Documentation SQL Server

Table de matière

I- Création de la base

1. Création de la base de données :

Toutes les données de la base de données sont organisées efficacement sous forme de tableaux. Une base de données peut être formée à partir d'une collection de plusieurs tables, où chaque table serait utilisée pour stocker un type particulier de données et la table elle-même serait liée les unes aux autres en utilisant certaines relations.

Pour faire la création d'une base de données, nous allons utiliser la commande :

```
CREATE DATABASE Nom_base_donnees ;
```

Exemple :

```
CREATE DATABASE TP_X ;
```

2. Création des tables :

Nous utilisons la commande CREATE pour créer une table. La table de l'exemple ci-dessus peut être créée avec le code suivant :

```
CREATE TABLE table_name  (
  Column1 DATATYPE,
  Column2 DATATYPE,
  Column3 DATATYPE,
  ...
);
```

Exemple :

```
CREATE TABLE student(
  ID INT NOT NULL,
  Name varchar(25),
  Phone varchar(12),
  Class INT
);
```

3. Type des données

Pour les chaines de caractères :

Types de données	Description
CHAR(size)	Chaîne de longueur fixe contenant des chiffres, des lettres ou des caractères spéciaux. La longueur peut varier de 0 à 255.
VARCHAR(size)	Chaîne de longueur variable dont la longueur peut varier de 0 à 65 535. Semblable à CHAR.
TEXT(size)	Peut contenir une chaîne d'une taille maximale de 65 536 octets.
TINY TEXT	Peut contenir une chaîne de 255 caractères maximum.
MEDIUM TEXT	Peut contenir une chaîne de 16777215 caractères maximum.
LONG TEXT	Peut contenir une chaîne de 4294967295 caractères maximum.
BINARY(size)	Similaire à CHAR() mais stocke des chaînes d'octets binaires.
VARBINARY(size)	Similaire à VARCHAR() mais stocke des chaînes d'octets binaires.
BLOB(size)	Contient des blobs jusqu'à 65536 octets.
TINYBLOB	Il est utilisé pour les grands objets binaires et a une taille maximale de 255 octets.
MEDIUMBLOB	Contient des blobs jusqu'à 16777215 octets.
LONGBLOB	Contient des blobs jusqu'à 4294967295 octets.

Pour les types de données numériques :

Types de données	*Description*
BIT(size)	Type de valeur binaire, où la taille varie de 1
INT(size)	Entier avec des valeurs dans la plage signée de -2147483648 à 2147483647 et des valeurs dans la plage non signée de 0 à 4294967295.
TINYINT(size)	Entier avec des valeurs dans la plage signée de -128 à 127 et des valeurs dans la plage non signée de 0 à 255. SMALLINT(taille)Entier avec des valeurs dans la plage signée de -32768 à 32767 et des valeurs dans la plage non signée de 0 à 65535.
MEDIUMINT(size)	Entier avec des valeurs dans la plage signée de -8388608 à 8388607 et des valeurs dans la plage non signée de 0 à 16777215.
BIGINT(size)	Entier avec des valeurs dans la plage signée de 9223372036854770000 à 9223372036854770000 et des valeurs dans la plage non signée de 0 à 18446744073709551615.
BOOLEAN	Les valeurs booléennes où 0 est considéré comme FALSE et les valeurs non nulles sont considérées comme TRUE.
FLOAT	(p) Le nombre à virgule flottante est stocké. Si le paramètre de précision est défini entre 0 et 24, le type est FLOAT(), sinon s'il est compris entre 25 et 53, le type de données est DOUBLE()

Pour les types de données Date/Heure :

Types de données	Description
DATE	Stocke la date au format AAAA-MM-JJ avec des dates comprises entre « 1000-01-01 » et « 9999-12-31 ».
TIME(fsp)	Stocke l'heure au format hh:mm:ss avec des heures comprises entre '-838:59:59' et '838:59:59'.
DATETIME(fsp)	Stocke une combinaison de date et d'heure au format AAAA-MM-JJ et hh:mm:ss, avec des valeurs comprises entre '1000-01-01 00:00:00' et '9999-12-31 23:59 : 59'.
TIMESTAMP(fsp)	Il stocke des valeurs relatives à l'époque Unix, essentiellement un horodatage Unix. Les valeurs sont comprises entre '1970-01-01 00:00:01' UTC et '2038-01-09 03:14:07' UTC.
YEAR	Stocke les valeurs des années sous forme de nombre à 4 chiffres, avec une plage comprise entre -1901 et 2155.

4. Clauses en SQL

Les clauses sont des fonctions intégrées disponibles dans SQL et sont utilisées pour filtrer et analyser rapidement les données permettant à l'utilisateur d'extraire efficacement les informations requises de la base de données.

Le tableau ci-dessous répertorie certaines des clauses SQL importantes et leur description avec des exemples :

Clause	Description	Exemple
WHERE	Utilisé pour sélectionner des données dans la base de données en fonction de certaines conditions.	SELECT * from Employee WHERE age >= 18;
ORDER BY	Utilisé pour trier les données dans l'ordre croissant ou décroissant.	SELECT * FROM student ORDER BY age ASC
GROUP BY	Regroupe les lignes qui ont les mêmes valeurs dans des lignes récapitulatives.	SELECT COUNT(StudentID), State FROM Students GROUP BY State;
HAVING	Spécifie une condition de recherche pour un groupe ou un agrégat. HAVING ne peut être utilisé qu'avec l'instruction SELECT. HAVING est généralement utilisé avec une clause GROUP BY. Lorsque GROUP BY n'est pas utilisé, il existe un seul groupe agrégé implicite.	SELECT COUNT(StudentID), State FROM Students GROUP BY State HAVING StudentID='12'

5. Opérateurs SQL

Les opérateurs sont utilisés dans SQL pour former des expressions complexes qui peuvent être évaluées pour coder des requêtes plus complexes et extraire des données plus précises d'une base de données.

Il existe 3 principaux types d'opérateurs : les opérateurs arithmétiques, de comparaison et logiques, chacun étant décrit ci-dessous.

a. Opérateurs arithmétiques

Les opérateurs arithmétiques permettent à l'utilisateur d'effectuer des opérations arithmétiques en SQL. Le tableau ci-dessous présente la liste des opérateurs arithmétiques disponibles en SQL :

Operateur	Description
+	Ajout
-	Soustraction
*	Multiplication
/	Division
%	Modulo

b. Opérateurs au niveau du bit

Les opérateurs au niveau du bit sont utilisés pour effectuer des opérations de manipulation de bits dans SQL. Le tableau ci-dessous présente la liste des opérateurs au niveau du bit disponibles en SQL :

Operateur	Description
\| | OR |
& | AND |
^ | XOR |

c. Opérateurs logiques

Les opérateurs logiques sont habitués à combiner 2 ou plusieurs énoncés relationnels en 1 énoncé composé dont la valeur de vérité est évaluée comme un tout. Le tableau ci-dessous présente les opérateurs logiques SQL avec leur description :

Operateur	Description
ALL	Renvoie Vrai si toutes les sous-requêtes remplissent la condition donnée.
AND	Renvoie Vrai si toutes les conditions s'avèrent vraies
ANY	Vrai si l'une des sous-requêtes répond à la condition donnée

BETWEEN	Vrai si l'opérande se situe dans la plage des conditions
EXISTS	Vrai si la sous-requête renvoie un ou plusieurs enregistrements
IN	Renvoie Vrai si les opérandes d'au moins un des opérandes d'une liste d'expressions donnée
LIKE	Renvoie Vrai si l'opérande et un motif donné correspondent.
NOT	Affiche un enregistrement si l'ensemble des conditions données est FAUX
OR	Renvoie Vrai si l'une des conditions s'avère être Vrai
SOME	Renvoie Vrai si l'une des sous-requêtes répond à la condition donnée.

*NB : si nous cherchons que si la valeur d'une colonne est « NULL », nous utilisons l'opérateur « IS NULL ». Exemple SELECT * FROM Table WHERE column1 IS NULL*

6. Clés en SQL

Une base de données se compose de plusieurs tables et ces tables et leur contenu sont liés les uns aux autres par certaines relations/conditions. Pour identifier chaque ligne de ces tables de manière unique, nous utilisons des clés SQL. Une clé SQL peut être une colonne unique ou un groupe de colonnes utilisées pour identifier de manière unique les lignes d'une table. Les clés SQL sont un moyen de s'assurer qu'aucune ligne n'aura de valeurs en double. Ils sont également un moyen d'établir des relations entre plusieurs tables dans une base de données.

a. Clé primaire

Ils identifient de manière unique une ligne dans une table.

- Une seule clé primaire pour une table. (Un cas particulier est une clé composite, qui peut être formée par la composition de 2 colonnes ou plus, et agir comme une seule clé candidate.)
- La colonne de clé primaire ne peut pas contenir de valeurs NULL.
- La clé primaire doit être unique pour chaque ligne.

Syntaxe SQL :

```
CREATE TABLE nom_table (
  Colonne1 type NOT NULL,
  Colonne2 type,
  Colonne3 type,
  ... ,
  PRIMARY KEY (Colonne1)
);
```

Exemple :

```
CREATE TABLE Client (
  ClientID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  PRIMARY KEY (ClientID)
);
```

b. Clé étrangaire

Les clés étrangères sont des clés qui référencent les clés primaires d'une autre table. Ils établissent une relation entre 2 tables et les relient.

Syntaxe SQL :

```
CREATE TABLE nom_table2 (
  Colonne1 type NOT NULL,
  Colonne2 type,
  Colonne3 type,
  ... ,
  PRIMARY KEY (Colonne1),
  FOREIGN KEY (Colonne3) REFERENCES nom_table1(nom_table1.Colonne1)
);
```

Exemple :

```
CREATE TABLE Commande (
  OrderID int NOT NULL,
  OrderNumber int NOT NULL,
  ClientID int,
  PRIMARY KEY (OrderID),
  FOREIGN KEY (ClientID) REFERENCES Client(ClientID)
);
```

II- CRUD Operations in SQL

CRUD est une abréviation pour créer, lire, mettre à jour et supprimer. Ces 4 opérations comprennent les opérations de base de données les plus élémentaires. Les commandes pertinentes pour ces 4 opérations en SQL sont :

- Create: INSERT
- Read: SELECT
- Update: UPDATE
- Delete: DELETE

1. INSERT

INSERT : Pour insérer de nouvelles données dans une base de données, nous utilisons l'instruction INSERT INTO.

Syntaxe SQL :

```
INSERT INTO nom_table(column1, column2, ....)
```

```
VALUES(value1, value2, ....)
```

Exemple :

```
INSERT INTO student(ID, name, phone, class)
VALUES(1, 'Scaler', '+1234-4527', 12)

INSERT INTO student(ID, name, phone, class)
VALUES(1, 'Scaler', '+1234-4527', 12),
(2, 'Ahmed', '+4321-7654', 11);
```

2. SELECT

SELECT : Nous utilisons l'instruction select pour effectuer l'opération de lecture (R) de CRUD.

Syntaxe SQL :

```
SELECT column1,column2,.. FROM nom_table;
```

Exemple :

```
SELECT name,class FROM student;
```

3. UPDATE

UPDATE : est le composant "U" de CRUD. La commande Mettre à jour est utilisée pour mettre à jour le contenu de colonnes spécifiques de lignes spécifiques.

Syntaxe SQL :

```
UPDATE nom_table
SET column1=value1,column2=value2,... WHERE conditions...;
```

Exemple :

```
UPDATE students
SET phone = '+1234-9876'
WHERE ID = 2;
```

4. DELETE

La commande DELETE est utilisée pour supprimer ou supprimer certaines lignes d'une table. C'est le composant "D" de CRUD.

Syntaxe SQL :

```
DELETE FROM nom_table
WHERE condition1, condition2, ...;
```

Exemple :

```
DELETE FROM student
WHERE class = 11;
```

La commande TRUNCATE permet de supprimer toutes les données d'une table sans supprimer la table en elle-même. En d'autres mots, cela permet de purger la table. Cette instruction diffère de la commande DROP qui a pour but de supprimer les données ainsi que la table qui les contient

Syntaxe SQL :

```
TRUNCATE TABLE nom_table
```

Exemple :

```
TRUNCATE TABLE student
```

III- Modification des tables

Pour effectuer des modifications à une table dans une base de données nous allons utiliser la commande « ALTER TABLE »

Syntaxe SQL :

```
ALTER TABLE nom_table
Instructions
```

1. Ajouter une colonne

Pour ajouter une nouvelle colonne à une table déjà créé nous utilisons l'instruction « ADD »

Syntaxe SQL :

```
ALTER TABLE nom_table
ADD nom_colonne type_donnees
```

Exemple :

```
ALTER TABLE student
ADD email_address VARCHAR(255)
```

2. Supprimer une colonne

Une syntaxe permet également de supprimer une colonne pour une table. Il y a 2 manières totalement équivalente pour supprimer une colonne :

Syntaxe SQL :

```
ALTER TABLE nom_table
DROP nom_colonne
```

```
ALTER TABLE nom_table
DROP COLUMN nom_colonne
```

Exemple :

```
ALTER TABLE student
DROP email_address VARCHAR(255)
```

Attention : Si la colonne est liée à une ou plusieurs contraintes, il faut supprimer les contraintes avant de supprimer la colonne.

3. Renommer une colonne

Pour SQL Server le renommage est un peu particulier. Il existe une fonction qui fait le renommage de la colonne :

Syntaxe SQL :

```
EXEC sp_rename 'nom_table.ancien_colonne', 'nouvelle_colonne';
```

Exemple :

```
EXEC sp_rename 'student.prenm', 'prenom';
```

4. Changer le type de la colonne

Pour modifier le type de la colonne, nous utilisons :

Syntaxe SQL :

```
ALTER TABLE  nom_table
ALTER COLUMN  colonne nouveau_type;
```

Exemple :

```
ALTER TABLE  client
ALTER COLUMN  nom  varchar(400);
```

5. Ajouter une clé

Après la création de la table, nous pouvons ajouter des contraintes sur la table. Par exemple rendre une colonne existante la clé primaire d'une table.

Syntaxe SQL :

```
ALTER TABLE  nom_table
ADD CONSTRAINT  PK_nom_table  PRIMARY KEY (colonne1);
```

Exemple :

```
ALTER TABLE  students
ADD CONSTRAINT  PK_students  PRIMARY KEY (ID);
```

Nous pouvons également ajouter une clé étrangère.

Syntaxe SQL :

```
ALTER TABLE nom_table1
ADD CONSTRAINT FK_nom_table1 Foreign KEY (colonne3)
references nom_table 2 (colonne1)
```

Exemple :

```
ALTER TABLE student
ADD CONSTRAINT FK_student Foreign KEY (Classe)
references Classe(ClasseID)
```

IV- Jointure des tables :

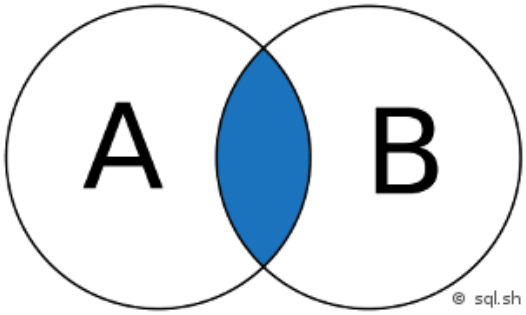
1. Types de jointures

Il y a plusieurs méthodes pour associer 2 tables ensemble. Voici la liste des différentes techniques qui sont utilisées :

- **INNER JOIN :** jointure interne pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est l'une des jointures les plus communes.
- **CROSS JOIN :** jointure croisée permettant de faire le produit cartésien de 2 tables. En d'autres mots, permet de joindre chaque ligne d'une table avec chaque ligne d'une seconde table. Attention, le nombre de résultats est en général très élevé.
- **LEFT JOIN (ou LEFT OUTER JOIN) :** jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifiée dans l'autre table.
- **RIGHT JOIN (ou RIGHT OUTER JOIN) :** jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifiée dans l'autre table.
- **FULL JOIN (ou FULL OUTER JOIN) :** jointure externe pour retourner les résultats quand la condition est vraie dans au moins une des 2 tables.
- **SELF JOIN :** permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.
- **NATURAL JOIN :** jointure naturelle entre 2 tables s'il y a au moins une colonne qui porte le même nom entre les 2 tables SQL
- **UNION JOIN :** jointure d'union

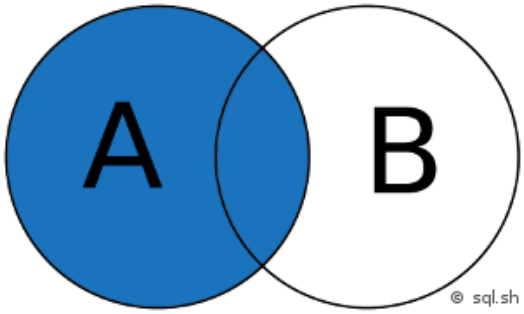
2. Exemples de Jointures

a. Inner join



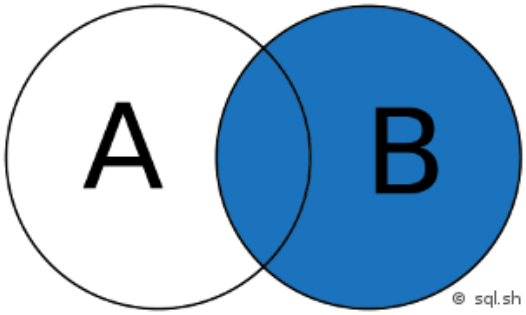

```
SELECT *
FROM A
INNER JOIN B  ON A.KEY=B.KEY
```

b. Left join



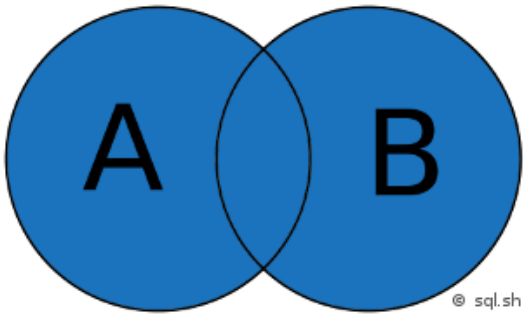

```
SELECT *
FROM A
LEFT JOIN B  ON A.KEY=B.KEY
```

c. Right join



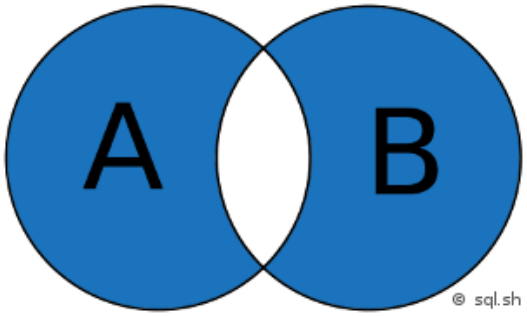

```
SELECT *
FROM A
RIGHT JOIN B  ON A.KEY=B.KEY
```

d. Full join




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

e. FULL JOIN (sans intersection)




```
SELECT *
FROM A
FULL JOIN B  ON A.KEY=B.KEY
WHERE A.KEY IS NULL or B.KEY IS NULL
```

V- Les requêtes imbriquées :

1. Définition

Dans le langage SQL une requête imbriquée (aussi appelé « sous-requête » ou « requête en cascade ») consiste à exécuter une requête à l'intérieur d'une autre requête. Une requête imbriquée est souvent utilisée au sein d'une clause WHERE ou de HAVING pour remplacer une ou plusieurs constantes.

2. Utilisation des requêtes imbriquées

a. Requête imbriquée qui retourne un seul résultat

L'exemple ci-dessous est un exemple typique d'une sous-requête qui retourne un seul résultat à la requête principale.

Syntaxe SQL :

```
SELECT *
FROM nom_table
WHERE nom_colonne = (
SELECT TOP (1) nom_colonne
FROM nom_table2
)
```

Cet exemple montre une requête interne (celle sur "table2") qui renvoi une seule valeur. La requête externe quant à elle, va chercher les résultats de "table" et filtre les résultats à partir de la valeur retournée par la requête interne.

Exemple :

```
SELECT Nom,Prenom
FROM Acteur
WHERE NOM = (
SELECT TOP (1) Nom_acteur FROM ROLE
WHERE Nom_Role='Mr White'
)
```

A noter : Il est possible d'utiliser n'importe quel opérateur d'égalité tel que =, >, <, >=, <= ou <>.

b. Requête imbriquée qui retourne une colonne

Une requête imbriquée peut également retournée une colonne entière. Dès lors, la requête externe peut utiliser la commande « IN » pour filtrer les lignes qui possèdent une des valeurs retournées par la requête interne. L'exemple ci-dessous met en évidence un tel cas de figure :

```
SELECT *
FROM nom_table
WHERE nom_colonne IN (
SELECT nom_colonne
FROM nom_table2
WHERE condition
)
```

Exemple :

```
SELECT Nom_acteur, ID_film
FROM Role
WHERE ID_film IN (
SELECT ID_film FROM Film
WHERE Nom_Realisateur='Woo'
)
```