aussi l'état de la mémoire et la moyenne des lectures écritures sur le disque comme ci-dessous :



A ce sujet, il est de bon ton de dédier une machine (avec un serveur) pour faire du monitoring si votre BDD est installé sur plusieurs serveurs. Ainsi en isolant une machine de test, vous avez une meilleure vue sur l'activité de vos différents serveurs.

Dans la liste des compteurs, nous pouvons bien sûr voir tous les composants actifs de Windows.

Nous pouvons mettre des compteurs en surillance en cliquant sur le bouton entouré en orange ci-dessous.



Cette mise en gras de notre compteur nous donne une meilleure idée sur l'activité du temps processeur par exemple.

Il faut surveiller l'utilisation de la mémoire en s'aidant des conseils ci-dessous :

Utilisation de la mémoire et du fichier d'échange	Objet : Compteur	Instruction
	Mémoire : Octets disponibles	Ce compteur doit être supérieur à 5 000 Ko
	Mémoire : Pages/s	Ce compteur ne doit jamais être constamment supérieur à zéro
Utilisation de la mémoire et du tampon	Objet : Compteur	Instruction
	Processus : Défauts de page/s/Instance sqlservr	Un chiffre élevé indique une pagination excessive
	Processus : Jeu de travail/ Instance sqlservr	Ce compteur doit être supérieur à 5 000 Ko
	SQL Server: Buffer Manager: Buffer Cache Hit Ratio	Ce compteur doit être supérieur à 90 %
	SQL Server: Buffer Manager: Total Pages	Un chiffre peu élevé peut indiquer des E/S du disque fréquentes et il peut être nécessaire d'ajouter de la mémoire
	SQL Server: Memory Manager: Total Server Memory	Si la valeur de ce compteur est proche de la quantité de mémoire physique, il peut être nécessaire d'ajouter de la mémoire

Il faut surveiller l'utilisation des threads et du processeur en s'aidant des conseils ci-dessous :

Objet : Compteur	Instruction
Processeur : %Temps processeur	Ce compteur doit généralement être inférieur à 90 %
Système : Changements de contexte/s	Sur un ordinateur multiprocesseur, si ce compteur atteint 8 000 et que le compteur Processeur : % Temps processeur dépasse 90 %, prévoyez l'activation de la planification en mode fibre de SQL Server
Système : Longueur de la file du processeur	Ce compteur ne doit jamais être constamment supérieur à 2
Processeur : %Temps Privilégié	La valeur de ce compteur doit être aussi faible que possible
Processeur : %Temps Utilisateur	Ce compteur donne le temps d'exécution des processus utilisateur, tel que SQL Server, exprimé en pourcentage

Il faut pratiquer une surveillance des E/S du disque dur en s'aidant des conseils ci-dessous :

Objet : Compteur	Instruction
Disque physique : % Temps du disque	Ce compteur doit toujours être inférieur à 90 %
Disque physique : Long. moy. de file d'attente du disque	Ce compteur ne doit jamais être égal à plus de 2 fois le nombre de piles
Disque physique : Lectures disque/s	La valeur de ce compteur doit toujours être inférieure à la capacité du sous-système de votre disque dur
Disque physique : Écritures disque/s	La valeur de ce compteur doit toujours être inférieure à la capacité du sous-système de votre disque dur

Il faut pratiquer une surveillance des requêtes qui ne s'effectuent pas correctement :

- Identification des performances des requêtes (Gestionnaire de profils SQL Server, traces et procédures stockées).
- Causes de la médiocrité des performances (Réseau, mémoire, statistiques, index et structure des requêtes).

Conseils pratiques :

- Définissez une référence en matière de performances
- Surveiller les performances réelles
- Identifier les goulets d'étranglements liés aux performances
- Utilisez les outils appropriés.

3- Haute Disponibilité et Récupération après Sinistre : Concepts de réPLICATION, clustering, et stratégies de backup.

1. Introduction

Dans un environnement professionnel, une base de données doit être disponible en permanence et tolérante aux pannes. Une interruption de service peut entraîner des pertes financières, une baisse de productivité ou une atteinte à l'image de l'entreprise.

SQL Server propose plusieurs **mécanismes de haute disponibilité (HA)** et de **récupération après sinistre (DR)** afin de garantir la continuité de service et la protection des données.

2. Concepts clés

- **Haute disponibilité (HA) :**
Objectif = assurer un accès constant aux données, même en cas de panne matérielle, logicielle ou réseau.
Exemple : bascule automatique (failover) d'un serveur défaillant vers un serveur de secours.
- **Récupération après sinistre (DR) :**
Objectif = restaurer les données et les services après un incident majeur (ex. : incendie, perte totale du datacenter).
Exemple : restauration à partir de sauvegardes ou bascule vers un site distant.

3. Mécanismes de haute disponibilité dans SQL Server

3.1. Clustering (Failover Cluster Instance – FCI)

- **Principe** : plusieurs serveurs (nœuds) partagent un stockage commun.
- En cas de panne d'un nœud, un autre prend automatiquement le relais.
- **Avantages** : continuité du service, transparent pour les applications.
- **Limites** : nécessite un stockage partagé (SAN) et une configuration complexe.

3.2. Groupes de disponibilité Always On (Always On Availability Groups)

- **Principe** : répliquer une base de données primaire vers une ou plusieurs bases secondaires.
- **Caractéristiques** :
 - Réplication synchrone ou asynchrone.
 - Possibilité de bascule automatique vers une base secondaire.
 - Les bases secondaires peuvent être utilisées en lecture seule (reporting, sauvegardes).
- **Avantages** : haute disponibilité et récupération après sinistre combinées.
- **Limites** : nécessite SQL Server Enterprise (sauf certaines éditions récentes qui offrent une version limitée).

3.3. Réplication

- **Principe** : distribuer les données d'une base source (publisher) vers une ou plusieurs cibles (subscribers).

- **Types de réPLICATION :**
 - **Snapshot** : copie ponctuelle.
 - **Transactionnelle** : réplique en temps quasi réel.
 - **Fusion** : permet aux abonnés de modifier aussi les données (synchronisation bidirectionnelle).
- **Avantages** : utile pour la distribution de données (reporting, mobilité, etc.).
- **Limites** : pas conçu comme une vraie solution de haute disponibilité (plus orienté distribution).

4. Stratégies de sauvegarde (Backup & Restore)

La sauvegarde reste la **base de toute stratégie de DR**. SQL Server propose différents types de sauvegardes :

- **Sauvegarde complète** : copie intégrale de la base.
- **Sauvegarde différentielle** : copie uniquement des changements depuis la dernière sauvegarde complète.
- **Sauvegarde du journal des transactions (log backup)** : permet la restauration point-in-time (jusqu'à un instant précis).
- **Sauvegarde en continu avec log shipping** : envoi automatique des journaux vers un serveur secondaire pour une reprise rapide.

Bonnes pratiques :

- Planifier des sauvegardes régulières (selon la criticité des données).
- Tester périodiquement la restauration (une sauvegarde non testée n'est pas une vraie sauvegarde).
- Externaliser les sauvegardes (site distant, cloud).
- Définir un **RPO (Recovery Point Objective)** et un **RTO (Recovery Time Objective)** adaptés aux besoins métiers.

5. Stratégies combinées

Dans la réalité, on combine souvent plusieurs solutions :

- **Clustering ou Always On** pour la haute disponibilité locale.
- **Sauvegardes + Log shipping** pour la reprise après sinistre sur un site distant.
- **RéPLICATION** pour distribuer les données à des serveurs de reporting.

6. Conclusion

La haute disponibilité et la récupération après sinistre sont essentielles pour toute base de données critique.

SQL Server offre une large gamme de solutions allant du **clustering** à **Always On**, en passant par la **réPLICATION** et des stratégies avancées de **sauvegarde/restauration**.

Le choix dépend du budget, de la complexité souhaitée et surtout des objectifs métiers en termes de RTO/RPO.

1. RÉPLICATION SIMPLE - AVEC BACKUPS LOGS EN TEMPS RÉEL

-- STRATÉGIE SIMPLIFIÉE SANS RÉPLICATION COMPLEXE

-- 1. CONFIGURATION DE BASE POUR HAUTE DISPOSNIBILITÉ

```
USE master;
```

```
GO
```

-- Vérifier l'état de la base

```
SELECT
    name AS database_name,
    recovery_model_desc,
    state_desc,
    user_access_desc
FROM sys.databases
WHERE name = 'TechShop';
GO
```

-- 2. PASSER EN MODÈLE DE RÉCUPÉRATION COMPLET

```
ALTER DATABASE TechShop SET RECOVERY FULL;
GO
```

-- 3. CRÉER UNE BASE MIRROIR SIMPLIFIÉE (pour la démonstration)

```
CREATE DATABASE TechShop_Mirror;
GO
```

-- Copier la structure (sans les données pour la démo)

```
USE TechShop_Mirror;
GO
```

-- Créer les mêmes tables vides

```
CREATE TABLE Clients (
    id INT PRIMARY KEY IDENTITY(1,1),
    nom NVARCHAR(100) NOT NULL,
    email NVARCHAR(255) UNIQUE NOT NULL,
    solde DECIMAL(10,2) DEFAULT 1000.00
);
```

```
CREATE TABLE Produits (
    id INT PRIMARY KEY IDENTITY(1,1),
    nom NVARCHAR(100) NOT NULL,
    prix DECIMAL(10,2) NOT NULL,
    stock INT NOT NULL
);
GO
```

2. STRATÉGIE DE BACKUP AUTOMATISÉE SIMPLE

--  BACKUP SIMPLE ET EFFICACE

-- 1. PROCÉDURE DE BACKUP COMPLET QUOTIDIEN

```
CREATE OR ALTER PROCEDURE sp_BackupSimpleComplet
AS
```

```

BEGIN
    DECLARE @chemin NVARCHAR(500) = 'C:\Backups\' ;
    DECLARE @fichier NVARCHAR(500) ;
    DECLARE @date NVARCHAR(20) = FORMAT(GETDATE(), 'yyyyMMdd_HHmmss') ;

    SET @fichier = @chemin + 'TechShop_Full_' + @date + '.bak' ;

    -- Créer le dossier si nécessaire
    EXEC xp_create_subdir @chemin;

    BACKUP DATABASE TechShop
    TO DISK = @fichier
    WITH
        COMPRESSION,
        CHECKSUM,
        STATS = 5;

    PRINT '☒ Backup complet créé: ' + @fichier;
END;
GO

-- 2. BACKUP DIFFÉRENTIEL TOUTES LES 4 HEURES
CREATE OR ALTER PROCEDURE sp_BackupDifferentielSimple
AS
BEGIN
    DECLARE @chemin NVARCHAR(500) = 'C:\Backups\' ;
    DECLARE @fichier NVARCHAR(500) ;
    DECLARE @date NVARCHAR(20) = FORMAT(GETDATE(), 'yyyyMMdd_HHmmss') ;

    SET @fichier = @chemin + 'TechShop_Diff_' + @date + '.bak' ;

    BACKUP DATABASE TechShop
    TO DISK = @fichier
    WITH
        DIFFERENTIAL,
        COMPRESSION,
        STATS = 5;

    PRINT '☒ Backup différentiel créé: ' + @fichier;
END;
GO

-- 3. BACKUP DES LOGS TOUTES LES 30 MINUTES
CREATE OR ALTER PROCEDURE sp_BackupLogSimple
AS
BEGIN
    DECLARE @chemin NVARCHAR(500) = 'C:\Backups\' ;
    DECLARE @fichier NVARCHAR(500) ;
    DECLARE @date NVARCHAR(20) = FORMAT(GETDATE(), 'yyyyMMdd_HHmmss') ;

    SET @fichier = @chemin + 'TechShop_Log_' + @date + '.trn' ;

    BACKUP LOG TechShop
    TO DISK = @fichier
    WITH
        COMPRESSION,
        STATS = 5;

    PRINT '☒ Backup log créé: ' + @fichier;
END;
GO

```

3. RÉPLICATION MANUELLE SIMPLE POUR ÉTUDIANTS

```

-- RÉPLICATION "MAISON" - COMPRÉHENSIBLE POUR ÉTUDIANTS

-- 1. PROCÉDURE DE SYNCHRONISATION MANUELLE
CREATE OR ALTER PROCEDURE sp_SynchroniserClients
AS
BEGIN
    PRINT '☒ Synchronisation des clients...';

    -- Synchroniser les nouveaux clients
    INSERT INTO TechShop_Mirror.dbo.Clients (nom, email, solde)
    SELECT nom, email, solde
    FROM TechShop.dbo.Clients c
    WHERE NOT EXISTS (
        SELECT 1 FROM TechShop_Mirror.dbo.Clients m
        WHERE m.id = c.id
    );
    PRINT '☒ ' + CAST(@@ROWCOUNT AS NVARCHAR) + ' nouveaux clients synchronisés';

    -- Synchroniser les modifications
    UPDATE m
    SET m.nom = c.nom,
        m.email = c.email,
        m.solde = c.solde
    FROM TechShop_Mirror.dbo.Clients m
    INNER JOIN TechShop.dbo.Clients c ON m.id = c.id
    WHERE m.nom <> c.nom
        OR m.email <> c.email
        OR m.solde <> c.solde;
    PRINT '☒ ' + CAST(@@ROWCOUNT AS NVARCHAR) + ' clients mis à jour';
END;
GO

-- 2. SYNCHRONISATION DES PRODUITS
CREATE OR ALTER PROCEDURE sp_SynchroniserProduits
AS
BEGIN
    PRINT '☒ Synchronisation des produits...';

    -- Synchroniser les nouveaux produits
    INSERT INTO TechShop_Mirror.dbo.Produits (nom, prix, stock)
    SELECT nom, prix, stock
    FROM TechShop.dbo.Produits p
    WHERE NOT EXISTS (
        SELECT 1 FROM TechShop_Mirror.dbo.Produits m
        WHERE m.id = p.id
    );
    PRINT '☒ ' + CAST(@@ROWCOUNT AS NVARCHAR) + ' nouveaux produits synchronisés';

    -- Synchroniser les modifications de stock/prix
    UPDATE m
    SET m.nom = p.nom,
        m.prix = p.prix,
        m.stock = p.stock
    FROM TechShop_Mirror.dbo.Produits m
    INNER JOIN TechShop.dbo.Produits p ON m.id = p.id
    WHERE m.nom <> p.nom
        OR m.prix <> p.prix
        OR m.stock <> p.stock;
    PRINT '☒ ' + CAST(@@ROWCOUNT AS NVARCHAR) + ' produits mis à jour';

```

```

END;
GO

-- 3. PROCÉDURE DE SYNCHRONISATION COMPLÈTE
CREATE OR ALTER PROCEDURE sp_SynchroniserComplete
AS
BEGIN
    PRINT '⌚ DÉBUT SYNCHRONISATION COMPLÈTE';
    PRINT '=====';
    EXEC sp_SynchroniserClients;
    EXEC sp_SynchroniserProduits;

    PRINT '☑ SYNCHRONISATION TERMINÉE';
END;
GO

-- 2. Lancer le test complet
EXEC sp_TestCompleteHADR;

```

4. 🚨 RÉCUPÉRATION APRÈS SINISTRE SIMPLIFIÉE : Restauration d'une base à un point dans le temps

Travaux pratiques (22) :

Pour restaurer votre BDD depuis une limite temporelle, nous pouvons soit nous servir au choix soit de notre journal de transactions ou soit comme d'utiliser l'option STOPAT en déterminant une date de notre choix concernant la sauvegarder à restaurer.

Nous allons procéder par étapes. Nous allons créer une BDD toute fraîche que l'on va nommer **InTimeDB**.

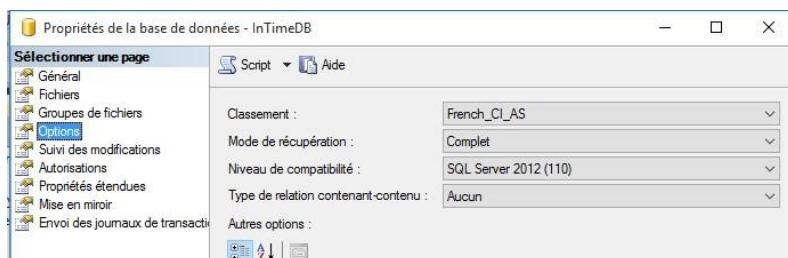
```

CREATE DATABASE InTimeDB;

/* Création de la BDD InTimeDB. */

```

- ➡ Actualiser votre liste de BDD, puis faites un clic-droit sur notre BDD
- InTimeDB > Propriétés.** ➡ Aller dans **Options > Mode de récupération :**
- Complet**



Pour restaurer votre BDD de manière complète et pas simple ? Pour la simple et bonne raison qu'en choisissant un « RECOVERY MODEL » « FULL », le mode de récupération exigera des sauvegardes du fichier journal, ce qui n'est pas le cas d'une restauration simple. En effet, c'est grâce au fichier log que vous pourrez pratiquer une récupération de votre BDD jusqu'à un point spécifique dans le temps, en supposant bien sûr que nos sauvegardes ont été effectuées jusqu'à ce point.

► Nous allons créer maintenant la table **SPORTIFS**.

```
USE InTimeDB

CREATE TABLE SPORTIFS (
    ID INT IDENTITY
    (1,1) PRIMARY KEY,
    NOM VARCHAR(50),
    SPORT VARCHAR(50)
);

/* Script création Table SPORTIFS */
```

► Insérons maintenant des données.

```
INSERT INTO SPORTIFS

VALUES
('Rougerie',
'RUGBY'),
('Jordan',
'BASKET'),
('Bolt', 'ATHLE');

/* Insertion data SPORTIFS à 1:04 */
```

► Faisons maintenant une sauvegarde complète de notre BDD.

```
BACKUP DATABASE InTimeDB

TO DISK = 'F:\SQL_BACKUP\InTimeDBFullBackup.bak';

/* Sauvegarde complète de InTimeDB à 1:08 */
```

► Allons maintenant jeter un œil sur les détails de notre backup dans msdb.
Faisons attention à l'horaire.

```
SELECT BS.database_name, BS.backup_finish_date,
BMF.logical_device_name,BMF.physical_device_name, *

FROM msdb..backupset BS

INNER JOIN
msdb..backupmediafamily
BMF ON BS.media_set_id =
BMF.media_set_id WHERE
BS.database_name =
'InTimeDB';

/* détail de la sauvegarde */
```

Résultats

	Résultats	Messages		
1	database_name	backup_finish_date	logical_device_name	physical_device_name
	InTimeDB	2018-02-09 01:08:18.000	NULL	F:\SQL_BACKUP\InTimeDBFullBackup.bak

► Nous allons maintenant insérer 2 nouvelles lignes, 2 tennismen.

```
INSERT INTO SPORTIFS
VALUES
('Federer',
'TENNIS'),
('Nadal',
'TENNIS');
/* Insertion des tennismen à 1:19 */
```

► Nous allons lire la table.

```
SELECT *
FROM SPORTIFS
```

Résultats

	Résultats	Messages	
	ID	NOM	SPORT
1	1	Rougerie	RUGBY
2	2	Jordan	BASKET
3	3	Bolt	ATHLE
4	4	Federer	TENNIS
5	5	Nadal	TENNIS

► Nous allons supprimer toutes les données de la table.

```
DELETE FROM SPORTIFS;
/* Suppression des données de la table SPORTIFS à 1:30 */
```

► Mettons à jour la table comme ceci.

```
UPDATE SPORTIFS
SET Nom = 'Mandanda' WHERE ID = 1;
/* MAJ de la table à 1:34 */
```

Rien ne s'est produit, car en effet un collègue de travail bien distrait a supprimé les données de la table. Nous allons donc la restaurer avec un point temporel. Nous allons choisir de la remettre en état juste après l'ajout des 2 tennismen et avant la suppression soit à 1:29 :59.

► La première chose à faire est de sauvegarder le journal des transactions.

```
BACKUP LOG InTimeDB
TO DISK ='F:\SQL_BACKUP\InTimeDBFullBackup.log';
/* MAJ de la table à 1:34 */
```

- ▶ Etant donné que nous sommes des administrateurs prudents, nous allons passer en mode mono-utilisateur. C'est-à-dire que toutes les connexions à la BDD sont fermées sans avertissement. Seul un utilisateur peut se connecter à la BDD.

```
USE master GO
ALTER DATABASE InTimeDB SET SINGLE_USER
WITH ROLLBACK IMMEDIATE;
/* Déclenchement du single user */
```

- ▶ A 1:08, j'ai effectué une sauvegarde complète. A ce moment-là, la table SPORTIFS contenait 3 lignes. Nous allons la restaurer, cette sauvegarde complète de 1:08.

```
RESTORE DATABASE InTimeDB
FROM DISK
='F:\SQL_BACKUP\InTimeDBFullBackup.bak' WITH FILE =1, NORECOVERY
GO
/* Restauration de la sauvegarde de 1:08 */
```

Résultats

```
Msg 3159, Niveau 16, État 1, Ligne 1
La fin du journal pour la base de données "InTimeDB" n'a pas été sauvegardée. Utilisez
BACKUP LOG WITH NORECOVERY pour sauvegarder le journal s'il contient des travaux que vous
ne voulez pas perdre. Utilisez la clause WITH REPLACE ou WITH STOPAT de l'instruction
RESTORE pour remplacer simplement le contenu du journal.
Msg 3013, Niveau 16, État 1 RESTORE DATABASE s'est terminé anormalement.
```

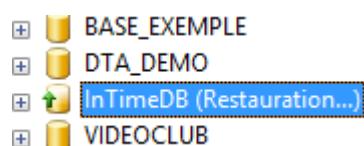
- ▶ Nous obtenons ce message d'erreur. SQL Server nous demande de faire une sauvegarde de notre fichier journal (une nouvelle fois) en mode NO RECOVERY. Nous allons nous exécuter. Pourquoi ? En mode NORECOVERY, SQL Server restaure la BDD et les transactions et séquences en cours, ce qui n'est pas le cas du mode RECOVERY qui annule les transactions. Beaucoup de DBA préfère restaurer en mode NO RECOVERY.

```
BACKUP LOG InTimeDB
TO DISK ='F:\SQL_BACKUP\InTimeDBFullBackup.log' WITH NORECOVERY;
```

- ▶ Restaurons à nouveau la sauvegarde complète.

```
RESTORE DATABASE InTimeDB
FROM DISK ='F:\SQL_BACKUP\InTimeDBFullBackup.bak' WITH FILE =1, NORECOVERY
```

- ▶ Actualisons notre BDD, nous notons qu'elle est passée en mode restauration.



➔ A présent remettons notre BDD telle qu'elle était à 1:29:59.

```
RESTORE LOG InTimeDB
FROM DISK = 'F:\SQL_BACKUP\InTimeDBFullBackup.log'
WITH FILE =1, RECOVERY, STOPAT='09 février 2018 01:29:59'
```

➔ Actualisez la BDD, vous noterez qu'elle a été restaurée. Mais ce n'est pas fini !! ➔ Refaisons une lecture de notre table SPORTIFS.

```
SELECT *
FROM SPORTIFS
```

Résultats

	ID	NOM	SPORT
1	1	Rougerie	RUGBY
2	2	Jordan	BASKET
3	3	Bolt	ATHLE
4	4	Federer	TENNIS
5	5	Nadal	TENNIS

Nous pouvons effectuer notre mise à jour, qui fonctionnera cette fois.

```
UPDATE SPORTIFS
SET Nom = 'Mandanda' WHERE ID = 1;
/* MAJ de la table à 1:34 */
```

➔ Aussi, bien que dans les faits on soit sorti du SINGLE_USER, assurons que le travail est bien terminé en remettant notre BDD en mode multi-utilisateur.

```
ALTER DATABASE InTimeDB SET MULTI_USER;
/* BDD en mode multi-user */
```

4- Atelier Pratique 4 : Audit des performances d'une base de données existante et proposition d'un plan d'amélioration

Contexte

Vous êtes consultant(e) en performance pour **TechShop**, une boutique en ligne qui rencontre des lenteurs depuis l'augmentation du trafic. Votre mission : analyser la base de données et proposer un plan d'amélioration.

Objectifs de l'Atelier

Partie 1 : Diagnostic des Performances

1. Identifier les requêtes les plus lentes
2. Analyser l'utilisation des index
3. Déetecter les goulots d'étranglement
4. Évaluer l'utilisation des ressources

Partie 2 : Plan d'Amélioration

1. Proposer des index manquants
2. Optimiser les requêtes problématiques
3. Suggérer des améliorations architecturales
4. Établir un plan de monitoring

Exercices à Réaliser

Exercice 1 : Analyse des Requêtes Lentes

- Trouver les 10 requêtes les plus longues
- Analyser leur impact sur les performances

Exercice 2 : Audit des Index

- Identifier les tables sans index ou avec des index inefficaces
- Proposer des index manquants

Exercice 3 : Analyse de l'Utilisation des Ressources

- Vérifier l'utilisation CPU, mémoire et disque
- Identifier les contraintes matérielles

Exercice 4 : Détection des Blocages

- Analyser les blocages entre transactions
- Proposer des solutions pour les réduire

Livrables Attendus

Rapport d'Audit (à compléter) :

1. ÉTAT DES LIEUX :

- Requêtes critiques : _____
- Temps moyen de réponse : _____
- Principaux goulets : _____

2. RECOMMANDATIONS :

- Index à créer : _____
- Requêtes à optimiser : _____
- Paramètres à ajuster : _____

3. PLAN D'ACTION :

- Actions immédiates (J+7) : _____
- Actions court terme (J+30) : _____
- Actions long terme (J+90) : _____

Durée de l'Atelier

- **Diagnostic** : 45 minutes
- **Analyse et recommandations** : 45 minutes
- **Présentation des résultats** : 30 minutes

 Votre mission : Rédiger un rapport clair et actionnable pour l'équipe technique de TechShop !