2021

SQL sous SQL Server

Les bases du Simple Query Language

Olivier D.

Table des matières

1 Les bases du SQL

SQL Server et un SGBDR, c'est-à-dire un **S**ystème de **G**estion de **B**ases de **D**onnées **R**elationnelles.

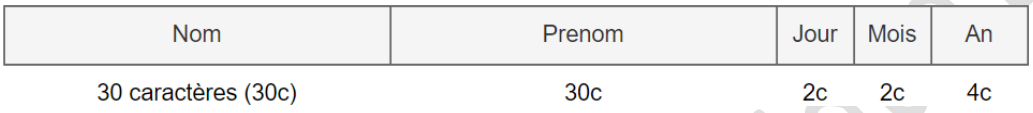
1.1 Plusieurs types de données

- **Données permanentes :** durée de vie très longue, commun, bouge peu (ex. : personnels salariés)
- **Données de mouvement :** durée de vie courte, bouge beaucoup (ex. : réservation d'un billet de train)
- **Données de travail :** extraites par calcul des données permanentes et de mouvement, durée de vie très courte
- **Données historiques :** archives. Stockées sur bandes dans une salle obscure

1.2 Organisation des données dans un fichier

Exemple : données d'une personne contenant Nom, Prénom et Date de naissance

Première méthode : direct



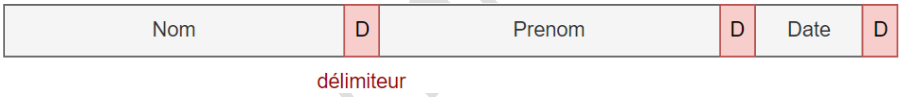
Méthode directe

Prénom du 1er client : du 31e au 60e C (caractère)

Prénom du 5e client : 68 x 4 + 31

- Avantage : facile à classer
- Inconvénient : limité aux n caractères (pas de nom de plus de 30 C)

Deuxième méthode : séquentiel

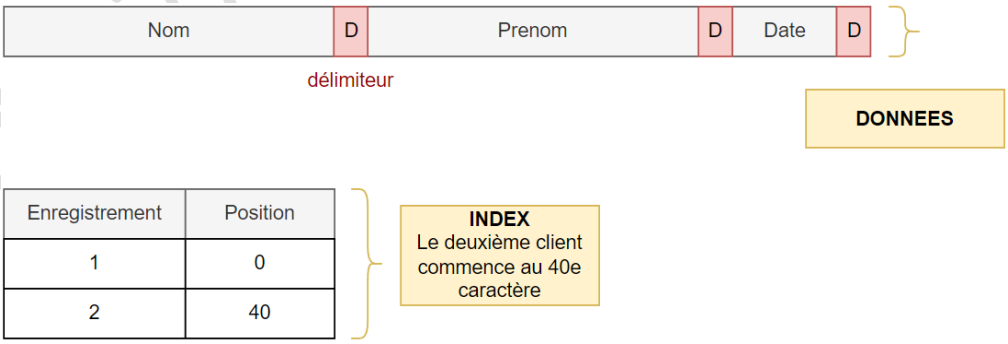


Méthode séquentielle

Taille des champs variable

- Avantage : plus de problème de taille des champs
- Inconvénient : recherches longues (il faut lire les données de façon séquentielle à cause des délimiteurs)

Troisième méthode : séquentiel indexé

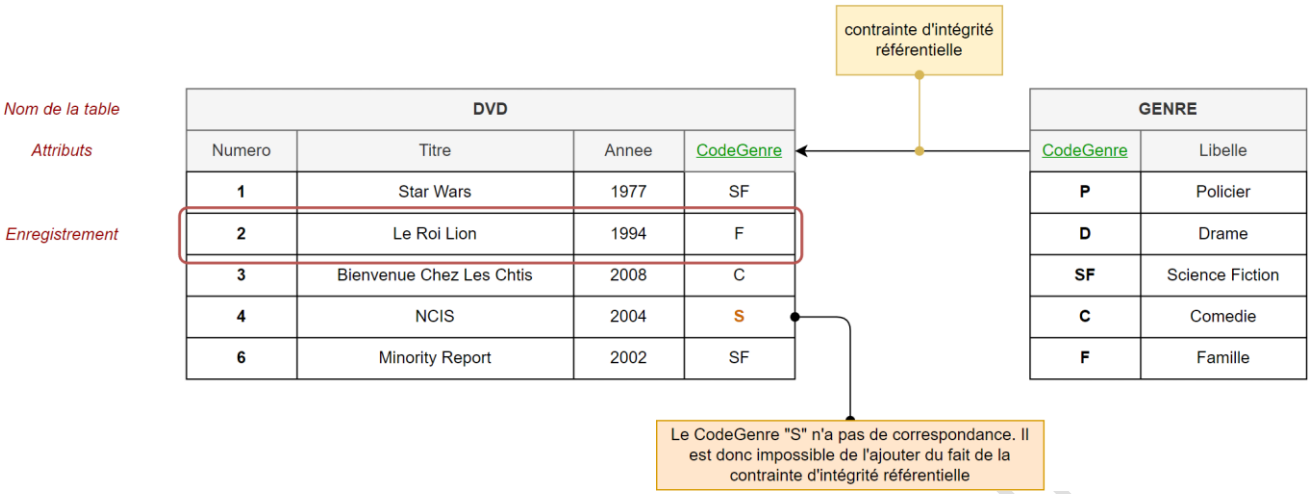


Méthode séquentielle indexée

- Avantages : pas de problème de taille des champs, recherches un peu plus rapides
- Inconvénient : réindexer lors de chaque modification

1.3 Tables et relations

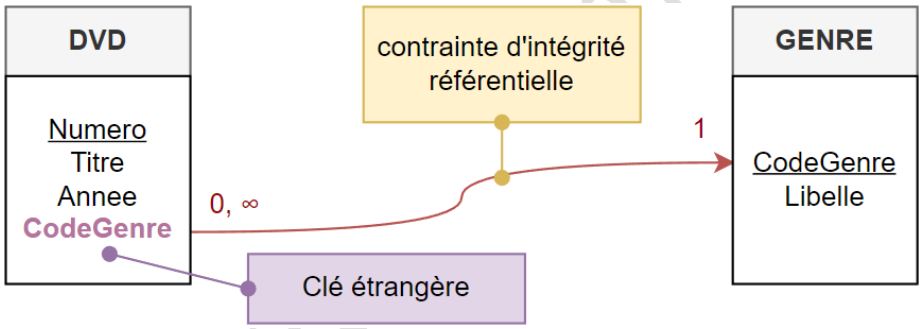
Système utilisé pour les BDD : système relationnel



Exemple de relation entre des films et leur genre

Contrainte d'intégrité relationnelle : la colonne CodeGenre doit exister dans la table GENRE.

Si on supprime un élément de la table GENRE, il faut s'assurer qu'aucun élément de la table DVD n'y fasse référence !



L'attribut souligné est la clé unique (clé primaire de la table)

0,∞ : un enregistrement de la table GENRE peut correspondre à de 0 à plusieurs enregistrements de la table DVD

1 : un enregistrement de la table DVD peut correspondre à 1 seul enregistrement de la table GENRE

Il y a toujours une contrainte d'intégrité allant de la clé étrangère vers la clé primaire

La clé primaire détermine les autres attributs : le numéro détermine le titre (c'est la clé d'unicité des lignes de la table), l'année, le code genre

2 SQL server

SQL Server fournit un ensemble de services :

- SQLServerExpress : version gratuite
- SQLServer : version payante (guide pour le licensing)

Application graphique pour faire du SQL : SQL Server Management Studio

Il faut ajouter les utilisateurs de SQLServer au groupe Administrateur avant

2.1 DDL (*Data Definition Language*) : Structure des Bases de Données (BDD)

- CREATE : créer un objet
- ALTER : modifier un objet
- DROP : supprimer un objet

Créer une base de données et l'utiliser

```
CREATE DATABASE GestionEmployes
GO                                          # Pour séparer les instructions
USE GestionEmployes
```

Créer, modifier, supprimer une table et son contenu

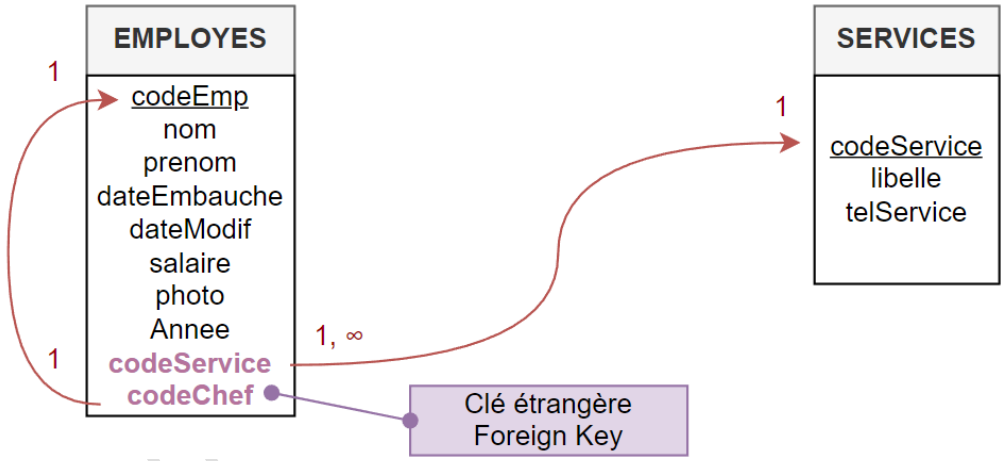
```
CREATE TABLE Services(
        codeService CHAR(5)    CONSTRAINT pk_services PRIMARY KEY,
        libelle VARCHAR(30)    NOT NULL
) ;
ALTER TABLE Services ADD telService CHAR(3) ;
ALTER TABLE Services ALTER COLUMN telService CHAR(10) ;
ALTER TABLE Services ADD CONSTRAINT un_services_libelle UNIQUE(libelle) ;
DROP TABLE Services ;
```

Clé primaire : indexé, relations possibles entre les tables, unique (identifie de façon unique un enregistrement)

Exemple de la création des 2e et 3e tables et de leurs relations :

```
CREATE TABLE Employes(
        codeEmp         UNIQUEIDENTIFIER            CONSTRAINT pk_employes PRIMARY KEY,
        nom             VARCHAR(20)             NOT NULL,
        prenom          VARCHAR(20),
        dateEmbauche    DATETIME                NULL,
        dateModif       DATETIME                NULL,
        salaire         NUMERIC(8,2)            DEFAULT 0
                CONSTRAINT ck_employes_salaire CHECK(salaire >= 0),
        photo           IMAGE,
        codeService     CHAR(5)                 NOT NULL,
        codeChef        UNIQUEIDENTIFIER        NULL,
                CONSTRAINT ck_employes_dates
        CHECK   (dateModif IS NULL OR dateModif >= dateEmbauche)
) ;
ALTER TABLE Employes ADD
        CONSTRAINT fk_employes_services      FOREIGN KEY(codeService)
                REFERENCES Services(CodeService),
        CONSTRAIN fk_employes_employes       FOREIGN KEY(codeChef)
                REFERENCES Employes(CodeEmp)
;
```

Le schéma obtenu est le suivant :



Utilisation des clés étrangères

Les types de contraintes :

```
CONSTRAINT <nom de la contrainte>
```

- Clé primaire : … PRIMARY KEY(nom de la/des clés primaires)
- Clé étrangère : … FOREIGN KEY(champ local) REFERENCES table(champ distant)
- Vérifications : … CHECK(conditions)
- Clé secondaire : … UNIQUE(champ)

Ajout d'une table ayant une clé primaire composite :

```
CREATE TABLE Conges(
	codeEmp UNIQUEIDENTIFIER	NOT NULL
		CONSTRAINT fk_conges_employes
		REFERENCES employes(codeEmp),
	annee		NUMERIC(4)		NOT NULL,
	nbjourAcquis	NUMERIC(2)		DEFAULT 30,
	CONSTRAINT pk_conges PRIMARY KEY(codeEmp, annee)
);
```

Ajout d'une clé étrangère composite:

```
CREATE TABLE congesMens(
	codeEmp		UNIQUEIDENTIFIER		NOT NULL,
	annee			NUMERIC(4)		NOT NULL,
	mois			NUMERIC(2)		NOT NULL
		CONSTRAINT ck_congesMens_mois		CHECK(mois BETWEEN 1 AND 12),
	nbJoursPris		NUMERIC(2)		DEFAULT 0,
		CONSTRAINT pk_congesMens		PRIMARY KEY(codeEmp, annee, mois),
		CONSTRAINT fk_congesMens_conges		FOREIGN KEY(codeEmp, annee)
		REFERENCES Conges(codeEmp, annee)
);
```

Suppression en cascade des enregistrements :

```
ALTER TABLE congesMens DROP CONSTRAINT fk_congesMens_conges;
ALTER TABLE conges DROP CONSTRAINT fk_conges_employes;

ALTER TABLE congesMens ADD CONSTRAINT fk_congesMens_conges
	FOREIGN KEY(codeEmp, annee) REFERENCES conges(codeEmp, annee)
	ON DELETE CASCADE;

ALTER TABLE conges ADD CONSTRAINT fk_conges_employes
	FOREIGN KEY(codeEmp) REFERENCES employes(codeEmp)
	ON DELETE CASCADE;
```

Les expressions courantes :

- `NOT NULL` valeur nulle interdite
- `NULL` (ou rien, valeur par défaut) valeur nulle autorisée
- `ON DELETE CASCADE` (après `FOREIGN KEY`) supprime les données des tables affiliées
- `DEFAUT GETDATE()` valeur par défaut = date du jour

Suppression de tables :

Méthode 1 : supprimer les tables en « suivant » les flèches des relations afin de ne pas causer d'incohérence dans les `CONSTRAINT … FOREIGN KEY …`

```
DROP TABLE congesMens ;
DROP TABLE conges ;
DROP TABLE employes ;
DROP TABLE services;
```

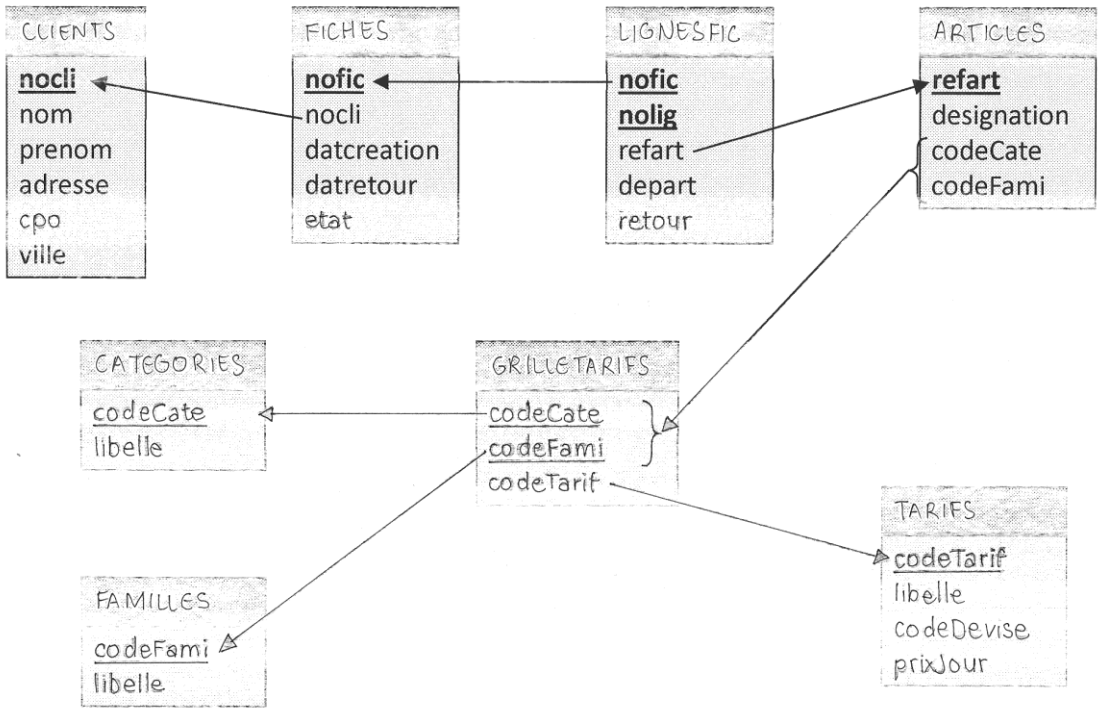
Méthode 2 : supprimer les contraintes de clés étrangères puis supprimer les tables (dans n'importe quel ordre)

```
ALTER TABLE congesMens DROP CONSTRAINT fk_congesMens_conges ;
ALTER TABLE conges DROP CONSTRAINT fk_conges_employes ;
ALTER TABLE employes DROP CONSTRAINT fk_employes_services ;
```

Nota : Il n'est pas utile le supprimer la contrainte interne à la table employés.

```
DROP TABLE employes ;
DROP TABLE services;
DROP TABLE congesMens ;
DROP TABLE conges ;
```

3 TD1 : DML – Refaire un schéma en utilisant SQL




```
CREATE DATABASE Location
GO
USE Location
```

```
CREATE TABLE categories(
codeCate        CHAR(5)
CONSTRAINT pk_categories PRIMARY KEY,
libelle         VARCHAR(80)    NOT NULL,
CONSTRAINT un_categories_libelle     UNIQUE(libelle));
```

```
CREATE TABLE familles(
codeFami        CHAR(5)
CONSTRAINT pk_familles PRIMARY KEY,
libelle         VARCHAR(80)    NOT NULL,
CONSTRAINT un_familles_libelle       UNIQUE(libelle));
```

```
CREATE TABLE tarifs(
codeTarif       CHAR(5)
CONSTRAINT pk_tarifs PRIMARY KEY,
libelle         VARCHAR(80)    NOT NULL,
codeDevise      CHAR(2),
prixJour        NUMERIC(10,2),
CONSTRAINT ck_tarifs_prixJour        CHECK(prixJour >= 0));
CREATE TABLE grilletarifs(
codeCate        CHAR(5)        NOT NULL
CONSTRAINT fk_grilletarifs_categories
REFERENCES categories(codeCate),
codeFami        CHAR(5)        NOT NULL
CONSTRAINT fk_grilletarifs_familles
```

```
REFERENCES familles(codeFami),
codeTarif      CHAR(5)        NOT NULL
CONSTRAINT fk_grilletarifs_tarifs
REFERENCES tarifs(codeTarif),
CONSTRAINT pk_grilletarifs    PRIMARY KEY(codeCate,codeFami));
```

```
CREATE TABLE articles(
refart  CHAR(8)
CONSTRAINT pk_articles PRIMARY KEY,
designation    VARCHAR(80)    NOT NULL,
codeCate       CHAR(5)        NOT NULL,
codeFami       CHAR(5)        NOT NULL,
CONSTRAINT fk_articles_grilletarifs         FOREIGN KEY(codeCate,codeFami)
       REFERENCES grilletarifs(codeCate,codeFami));
```

```
CREATE TABLE clients(
nocli          NUMERIC(6)
CONSTRAINT pk_clients PRIMARY KEY,
nom            CHAR(30)       NOT NULL,
prenom         VARCHAR(30)    NOT NULL,
adresse        VARCHAR(120)   NOT NULL,
cpo            CHAR(5)        NOT NULL,
ville          CHAR(80)       DEFAULT 'Nantes',
CONSTRAINT ck_clients_cpo     CHECK(cpo BETWEEN 1000 AND 95999));
```

```
CREATE TABLE fiches(
nofic          NUMERIC(6)
CONSTRAINT pk_fiches PRIMARY KEY,
nocli          NUMERIC(6)
CONSTRAINT fk_fiches_clients
       REFERENCES clients(nocli),
datcreation    DATE           DEFAULT GETDATE(),
datretour      DATE,
etat           CHAR(2)        DEFAULT 'EC',
CONSTRAINT ck_fiches_etat     CHECK(etat = 'EC' OR etat = 'RE' OR etat = 'SO'),
CONSTRAINT ck_fiches_datretour       CHECK(datretour IS NULL OR datretour >= datcreation));
```

```
CREATE TABLE lignesfic(
nofic          NUMERIC(6)     NOT NULL
CONSTRAINT fk_lignesfic_fiches
       REFERENCES fiches(nofic),
nolig          NUMERIC(3)     NOT NULL,
refart  CHAR(8)
CONSTRAINT fk_lignesfic_articles
       REFERENCES articles(refart),
depart         DATE           DEFAULT GETDATE(),
retour         DATE,
CONSTRAINT pk_lignesfic              PRIMARY KEY(nofic,nolig),
CONSTRAINT ck_lignesfic_retour       CHECK(retour IS NULL OR retour >= depart));
```

4 DML (Data Manipulation Language) : Contenu de la BD

- INSERT : ajout de données
- UPDATE : modification de données
- DELETE : suppression de données
- SELECT : extraire des informations

Ajout de données (travail sur GestionEmployes)

```
INSERT [INTO] table[(ordre des champs)] VALUES(valeurs) ;
```

Déclaration et modification d'une variable :

```
DECLARE @variable TYPE;
SET @variable = valeur;
```

Exemple :

```
DECLARE @myid UNIQUEIDENTIFIER;
SET @myid = NEWID();
INSERT INTO employes(codeEmp,nom,prenom,dateEmbauche,salaire,codeService)
       VALUES(@myid,'Raimbaud','Arthur','25/01/2008',3500.20,'RESHU');
INSERT INTO Conges VALUES (@myid,2008,DEFAULT);
INSERT congesMens VALUES (@myid,2008,3,5);
INSERT congesMens VALUES (@myid,2008,5,5);
INSERT congesMens VALUES (@myid,2008,7,15);
INSERT congesMens VALUES (@myid,2008,12,5);
```

La variable doit être déclarée à chaque fois qu'on l'utilise. On rentre les valeurs codeEmp tant que @myid a la valeur du codeEmp de Arthur Raimbaud.

```
DECLARE @myid UNIQUEIDENTIFIER;
SET @myid = (SELECT codeEmp
FROM Employes
WHERE nom='Raimbaud' AND prenom ='Arthur);
INSERT INTO conges VALUES(@idR,2011,35);
INSERT INTO congesMens VALUES(@idR,2011,1,2);
INSERT INTO congesMens VALUES(@idR,2011,4,5);
INSERT INTO congesMens VALUES(@idR,2011,6,10);
```

On récupère la valeur de codeEmp correspondant à Arthur Raimbaud pour l'injecter dans la variable @myid et ainsi lui ajouter de nouveaux congés.

Les variables courantes

- `NEWID()` : génère un nouvel ID
- `(SELECT …)` : valeur du résultat
- `DEFAULT` : valeur par défaut
- `'chaine','date',123` : chaine, date ou nombre
- `@variable` : valeur de la variable
- `NULL` : non renseigné / valeur vide

Tests des contraintes

Le but est de tenter d'ajouter un enregistrement contenant des erreurs liées aux contraintes. Si un message d'erreur s'affiche avec le nom de la contrainte testée alors c'est gagné.

Exemple : test de la contrainte pk_employes :

```
/* test d'ajout d'une clef primaire déjà existante */
DECLARE @idR UNIQUEIDENTIFIER;
SET @idR = (SELECT codeEmp
FROM Employes
WHERE nom='Raimbaud' AND prenom='Arthur');
INSERT INTO Employes VALUES(@idR,'A','B','1/1/2010',NULL,1,NULL,'INFOR',NULL);
```

4.1 Modification des données

```
UPDATE table SET champ = valeur;
UPDATE Employes SET nom=UPPER(nom);

DECLARE @idBB UNIQUEIDENTIFIER;
SET @idBB = (SELECT codeEmp FROM Employes WHERE nom='BIG BOSS');
UPDATE employes SET codeChef=@IdBB WHERE nom IN ('MICHEL','RAIMBAUD');
```

4.2 Suppression des données

```
DELETE table [WHERE condition];
```

```
DELETE Employes WHERE nom =  'REGIS';
DELETE Employes;
```

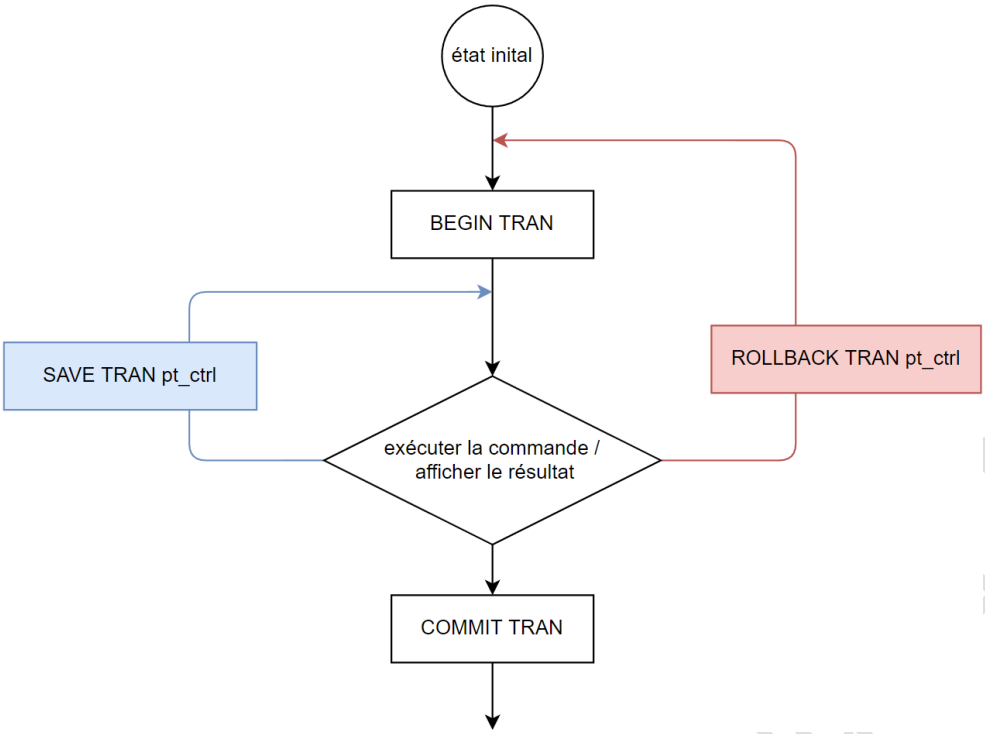
Attention aux contraintes de clé étrangères ! Attention aux suppressions en cascade !

4.3 Utilisation des transactions pour tester les modifications de contenu

```
BEGIN TRAN nom_de_la_transation
        MODIFICATION (INSERT / UPDATE / DELETE)
        SELECT * FROM table_modifiée;
ROLLBACK|COMMIT TRAN nom_de_la_transation
SELECT * FROM table_modifiée;
```

```
BEGIN TRAN sup
        DELETE Employes WHERE nom = 'REGIS';
        SELECT * FROM Employes;
ROLLBACK TRAN sup
SELECT * FROM Employes;
```

Attention aux contraintes de clé étrangères ! Attention aux suppressions en cascade !



Fonctionnement des transactions SQL Server

Gestion des points de contrôle

```
SELECT '' AS 'Avant', * FROM clients;
BEGIN TRAN TrySav
	INSERT INTO clients VALUES (20,'LAURENT','Amelie','rue Brassens',45100,NULL);
	SELECT '' AS 'Pendant1', * FROM clients;
SAVE TRAN pt_ctrl
	INSERT INTO clients VALUES (21,'DEHECQ','Olivier','ave Beauregard',44800,NULL);
	SELECT '' AS 'Pendant2', * FROM clients;
ROLLBACK TRAN pt_ctrl
COMMIT TRAN
SELECT '' AS 'Après', * FROM clients;
```

Validation tout ou rien

```
SET XACT_ABORT ON;                          # si une instruction de la TRAN rate, annule toute la TRAN

SELECT '' AS 'Avant', * FROM categories;
BEGIN TRAN TryTou
       INSERT INTO categories VALUES ('TOTO','Totos');
       SELECT '' AS 'Pendant1', * FROM categories;
       INSERT INTO categories VALUES ('PORTABLE','Ordinateur Portable');
COMMIT TRAN TryTou;
SELECT '' AS 'Après', * FROM categories;
```

4.4 L'instruction SELECT

Construction de la commande

```
SELECT valeur ;
SELECT [DISCTINCT] *|champs [valeur AS 'champ'] FROM table
```

```
SELECT NEWID();
SELECT GETDATE() AS 'Date du jour';
```

```
SELECT DATEPART(DW, GETDATE()) AS 'Jour de la semaine';
SELECT DATEPART(MONTH, GETDATE()) AS 'Mois Actuel';
SELECT DATEPART(WEEK, GETDATE()) AS 'Semaine dans lannee';
```

```
/* date dans 10 jours */
SELECT GETDATE()+10 AS 'Date dans 10 jours';
```

```
/* date dans 5 semaines */
SELECT DATEADD(WW,5,GETDATE()) AS 'Date dans 5 semaines';
```

```
/* Pour aller plus loin */
SELECT codeService FROM Employes;
SELECT DISTINCT codeService FROM Employes;
```

```
SELECT  UPPER(nom) AS 'Nom de l''employé',
               COALESCE(prenom, '<pas de prénom>') AS 'Prénom',
               CONVERT(CHAR, dateEmbauche, 103) AS 'Date d''embauche',
               Salaire=CONVERT(NUMERIC(6),ROUND(salaire,0))
FROM Employes;
/* Affichage des données d'une table */
SELECT * FROM Employes;
SELECT nom, prenom FROM Employes;
SELECT * FROM Employes WHERE dateEmbauche > '01/01/2009';
SELECT nom,prenom,codeService
FROM Employes WHERE dateEmbauche > '01/01/2009';
```

```
/* Calculs */
SELECT nom, ancienneté=DATEDIFF(MONTH, dateEmbauche,GETDATE())
FROM Employes ORDER BY ancienneté;
SELECT nom, salaire * 1.3 AS 'Salaire brut mensuel'
FROM Employes;
```

Calcul d'agrégat

C'est un calcul portant sur plusieurs enregistrements (`SUM` somme, `AVG` moyenne, `MAX`, `MIN`, `COUNT`).

```
SELECT COUNT(*) AS 'nombre d''employés'  --> calcul d'agregat cf. RP p. 88
FROM Employes;
SELECT SUM(salaire) AS 'Masse salariale'
FROM Employes;
SELECT AVG(salaire) AS 'salaire moyen' FROM Employes;
SELECT MIN(Salaire) AS 'Salaire minimum' FROM Employes;
SELECT MAX(Salaire) AS 'Salaire maximum' FROM Employes;
SELECT COUNT(prenom) AS 'nb d''employés ayant un prenom connu dans la base'
FROM Employes; --> compte les prenom != NULL
```

Grouper par critères : GROUP BY

```
/* masse salariale par service */
SELECT codeService,SUM(salaire) AS 'masse salariale du service'
FROM Employes
GROUP BY codeService;
SELECT codeService,CONVERT(NUMERIC(8,2),AVG(salaire)) AS 'sal moy/ service'
FROM Employes
GROUP BY codeService;
```

Conditions sur des ensembles : HAVING

```
SELECT codeService,CONVERT(NUMERIC(8,2),AVG(salaire)) AS 'salaire moyen par service'
FROM Employes
GROUP BY codeService
HAVING AVG(Salaire) > 2500;
```

Jointures : INNER JOIN

Les jointures permettent de faire des select sur les champs de plusieurs tables ayant des relations d'intégrités référentielles.

```
SELECT nom, Employes.codeService, Services.codeService,libelle
FROM Employes
INNER JOIN Services ON Employes.codeService = Services.codeService
ORDER BY libelle;
/* utilisation des alias */
SELECT nom, e.codeService, s.codeService,libelle
FROM Employes e
INNER JOIN Services s ON e.codeService = s.codeService
ORDER BY libelle;
```

Quand on utilise un alias (ici e pour Employes et s pour Services) il faut l'utiliser dans toute la requête SELECT.

Exemple : Calcul d'agrégat après jointure :

```
/* Jointure (voir schéma AAAA) */
SELECT nom,
       nbJoursAcquis,
       SUM(nbJoursPris) 'Nombre de jours pris'
FROM Employes e
INNER JOIN Conges c ON e.codeEmp = c.codeEmp
INNER JOIN congesMens m ON c.codeEmp=m.codeEmp AND c.annee=m.annee
WHERE c.annee = YEAR(GETDATE())
GROUP BY e.codeEmp, nom, nbJoursAcquis, c.annee;
```

Il est nécessaire de grouper par clé primaire (car unique) ainsi que par l'ensemble des valeurs affichées qui sont agrégées.

Pour exploiter un peu plus :

```
/* liste des personnes à qui il reste + de 10 jours de congés */
SELECT nom,
       nbJoursAcquis,
       SUM(nbJoursPris) 'Nombre de jours pris'
FROM Employes e
INNER JOIN Conges c ON e.codeEmp = c.codeEmp
INNER JOIN congesMens m ON c.codeEmp=m.codeEmp AND c.annee=m.annee
WHERE c.annee = YEAR(GETDATE())
GROUP BY e.codeEmp, nom,nbJoursAcquis, c.annee
HAVING nbJoursAcquis-SUM(nbJoursPris)>10;
```

Ici on joint 3 tables ensembles : Employes, Conges et congesMens

Trier : ORDER BY

```
DESC indique un ordre décroissant.
SELECT * FROM Employes ORDER BY salaire, dateEmbauche DESC;
```

Jointures de sous requêtes

Le but d'une jointure de sous requête et de pouvoir travailler sur plusieurs requêtes comme si les champs obtenus faisaient partie d'une grande table contenant les attributs de nom1 et de nom2.

```
SELECT * FROM (
  <première requête select>
  ) nom1
INNER JOIN (
  <deuxième requête select>
  ) nom2
ON  clauses*;
```

* On sélectionne les clauses de filtre après la jointure, par exemple `nom1.codeEmp=nom2.codeEmp`

```
SELECT * FROM (
  <première requête select>
  ) nom1
CROSS JOIN (
    <deuxième requête select>
  ) nom2;
```

Type	Cout	Nom
Luge	150	Boulier
Ski	340	Boulier
Surf	110	Boulier

Jointure **INNER JOIN**

Nom	Annee	Jours
Boulier	2021	35
Boulier	2022	40

Résultat :

Type	Cout	Nom	Annee	Jours
Luge	150	Boulier	2021	35
Ski	340	Boulier	2021	35
Surf	110	Boulier	2021	35
Luge	150	Boulier	2022	40
Ski	340	Boulier	2022	40
Surf	110	Boulier	2022	40

Pour toutes les jointures INNER JOIN, le fonctionnement est le même

```
SELECT * FROM (
SELECT t.prixJour * (DATEDIFF(DD,l.depart,l.retour)+1) AS 'montant',
nolig AS 'ligne', nom, prenom
FROM clients c
INNER JOIN fiches f ON c.nocli = f.nocli
INNER JOIN lignesfic l ON l.nofic = f.nofic
INNER JOIN articles a ON l.refart = a.refart
INNER JOIN grilletarifs g ON g.codeCate = a.codeCate AND g.codeFami = a.codeFami
INNER JOIN tarifs t ON t.codeTarif = g.codeTarif
WHERE UPPER(nom)='BOUTAUD' AND UPPER(prenom)='SABINE'
) req1
INNER JOIN (
SELECT SUM(t.prixJour * (DATEDIFF(DD,l.depart,l.retour)+1)) AS 'montant total', nom, prenom
FROM clients c
INNER JOIN fiches f ON c.nocli = f.nocli
```

```
INNER JOIN lignesfic l ON l.nofic = f.nofic
INNER JOIN articles a ON l.refart = a.refart
INNER JOIN grilletarifs g ON g.codeCate = a.codeCate AND g.codeFami = a.codeFami
INNER JOIN tarifs t ON t.codeTarif = g.codeTarif
WHERE UPPER(nom)='BOUTAUD' AND UPPER(prenom)='SABINE'
GROUP BY nom, prenom,l.depart,l.retour
) req2
ON req1.nom=req2.nom AND req1.prenom=req2.prenom;
```

Jointures externes : [FULL|LEFT|RIGHT] OUTER JOIN

Le but d'une jointure externe est d'afficher les enregistrements qui ne se retrouvent dans la table jointe (Employés sans Services par exemple). Il est possible d'afficher :

- Soit l'intégralité des personnels + les services étant affectés à des employés
- Soit les personnels étant affectés à des services + l'intégralité des services
- Soit l'intégralité des personnels + l'intégralité des services

FULL OUTER JOIN :

ARTICLES a	
Type	RefArticle
Luge	A
Ski	B
Surf	C

LIGNESFIC l		
RefArticle	NumeroFichier	NumeroLigne
A	1001	1
C	1001	2
A	1002	1

Résultat :

ARTICLES | X | LIGNESFIC				
Type	a.RefArticle	l.RefArticle	NumeroFichier	NumeroLigne
Luge	A	A	1001	1
Surf	C	C	1001	2
Luge	A	A	1002	1
Ski	B	***NULL***	***NULL***	***NULL***

INNER JOIN (jointure interne)

OUTER JOIN (jointure externe)

Jointures internes et externes

```
SELECT a.refart,a.designation,
COUNT(l.refart) AS 'NbTotalLoc par Article'
FROM Articles a
FULL OUTER JOIN LignesFic l ON l.refart=a.refart
GROUP BY a.refart,a.designation;
```

Dans ce cas présent, il faut `COUNT(l.refart)` qui dira que B se retrouve 0 fois dans la table LignesFic ! Si on met `COUNT(a.refart)`, il indique qu'il y a 1 ligne pour B.

[LEFT|RIGHT] OUTER JOIN :

```
SELECT … FROM employes LEFT OUTER JOIN service ON …
```

On retrouve **tous les enregistrements de la table de gauche** (employés)

Et dans la table de droite (services), on ne trouve que ceux qui existent dans employés

```
SELECT … FROM employes RIGHT OUTER JOIN service ON …
```

On retrouve **tous les enregistrements de la table de droite** (services)

Et dans la table de gauche (employes), on ne trouve que ceux qui existent dans employes

Auto-jointures : jointures dans une même table

C'est pour utiliser les relations d'une table sur elle-même (cas des employés et de leurs chefs)

EMPLOYES e		
CodeEmploye	Nom	codeChef
01	Jean Vintrois	NULL
02	Marc Aurele	03
03	Jules Cesar	01
04	Premier Decurion	03

EMPLOYES c		
CodeEmploye	Nom	codeChef
01	Jean Vintrois	NULL
02	Marc Aurele	03
03	Jules Cesar	01
04	Premier Decurion	03

Résultat :

EMPLOYES e		
CodeEmploye	Nom	Son Chef
02	Marc Aurele	Jules Cesar
03	Jules Cesar	Jean Vintrois
04	Premier Decurion	Jules Cesar

Utilisation des auto-jointures

```
/* nom du chef direct */
SELECT e.nom AS 'Employé', c.nom AS 'Chef'
FROM Employes e
INNER JOIN Employes c ON e.codeChef=c.codeEmp
ORDER BY c.nom,e.nom;
```

Bouba n'apparait pas dans le résultat car la valeur de codeChef est NULL. Si on veut l'afficher, il faut faire un LEFT OUTER JOIN :

```
/* nom du chef direct : affiche les sans chef */
SELECT e.nom AS 'Employé', COALESCE(c.nom,'-') AS 'Chef'
FROM Employes e
LEFT OUTER JOIN Employes c ON e.codeChef=c.codeEmp
ORDER BY c.nom,e.nom;
```

Dans le même ordre d'idées, la requête suivante permet d'afficher le nombre de subordonnés qu'un chef a sous son commandement direct :

```
/* Compter le nombre de subordonnés */
SELECT c.nom AS 'Nom du chef',COUNT(e.codeEmp) AS 'NbreSubordonnés'
FROM Employes e
RIGHT JOIN Employes c ON e.codeChef=c.codeEmp
GROUP BY c.nom,c.codeEmp
ORDER BY NbreSubordonnés DESC;
```

4.5 Les vues

Une vue est un filtre d'affichage d'un ou plusieurs tables selon les colonnes, selon les critères, selon les enregistrements.

Le tableau représente une table. En vert, la valeur qu'on veut faire apparaitre dans la vue.

- **La vue est enregistrée définitivement dans la BD ; on y accès comme une table**
- Si les données sont modifiées à partir des tables d'origine, cela met à jour va vue (et réciproquement).

```
CREATE VIEW nomVue AS
SELECT nom_des_champs FROM table
WHERE code_service='RESHU'
[WITH CHECK OPTION];
```

`WITCH CHECK OPTIONS` empêche l'ajout de données (avec `INSERT INTO nomVue`) qui sortent du cadre de la vue.

Très utile pour donner les droits à une personne de modifier un seul service.

TOP

```
SELECT TOP 3 nom, salaire FROM Employes ORDER BY salaire DESC;
```

Les 3 plus gros salaires.

Les tables temporaire (dure le temps de la session)

```
SELECT AVG(salaire) AS 'salaire moyen' INTO #T1 FROM Employes;
```

Les données de la table sont figées, c'est l'équivalent d'une photo à un instant donné.

Les tables instantanées (tables CTE)

```
WITH ChefCTE AS (
SELECT codeEmp, nom, prenom FROM Employes e)
SELECT e.nom,e.prenom,c.nom AS 'son chef' FROM chefCTE c
RIGHT OUTER JOIN Employes e ON e.codeChef=c.codeEmp;
```

La table `ChefCTE` contient un instantané de la requête entre parenthèse puis est supprimée dès son utilisation dans la requête suivante.

Les valeurs auto incrémentées

```
CREATE TABLE Tests(
numTest INT IDENTITY(100,2) CONSTRAINT pk_tests PRIMARY KEY,
nomTest VARCHAR(10)
);
```

Le type doit être INT, IDENTITY(100,2) indique (début à 100, augmente de 2 en 2)

```
INSERT Tests VALUES('Polio');
```

Un nouvel enregistrement est créé puisqu'on NE DOIT PAS entrer la valeur de IDENTITY (elle est générée toute seule).

```
SELECT * FROM Tests ;
```

numTest	nomTest
100	Polio

```
INSERT Tests VALUES('BCG');
SELECT * FROM Tests ;
```

numTest	nomTest
100	Polio
102	BCG