

# GUIDE COMPLET D'APPRENTISSAGE SQL

## 1. Introduction à SQL

### Qu'est-ce que SQL ?

**SQL (Structured Query Language)** est un langage standardisé pour gérer et manipuler les bases de données relationnelles. Il permet de :

- Créer et modifier la structure des bases de données
- Insérer, modifier et supprimer des données
- Interroger et récupérer des informations
- Contrôler l'accès et la sécurité

### SGBD vs SQL

- **SGBD (Système de Gestion de Base de Données)** : Le logiciel qui stocke et gère les données
  - Exemples : SQL Server, MySQL, PostgreSQL, Oracle, SQLite
- **SQL** : Le langage utilisé pour communiquer avec le SGBD

### Types de SQL

1. **DDL (Data Definition Language)** : Structure des données
  - CREATE, ALTER, DROP
2. **DML (Data Manipulation Language)** : Manipulation des données
  - SELECT, INSERT, UPDATE, DELETE
3. **DCL (Data Control Language)** : Contrôle d'accès
  - GRANT, REVOKE, DENY
4. **TCL (Transaction Control Language)** : Contrôle des transactions
  - BEGIN, COMMIT, ROLLBACK

### Environnements de développement

- **SSMS (SQL Server Management Studio)** : Interface graphique complète
- **Azure Data Studio** : Multiplateforme, moderne
- **sqlcmd** : Interface en ligne de commande
- **Visual Studio Code** avec extensions SQL

## 2. Bases de données et tables

### Créer une base de données



-- Créer une nouvelle base de données  
**CREATE DATABASE BibliothequeDB;**

-- Utiliser la base de données  
**USE BibliothequeDB;**

-- Supprimer une base de données (attention !)  
**DROP DATABASE BibliothequeDB;**

## Créer une table



```
CREATE TABLE Auteurs (
    AuteurID INT IDENTITY(1,1) PRIMARY KEY,
    Nom VARCHAR(50) NOT NULL,
    Prenom VARCHAR(50) NOT NULL,
    DateNaissance DATE,
    Nationalite VARCHAR(30) DEFAULT 'Non spécifiée',
    Email VARCHAR(100) UNIQUE,
    EstActif BIT DEFAULT 1
);
```

## Types de données principaux



-- Types numériques

**INT** -- Nombre entier (-2,147,483,648 à 2,147,483,647)

**BIGINT** -- Grand entier

**DECIMAL(10,2)** -- Nombre décimal (10 chiffres, 2 après virgule)

**FLOAT** -- Nombre à virgule flottante

-- Types texte

**CHAR(10)** -- Chaîne fixe de 10 caractères

**VARCHAR(50)** -- Chaîne variable jusqu'à 50 caractères

**NVARCHAR(MAX)** -- Chaîne Unicode de taille variable

**TEXT** -- Texte long (déprécié, utiliser VARCHAR(MAX))

-- Types date/heure

**DATE** -- Date (YYYY-MM-DD)

**TIME** -- Heure (HH:MM:SS)

**DATETIME** -- Date et heure

**DATETIME2** -- Version améliorée de DATETIME

-- Autres types

**BIT** -- Booléen (0 ou 1)

**UNIQUEIDENTIFIER** -- GUID

**VARBINARY** -- Données binaires

## Contraintes



sql

**CREATE TABLE** Livres (

LivreID **INT IDENTITY(1,1) PRIMARY KEY**, -- Clé primaire auto-incrémentée

Titre **VARCHAR(200) NOT NULL**, -- Obligatoire

ISBN **VARCHAR(13) UNIQUE NOT NULL**, -- Unique et obligatoire

Prix **DECIMAL(8,2) CHECK (Prix > 0)**, -- Vérification de valeur

DatePublication **DATE DEFAULT GETDATE()**, -- Valeur par défaut

AuteurID **INT**,

**FOREIGN KEY** (AuteurID) **REFERENCES** Auteurs(AuteurID) -- Clé étrangère

);

## 3. Manipulation de données (DML)

### **INSERT INTO : Ajouter des données**



-- Insertion simple

```
INSERT INTO Auteurs (Nom, Prenom, DateNaissance, Nationalite)
VALUES ('Hugo', 'Victor', '1802-02-26', 'Française');
```

-- Insertion multiple

```
INSERT INTO Auteurs (Nom, Prenom, Nationalite) VALUES
('Camus', 'Albert', 'Française'),
('Tolkien', 'J.R.R.', 'Britannique'),
('Asimov', 'Isaac', 'Américaine');
```

-- Insertion depuis une autre table

```
INSERT INTO ArchiveAuteurs
SELECT * FROM Auteurs WHERE EstActif = 0;
```

### **SELECT : Lire des données**



-- Sélection basique

```
SELECT * FROM Auteurs;
```

-- Sélection de colonnes spécifiques

```
SELECT Nom, Prenom FROM Auteurs;
```

-- Avec alias

```
SELECT
```

```
    Nom AS NomFamille,
```

```
    Prenom AS PrenomAuteur,
```

```
    CONCAT(Prenom, ' ', Nom) AS NomComplet
```

```
FROM Auteurs;
```

-- Limiter les résultats

```
SELECT TOP 10 * FROM Livres;
```

-- Tri des résultats

```
SELECT * FROM Auteurs
```

```
ORDER BY Nom ASC, Prenom DESC;
```

-- Éliminer les doublons

```
SELECT DISTINCT Nationalite FROM Auteurs;
```

## UPDATE : Modifier des données



sql

-- Modification simple

**UPDATE** Auteurs

**SET** Email = 'victor.hugo@exemple.com'

**WHERE** AuteurID = 1;

-- Modification multiple

**UPDATE** Livres

**SET** Prix = Prix \* 1.1 -- Augmentation de 10%

**WHERE** DatePublication < '2020-01-01';

-- Modification conditionnelle

**UPDATE** Auteurs

**SET** EstActif = 0

**WHERE** DateNaissance < '1900-01-01';

## DELETE : Supprimer des données



-- Suppression avec condition

**DELETE FROM** Auteurs

**WHERE** EstActif = 0;

-- Suppression avec jointure

**DELETE L**

**FROM** Livres L

**INNER JOIN** Auteurs A **ON** L.AuteurID = A.AuteurID

**WHERE** A.Nationalite = 'Inconnue';

-- Vider une table (plus rapide que DELETE)

**TRUNCATE TABLE** LogConnexions;

## 4. Filtres et conditions

### WHERE : Filtrer les résultats



-- Conditions simples

```
SELECT * FROM Livres WHERE Prix > 20;
```

```
SELECT * FROM Auteurs WHERE Nationalite = 'Française';
```

-- Opérateurs de comparaison

```
SELECT * FROM Livres WHERE Prix >= 15 AND Prix <= 50;
```

```
SELECT * FROM Livres WHERE Prix BETWEEN 15 AND 50; -- Équivalent
```

-- IN : Valeurs multiples

```
SELECT * FROM Auteurs
```

```
WHERE Nationalite IN ('Française', 'Britannique', 'Américaine');
```

-- LIKE : Recherche de motifs

```
SELECT * FROM Auteurs WHERE Nom LIKE 'Hug%'; -- Commence par "Hug"
```

```
SELECT * FROM Auteurs WHERE Nom LIKE '%man'; -- Finit par "man"
```

```
SELECT * FROM Auteurs WHERE Nom LIKE '%ar%'; -- Contient "ar"
```

```
SELECT * FROM Auteurs WHERE Nom LIKE '_ugo'; -- 4 lettres, finit par "ugo"
```

-- IS NULL / IS NOT NULL

```
SELECT * FROM Auteurs WHERE Email IS NULL;
```

```
SELECT * FROM Auteurs WHERE Email IS NOT NULL;
```

-- Combinaisons logiques

```
SELECT * FROM Livres
```

```
WHERE (Prix > 20 OR DatePublication > '2020-01-01')
```

```
AND AuteurID IS NOT NULL;
```

-- NOT : Négation

```
SELECT * FROM Auteurs
```

```
WHERE NOT (Nationalite = 'Française' OR Nationalite = 'Britannique');
```

## 5. Fonctions SQL

### Fonctions d'agrégation



-- Comptage

```
SELECT COUNT(*) AS NombreTotal FROM Livres;  
SELECT COUNT>Email) AS AuteursAvecEmail FROM Auteurs; -- Ignore les NULL  
SELECT COUNT(DISTINCT Nationalite) AS NationalitesDifferentes FROM Auteurs;
```

-- Calculs numériques

```
SELECT  
    SUM(Prix) AS PrixTotal,  
    AVG(Prix) AS PrixMoyen,  
    MIN(Prix) AS PrixMinimum,  
    MAX(Prix) AS PrixMaximum  
FROM Livres;
```

-- Avec groupement

```
SELECT  
    Nationalite,  
    COUNT(*) AS NombreAuteurs,  
    AVG(DATEDIFF(YEAR, DateNaissance, GETDATE())) AS AgeMoyen  
FROM Auteurs  
GROUP BY Nationalite;
```

## Fonctions de chaîne



sql

**SELECT**

Nom,  
**LEN(Nom) AS LongueurNom,** -- *Longueur*  
**UPPER(Nom) AS NomMajuscule,** -- *Majuscules*  
**LOWER(Nom) AS NomMinuscule,** -- *Minuscules*  
**SUBSTRING(Nom, 1, 3) AS TroisPremieresLettres,** -- *Sous-chaine*  
**LEFT(Nom, 2) AS DeuxPremieresLettres,** -- *Gauche*  
**RIGHT(Nom, 2) AS DeuxDernieresLettres,** -- *Droite*  
**REVERSE(Nom) AS NomInverse,** -- *Inverse*  
**REPLACE(Nom, 'a', '@') AS NomModifie,** -- *Remplacement*  
**LTRIM(RTRIM(' ' + Nom + ' ')) AS NomSansEspaces** -- *Suppression espaces*  
**FROM Auteurs;**

-- *Concaténation*

**SELECT**

**CONCAT(Prenom, ' ', Nom) AS NomComplet,**  
**Prenom + ' ' + Nom AS NomComplet2** -- *Alternative*  
**FROM Auteurs;**

## Fonctions de date



**SELECT**

**GETDATE() AS DateActuelle,** -- *Date/heure actuelle*  
**GETUTCDATE() AS DateUTC,** -- *Date/heure UTC*  
**DATEADD(YEAR, 1, GETDATE()) AS DansUnAn,** -- *Ajouter du temps*  
**DATEADD(MONTH, -6, GETDATE()) AS IlYa6Mois,**  
**DATEDIFF(YEAR, DateNaissance, GETDATE()) AS Age,** -- *Différence*  
**DATEDIFF(DAY, DatePublication, GETDATE()) AS JoursDepuisPublication,**  
**YEAR(DateNaissance) AS AnneeNaissance,** -- *Extraire l'année*  
**MONTH(DateNaissance) AS MoisNaissance,** -- *Extraire le mois*  
**DAY(DateNaissance) AS JourNaissance,** -- *Extraire le jour*  
**DATENAME(WEEKDAY, DateNaissance) AS JourSemaine,** -- *Nom du jour*  
**FORMAT(DateNaissance, 'dd/MM/yyyy') AS DateFormatee** -- *Formatage*  
**FROM Auteurs;**

## Fonctions conditionnelles



-- CASE WHEN

**SELECT**

Nom,

Prix,

**CASE**

WHEN Prix < 20 THEN 'Bon marché'

WHEN Prix BETWEEN 20 AND 50 THEN 'Prix moyen'

ELSE 'Cher'

END AS CategoriePrix

**FROM** Livres;

-- IIF (SQL Server 2012+)

**SELECT**

Nom,

IIF(EstActif = 1, 'Actif', 'Inactif') **AS** Statut

**FROM** Auteurs;

-- ISNULL / COALESCE

**SELECT**

Nom,

ISNULL>Email, 'Pas d"email') **AS** EmailOuDefaut,

COALESCE>Email, 'inconnu@example.com') **AS** EmailAvecDefaut

**FROM** Auteurs;

---

## 6. Jointures (JOINS)

### Préparation des données d'exemple



-- Créons quelques tables pour les exemples

```
CREATE TABLE Categories (
    CategorieID INT PRIMARY KEY,
    NomCategorie VARCHAR(50)
);
```

```
CREATE TABLE Emprunts (
    EmpruntID INT IDENTITY(1,1) PRIMARY KEY,
    LivreID INT,
    NomEmprunteur VARCHAR(100),
    DateEmprunt DATE,
    DateRetour DATE
);
```

-- Données d'exemple

```
INSERT INTO Categories VALUES (1, 'Fiction'), (2, 'Science'), (3, 'Histoire');
INSERT INTO Emprunts (LivreID, NomEmprunteur, DateEmprunt) VALUES
(1, 'Marie Dubois', '2024-01-15'),
(2, 'Jean Martin', '2024-01-20');
```

## INNER JOIN



sql

-- Jointure interne : seulement les enregistrements qui correspondent

```
SELECT
    L.Titre,
    A.Nom,
    A.Prenom,
    C.NomCategorie
FROM Livres L
INNER JOIN Auteurs A ON L.AuteurID = A.AuteurID
INNER JOIN Categories C ON L.CategorieID = C.CategorieID;
```

## LEFT JOIN



sql

-- Jointure gauche : tous les auteurs, même sans livres

**SELECT**

```
A.Nom,  
A.Prenom,  
L.Titre,  
ISNULL(L.Titre, 'Aucun livre') AS TitreOuMessage
```

**FROM** Auteurs A

**LEFT JOIN** Livres L **ON** A.AuteurID = L.AuteurID

**ORDER BY** A.Nom;

## RIGHT JOIN



-- Jointure droite : tous les livres, même sans auteur

**SELECT**

```
L.Titre,  
A.Nom,  
A.Prenom  
FROM Auteurs A  
RIGHT JOIN Livres L ON A.AuteurID = L.AuteurID;
```

## FULL OUTER JOIN



-- Jointure complète : tous les enregistrements des deux tables

**SELECT**

```
A.Nom,  
L.Titre  
FROM Auteurs A  
FULL OUTER JOIN Livres L ON A.AuteurID = L.AuteurID;
```

## Jointures multiples



sql

```
-- Exemple complexe avec plusieurs jointures
SELECT
    A.Nom + ',' + A.Prenom AS AuteurComplet,
    L.Titre,
    C.NomCategorie,
    E.NomEmprunteur,
    E.DateEmprunt,
    CASE
        WHEN E.DateRetour IS NULL THEN 'En cours'
        ELSE 'Retourné'
    END AS StatutEmprunt
FROM Livres L
INNER JOIN Auteurs A ON L.AuteurID = A.AuteurID
LEFT JOIN Categories C ON L.CategorieID = C.CategorieID
LEFT JOIN Emprunts E ON L.LivreID = E.LivreID
ORDER BY A.Nom, L.Titre;
```

## Auto-jointure



sql

```
-- Exemple : trouver les auteurs de même nationalité
SELECT
    A1.Nom + ' ' + A1.Prenom AS Auteur1,
    A2.Nom + ' ' + A2.Prenom AS Auteur2,
    A1.Nationalite
FROM Auteurs A1
INNER JOIN Auteurs A2 ON A1.Nationalite = A2.Nationalite
    AND A1.AuteurID < A2.AuteurID -- Éviter les doublons
ORDER BY A1.Nationalite;
```

---

## 7. Groupements et regroupement de données

### GROUP BY



sql

-- Groupement simple

**SELECT**

Nationalite,

**COUNT(\*) AS NombreAuteurs**

**FROM Auteurs**

**WHERE EstActif = 1**

**GROUP BY Nationalite**

**ORDER BY NombreAuteurs DESC;**

-- Groupement multiple

**SELECT**

**YEAR(DatePublication) AS Annee,**

**COUNT(\*) AS NombreLivres,**

**AVG(Prix) AS PrixMoyen,**

**MIN(Prix) AS PrixMin,**

**MAX(Prix) AS PrixMax**

**FROM Livres**

**GROUP BY YEAR(DatePublication)**

**ORDER BY Annee;**

## HAVING



sql

-- HAVING : filtrer après groupement (WHERE ne fonctionne pas avec les fonctions d'agrégation)

**SELECT**

Nationalite,

**COUNT(\*) AS NombreAuteurs,**

**AVG(DATEDIFF(YEAR, DateNaissance, GETDATE())) AS AgeMoyen**

**FROM Auteurs**

**GROUP BY Nationalite**

**HAVING COUNT(\*) >= 2 -- Seulement les nationalités avec au moins 2 auteurs**

**AND AVG(DATEDIFF(YEAR, DateNaissance, GETDATE())) < 80**

**ORDER BY NombreAuteurs DESC;**

## Sous-requêtes (Subqueries)



sql

-- Sous-requête dans WHERE

```
SELECT *
FROM Livres
WHERE Prix > (SELECT AVG(Prix) FROM Livres);
```

-- Sous-requête dans FROM

```
SELECT
    Categorie,
    NombreLivres,
    CASE
        WHEN NombreLivres > MoyenneLivresParCategorie THEN 'Au-dessus'
        ELSE 'En dessous ou égal'
    END AS ComparaisonMoyenne
FROM (
    SELECT
        C.NomCategorie AS Categorie,
        COUNT(L.LivreID) AS NombreLivres,
        (SELECT COUNT(*) * 1.0 / COUNT(DISTINCT CategorieID) FROM Livres) AS MoyenneLivresParCategorie
    FROM Categories C
    LEFT JOIN Livres L ON C.CategorieID = L.CategorieID
    GROUP BY C.CategorieID, C.NomCategorie
) AS StatistiquesCategories;
```

-- Sous-requête corrélée

```
SELECT
    A.Nom,
    A.Prenom,
    (SELECT COUNT(*) FROM Livres L WHERE L.AuteurID = A.AuteurID) AS NombreLivres
FROM Auteurs A;
```

-- EXISTS

```
SELECT *
FROM Auteurs A
WHERE EXISTS (
    SELECT 1 FROM Livres L WHERE L.AuteurID = A.AuteurID AND L.Prix > 50
);
```

-- IN avec sous-requête

```
SELECT *
FROM Livres
WHERE AuteurID IN (
```

```
SELECT AuteurID FROM Auteurs WHERE Nationalite = 'Française'
```

);

## CTE (Common Table Expressions)



sql

```
-- CTE simple
WITH AuteursProductifs AS (
    SELECT
        AuteurID,
        COUNT(*) AS NombreLivres
    FROM Livres
    GROUP BY AuteurID
    HAVING COUNT(*) > 1
)
SELECT
    A.Nom,
    A.Prenom,
    AP.NombreLivres
FROM Auteurs A
INNER JOIN AuteursProductifs AP ON A.AuteurID = AP.AuteurID;
```

```
-- CTE récursive (exemple : hiérarchie)
WITH HierarchieCategories AS (
    -- Ancre : catégories de niveau 0
    SELECT CategorieID, NomCategorie, ParentID, 0 AS Niveau
    FROM Categories
    WHERE ParentID IS NULL

    UNION ALL

    -- Partie récursive
    SELECT C.CategorieID, C.NomCategorie, C.ParentID, H.Niveau + 1
    FROM Categories C
    INNER JOIN HierarchieCategories H ON C.ParentID = H.CategorieID
)
SELECT * FROM HierarchieCategories;
```

## 8. Index, vues et performances

### Index



sql

-- Créer un index simple

```
CREATE INDEX IX_Livres_Titre ON Livres(Titre);
```

-- Index composé

```
CREATE INDEX IX_Livres_Auteur_Prix ON Livres(AuteurID, Prix);
```

-- Index unique

```
CREATE UNIQUE INDEX IX_Auteurs_Email ON Auteurs>Email);
```

-- Index avec conditions

```
CREATE INDEX IX_Livres_Prix_Chers ON Livres(Prix)
```

```
WHERE Prix > 50;
```

-- Supprimer un index

```
DROP INDEX IX_Livres_Titre ON Livres;
```

-- Voir les index d'une table

```
SELECT
```

```
i.name AS NomIndex,  
i.type_desc AS TypeIndex,  
c.name AS Colonne
```

```
FROM sys.indexes i
```

```
INNER JOIN sys.index_columns ic ON i.object_id = ic.object_id AND i.index_id = ic.index_id
```

```
INNER JOIN sys.columns c ON ic.object_id = c.object_id AND ic.column_id = c.column_id
```

```
WHERE i.object_id = OBJECT_ID('Livres');
```

## Avantages et inconvénients des index

### Avantages :

- Accélèrent les requêtes SELECT avec WHERE, JOIN, ORDER BY
- Améliorent les performances des contraintes UNIQUE et PRIMARY KEY
- Peuvent forcer l'unicité des données

### Inconvénients :

- Ralentissent les opérations INSERT, UPDATE, DELETE
- Consomment de l'espace disque supplémentaire
- Nécessitent de la maintenance

## Vues (Views)





-- Créer une vue simple

CREATE VIEW VueAuteursActifs AS

SELECT

AuteurID,

Nom,

Prenom,

Nationalite,

Email

FROM Auteurs

WHERE EstActif = 1;

-- Utiliser la vue

SELECT \* FROM VueAuteursActifs WHERE Nationalite = 'Française';

-- Vue complexe avec jointures

CREATE VIEW VueLivresDetailles AS

SELECT

L.LivreID,

L.Titre,

L.ISBN,

L.Prix,

A.Nom + ' ' + A.Prenom AS Auteur,

A.Nationalite,

C.NomCategorie,

CASE

WHEN E.EmpruntID IS NOT NULL AND E.DateRetour IS NULL THEN 'Emprunté'

ELSE 'Disponible'

END AS Statut

FROM Livres L

INNER JOIN Auteurs A ON L.AuteurID = A.AuteurID

LEFT JOIN Categories C ON L.CategorieID = C.CategorieID

LEFT JOIN Emprunts E ON L.LivreID = E.LivreID AND E.DateRetour IS NULL;

-- Modifier une vue

ALTER VIEW VueAuteursActifs AS

SELECT

AuteurID,

Nom,

Prenom,

Nationalite,

Email,

```
DATEDIFF(YEAR, DateNaissance, GETDATE()) AS Age  
FROM Auteurs  
WHERE EstActif = 1;
```

-- Supprimer une vue  
**DROP VIEW** VueAuteursActifs;

## Optimisation des requêtes



-- Afficher le plan d'exécution

```
SET SHOWPLAN_ALL ON;  
SELECT * FROM Livres WHERE Prix > 20;  
SET SHOWPLAN_ALL OFF;
```

-- Statistiques d'E/S

```
SET STATISTICS IO ON;  
SELECT * FROM Livres L  
INNER JOIN Auteurs A ON L.AuteurID = A.AuteurID;  
SET STATISTICS IO OFF;
```

-- Statistiques de temps

```
SET STATISTICS TIME ON;  
SELECT COUNT(*) FROM Livres;  
SET STATISTICS TIME OFF;
```

-- Conseils d'optimisation

- 1. Utiliser des index appropriés
- 2. Éviter **SELECT \***
- 3. Utiliser **WHERE** pour limiter les résultats
- 4. Préférer **EXISTS** à **IN** pour les sous-requêtes
- 5. Utiliser les jointures plutôt que les sous-requêtes corrélées quand possible

## 9. Transactions et contrôle

### Transactions de base



-- Transaction simple

BEGIN TRANSACTION;

UPDATE Livres SET Prix = Prix \* 1.1 WHERE CategorieID = 1;

INSERT INTO LogModifications (Table\_Modifiee, Action, DateModification)

VALUES ('Livres', 'Augmentation prix fiction', GETDATE());

COMMIT TRANSACTION; -- Confirmer les changements

-- OU

-- ROLLBACK TRANSACTION; -- Annuler les changements

### Gestion d'erreurs avec TRY-CATCH



sql

```
BEGIN TRANSACTION;

BEGIN TRY
    -- Opérations risquées
    INSERT INTO Auteurs (Nom, Prenom, Email)
    VALUES ('Nouveau', 'Auteur', 'email@existant.com'); -- Peut violer UNIQUE

    UPDATE Livres SET Prix = -10 WHERE LivreID = 1; -- Peut violer CHECK

    -- Si tout va bien, confirmer
    COMMIT TRANSACTION;
    PRINT 'Transaction réussie';

END TRY
BEGIN CATCH
    -- En cas d'erreur, annuler
    ROLLBACK TRANSACTION;

    -- Afficher l'erreur
    PRINT 'Erreur survenue : ' + ERROR_MESSAGE();
    PRINT 'Numéro erreur : ' + CAST(ERROR_NUMBER() AS VARCHAR);
    PRINT 'Ligne : ' + CAST(ERROR_LINE() AS VARCHAR);

END CATCH;
```

## Points de sauvegarde



sql

**BEGIN TRANSACTION;**

**INSERT INTO** Auteurs (Nom, Prenom) **VALUES** ('Test1', 'Auteur1');

**SAVE TRANSACTION** Point1; -- *Point de sauvegarde*

**INSERT INTO** Auteurs (Nom, Prenom) **VALUES** ('Test2', 'Auteur2');

-- *Quelque chose se passe mal...*

**ROLLBACK TRANSACTION** Point1; -- *Revenir au point de sauvegarde*

**INSERT INTO** Auteurs (Nom, Prenom) **VALUES** ('Test3', 'Auteur3');

**COMMIT TRANSACTION;**

## Niveaux d'isolation



-- READ UNCOMMITTED : Lecture des données non validées  
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;  
SELECT \* FROM Livres;

-- READ COMMITTED : Lecture des données validées uniquement (défaut)  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  
SELECT \* FROM Livres;

-- REPEATABLE READ : Lecture répétable  
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
BEGIN TRANSACTION;  
SELECT \* FROM Livres WHERE Prix > 20; -- Première lecture  
-- ... autres opérations ...  
SELECT \* FROM Livres WHERE Prix > 20; -- Même résultat garanti  
COMMIT;

-- SERIALIZABLE : Isolation complète  
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
BEGIN TRANSACTION;  
SELECT \* FROM Livres; -- Aucune modification possible par d'autres sessions  
COMMIT;

-- SNAPSHOT : Isolation par versioning  
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;  
SELECT \* FROM Livres;

## Verrous explicites



sql

-- Verrou partagé (SHARED)

```
SELECT * FROM Livres WITH (HOLDLOCK, ROWLOCK) WHERE LivreID = 1;
```

-- Verrou exclusif

```
SELECT * FROM Livres WITH (XLOCK) WHERE LivreID = 1;
```

-- Verrou de mise à jour

```
SELECT * FROM Livres WITH (UPDLOCK) WHERE Prix < 20;
```

-- Pas de verrous (risqué)

```
SELECT * FROM Livres WITH (NOLOCK);
```

---

## 10. Sécurité et utilisateurs

### Créer des connexions et utilisateurs



sql

-- Créer une connexion SQL Server

```
CREATE LOGIN BiblioUser  
WITH PASSWORD = 'MotDePasse123!',  
    DEFAULT_DATABASE = BibliothequeDB,  
    CHECK_EXPIRATION = ON,  
    CHECK_POLICY = ON;
```

-- Créer un utilisateur dans la base de données

```
USE BibliothequeDB;  
CREATE USER BiblioUser FOR LOGIN BiblioUser;
```

-- Connexion Windows

```
CREATE LOGIN [DOMAINE\UtilisateurWindows] FROM WINDOWS;  
CREATE USER UtilisateurWindows FOR LOGIN [DOMAINE\UtilisateurWindows];
```

### Rôles de base de données



sql

-- Créer un rôle personnalisé

**CREATE ROLE LecteurBibliotheque;**

-- Ajouter un utilisateur à un rôle

**ALTER ROLE LecteurBibliotheque ADD MEMBER BiblioUser;**

-- OU

**EXEC sp\_addrolemember 'LecteurBibliotheque', 'BiblioUser';**

-- Rôles prédéfinis utiles

**ALTER ROLE db\_datareader ADD MEMBER BiblioUser;**

-- Lecture seule

**ALTER ROLE db\_datawriter ADD MEMBER BiblioUser;**

-- Écriture

**ALTER ROLE db\_ddladmin ADD MEMBER BiblioUser;**

-- DDL (CREATE, ALTER, DROP)

**ALTER ROLE db\_owner ADD MEMBER BiblioUser;**

-- Propriétaire (tous droits)

## Permissions GRANT, REVOKE, DENY



sql

-- GRANT : Accorder des permissions

GRANT SELECT ON Auteurs TO BiblioUser;

GRANT SELECT, INSERT, UPDATE ON Livres TO LecteurBibliotheque;

GRANT EXECUTE ON SCHEMA::dbo TO BiblioUser; -- Exécuter toutes les procédures

-- Permissions spécifiques sur colonnes

GRANT SELECT (Nom, Prenom) ON Auteurs TO LecteurBibliotheque;

GRANT UPDATE (Prix) ON Livres TO BiblioUser;

-- REVOKE : Révoquer des permissions

REVOKE INSERT ON Livres FROM BiblioUser;

REVOKE ALL ON Auteurs FROM LecteurBibliotheque;

-- DENY : Interdire explicitement (plus fort que GRANT)

DENY DELETE ON Auteurs TO BiblioUser;

DENY SELECT ON sys.tables TO LecteurBibliotheque; -- Interdire la lecture des métadonnées

-- Voir les permissions

SELECT

p.permission\_name,

p.state\_desc,

pr.name AS principal\_name,

o.name AS object\_name

FROM sys.database\_permissions p

LEFT JOIN sys.objects o ON p.major\_id = o.object\_id

LEFT JOIN sys.database\_principals pr ON p.grantee\_principal\_id = pr.principal\_id

WHERE pr.name = 'BiblioUser';

## Schémas et sécurité



sql

-- Créer un schéma

**CREATE SCHEMA** Gestion **AUTHORIZATION** BiblioUser;

-- Créer des objets dans le schéma

**CREATE TABLE** Gestion.Statistiques (

StatID **INT IDENTITY(1,1) PRIMARY KEY**,  
NomStat **VARCHAR(100)**,  
Valeur **INT**,  
DateCalcul **DATETIME DEFAULT GETDATE()**

);

-- Accorder des permissions sur le schéma

**GRANT SELECT, INSERT ON SCHEMA::Gestion TO LecteurBibliotheque;**

-- Transférer la propriété d'un schéma

**ALTER AUTHORIZATION ON SCHEMA::Gestion TO dbo;**

## Audit et surveillance



sql

```
-- Activer l'audit SQL Server (niveau serveur - nécessite des droits sysadmin)
CREATE SERVER AUDIT BibliothequeAudit
```

```
TO FILE (FILEPATH = 'C:\Audit\')
```

```
WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE);
```

```
-- Spécification d'audit pour la base de données
```

```
CREATE DATABASE AUDIT SPECIFICATION BibliothequeAuditSpec
```

```
FOR SERVER AUDIT BibliothequeAudit
```

```
ADD (SELECT, INSERT, UPDATE, DELETE ON Livres BY PUBLIC),
```

```
ADD (SELECT ON Auteurs BY PUBLIC);
```

```
-- Activer l'audit
```

```
ALTER SERVER AUDIT BibliothequeAudit WITH (STATE = ON);
```

```
ALTER DATABASE AUDIT SPECIFICATION BibliothequeAuditSpec WITH (STATE = ON);
```

```
-- Trigger d'audit personnalisé
```

```
CREATE TRIGGER TR_Livres_Audit ON Livres
```

```
AFTER INSERT, UPDATE, DELETE
```

```
AS
```

```
BEGIN
```

```
    INSERT INTO LogAudit (TableName, Action, UserName, Date, OldValues, NewValues)
```

```
    SELECT
```

```
        'Livres',
```

```
        CASE
```

```
            WHEN EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted) THEN 'UPDATE'
```

```
            WHEN EXISTS(SELECT * FROM inserted) THEN 'INSERT'
```

```
            ELSE 'DELETE'
```

```
        END,
```

```
        SUSER_NAME(),
```

```
        GETDATE(),
```

```
        (SELECT * FROM deleted FOR JSON AUTO),
```

```
        (SELECT * FROM inserted FOR JSON AUTO);
```

```
END;
```

---

## 11. Sauvegarde et restauration (SQL Server)

### Sauvegarde complète



sql

-- Sauvegarde complète de base de données  
**BACKUP DATABASE** BibliothequeDB  
**TO DISK** = 'C:\Backups\BibliothequeDB\_Full.bak'  
**WITH**  
    **DESCRIPTION** = 'Sauvegarde complète de la bibliothèque',  
    **NAME** = 'BibliothequeDB-Full Database Backup',  
    **COMPRESSION**, -- Compression (SQL Server 2008+)  
    **CHECKSUM**, -- Vérification d'intégrité  
    **INIT**; -- Écraser le fichier existant

-- Sauvegarde avec plusieurs fichiers  
**BACKUP DATABASE** BibliothequeDB  
**TO DISK** = 'C:\Backups\BibliothequeDB\_1.bak',  
    **DISK** = 'C:\Backups\BibliothequeDB\_2.bak',  
    **DISK** = 'C:\Backups\BibliothequeDB\_3.bak'  
**WITH COMPRESSION, FORMAT;**

## Sauvegarde différentielle et du journal



-- Sauvegarde différentielle (seulement les changements depuis la dernière complète)  
**BACKUP DATABASE** BibliothequeDB  
**TO DISK** = 'C:\Backups\BibliothequeDB\_Diff.bak'  
**WITH DIFFERENTIAL, COMPRESSION, CHECKSUM;**

-- Sauvegarde du journal de transactions (mode de récupération FULL requis)  
**BACKUP LOG** BibliothequeDB  
**TO DISK** = 'C:\Backups\BibliothequeDB\_Log.trn'  
**WITH COMPRESSION, CHECKSUM;**

-- Changer le mode de récupération  
**ALTER DATABASE** BibliothequeDB **SET RECOVERY FULL;**

## Restauration



sql

-- Restauration simple (base arrêtée)

**RESTORE DATABASE** BibliothequeDB

**FROM DISK** = 'C:\Backups\BibliothequeDB\_Full.bak'

**WITH REPLACE, CHECKSUM;**

-- Restauration avec changement d'emplacement

**RESTORE DATABASE** BibliothequeDB\_Test

**FROM DISK** = 'C:\Backups\BibliothequeDB\_Full.bak'

**WITH**

MOVE 'BibliothequeDB' **TO** 'C:\Data\BibliothequeDB\_Test.mdf,

MOVE 'BibliothequeDB\_Log' **TO** 'C:\Data\BibliothequeDB\_Test.ldf',

**REPLACE;**

-- Restauration point-in-time avec journal

**RESTORE DATABASE** BibliothequeDB

**FROM DISK** = 'C:\Backups\BibliothequeDB\_Full.bak'

**WITH NORECOVERY, REPLACE;**

**RESTORE DATABASE** BibliothequeDB

**FROM DISK** = 'C:\Backups\BibliothequeDB\_Diff.bak'

**WITH NORECOVERY;**

**RESTORE LOG** BibliothequeDB

**FROM DISK** = 'C:\Backups\BibliothequeDB\_Log.trn'

**WITH STOPAT** = '2024-01-15 14:30:00';

-- Voir les informations de sauvegarde

**RESTORE HEADERONLY** **FROM DISK** = 'C:\Backups\BibliothequeDB\_Full.bak';

**RESTORE FILELISTONLY** **FROM DISK** = 'C:\Backups\BibliothequeDB\_Full.bak';

## Scripts d'automatisation



sql

```
-- Procédure stockée de sauvegarde automatisée
CREATE PROCEDURE sp_BackupBibliotheque
    @BackupType VARCHAR(10) = 'FULL' -- FULL, DIFF, LOG
AS
BEGIN
    DECLARE @FileName VARCHAR(255);
    DECLARE @BackupPath VARCHAR(255) = 'C:\Backups\';
    DECLARE @DatabaseName VARCHAR(128) = 'BibliothequeDB';

    -- Générer nom de fichier avec date/heure
    SET @FileName = @BackupPath + @DatabaseName + '_' + @BackupType + '_' +
        CONVERT(VARCHAR, GETDATE(), 112) + '_' +
        REPLACE(CONVERT(VARCHAR, GETDATE(), 108), ':', '') +
        CASE @BackupType
            WHEN 'LOG' THEN '.trn'
            ELSE '.bak'
        END;

    IF @BackupType = 'FULL'
    BEGIN
        BACKUP DATABASE @DatabaseName
        TO DISK = @FileName
        WITH COMPRESSION, CHECKSUM, INIT;
    END
    ELSE IF @BackupType = 'DIFF'
    BEGIN
        BACKUP DATABASE @DatabaseName
        TO DISK = @FileName
        WITH DIFFERENTIAL, COMPRESSION, CHECKSUM, INIT;
    END
    ELSE IF @BackupType = 'LOG'
    BEGIN
        BACKUP LOG @DatabaseName
        TO DISK = @FileName
        WITH COMPRESSION, CHECKSUM, INIT;
    END

    PRINT 'Sauvegarde ' + @BackupType + ' terminée : ' + @FileName;
END;

-- Utilisation
```

```
EXEC sp_BackupBibliotheque 'FULL';
EXEC sp_BackupBibliotheque 'DIFF';
EXEC sp_BackupBibliotheque 'LOG';
```

## Maintenance et vérification



-- Vérifier l'intégrité de la base de données

```
DBCC CHECKDB('BibliothequeDB') WITH NO_INFOMSGS;
```

-- Réorganiser les index

```
ALTER INDEX ALL ON Livres REORGANIZE;
```

-- Reconstruire les index

```
ALTER INDEX ALL ON Livres REBUILD WITH (ONLINE = OFF);
```

-- Mettre à jour les statistiques

```
UPDATE STATISTICS Livres;
```

-- Réduire la taille de la base de données (à utiliser avec précaution)

```
DBCC SHRINKDATABASE('BibliothequeDB', 10); -- Garder 10% d'espace libre
```

---

## 12. Projets pratiques

### Projet 1 : Système de gestion de bibliothèque

#### Structure complète de la base de données



-- Base de données

CREATE DATABASE BibliothequeComplete;

USE BibliothequeComplete;

-- Table des auteurs

CREATE TABLE Auteurs (

AuteurID INT IDENTITY(1,1) PRIMARY KEY,  
Nom VARCHAR(50) NOT NULL,  
Prenom VARCHAR(50) NOT NULL,  
DateNaissance DATE,  
DateDeces DATE,  
Nationalite VARCHAR(30),  
Biographie TEXT,  
Email VARCHAR(100) UNIQUE,  
EstActif BIT DEFAULT 1,  
DateCreation DATETIME DEFAULT GETDATE(),  
DateModification DATETIME DEFAULT GETDATE()  
);

-- Table des catégories

CREATE TABLE Categories (

CategorieID INT IDENTITY(1,1) PRIMARY KEY,  
NomCategorie VARCHAR(50) NOT NULL UNIQUE,  
Description TEXT,  
CategorieParentID INT,  
FOREIGN KEY (CategorieParentID) REFERENCES Categories(CategorieID)  
);

-- Table des éditeurs

CREATE TABLE Editeurs (

EditeurID INT IDENTITY(1,1) PRIMARY KEY,  
NomEditeur VARCHAR(100) NOT NULL,  
Adresse VARCHAR(200),  
Ville VARCHAR(50),  
CodePostal VARCHAR(10),  
Pays VARCHAR(50),  
Telephone VARCHAR(20),  
Email VARCHAR(100),  
SiteWeb VARCHAR(200)  
);

-- Table des livres

**CREATE TABLE** Livres (

LivreID **INT IDENTITY(1,1) PRIMARY KEY**,  
Titre **VARCHAR(200) NOT NULL**,  
ISBN **VARCHAR(13) UNIQUE**,  
DatePublication **DATE**,  
NombrePages **INT CHECK (NombrePages > 0)**,  
Prix **DECIMAL(8,2) CHECK (Prix >= 0)**,  
Quantite **INT DEFAULT 1 CHECK (Quantite >= 0)**,  
CategorieID **INT**,  
EditeurID **INT**,  
Resume **TEXT**,  
LangueOrigine **VARCHAR(30) DEFAULT 'Français'**,  
DateAjout **DATETIME DEFAULT GETDATE()**,  
EstDisponible **BIT DEFAULT 1**,  
**FOREIGN KEY** (CategorieID) **REFERENCES** Categories(CategorieID),  
**FOREIGN KEY** (EditeurID) **REFERENCES** Editeurs(EditeurID)

);

-- Table de liaison auteurs-livres (relation many-to-many)

**CREATE TABLE** AuteursLivres (

AuteurID **INT**,  
LivreID **INT**,  
RoleAuteur **VARCHAR(20) DEFAULT 'Auteur' -- Auteur, Co-auteur, Traducteur, etc.**  
**PRIMARY KEY** (AuteurID, LivreID),  
**FOREIGN KEY** (AuteurID) **REFERENCES** Auteurs(AuteurID),  
**FOREIGN KEY** (LivreID) **REFERENCES** Livres(LivreID)

);

-- Table des membres/utilisateurs

**CREATE TABLE** Membres (

MembreID **INT IDENTITY(1,1) PRIMARY KEY**,  
NumeroMembre **VARCHAR(20) UNIQUE NOT NULL**,  
Nom **VARCHAR(50) NOT NULL**,  
Prenom **VARCHAR(50) NOT NULL**,  
DateNaissance **DATE**,  
Adresse **VARCHAR(200)**,  
Ville **VARCHAR(50)**,  
CodePostal **VARCHAR(10)**,  
Telephone **VARCHAR(20)**,  
Email **VARCHAR(100) UNIQUE**,

```
DateInscription DATETIME DEFAULT GETDATE(),
TypeMembre VARCHAR(20) DEFAULT 'Standard', -- Standard, Premium, Étudiant
EstActif BIT DEFAULT 1,
DateExpirationCarte DATE
);
```

-- Table des emprunts

```
CREATE TABLE Emprunts (
EmpruntID INT IDENTITY(1,1) PRIMARY KEY,
MembreID INT NOT NULL,
LivreID INT NOT NULL,
DateEmprunt DATETIME DEFAULT GETDATE(),
DateRetourPrevue DATE NOT NULL,
DateRetourReel DATETIME,
Statut VARCHAR(20) DEFAULT 'En cours', -- En cours, Retourné, En retard
Commentaires TEXT,
FOREIGN KEY (MembreID) REFERENCES Membres(MembreID),
FOREIGN KEY (LivreID) REFERENCES Livres(LivreID)
);
```

-- Table des réservations

```
CREATE TABLE Reservations (
ReservationID INT IDENTITY(1,1) PRIMARY KEY,
MembreID INT NOT NULL,
LivreID INT NOT NULL,
DateReservation DATETIME DEFAULT GETDATE(),
DateExpiration DATE,
Statut VARCHAR(20) DEFAULT 'Active', -- Active, Annulée, Satisfait
FOREIGN KEY (MembreID) REFERENCES Membres(MembreID),
FOREIGN KEY (LivreID) REFERENCES Livres(LivreID)
);
```

-- Table des amendes

```
CREATE TABLE Amendes (
AmendeID INT IDENTITY(1,1) PRIMARY KEY,
EmpruntID INT NOT NULL,
MontantAmende DECIMAL(6,2) NOT NULL,
DateAmende DATETIME DEFAULT GETDATE(),
Motif VARCHAR(100),
EstPayee BIT DEFAULT 0,
DatePaiement DATETIME,
```

**FOREIGN KEY** (EmpruntID) **REFERENCES** Emprunts(EmpruntID)

);

## Procédures stockées essentielles



sql

-- Procédure d'emprunt

CREATE PROCEDURE sp\_EmprunterLivre

    @MembreID INT,

    @LivreID INT,

    @DureeEmprunt INT = 21 -- 21 jours par défaut

AS

BEGIN

    SET NOCOUNT ON;

    DECLARE @NombreEmpruntsActuels INT;

    DECLARE @LivreDisponible BIT;

    DECLARE @EmpruntID INT;

BEGIN TRY

    BEGIN TRANSACTION;

    -- Vérifier si le membre peut emprunter

    SELECT @NombreEmpruntsActuels = COUNT(\*)

    FROM Emprunts

    WHERE MembreID = @MembreID AND Statut = 'En cours';

    IF @NombreEmpruntsActuels >= 5

        BEGIN

            RAISERROR('Le membre a déjà 5 emprunts en cours', 16, 1);

            RETURN;

        END

    -- Vérifier si le livre est disponible

    SELECT @LivreDisponible = CASE WHEN Quantite > 0 THEN 1 ELSE 0 END

    FROM Livres WHERE LivreID = @LivreID;

    IF @LivreDisponible = 0

        BEGIN

            RAISERROR('Le livre n''est pas disponible', 16, 1);

            RETURN;

        END

    -- Créer l'emprunt

    INSERT INTO Emprunts (MembreID, LivreID, DateRetourPrevue)

    VALUES (@MembreID, @LivreID, DATEADD(DAY, @DureeEmprunt, GETDATE()));

```
SET @EmpruntID = SCOPE_IDENTITY();
```

-- Décrémenter la quantité disponible

```
UPDATE Livres
```

```
SET Quantite = Quantite - 1
```

```
WHERE LivreID = @LivreID;
```

```
COMMIT TRANSACTION;
```

```
PRINT 'Emprunt créé avec succès. ID: ' + CAST(@EmpruntID AS VARCHAR);
```

```
END TRY
```

```
BEGIN CATCH
```

```
ROLLBACK TRANSACTION;
```

```
THROW;
```

```
END CATCH
```

```
END;
```

-- Procédure de retour

```
CREATE PROCEDURE sp_RetournerLivre
```

```
    @EmpruntID INT
```

```
AS
```

```
BEGIN
```

```
    SET NOCOUNT ON;
```

```
    DECLARE @LivreID INT;
```

```
    DECLARE @DateRetourPrevue DATE;
```

```
    DECLARE @JoursRetard INT;
```

```
BEGIN TRY
```

```
    BEGIN TRANSACTION;
```

-- Récupérer les informations de l'emprunt

```
    SELECT
```

```
        @LivreID = LivreID,
```

```
        @DateRetourPrevue = DateRetourPrevue
```

```
    FROM Emprunts
```

```
    WHERE EmpruntID = @EmpruntID AND Statut = 'En cours';
```

```
    IF @LivreID IS NULL
```

```
        BEGIN
```

```
RAISERROR('Emprunt non trouvé ou déjà retourné', 16, 1);
RETURN;
END
```

-- Marquer comme retourné

```
UPDATE Emprunts
SET DateRetourReel = GETDATE(),
Statut = 'Retourné'
WHERE EmpruntID = @EmpruntID;
```

-- Incrémenter la quantité disponible

```
UPDATE Livres
SET Quantite = Quantite + 1
WHERE LivreID = @LivreID;
```

-- Calculer les jours de retard

```
SET @JoursRetard = DATEDIFF(DAY, @DateRetourPrevue, GETDATE());
```

-- Créer une amende si en retard

```
IF @JoursRetard > 0
BEGIN
    INSERT INTO Amendes (EmpruntID, MontantAmende, Motif)
    VALUES (@EmpruntID, @JoursRetard * 0.50, 'Retard de ' + CAST(@JoursRetard AS VARCHAR) + ' jour(s)');
END
```

```
COMMIT TRANSACTION;
```

```
PRINT 'Livre retourné avec succès';
```

```
IF @JoursRetard > 0
    PRINT 'Amende appliquée: ' + CAST(@JoursRetard * 0.50 AS VARCHAR) + '€';
```

```
END TRY
```

```
BEGIN CATCH
```

```
ROLLBACK TRANSACTION;
```

```
THROW;
```

```
END CATCH
```

```
END;
```

## Requêtes de reporting



sql

-- Rapport des livres les plus empruntés

CREATE VIEW RapportLivresPopulaires AS

SELECT

L.Titre,

STRING\_AGG(A.Nom + ' ' + A.Prenom, ',') AS Auteurs,

C.NomCategorie,

COUNT(E.EmpruntID) AS NombreEmprunts,

AVG(CAST(DATEDIFF(DAY, E.DateEmprunt, ISNULL(E.DateRetourReel, GETDATE())) AS FLOAT)) AS DureeMois

FROM Livres L

LEFT JOIN AuteursLivres AL ON L.LivreID = AL.LivreID

LEFT JOIN Auteurs A ON AL.AuteurID = A.AuteurID

LEFT JOIN Categories C ON L.CategorieID = C.CategorieID

LEFT JOIN Emprunts E ON L.LivreID = E.LivreID

GROUP BY L.LivreID, L.Titre, C.NomCategorie;

-- Rapport des membres actifs

CREATE VIEW RapportMembresActifs AS

SELECT

M.NumeroMembre,

M.Nom + ' ' + M.Prenom AS NomComplet,

M.TypeMembre,

COUNT(E.EmpruntID) AS TotalEmprunts,

COUNT(CASE WHEN E.Statut = 'En cours' THEN 1 END) AS EmpruntsEnCours,

COUNT(CASE WHEN E.DateRetourReel > E.DateRetourPrevue THEN 1 END) AS EmpruntsEnRetard,

ISNULL(SUM(AM.MontantAmende), 0) AS TotalAmendes,

ISNULL(SUM(CASE WHEN AM.EstPayee = 0 THEN AM.MontantAmende ELSE 0 END), 0) AS AmendesNonPayees

FROM Membres M

LEFT JOIN Emprunts E ON M.MembreID = E.MembreID

LEFT JOIN Amendes AM ON E.EmpruntID = AM.EmpruntID

WHERE M.EstActif = 1

GROUP BY M.MembreID, M.NumeroMembre, M.Nom, M.Prenom, M.TypeMembre;

-- Tableau de bord général

CREATE PROCEDURE sp\_TableauDeBord

AS

BEGIN

-- Statistiques générales

SELECT

'Statistiques Générales' AS Categorie,

(SELECT COUNT(\*) FROM Livres) AS TotalLivres,

(SELECT COUNT(\*) FROM Membres WHERE EstActif = 1) AS MembresActifs,

```
(SELECT COUNT(*) FROM Emprunts WHERE Statut = 'En cours') AS EmpruntsEnCours,  
(SELECT COUNT(*) FROM Reservations WHERE Statut = 'Active') AS ReservationsActives;
```

-- Top 5 des livres les plus empruntés ce mois

```
SELECT TOP 5
```

```
'Top Livres du Mois' AS Categorie,  
L.Titre,  
COUNT(E.EmpruntID) AS EmpruntsDeuxMois
```

```
FROM Livres L
```

```
INNER JOIN Emprunts E ON L.LivreID = E.LivreID  
WHERE E.DateEmprunt >= DATEADD(MONTH, -1, GETDATE())  
GROUP BY L.LivreID, L.Titre  
ORDER BY EmpruntsDeuxMois DESC;
```

-- Membres avec amendes impayées

```
SELECT
```

```
'Amendes Impayées' AS Categorie,  
M.NumeroMembre,  
M.Nom + ' ' + M.Prenom AS Membre,  
SUM(AM.MontantAmende) AS TotalDu
```

```
FROM Membres M
```

```
INNER JOIN Emprunts E ON M.MembreID = E.MembreID  
INNER JOIN Amendes AM ON E.EmpruntID = AM.EmpruntID  
WHERE AM.EstPayee = 0  
GROUP BY M.MembreID, M.NumeroMembre, M.Nom, M.Prenom  
HAVING SUM(AM.MontantAmende) > 0  
ORDER BY TotalDu DESC;
```

```
END;
```

## Projet 2 : Système de gestion scolaire simplifié



sql

-- Base de données école

CREATE DATABASE EcoleDB;

USE EcoleDB;

-- Tables principales

CREATE TABLE Professeurs (

ProfesseurID INT IDENTITY(1,1) PRIMARY KEY,

NumeroEmploye VARCHAR(10) UNIQUE,

Nom VARCHAR(50) NOT NULL,

Prenom VARCHAR(50) NOT NULL,

Specialite VARCHAR(50),

Email VARCHAR(100),

DateEmbauche DATE,

Salaire DECIMAL(10,2)

);

CREATE TABLE Classes (

ClasseID INT IDENTITY(1,1) PRIMARY KEY,

NomClasse VARCHAR(20) NOT NULL,

Niveau VARCHAR(20),

ProfesseurPrincipalID INT,

AnneeScolaire VARCHAR(9), -- 2023-2024

FOREIGN KEY (ProfesseurPrincipalID) REFERENCES Professeurs(ProfesseurID)

);

CREATE TABLE Eleves (

EleveID INT IDENTITY(1,1) PRIMARY KEY,

NumeroEleve VARCHAR(10) UNIQUE,

Nom VARCHAR(50) NOT NULL,

Prenom VARCHAR(50) NOT NULL,

DateNaissance DATE,

ClasseID INT,

AdresseParents VARCHAR(200),

TelephoneParents VARCHAR(20),

FOREIGN KEY (ClasseID) REFERENCES Classes(ClasseID)

);

CREATE TABLE Matieres (

MatiereID INT IDENTITY(1,1) PRIMARY KEY,

NomMatiere VARCHAR(50) NOT NULL,

Coefficient DECIMAL(3,2) DEFAULT 1.0,

Description TEXT

);

CREATE TABLE Notes (

NoteID INT IDENTITY(1,1) PRIMARY KEY,  
EleveID INT NOT NULL,  
MatiereID INT NOT NULL,  
ProfesseurID INT NOT NULL,  
Note DECIMAL(4,2) CHECK (Note >= 0 AND Note <= 20),  
TypeEvaluation VARCHAR(20), -- Contrôle, Devoir, Oral, etc.  
DateEvaluation DATE,  
Commentaire TEXT,  
FOREIGN KEY (EleveID) REFERENCES Eleves(EleveID),  
FOREIGN KEY (MatiereID) REFERENCES Matieres(MatiereID),  
FOREIGN KEY (ProfesseurID) REFERENCES Professeurs(ProfesseurID)

);

-- Vue pour les bulletins

CREATE VIEW BulletinsEleves AS

SELECT

E.NumeroEleve,  
E.Nom + ' ' + E.Prenom AS NomComplet,  
C.NomClasse,  
M.NomMatiere,  
AVG(N.Note) AS MoyenneMatiere,  
M.Coefficient,  
AVG(N.Note) \* M.Coefficient AS NoteCoefficients  
FROM Eleves E  
INNER JOIN Classes C ON E.ClasseID = C.ClasseID  
INNER JOIN Notes N ON E.EleveID = N.EleveID  
INNER JOIN Matieres M ON N.MatiereID = M.MatiereID  
GROUP BY E.EleveID, E.NumeroEleve, E.Nom, E.Prenom, C.NomClasse,  
M.MatiereID, M.NomMatiere, M.Coefficient;



## BONUS : Outils utiles

### SQL Server Management Studio (SSMS)

Fonctionnalités principales :

- Interface graphique complète pour SQL Server

- Éditeur de requêtes avec coloration syntaxique et IntelliSense
- Générateur de diagrammes de base de données
- Outils de sauvegarde/restauration
- Moniteur d'activité et optimiseur de requêtes
- Importation/exportation de données

## Raccourcis utiles :

- Ctrl + E : Exécuter la requête
- Ctrl + Shift + U : Majuscules
- Ctrl + Shift + L : Minuscules
- Ctrl + K, Ctrl + C : Commenter
- Ctrl + K, Ctrl + U : Décommenter
- F5 : Exécuter
- Ctrl + L : Afficher le plan d'exécution

## Azure Data Studio

### Avantages :

- Multiplateforme (Windows, macOS, Linux)
- Interface moderne et légère
- Support des notebooks SQL
- Extensions disponibles
- Compatible avec SQL Server, PostgreSQL, etc.

## sqlcmd (Command Line Interface)



bash

```
# Connexion
sqlcmd -S serveur -d base_de_donnees -E #Authentification Windows
sqlcmd -S serveur -d base_de_donnees -U utilisateur -P motdepasse
```

```
# Exécution de scripts
sqlcmd -S serveur -d base_de_donnees -i script.sql -o resultat.txt
```

```
# Variables
sqlcmd -v NomVariable="Valeur" -i script_avec_variables.sql
```

## Outils de modélisation ERD

### Outils recommandés :

- **SQL Server Management Studio** : Diagrammes intégrés
- **Lucidchart** : En ligne, collaboratif
- **Draw.io** : Gratuit, en ligne
- **MySQL Workbench** : Gratuit, pour MySQL mais peut servir de modèle

- **dbdiagram.io** : Spécialisé dans les bases de données

## Exemple de script ERD en texte



```
-- Relations principales de notre système bibliothèque
/*
AUTEURS ||--o{ AUTEURS_LIVRES }o--|| LIVRES
LIVRES }o--|| CATEGORIES
LIVRES }o--|| EDITEURS
LIVRES ||--o{ EMPRUNTS }o--|| MEMBRES
EMPRUNTS ||--o{ AMENDES
LIVRES ||--o{ RESERVATIONS }o--|| MEMBRES
CATEGORIES ||--o{ CATEGORIES (auto-référence)
*/

```

## Conseils de bonnes pratiques

- Nommage cohérent** : Utilisez des conventions (PascalCase pour les tables, camelCase pour les colonnes)
- Commentaires** : Documentez vos requêtes complexes
- Indentation** : Rendez votre code lisible
- Sauvegarde régulière** : Automatisez vos sauvegardes
- Index appropriés** : Créez des index sur les colonnes fréquemment utilisées dans WHERE et JOIN
- \*\*Évitez SELECT \*\*\*** : Spécifiez les colonnes nécessaires
- Transactions** : Utilisez-les pour les opérations critiques
- Gestion d'erreurs** : Implémentez TRY-CATCH pour les procédures stockées

## Exercices pratiques par niveau

### Niveau débutant

1. Créer une base de données "MaPremiereDB" avec une table "Produits"
2. Insérer 10 produits avec nom, prix, catégorie
3. Sélectionner tous les produits dont le prix > 50€
4. Modifier le prix de tous les produits de la catégorie "Électronique" (+10%)
5. Supprimer les produits obsolètes

### Niveau intermédiaire

1. Créer un système de commandes avec clients, commandes, et détails
2. Utiliser des jointures pour afficher commandes avec informations client
3. Calculer le total de chaque commande
4. Trouver les clients qui n'ont jamais commandé
5. Créer une vue des meilleures ventes

## Niveau avancé

1. Implémenter un système d'audit avec triggers
  2. Créer des procédures stockées avec gestion d'erreurs
  3. Optimiser les requêtes avec des index appropriés
  4. Implémenter la sécurité avec rôles et permissions
  5. Créer un système de sauvegarde automatisé
- 

Ce guide vous donne une base solide pour maîtriser SQL de A à Z. Pratiquez régulièrement et n'hésitez pas à expérimenter avec vos propres projets !