Secteur Tertiaire Informatique
Filière étude - développement

Activité « Développer la persistance des données »

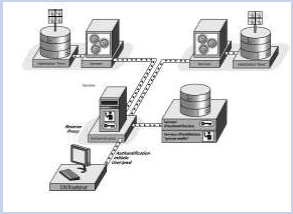
LDD : Base de données – Table – Index – Vue

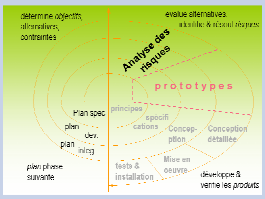
Accueil

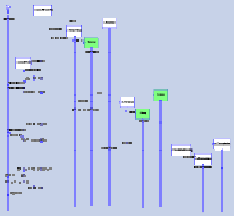
Apprentissage

Période en entreprise

Evaluation







JPF

Sommaire

Suppléments

1 . Composants d'une Base de Données

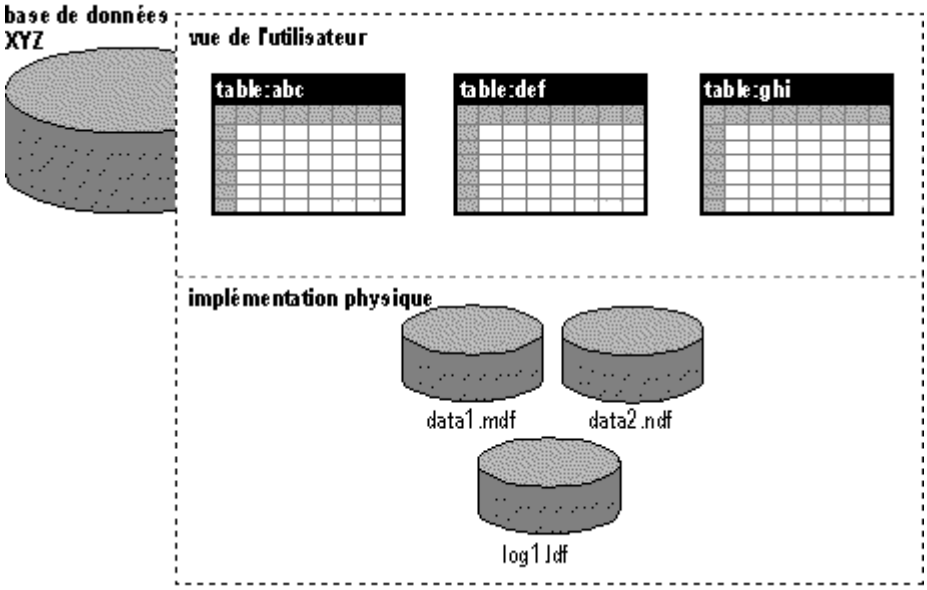
Une base de données se compose d'un ou plusieurs fichiers.
Ils sont créés automatiquement lors de la création de la base.
Ils sont propres à la BD et ne peuvent être partagés avec d'autres BD.
Ils sont situés sur un ou plusieurs disques. La mise en œuvre physique est du ressort des administrateurs de la BD. Elle est transparente pour les utilisateurs.

Chaque instance de SQL Server possède quatre bases de données Système :

- Master : contient toutes les informations systèmes ainsi que les informations concernant toutes les autres BD, les connexions et les paramètres de configuration.
- Tempdb : enregistre les tables temporaires et les procédures stockées des utilisateurs. Elle est créée à chaque fois que SQL Server est démarré.
- Model : sert de modèle lors de la création d'une BD.
- MSDB : utilisé pour programmer alertes et travaux.

Chaque base possède au moins :

- un fichier de données
- un fichier « Journal des transactions ».



Le **fichier primaire** (d'extension .MDF) est le point de départ de la base de données.
Les fichiers secondaires, facultatifs (d'extension .NDF) contiennent toutes les données et objets qui ne se trouvent pas dans le fichier primaire.
Les **fichiers journaux** (d'extension .LDF) contiennent toutes les informations relatives aux transactions effectuées et permettent de récupérer la base en cas d'incident.

Considérations relatives aux fichiers de base de données

- la création d'une base de donnée s'effectue par copie de la base Model, comprenant les tables système
- les données sont stockées dans des pages de 8 Ko d'espace disque contigu
- les lignes ne peuvent pas s'étendre sur plusieurs pages
- les tables, les index sont stockés dans des extensions (une extension = 8 pages contiguës donc 64 Ko)
  8 objets peuvent se partager une extension (extension mixte).
  Lorsqu'une table dépasse 8 pages, elle utilise ses propres extensions (extension uniforme).
- La taille du journal des transactions représente 25% de la taille des données.

Mode de fonctionnement du journal des transactions

Le journal des transactions consigne les modifications apportées aux données, via les instructions INSERT, UPDATE ou DELETE.

1. Lorsque la modification est exécutée, les pages de données concernées sont chargées dans une zone mémoire appelée cache de données
2. Chaque instruction de modification de données est enregistrée
3. dans le journal des transactions au moment ou elle est exécutée. La modification est toujours enregistrée dans le fichier journal, avant d'être appliquée à la base (journal à écriture anticipée).
4. Le processus de point de contrôle écrit toutes les transactions validées (par une instruction COMMIT TRANSACTION) dans la base.

En mode de fonctionnement normal, le processus de point de contrôle vérifie régulièrement que pour toutes les transactions validées, les données correspondantes ont bien été réécrites dans la base. Un point de contrôle est créé
En cas de défaillance du système, le processus de récupération automatique s'exécute au redémarrage de SQL Server.

Structure logique d'une Base de Données

Une base de données est structurée en objets (visibles dans l'explorateur d'objets) :

- **Schémas : il s'agit de « conteneurs » de tables**
- **Tables** : composées de colonnes structurées (nom, type, longueur) et de lignes
- **Par table :**
  - **Index** : données organisées pour faciliter la recherche. Les index associent la valeur d'une colonne avec l'adresse physique de la ligne.
  - **Contraintes** : règles permettant de garantir l'intégrité référentielle de la BD.
  - **Déclencheurs** : scripts utilisateurs déclenchés automatiquement par une instruction Insert, Update ou Delete.
- **Vues** : une vue est le résultat d'une requête sur une ou plusieurs tables. Elle ne correspond à aucun stockage physique.
- **Programmabilité** :
  - **Procédures stockées** (utilisateur) : groupe d'instructions compilées en un seul plan d'exécution.
  - **Fonctions** (utilisateur) : code encapsulé en vue de sa réutilisation pour faciliter les requêtes.
  - **Types :** les types de données systèmes ou ceux créés par l'utilisateur (voir Supplément A)
  - **Règles et valeurs par défaut.**
- **Stockage :** objets en rapport avec le stockage.
- **Sécurité :** utilisateurs, rôles …

DDL : langage de création des objets de la BD

Pour manipuler des objets de la Base de données, on utilise le DDL : Data Description Language. (ou LDD : Langage de Description de Données)

CREATE, ALTER et DROP.

Ces instructions permettent de créer, modifier ou supprimer des objets tels que :

- Database
- Schema
- Table
- View
- Index
- Eléments de programmation : fonctions, déclencheurs, procédures stockées
- Types de données …

2 . Création d'une Base de données

Une base de données peut être créée de deux façons :

- à l'aide de l'interface graphique SQL Server Management studio, en cliquant avec le bouton droit sur Base de données puis, dans le menu contextuel, en sélectionnant Nouvelle Base de données
- par l'instruction CREATE DATABASE de Transact-Sql.

Le journal des transactions sera créé automatiquement, quelque soit le processus utilisé.

Création à l'aide de l'interface graphique SQL Server Management studio

- Dans la page **Général**, indiquer le nom et le propriétaire de la base ; une boite de dialogue permet de sélectionner la connexion qui sera propriétaire de la base. La création d'une base de données comprend l'affectation d'un nom, la déclaration de la taille et de l'emplacement des fichiers. ; on précise également la taille maximale autorisée et l'incrément de croissance autorisée (0 indiquera qu'aucune croissance n'est autorisée) :

  Les options de croissance automatique des fichiers (par défaut) évitent les tâches administratives liées à la croissance manuelle de la base ; elles réduisent également le risque que la base de données manque d'espace disque de façon inattendue.

  Les groupes de fichiers sont essentiellement destinés aux bases de données importantes (> 1GO) et aux tâches d'administration avancées : leur principal intérêt est d'améliorer le temps de réponse de la base, en autorisant la création de ses fichiers sur plusieurs disques et/ou l'accès par de multiples contrôleurs de disque.

  Dans la zone chemin d'accès, saisir le chemin d'accès complet au fichier de données (emplacement par défaut sélectionné) ; le fichier primaire doit porter l'extension **.mdf**. Les fichiers de données secondaires constituent un espace supplémentaire pour stocker des données et doivent porter l'extension **.ndf**.

- Dans la page **Options**, sélectionner un classement pour la base de données :

  French_CI_AS : langue française, **C**ase **I**nsensitive (Casse non prise en compte), **A**ccent **S**ensitive (Accent pris en compte)

Création par l'instruction CREATE DATABASE de Transact-SQL

CREATE DATABASE *nom_base*
[ON
    [PRIMARY]
    (
        [NAME = *Nom_fichier_logique*,
        FILENAME ='*nom complet du fichier (avec chemin d'accès)*'
        [, SIZE = taille [KB | MB | GB | TB]]
        [, MAXSIZE = *taille_max* / UNLIMITED]
        [, FILEGROWTH = *incrément_croissance*])
)
[LOG ON {
    (
        [NAME = *Nom_fichier_logique*,
        FILENAME ='*nom complet du fichier (avec chemin d'accès)*'
        [, SIZE = taille [KB | MB | GB | TB]]
        [, MAXSIZE = *taille_max* / UNLIMITED]
        [, FILEGROWTH = *incrément_croissance*])
) }]
]
[COLLATE *nom-séquence_classement*] (voir supplément C)
[WITH DB_CHAINING {ON | OFF} | TRUSTWORTHY {ON | OFF}]

nom-séquence_classement

Indique le classement par défaut de la base de données. Le nom du classement peut être un nom de classement Windows ou SQL. S'il n'est pas spécifié, la base de données est affectée au classement par défaut de l'instance SQL Server

DB_CHAINING {ON | OFF}
Lorsque ON est spécifié, la base de données peut être la source ou la cible d'un chaînage des propriétés des bases de données croisées.

Lorsque OFF est spécifié, la base de données ne peut pas participer au chaînage des propriétés des bases de données croisées. La valeur par défaut est OFF.

TRUSTWORTHY {ON | OFF}
Lorsque ON est spécifié, les modules de base de données (par exemples les vues, les fonctions définies par l'utilisateur ou les procédures stockées) utilisant le contexte d'emprunt d'identité peuvent accéder à des ressources en dehors de la base de données à modifier obligatoirement par Transact-Sql :

ALTER DATABASE *nom_base* SET TRUSTWORTHY ON.

Exemple 1: Création d'une base de données appelée Ventes avec un fichier de données primaires de 1Go et un fichier journal de 500Mo.
Le fichier de données primaire peut croître jusqu'à 5Go par incrément de 5%.
Le fichier journal peut croître par incrément de 25Mo.

```
CREATE DATABASE Ventes
ON PRIMARY
      (NAME = Ventes_data,
      FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata.mdf'',
      SIZE = 1GB, MAXSIZE = 5GB, FILEGROWTH = 5%)
LOG ON
      (NAME = 'Ventes_log',
       FILENAME ='c: \program files\Microsoft SQL Server\mssql\data\ventelog.ldf',
       SIZE = 500MB, FILEGROWTH = 5%)
```

Exemple 2: On aurait pu créer cette base avec 2 fichiers de données

```
CREATE DATABASE Ventes
ON PRIMARY
      (NAME = Ventes_data1,
       FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata1.mdf',
       SIZE = 1GB, MAXSIZE = 5GB, FILEGROWTH = 5%),
      (NAME = Ventes_data2,
       FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventedata2.ndf',
       SIZE = 500MB, MAXSIZE = 2GB, FILEGROWTH = 10%)
LOG ON
      (NAME = 'Ventes_log',
       FILENAME = 'c: \program files\Microsoft SQL Server\mssql\data\ventelog.ldf',
       SIZE = 500MB, FILEGROWTH = 5%)
```

Exemple 3: On aurait pu créer cette base avec 2 fichiers de données, répartis dans 2 groupes de fichiers distincts

```
CREATE DATABASE Ventes
ON PRIMARY
      (NAME = Ventes_data1,
       FILENAME = 'c:\program files\Microsoft SQL Server\mssql\data\ventedata1.mdf',
       SIZE = 1GB, MAXSIZE = 5GB, FILEGROWTH = 5%),
FILEGROUP Histo
      (NAME = Ventes_data2,
       FILENAME = 'c:\program files\Microsoft SQL Server\mssql\data\ventedata2.ndf',
       SIZE = 500MB, MAXSIZE = 2GB, FILEGROWTH = 10%)
LOG ON
      (NAME = 'Ventes_log',
       FILENAME = 'c:\program files\Microsoft SQL Server\mssql\data\ventelog.ldf',
       SIZE = 500MB, FILEGROWTH = 5%)
```

Dans cet exemple, tout objet de la base de données peut être créé soit dans le groupe PRIMARY, soit dans le groupe HISTO.

Un fichier primaire .mdf se trouve toujours sur le groupe PRIMARY, et contient les tables système. Il contient également tous les objets non affectés aux groupes de fichiers utilisateur.

La répartition des données sur des fichiers ou des groupes de fichiers peut améliorer les performances (répartition des E/S), améliorer les temps de sauvegarde (sauvegarde par groupe de fichiers).

Le principal avantage des groupes de fichiers est de pouvoir répartir manuellement les objets sur un groupe de fichiers donné, alors que ce n'est pas possible sur un fichier de données.

Suppression d'une base de données

```
DROP DATABASE nom_base  [ , nom_base  … ]
```

3 . Création des Tables

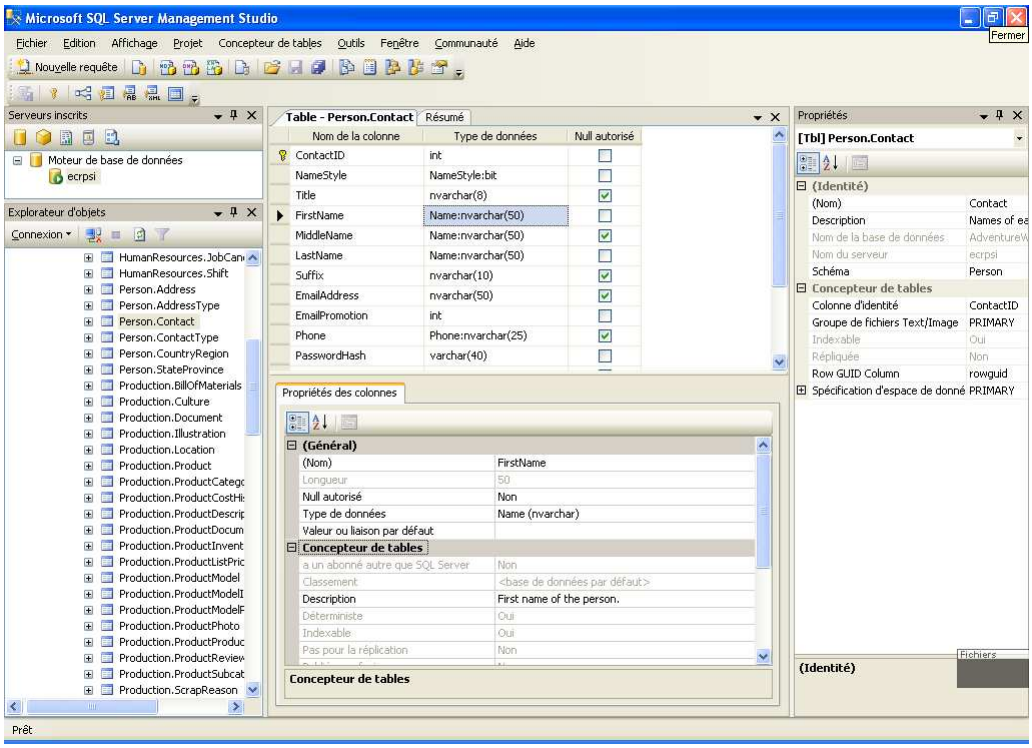
SQL Server limite à :

- Deux milliards de tables par base de données
- 1024 colonnes par table
- 8060 octets par ligne

Une table peut être créée à l'aide de SQL Server Management Studio ou grâce à l'instruction CREATE TABLE de Transact-Sql.

Création àl'aide de l'interface graphique SQL Server Manager studio

Sélectionner **Nouvelle table** dans le menu contextuel du nœud **Tables** de la base de données choisie.



La table sera facilement conçue au moyen des vues fournies :

- la vue Table qui permet de visualiser l'ensemble des colonnes de la table,
- la vue Propriétés des colonnes permet de spécifier pour la colonne sélectionnée : son nom, le type de données (voir supplément A), des contraintes, des valeurs par défaut (valeurs nulles autorisées ou non).
- la vue Propriétés de la table permet de définir son nom, sa description et son schéma.

Création par l'instruction CREATE TABLE de Transact-SQL

```
CREATE TABLE nom_table
    ( nom_colonne   type_données   [Contrainte]        [,...n] )
```

Exemple : Création de la table **Employe** pour laquelle les champs Matricule, Nom, Prénom, DateEntrée ne peuvent être nuls.

```
CREATE TABLE Employe
    (Matricule    SMALLINT        NOT NULL    IDENTITY(1,1)    ,
     Nom          VARCHAR(25)     NOT NULL                     ,
     Prenom       VARCHAR (15)    NOT NULL                     ,
     Adresse      VARCHAR (30)    NULL                         ,
     Cp           ZIPCODE                                      ,
     DateEntree   DATETIME        NOT NULL                     ,
     DateFin      DATETIME        NULL)
```

La propriété **IDENTITY** permet de créer des colonnes (appelées colonnes d'identité) contenant des valeurs numériques séquentielles générées par le système (champ compteur) et identifiant de façon unique chaque ligne ajoutée à une table. Une colonne d'identité sert souvent de clé primaire.
Les paramètres précisent la valeur de départ et l'incrément ; il ne peut y avoir qu'une seule colonne d'identité par table ; une colonne d'identité doit être de type entier, numérique ou décimal et n'autorise pas les valeurs nulles.

Exemple : Création de la table Client dont la colonne **CodeCli** est créée avec le type de données **uniqueidentifier**, en utilisant une valeur par défaut généré par la fonction **NEWID**.

```
CREATE TABLE Client
    (Codecli    UNIQUEIDENTIFIER  NOT NULL  DEFAULT  NEWID( ),
     NomCli     CHAR(30)                   NOT NULL)
```

Le type de données **uniqueidentifier** contient un numéro d'identification stocké comme une chaîne binaire sur 16 octets et sert à stocker un identificateur globalement unique (GUID).
La fonction **NEWID** crée un identificateur unique (GUID) stocké à l'aide du type de données uniqueidentifier.
Le fait de laisser SQL Server fournir automatiquement des valeurs de clé plutôt que de laisser l'application cliente s'en charger peut améliorer les performances.

Une table peut être modifiée par l'instruction Tansact-Sql ALTER TABLE.
Cette instruction permet d'ajouter, supprimer et modifier des colonnes.

Une table peut être supprimée par l'instruction Tansact-Sql DROP TABLE.

4 . Intégrité des données

On distingue 4 types d'intégrité :

- **L'intégrité de domaine** (ou de colonne) consiste à préciser l'ensemble des valeurs valides dans une colonne et à indiquer si les valeurs nulles sont autorisées.
- **L'intégrité d'entité** (ou de table) consiste à s'assurer que toutes les lignes d'une table possèdent un identificateur unique.
- **L'intégrité référentielle** garantit le maintien d'une relation existant entre la clé primaire d'une table référencée et la clé étrangère dans chacune des tables qui la référencent.
- **L'intégrité définie par l'utilisateur** permet de définir de règles d'entreprise spécifiques qui n'entrent dans aucune des autres catégories d'intégrité.

Les contraintes

Les contraintes permettent d'assurer l'intégrité des données. Elles garantissent la validité des valeurs saisies dans les colonnes et le maintien des relations entre les tables.

Type de contrainte		Description
PRIMARY KEY	Clé Primaire	Identifie chaque ligne de façon unique, interdit les doublons et garantit la création d'un index pour améliorer les performances. Les valeurs NULL ne sont pas acceptées. Une table ne peut comporter qu'une seule clé primaire.
FOREIGN KEY	Clé étrangère	Définit une colonne ou ensemble de colonnes dont les valeurs sont égales à la clé primaire de la table référencée.
UNIQUE	Unicité	Empêche la création de doublons dans la colonne non clé primaire et garantit la création d'un index pour améliorer les performances. Les valeurs NULL sont acceptées.
CHECK	Validation	Fournit une règle pour les valeurs d'une colonne.
DEFAULT	Valeur par défaut	Valeur par défaut de la colonne (lors d'un INSERT)
NOT NULL	Champ non nul	La valeur de la colonne doit être définie.

Les contraintes s'appliquent sur des colonnes ou sur des tables entières :

- Une contrainte de colonne fait partie de la définition de la colonne et ne s'applique qu'à celle-ci.
- Une contrainte de table est déclarée indépendamment de la définition d'une colonne et peut s'appliquer à plusieurs colonnes d'une même table. Dès lors que la contrainte porte sur plusieurs colonnes, elle doit être définie au niveau table.

Elles se mettent en œuvre dans les instructions CREATE TABLE ou ALTER TABLE.

Exemple 1 : Création de la table **Employe** pour laquelle :

- Les champs Matricule, Nom, Prénom, DateEntrée ne peuvent être nuls
- La clé primaire est le champ Matricule, champ compteur dont la valeur initiale est 1, et l'incrément de 1
- La colonne Nom ne peut contenir de doublons

```
CREATE TABLE Employe
(Matricule   SMALLINT      NOT NULL  IDENTITY (1, 1)  PRIMARY KEY,
 Nom         VARCHAR (25)  NOT NULL  UNIQUE ,
 Prenom      VARCHAR (15)  NOT NULL,
 Adresse     VARCHAR (30)  NULL,
 Cp          ZIPCODE,
 DateEntree  DATETIME      NOT NULL,
 DateFin     DATETIME      NULL)
```

Remarque : En définissant une contrainte UNIQUE sur une ou plusieurs colonnes, SQL Server crée automatiquement un index unique (voir les index).

On aurait pu coder différemment, en reportant les contraintes après la définition des colonnes :

```
CREATE TABLE Employe
(Matricule      SMALLINT      NOT NULL   IDENTITY (1,1),
 Nom            VARCHAR(25)   NOT NULL,
 Prenom         VARCHAR (15)  NOT NULL,
 Adresse        VARCHAR (30)  NULL,
 Cp             ZIPCODE,
 DateEntree     DATETIME      NOT NULL,
 DateFin        DATETIME      NULL,
 CONSTRAINT PK_Employe*   PRIMARY KEY (Matricule),
 CONSTRAINT UK_Employe*   UNIQUE (Nom) )
```

** une contrainte décrite indépendamment d'une colonne doit obligatoirement porter un nom. Il est judicieux de donner un nom explicite car il sera affiché en cas d'erreur.*

Pour une clé primaire constituée de 2 colonnes, on doit coder une contrainte de table :

```
CREATE TABLE Employe
(Matricule      SMALLINT      NOT NULL   IDENTITY (1,1),
 Nom            VARCHAR (25)  NOT NULL UNIQUE,
 Prenom         VARCHAR (15)  NOT NULL,
 Adresse        VARCHAR (30)  NULL,
 Cp             ZIPCODE,
 DateEntree     DATETIME      NOT NULL,
 DateFin        DATETIME      NULL,
 CONSTRAINT PK_matricule  PRIMARY KEY (Nom, Prenom) )
```

Exemple 2 : Ajout d'une contrainte DEFAULT qui insère la valeur '*Inconnu*' dans la colonne du prénom, si aucun prénom n'a été saisi. Il faut utiliser ALTER TABLE.

```
ALTER TABLE Employe
ADD CONSTRAINT DF_Employe_Prenom DEFAULT '*Inconnu*' FOR Prenom
```

Remarque : Pour obtenir par défaut la date système comme DateEntree, on code :

```
ALTER TABLE Employe
ADD CONSTRAINT DF_Employe_Entree DEFAULT  getDate() FOR DateEntree
```

Exemple 3 : Ajout d'une contrainte CHECK qui assure que la date de résiliation du contrat est bien postérieure à la date d'embauche.

```
ALTER TABLE Employe
ADD CONSTRAINT CK_ Employe CHECK (DateEntree < DateFin)
```

Une contrainte CHECK peut être créée avec n'importe quelle expression logique (booléenne) qui retourne TRUE ou FALSE sur la base des opérateurs logiques.

Exemples :

- Pour contrôler qu'une colonne salaire est comprise entre 1500€ et 10000€, l'expression logique est la suivante :

  ```
  Salaire >= 1500 AND salaire <= 10000
  ```

- Pour s'assurer qu'une colonne Departement, de typer char, est bien composée de 2 chiffres, l'expression logique sera de type :

  ```
  Departement like ('[0-9][0-9]')
  ```

Pour désactiver la contrainte :

```
ALTER TABLE Employe NOCHECK CONSTRAINT CK_ Employe
```

Pour réactiver la contrainte :

```
ALTER TABLE Employe CHECK CONSTRAINT CK_ Employe
```

Exemple 4 : Création de la table **Client** pour laquelle :

- Les champs CodeCli, Nom, CodeRep ne peuvent être nuls
- La clé primaire est le champ CodeCli
- La contrainte FOREIGN KEY, basée sur la colonne **Coderep** référence le champ **CodeRep** de la table **Représentant** pour garantir que tout client est associé à un représentant existant

```
CREATE TABLE Client
    (CodeCli   SMALLINT        NOT NULL   IDENTITY(1,1),
    Nom        VARCHAR(25)     NOT NULL,
    Coderep    SMALLINT        NOT NULL,
    CONSTRAINT PK_Codecli      PRIMARY KEY (CodeCli),
    CONSTRAINT FK_ClientRep    FOREIGN KEY (Coderep)
                               REFERENCES Representant (CodeRep))
```

Pour pouvoir créer une telle relation au niveau du schéma de la base, il est nécessaire que la colonne **CodeRep** soit clé primaire de la table REPRESENTANT.

Dans le cas d'une relation entre les tables Client et Représentant, le fait que l'intégrité empêche de supprimer un représentant est plutôt une bonne chose : un représentant quittant sa société, n'emmène pas tous ses clients avec lui ….

***

Il n'en serait pas de même entre les tables Commande et DétailCommande : en effet, si un client souhaite annuler sa commande, il est important de supprimer au même moment, tout le détail ….

```
ALTER TABLE.DetailCommande
    ADD CONSTRAINT FK_DetailCommande_Commande FOREIGN KEY (NO_Cde)
    REFERENCES Commande (NO_Cde)      ON DELETE CASCADE
```

On aurait également pu prévoir que la modification d'un numéro de commande dans la table Commande entraîne la répercussion automatique de la modification du numéro de commande dans la table DétailCommande.

```
ALTER TABLE.DetailCommande
    ADD CONSTRAINT  FK_DetailCommande_Commande FOREIGN KEY (NO_Cde)
    REFERENCES Commande (NO_Cde)      ON UPDATE CASCADE
```

On aurait pu également choisir que toutes les valeurs qui composent la clé étrangère soient :

- NULL
- à leur valeur par défaut

si la ligne correspondante dans la table parente est supprimée ou mise à jour.

Pour que cette contrainte soit exécutée, les colonnes de clé étrangère doivent accepter les valeurs NULL / des valeurs par défaut.

Si une colonne peut avoir une valeur NULL et qu'il n'existe aucun ensemble de valeurs par défaut explicite, NULL devient la valeur par défaut implicite de la colonne SET DEFAULT

Syntaxe Transact-Sql complète :

ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }
ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

Remarque : par défaut l'option retenue est NO ACTION qui empêche de supprimer ou mettre à jour la ligne de la table parente dès lors qu'il existe des « lignes affiliées ».

Valeurs par défaut

Les valeurs par défaut sont des objets qui peuvent être liés à une ou plusieurs colonnes, ou à des types de données utilisateurs. Elles peuvent être définies, puis réutilisées.

Elles ne sont pas normalisées ANSI : elles sont proposées pour des raisons de compatibilité ascendante, mais **il est préférable de privilégier les contraintes CHECK et DEFAULT.**

Une valeur par défaut est créée par l'instruction CREATE DEFAULT, puis doit être associée à une colonne ou un type de données utilisateur en exécutant une procédure système sp_bindefault.

Création d'une valeur par défaut sous Transact-SQL :

CREATE DEFAULT [*nom_schéma*.]*nom-par-défaut* AS *expression_constante*
EXEC sp-bindefault *nom-par-défaut* , *nom_de_colonne*

Exemple : Positionnement de la valeur 'Inconnu' si la colonne Prénom n'a pas été saisie

CREATE DEFAULT ***Pinconnu*** AS *'Inconnu'*
EXEC sp_bindefault ***Pinconnu*** , ***Prenom***

Sppression d'une valeur par défaut

DROP DEFAULT [*propriétaire*.]*nom-par-défaut* [, [*propriétaire*.]*nom-par-défaut*…]

5 . Les Index

SQL Server 2005 stocke les lignes d'une table dans des pages de données.
Pour accéder aux données, SQL Server effectue une analyse de la table ou un accès indexé.

L'organisation des pages de données s'effectue :

- **Pour les tables dépourvues d'index (segments de mémoire)**
  Les lignes de données ne sont pas stockées dans un ordre particulier et les pages ne sont pas chaînées.
  Lors d'une **analyse de table**, SQL Server parcourt les pages et balaye ainsi toutes les lignes de la table en extrayant celles qui satisfont à la requête.
  Adapté aux tables de petite dimension ou lorsqu'un pourcentage élevé de lignes est renvoyé.

- **Pour les tables possédant un index**
  Un Index est composé de pages de 8Ko chaînées entre elles.
  Les pages sont rangées dans l'ordre de la clé. Une ligne d'index contient la valeur de la clé et un pointeur vers une autre page ou la page finale.
  Lorsqu'un index existe, l'optimiseur de requêtes détermine s'il est plus efficace d'analyser la table ou d'utiliser l'index.
  Lors d'un **accès indexé**, SQL Server parcourt l'index ; il ne lit et n'extrait que les lignes satisfaisant les critères de la requête.
  Adapté à l'accès à des lignes uniques ou lorsqu'un faible pourcentage de lignes est renvoyé.

Pour / Contre la création d'index :

De manière générale, les index accélèrent les requêtes qui effectuent des jointures de tables et des opérations de tri ou groupage.
Les index sont créés et actualisés en ordre croissant.
Lors d'une modification de données, SQL Server doit mettre à jour les index.
La gestion des index demande du temps et des ressources.

Les colonnes à indexer :

- Clés primaires, clés étrangères ou colonnes utilisées pour jointure
- Colonnes devant être triées (order by)
- Colonnes utilisées dans les requêtes GROUP BY
- Colonnes utilisées dans les fonctions d'agrégation

Les colonnes à ne pas indexer :

- Colonnes rarement référencées ou contenant des types bit, text ou image
- Colonnes contenant peu de valeurs uniques

Architecture d'un index

Un **index ordonné en clusters (CLUSTERED)** trie physiquement les lignes d'une table. Les feuilles de données sont chaînées entre elles directement et suivent l'ordre de l'index. Il a une performance optimale, mais les insertions deviendront difficiles.
Une table ne peut posséder **qu'un seul index ordonné en clusters**, car ses lignes ne peuvent être stockées que dans un seul ordre physique ; l'ordre physique des lignes de l'index et l'ordre des lignes de l'index sont identiques (il est conseillé de créer cet index avant tout autre).

Un **index non ordonné en clusters (NONCLUSTERED)** contient des pages crées selon l'ordre d'insertion. Une feuille d'index contient la valeur de la clé et un pointeur vers une autre page ou la page finale (on parle de tri logique).
Les index non ordonnés en clusters sont les index par défaut de SQL Server (249 maximum par table) et sont intéressants lorsque les utilisateurs souhaitent rechercher des données de plusieurs façons.

Exemple : sur une table client, un index ordonné en clusters serait construit sur le *code*, des index non ordonnés en clusters pourraient être construits sur les colonnes *nom* et *catégorie*.

Un index est automatiquement créé lorsqu'une contrainte PRIMARY KEY ou UNIQUE est ajoutée à une table.

Depuis SQL Server 2000, il est possible de créer des index sur des vues et des colonnes calculées.

SQL Server stocke les informations sur les index dans la vue sys.indexes.

Dans la plupart des cas, il est préférable d'utiliser la contrainte UNIQUE plutôt qu'un index unique.

L'instruction CREATE INDEX échouera s'il existe des doublons dans la colonne à indexer.

Création d'index

Seul le propriétaire de la table ou de la vue peut créer des index sur celle-ci, qu'il y ait ou non des données dans la table.

Un index peut être créée à l'aide de SQL Server Management Studio ou grâce à l'instruction CREATE INDEX de Transact-Sql.

Création avec SQL Server Management Studio

Choisir l'option **Nouvel Index** dans le menu contextuel du nœud **Index** de la table choisie.

Création par l'instruction CREATE INDEX de Transact-SQL

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX *nom_index*
ON { *table* | *view* } **(** *column* [ASC | DESC] [**,**...*n*] **)**
[WITH
[PAD INDEX= ON | OFF]
[[,] FILLFACTOR = *taux_remplissage]*
[[,] IGNORE_DUP_KEY = ON | OFF]
[[,] DROP_EXISTING]
[[,] STATISTICS_NORECOMPUTE]
[[,] SORT_IN_TEMPDB]
]
[ON *groupe_fichiers*]

UNIQUE
Si le mot-clé UNIQUE n'est pas spécifié dans l'instruction CREATE INDEX, les doublons sont autorisés dans l'index.

CLUSTERED | NONCLUSTERED
Spécifie si l'ordre physique des lignes est dépendant / indépendant de l'ordre des clés : si le mot clé CLUSTERED n'est pas spécifié, c'est un index non ordonné en clusters qui est créé

PAD INDEX= ON | OFF
Spécifie si le taux de remplissage indiqué dans le paramètre FILLFACTOR est appliqué ou non aux pages d'index L'option PAD_INDEX est utile seulement si FILLFACTOR est spécifié

FILLFACTOR = *taux_remplissage*
Taux de remplissage des pages d'index (entre 1 et 100) ; si la valeur est 0 ou 100, les pages sont remplies complètement.

IGNORE_DUP_KEY= ON | OFF
Lors de l'insertion d'une clé en double, avec la valeur ON, seules les lignes qui ne respectent pas l'unicité sont omises ; avec la valeur OFF, l'ensemble de la transaction est annulée.

Exemple 1 : Création d'un index nommé **IX_Nom** sur la colonne **Nom** de la table **EMPLOYES**.

CREATE INDEX **IX_Nom** ON **EMPLOYES** (**Nom**)

L'exécution de CREATE INDEX avec l'option DROP_EXISTING permet de reconstruire un index existant ou modifier ses caractéristiques :

- son type
  possibilité de transformer un index non ordonné en clusters en index ordonné en clusters (l'inverse n'étant pas possible) ou transformation d'un index unique en index non unique et inversement.
- Ses colonnes indexées
  possibilité d'ajouter ou retirer ou spécifier d'autres colonnes
- Ses options
  modification possible des options FILLFACTOR/PAD_INDEX
-

Exemple 2 : Re-construction de l'index existant IX_NOM.

CREATE UNIQUE INDEX **IX_Nom** ON **EMPLOYES (Nom, Prenom)**
WITH DROP_EXISTING

<u>Suppression d'index</u>

L'instruction **DROP INDEX** supprimera l'index.

DROP INDEX *nom_table.nom_index* [, *nom_table.nom_index* …]

6 . Les Vues

Une vue permet de stocker une requête prédéfinie sous forme d'objet de base de données, pour une utilisation ultérieure.

Avantages :

- Mise en valeur des données pour les utilisateurs. Les données utiles sont sélectionnées, les données inutiles ou stratégiques sont occultées.
- Masquage de la complexité de la base de données. Les requêtes complexes sont simplifiées.
- Simplification de la gestion des autorisations des utilisateurs. Les autorisations peuvent être données aux utilisateurs d'effectuer des requêtes uniquement par le biais de vues.
- Amélioration des performances. Les vues permettent de stocker les résultats de requêtes complexes. D'autres requêtes peuvent utiliser ces résultats récapitulatifs.
- Organisation des données pour les exporter vers d'autres applications.

Création des Vues

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH { ENCRYPTION | SCHEMABINDING | VIEW_METADATA } ]
AS  select_statement       [ WITH CHECK OPTION ]
```

Si vous ne spécifiez pas l'argument column, les colonnes de la vue prennent le même nom respectif que les colonnes indiquées dans l'instruction SELECT.

La clause SELECT moyennant quelques restrictions :

- pas de clause COMPUTE ou COMPUTE BY
- pas de clause ORDER BY sans clause TOP
- pas de référence à une table temporaire
- ….

comporte toutes les clauses d'une instruction SELECT classique.

ENCRYPTION

Le texte de création de l'instruction CREATE VIEW, stocké dans la table système **sys.syscomments** est chiffré. Attention : Pour décrypter une vue, il faut la supprimer et la recréer ou la modifier à partir du texte d'origine.

SCHEMABINDING

L'ajout de cette clause permet de garantir que les tables ou vues dépendantes ne seront pas supprimées.

VIEW_METADATA

Cette clause permet de retourner des informations de méta données sur la vue, et non pas sur les tables de base.

WITH CHECK OPTION

Cette option impose que toutes les instructions de modification de données exécutées sur la vue respectent le critère défini dans la clause Select.

Exemple : Création d'une vue permettant de faciliter la consultation des informations situées dans les 2 tables Employes et Depart sans visualiser la colonne salaire.

EMPLOYES (NOEMP, NOM, PRENOM, DEPT, SALAIRE)
DEPART (NODEPT, NOMDEPT)

```
CREATE VIEW dbo.VueEmp (NumEmp, Nom, Prenom, Dep)
AS
        SELECT Noemp, Nom,Prenom,Wdept
        FROM Employes
        JOIN Depart On Nodept = Wdept
```

Suppresion d'une Vue

La suppression d'une vue s'effectue par l'instruction **DROP VIEW.**

DROP VIEW [*propriétaire,*] *view_name* [, [*propriétaire,*] *view_name* …]

7 . Les Schémas

Les schémas sont des conteneurs d'objets qui simplifient la gestion et la création de sous-ensemble d'objets traités globalement. Les schémas sont distincts des utilisateurs. Ceux-ci possèdent des schémas et ils disposent toujours d'un schéma par défaut que le serveur utilise pour les objets non qualifiés des requêtes.

Création d'un schéma sous Transact-SQL

CREATE SCHEMA *nom-schema* AUTHORIZATION *nom_propriétaire*

Exemple 1 : Création du schéma Gescli dans la base Ventes, de propriétaire Gestion.

```
USE Ventes;
CREATE SCHEMA Gescli AUTHORIZATION Gestion
```

L'instruction CREATE SCHEMA peut également créer des tables et des vues dans le nouveau schéma et établir des autorisations GRANT, DENY ou REVOKE sur ces objets.

Exemple 2 : Création du schéma Gescli dans la base **Ventes** de propriétaire **Gestion** ; création de la table Clients pour laquelle l'autorisation Select à est accordée à Marketing, et est refusée à Stock.

```
USE Ventes;
CREATE SCHEMA Gescli AUTHORIZATION Gestion
CREATE TABLE Clients (codecli int, nomcli char(30))
GRANT SELECT TO Marketing
DENY SELECT TO Stock
```

Sous SQL Server Management Studio :

La création d'un schéma se fait en sélectionnant l'option **Nouveau schéma** dans le menu contextuel du nœud **Sécurité** de la base de données sélectionnée.

Sur la page **Général**, seront saisis le nom du schéma et son propriétaire (guidé en affichant la boite de dialogue rechercher les rôles et les utilisateurs, la sélection).

Le déplacement d'un objet vers un nouveau schéma se fait en sélectionnant l'option **Modifier** dans le menu contextuel de l'objet à déplacer. La vue **Propriétés** de l'objet sélectionné s'affiche (F4 si elle n'est pas visible).

Sous **Identité**, la liste déroulante Schémas affiche tous les schémas de la base, et permet de sélectionner le schéma cible.

Déplacement d'objet

Les objets peuvent être déplacés d'un schéma à un autre, mais uniquement dans la même base de données.

La modification d'un schéma par l'instruction **ALTER SCHEMA** consiste à modifier les objets contenus dans ce schéma ; lorsqu'un objet est transféré d'un schéma à un autre, les autorisations relatives à cet objet sont perdues.
Si le propriétaire de l'objet est un utilisateur ou un rôle spécifique, l'autorisation est conservée ; si la propriété de l'objet est définie à SCHEMA OWNER, cette propriété reste inchangée avec comme effet, l'affectation du propriétaire du schéma comme propriétaire de l'objet.

Syntaxe sous Transact-SQL

ALTER SCHEMA *nom_schéma_cible* TRANSFER *nom_objet*

Exemple 3 : Transfert de la table Société qui appartenait au schéma dbo vers le schéma Gescli

ALTER SCHEMA Gescli TRANSFER dbo.Société

Suppression d'un schéma

La suppression d'un schéma par l'instruction **DROP SCHEMA** ne pourra s'effectuer que si ce schéma ne contient plus aucun objet.

A – SUPPRESSION D'OBJETS

Cette situation est utile lorsqu'on utilise un script de création, qu'on peut être amené à relancer plusieurs fois.
Pour éviter des retours d'erreur, il convient de supprimer l'objet qui existe avant de le recréer avec les modifications apportées.

if object_id (*'nom-objet'***, 'type') is not null** **DROP OBJET** *nom-objet*

Le **type** est un code du système SQL Server :

U = table utilisateur
V = vue
C = contrainte Check
F = contrainte Foreign Key
PK = contrainte Primary key
D = contrainte Default
UQ = contrainte Unique
P = procedure stockée
FN = fonction scalaire
IF = fonction table

DROP OBJET est selon le cas :

DROP TABLE
DROP VIEW
DROP INDEX
DROP CONSTRAINT
DROP DEFAULT
DROP PROCEDURE

Exemple :

Supprimer la table dbo.client

if object_id ('dbo.client**,'U') is not null** **DROP TABLE** dbo.client

Pour supprimer la contrainte Foreign Key *fk_alpha* de la table *client*, on pourra procéder ainsi :

if object_id ('dbo.fk_alpha**,'F') is not null**
alter table client **DROP CONSTRAINT** fk_alpha

B – Les types de données

Type SQL Server		Nb Octets	Plage
Nombres entiers			
	bigint	8	-2^{63} à $2^{63}-1$
	int	4	-2^{31} à $2^{31}-1$
	smallint	2	-2^{15} (32768) à $2^{15}-1$(32767)
	tinyint	1	0 à 255
	bit	1	0 ou 1
Valeurs numériques exactes (*)			
	decimal[(p[,s])]	5 à 17	-10^{38} à $10^{38}-1$
	numeric[(p[,s])]	5 à 17	-10^{38} à $10^{38}-1$
Valeurs numériques approximatives			
	float[(n)] n<=24	4	-1,18E -38 à 3,40E +38
	float[(n)] 24<n<=53	8	-1,79E + 308 à 3,40E +308
	real	4	-3,40E + 38 à 3,40E + 38
Valeurs binaires			
	binary[(n)]	1 à 8000	
	varbinary[(n)]		
	varbinary[(max)]	jusqu'à $2^{31}-1$ (+ pointeur de 2 octets)	
	image	0 à $2^{31}-1$	
	uniqueidentifier	16	
Nombres			
	rowversion	8	Nombre unique
	timestamp	8	(synonyme de rowversion)
Date et heure			
	datetime	8 (2 x 4 octets)	01/01/1753 au 31/12/9999
	smalldatetime	4 (2 x 2 octets)	01/01/1960 au 06/06/2079
Monétaire			
	money	8	
	smallmoney	4	
Chaînes de caractères (1 octet par caractère)			
	char[(n)]	n = 1 à 8000 caractères	
	varchar[(n)]	n = 1 à 8000 caractères	
	varchar[(max)]	jusqu'à $2^{31}-1$ (+ pointeur de 2 octets)	
	text	0 à $2^{30}-1$ caractères	
Chaînes de caractères Unicode (2 octets par caractère)			
	nchar[(n)]	n = 1 à 4000 caractères	
	nvarchar[(n)]	n = 1 à 4000	
	nvarchar[(max)]	jusqu'à $2^{31}-1$ (+ pointeur de 2 octets)	

(*****) decimal(5,2) ou numeric(5,2) décrivent un nombre de 5 chiffres dont 2 décimales.

Types particuliers

- **Sqlvariant** (taille variable) : permet à une seule colonne de stocker plusieurs types de données
- **table** (taille variable) : utilisé pour stocker temporairement un ensemble de résultats pour traitement
- **Xml** (taille variable) : utilisé pour stocker des données XML

Les types de données numériques exacts permettent de spécifier exactement l'échelle et le degré de précision à utiliser (ex : 2 décimales).

Dans le cas des types de données numériques approximatives, les données sont stockées aussi précisément que possible à l'aide de nombre binaires (ex : fraction 1/3 = 0,3333…)

Choix des types de données :

- Longueur fixe lorsque la taille des données est cohérente
- Longueur variable lorsque la taille des données n'est pas constante
- Longueur maximale lorsque la taille des données excède la limite de longueur fixe ou longueur variable

Les caractères Unicode utilisent 2 octets par caractère et prennent en charge tous les caractères internationaux.

Les types de données utilisateur

Ils permettent d'affiner les types de données afin d'assurer la cohérence entre les données communes à plusieurs tables. Il est possible de leur associer des règles et des valeurs par défaut. Un type de données utilisateur est déterminé pour une base spécifique.

L'instruction Transact-SQL CREATE TYPE (remplace la procédure stockée système **sp_addtype** de SQL Server 2000) permet de créer ces types de données :

Syntaxe sous Transact-SQL :

CREATE TYPE [nom_schema.] *type_name*
{
FROM *type_système*[(*precision* [, *échelle*])]
[NULL | NOT NULL]| EXTERNAL NAME *nom_assembly* [.*nom_classe*]
}

Exemple 1 : création d'un type zipcode (code postal) de type caractère de 10 caractères n'autorisant pas les valeurs nulles

CREATE TYPE Gescli.zipcode FROM char(10) NOT NULL

Exemple 2 : création d'un type num_membre, nombre entier ne dépassant pas 30000

CREATE TYPE Gescli.num_membre FROM smallint

C – Critères de classement

Le critère de classement précise un indicateur de classement.

Le classement donne l'ordre des caractères à utiliser lorsque le tri par dictionnaire est utile.

- Pour les chaînes ASCII, il influence également la page de codes utilisée pour stocker les caractères.
- Pour les chaînes UNICODE, le classement ne contrôle que les règles de comparaison et la sensibilité à la casse.

Les chaînes **ASCII** correspondent aux types : char, varchar ou text.
Chaque caractère est codé sur 8 bits :

- De 0 à 127, les caractères sont identiques quelque soit le pays
- De 128 à 255 les caractères dépendent de la page de code utilisée pour représenter les caractères du pays.

Certains alphabets (japonais, coréen …) ne peuvent être codés en ASCII.

Les chaînes **UNICODE** correspondent aux types : nchar, nvarchar et ntext.
Chaque caractère est codé sur 16 bits.
Le codage UNICODE reprend les 128 premiers codes ASCII et attributs valables pour tous les pays. Il permet en plus de coder d'autres alphabets.

Le critère de classement précise aussi les styles de comparaison :

- Comparaison binaire
- Comparaison par dictionnaire, sensible aux accents et à la casse (majuscules …)

La clause COLLATE

Cette clause peut s'utiliser lors de la création de la base ou comme opérateur de conversion entre deux chaînes de caractères.

COLLATE ***SQL_collation_name***

Où :

<SQL_collation_name> :: = SQL_*SortRules*[**_Pref**]**_CP***Codepage*_<ComparisonStyle>

<ComparisonStyle> ::= *_CaseSensitivity_AccentSensitivity* | **_BIN**

SortRules

Chaîne identifiant l'alphabet ou la langue dont les règles de tri sont appliquées lorsque le tri du dictionnaire est spécifié. Exemples : Latin1_General ou Polonais.

Pref

Préférence pour les caractères majuscules.

Codepage

Nombre de 1 à 4 chiffres identifiant la page de codes utilisée par le classement. **CP1** indique la page de codes 1252. Le numéro de page de codes complet est indiqué pour toutes les autres pages de codes. Par exemple, **CP1251** correspond à la page de codes 1251 et **CP850** identifie la page de codes 850.

CaseSensitivity

CI ne distingue pas la casse, contrairement à **CS**.

AccentSensitivity

AI ne distingue pas les accents, contrairement à **AS**.

BIN

Indique l'ordre de tri binaire à utiliser.

Exemples de noms de classement Windows :

- **Latin1_General_CI_AS**
  Ce classement utilise les règles de tri du dictionnaire général Latin1, page de codes 1252. Non respect de la casse et respect des accents.

- **Estonian_CS_AS**
  Ce classement utilise les règles de tri du dictionnaire estonien, page de codes 1257. Respect de la casse et des accents.

- **Latin1_General_BIN**
  Ce classement utilise la page de codes 1252 et les règles de tri binaire. Les règles de tri du dictionnaire général Latin1 sont ignorées.

La clause COLLATE peut être spécifiée à plusieurs niveaux, notamment lors de :

- la création ou modification d'une base de données ;

  Vous pouvez utiliser la clause COLLATE de l'instruction CREATE DATABASE ou ALTER DATABASE pour spécifier le classement par défaut de la base de données. Vous pouvez également spécifier un classement lorsque vous créez une base de données à l'aide de SQL Server Management Studio. Si vous ne spécifiez pas de classement, le classement par défaut de l'instance de SQL Server sera appliqué à la base de données.

- la création ou modification d'une colonne dans une table ;

  Vous pouvez spécifier des classements pour chaque colonne de chaîne de caractères à l'aide de la clause COLLATE de l'instruction CREATE TABLE ou ALTER TABLE. Vous pouvez également spécifier un classement lorsque vous créez une table à l'aide de SQL Server Management Studio. Si vous ne spécifiez pas de classement, le classement par défaut de la base de données sera appliqué à la colonne.

  Vous pouvez également utiliser l'option database_default dans la clause COLLATE pour spécifier qu'une colonne d'une table temporaire utilise le classement par défaut de la base de données utilisateur pour la connexion au lieu de la base de données **tempdb**.

- la conversion du classement d'une expression.

  Vous pouvez utiliser la clause COLLATE pour attribuer un classement particulier à une expression de caractère. Le classement par défaut de la base de données active est attribué aux constantes et aux variables de caractères. Le classement des définitions de la colonne est affecté aux références de colonnes.