Notes sur le module : *Base de Données.*

Armel Pitelet

10 janvier 2022

Table des matières

1 BDD et SGBD

1.1 Généralités

Commencer par voir https://github.com/Authfix/dotnet-basics/blob/main/sql-server-basics.ipynb pour les définitions et principe de base. Pour se connecter (en local) il faut commencer par utiliser le serveur : (LocalDb)\MSSQLLocalDB.

Note :

Attention à l'installation d'une bdd en local. Elles sont pensé pour être seul et prennent donc toute la place et les ressources disponibles.

Note :

La base doit se situer sur la même machine que le gestionnaire de base. Pas forcément dans un *folder* spécifique mais sue la même *machine* (ou serveur, ou conteneur docker, ...).

Note :

Dans une base tout est objet.

Note :

Il existe deux type de backup : full → avec tout dedans. Incrémental → se base sur un backup full mais n'enregistre que le delta par rapport à cette partie la.

Remarque : *Sur la stratégie de backup classique.*

Exemple : Dimanche → full puis de Lundi à Samedi → incrémental.

Remarque : *Sur l'archivage.*

Une BDD n'est pas faite pour faire de l'archivage long terme (en dehors du backup) → à cause de l'obligation de passer par un système de gestion de base de données pour pouvoir lire dans la base. Pour de l'archive il vaut mieux avoir les données dans un fichier *clair* csv, json, dat,... aka pas un fichier binaire (dépendant d'un système de lecture particulier).

Remarque : *Sur la spécialisation des SGBD.*

Selon le type de donnée, il existe des SGBD qui sont plus efficaces que d'autres (données temps, données espace,...). SQL Server peut à peut prêt tout faire mais le fera moyennement bien (rapidité, efficacité, optimisation).

Note :

voir docker et powershell → cela aidera grandement pour toute la suite.

1.2 Contenue d'une table

On prend l'exemple de la table SalesLT.Adress (au début) et on déroule les différents menus.

Définition : **table**.

Liste de données obéissant à une certaine nomenclature, un certain format. Cela garantie que chaque ligne de données possèdera les même colonne. Faire click-droit → property pour plus de détails.

Remarque : *Columns.*

Contient les infos sur les différents colonnes de la tables.

Note :

Les type de valeur peuvent être différent des types du système. Pour gérer de l'upper case/lower case par exemple.

Note :

Le nom de chaque colonne est accompagné d'un type (ex : int) et de conditions particulière (ex : not null)

Remarque : *Sur varchar.*

Il existait varchar et aujourd'hui nvarchar (Pour nouveau varchar). Ce nvarchar(x) est accompagné de sa taille maximum x.

Remarque : *Sur les dates et le temps.*

Il faut toujours utiliser un temps universel et pas un temps localisé.

Remarque : *Unique identifier.*

Est un type de donnée en bdd qui correspond à un Guid.

Remarque : *rowid (rowguid).*

Est un identifier qui correspond à une ligne de base de donnée qui change (ou doit être changé) à chaque modification de la ligne. C'est un moyen de bloquer des updates parallèles (cela assure de repasser par la dernière version et pas une version antérieur). Attention cette mécanique n'est pas automatique, il faut l'implémenter.

Remarque : *Sur le nombre de ligne.*

Une BDD ne sait jamais combien de ligne elle possède exactement !

Remarque : *Keys*.
Dans ce dossier les différentes clefs de la table (voir ci dessous).

Définition : **PK**.
Primary Key : → garantie que chaque ligne de la table est unique. C'est un index ordonné : c'est à dire que c'est via elle qu'il est le plus rapide et efficace d'aller chercher une donnée. C'est l'équivalent d'un annuaire.
Remarque : *AK*.
Fait partie du mécanisme de rowguid.

Remarque : *Dans le folder keys*.
FK : plutôt Foreign Key qui permet de faire le liens entre les différentes tables et de garantir l'existence de la donnée du coté des deux tables. Convention de nommage : le nom des clefs indique le liens entre les table. Ex : FK_CustomerAdress_Adress_AdressID fait le lien entre CustomerAdress et Adress_AdressID.

Remarque : *Constraints*.
Dans ce folder on trouve des objet Contrainte qui vont s'appliquer sur des colonnes particulières. On peut par exemple donner des valeurs par défauts dans ces contraintes (comme le temps en cours au moment d'une modification ou bien pour donner par défaut un nouveau guid lors d'un changement de ligne). On peut également poser une contrainte d'unicité (sans passer par une primary key), Ex : une adresse email qui doit être unique.

Remarque : *Triggers*.
Permet d'écrire du code SQL qui réagit en fonction de certaines actions faites dans les tables. Ex : exécuter un code qui remplit une table X quand une table Y est mise à jour.

Note :
Pour faire des trigger il faut travailler avec du TSQL (language de programmation [spécifique à Microsoft]). Le trigger tend à disparaitre car on essaie d'éviter de maintenir du code coté bdd et coté applicatif en même temps.

Remarque : *Indexes*.
Permet de ranger la base pour l'optimiser en fonction d'un certain type de recherche. Ce n'est pas réellement cela, les index créent en fait un fichier qui garde de l'information utile pour savoir comment parcourir la base de donnée en fonction du type de recherche demandée. Attention : ces fichiers peuvent poser très lourd très vite. Cela engendre aussi une lenteur dans la suppression, l'update et l'insertion (puisqu'il faut mettre à jour les index en plus des données elle même).

Note :
Eviter de se servir des indexes tant que l'on a pas de problème de performance. De plus ce n'est pas la seule manière d'optimiser la lecture dans une base. La base peut également ne pas se servir du tout de ces index si elle *estime* qu'ils ne sont pas pertinent.

Note :
Les Clefs sont systématiquement (obligatoirement) accompagnées d'index (pour AK/PK mais pas forcément FK [qui lient deux tables différentes]).

Remarque : *Statistics.*

Ne sert pas souvent... La base de donnée reste dans un état d'incertitude sur beaucoup de chose (pour des raisons de performance) et garde des statistiques à la place de données exacts (comme le nombre de ligne par exemple). Les stats se mettent à jour pour éviter que la base ne se trompe trop souvent. C'est surtout un outils d'analyse pour comprendre exactement ce qu'il se passe dans la BDD. Pratiquement réservé aux développeurs spécialistes des BDD ou au moteur d'optimisation qui se basent sur les stats pour décider de la meilleur manière de faire une requète à la BDD (faite par un moteur d'éxécution).

2 Divers

Remarque : *Sur les schéma.*

Au niveau des nom des tables on a Schema.Nom_de_table. Le schéma permet de moduler le comportement des tables, par exemple pour donner des accès à certain et pas à d'autre. Ces schémas permettent de segmenter la BDD.

Note :

Une application ne doit jamais authoriser la modification des schémas de la bdd (sauf admin aka superuser). Il vaut mieux avoir 2 user différents avec deux roles différent plutôt qu'un seul type de user avec deux rôles différents.

Remarque : *Principe de List Priviledges.*

Il faut à tt pris éviter de donner un droit non nécessaire. L'idée est de conserver au maximum les données et surtout la structure de ces données.

3 Le SQL

liens utiles https://www.w3schools.com/sql/sql_top.asp, https://sql.sh/.

Mot clef : **SELECT**.

Permet de récupérer les données nécessaires (dont on a besoin). SELECT nom_de_colonne

Mot clef : **FROM**.

Permet de sélectionner la table que l'on veut : Schema.nom_de_table

Mot clef : **WHERE**.

Permet de sélectionner par filtre. WHERE : Predicat

Remarque : *Sur les prédicats.*

on a le =, comparaison exact, ou bien le LIKE 'untruc%' ou le % est un wildcard indiquant qu'il y aura un truc après. C'est du Regex.

Mot clef : **AND et OR**.

Pour *empiler* les prédicat. Utilisable avec un WHERE par exemple.

Note :

Les fichier.sql sont un bon endroit pour faire des calculs (au lieu de récupérer la donnée pour la traiter en externe). On utilise les fonction d'aggrégats

Remarque : *Quelques fonction d'aggrégat.*

MAX, SUM, MIN, COUNT, AVG, TOP (n) column_name1, column_name2 [remonte les n lignes des columns]...

Note :

Les mot clef ne sont pas case sensitive. MAX est équivalent à Max ou max

Mot clef : **ORDER BY**.

Order by column_name opt : ordonne la remontée des données par la colonne column_name dans l'ordre opt (ASC OU DSC). En principe on en met partout pour maitriser la remontée des données.

Mot clef : **GROUP BY**.

Group by column .Regroupe par valeur unique dans column. Nécessite un SELECT fun_aggregation() [et pas un simple SELECT column]

Mot clef : **JOIN**.

JOIN schema.table ON schema.table.column_interessante = schema.Autretable.column_interessante. schema.table.column_interessante est une colonne de FK qui font référence à schema.Autretable.column_interess Le nom des colonnes n'est pas forcément le même dans les deux tables mais le types doit rester cohérent. Le JOIN permet de mapper une table à une autre. Le join va joindre les primary keys de la table cible aux foreign keys de la table mère (celle qui demande la jointure).

Note :

On peut (et on doit) mettre des alias sur les noms quand tt devient trop verbeux.

Remarque : *Pour créer une base de donnée.*

Click droit sur *database* → new database. Va créer deux fichier, la ddb et le log. Option permet ensuite de changer des choses appliqués à toutes la ddb, ex : *collation* qui permet d'avoir du case sensitive ou non (à bien choisir au début car dure à changer et appliquer à toute la base !) : CI → case insensitive ou CS pour sensitive. AS ou AI pour Accent sensitive ou insensitive.

Remarque : *Lors de la création d'une table.*

Click-droit dans l'onglet *table* → new table. Les paramètres en dessous permettent une gestion plus fines des colonnes (comme identity qui définit ou non si le champ est une PK).

Note :

Une fois la table créée on peut générer automatiquement les scripts de CrUDe et plus.

Note :

Dans les scripts sql on peut (et on devrait) faire [schema].[table] au lieu de schema.table ou juste table.

Acronymes

DSP Nom simple. , *Glossaire :* DSP

Glossaire

DSP la description détaillé.