Formation T-SQL (Transact-SQL)

Prérequis pour la formation

- Instance SQL-Server
- SSMS
- Support de cours

احمد رضا OK
Alfoussein OK
Ali-Haïdar 2 OK
Aser 1/2 OK
Denis OK
Jade 1/2 OK
Nicolas OK
Romain OK
Sylvain ε-KO
Valentin OK

Comprendre l'architecture de Microsoft SQL Server

Architecture de base de SQL Server

SQL-Server : SGBDR de Microsoft (**SGBDR** : **S**ystème de **G**estion de **B**ases de **D**onnées **R**elationnelles) Base de Données Relationnelles

- Une relation est tout simplement une table !
- Les données sont stockées dans des tables.
- Les données répondent à une structure prédéfinie.

SQL : **S**tructured **Q**uery **L**anguage

No-SQL : Not **Only** SQL - Bases SchemaLess (Sans schéma - Non structurée -> Big Data) (MongoDB, Cassandra, Hadoop, ...)

Aujourd'hui, la plupart de SGBDR intègrent des fonctionnalités NoSQL.

Éditions et versions SQL Server

Après la version 7, SQL Server est millésimé depuis la version 2000 (2000, 2005, 2008, 2008-R2, 2012, 2014, 2016, 2017, 2019).
Chaque version se décline en éditions

- **Express** : gratuite mais limitée
- **Web** : suffisante pour la gestion d'un site Web
- **Standard** : certaines fonctionnalités sont limitées
- **Enterprise** : édition complète
- **Developper** : version gratuite de l'édition Enterprise tant que l'on ne l'utilise pas en production.

La version et l'édition sont obtenues par la requête suivante :

```
SELECT @@VERSION;
```

Démarrer avec SQL Server Management Studio (SSMS)

Pour s'interfacer avec SQL-Server, on peut utiliser

- **SSMS**
- **SQLCMD** (ligne de commande)
- Tout langage dès lors que l'on dispose du pilote adapté (PHP, .NET, ...)

Une instance est une installation d'une version et d'une édition de SQL-Server.
Sur une système, il peut exister plusieurs instances

- pouvoir avoir des versions/éditions différentes
- séparer les bases de comptes (la base de compte est propre à l'instance)
- configurations différentes (mémoire, stockage, processeurs, ...)

Découvrir le Transact-SQL (T-SQL)

Présenter T-SQL

SQL: Structured Query Language
Langage déclaratif : on écrit ce que l'on veut, le moteur se charge de déterminer la manière d'y accéder.

L'implémentation SQL de Microsoft, utilisée au sein de SQL-Server :

- comprend un grand nombre d'éléments conformes aux différents standards SQL
- comprend des éléments spécifiques à T-SQL

Le langage SQL est apparu en 1970 et est toujours présent aujourd'hui et continue d'évoluer...
Ce langage est composé de trois sous-langages

```
SQL
+----------------------------------+----------------------------------
+
DDL DML DCL
Data Definition Language Data Manipulation Language Data Control
Language
CREATE INSERT GRANT
ALTER UPDATE REVOKE
DROP DELETE DENY
SELECT
```

Comprendre les ensembles

Une table est une relation, un ensemble de lignes mettant en relation des valeurs de 1 ou plusieurs colonnes.

Comprendre les prédicats logiques

Un prédicat est l'expression d'une condition utilisée dans la clause **WHERE** d'une requête SQL (SELECT, UPDATE, DELETE)

Comprendre l'ordre logique des opérations dans les instructions SELECT

```
0 WITH
5 SELECT liste de colonnes à afficher -- critères de projection
1 FROM sources des données (une ou plusieurs tables) -- où trouve-t-on
les lignes
2 WHERE critères de sélection de lignes -- opération de restriction
3 GROUP BY liste de colonnes -- opération de regroupement
4 HAVING critères de sélection de regroupement -- restriction sur
regroupement
6 ORDER BY critères de tri d'affichage
7 OFFSET lignes à évacuer
8 FETCH lignes à conserver
```

Création d'une base d'exemple

Structure d'une base

Une base est découpée en schémas regroupant les divers objets que sont les tables, les vues, les fonctions et autres procédures stockées. Au sein d'un même schéma il ne peut y avoir deux objets portant le même nom. Par contre, deux objets peuvent avoir le même nom s'ils ne sont pas dans le même schéma.
Par défaut, une base est créée avec le seul schéma 'dbo' dans lequel seront créer tous les objets.
Il est possible de créer de nouveau schéma en fonction des besoins demandés.
En principe, le nom d'une table devrait être préfixé par le nom de son schéma, par exemple hr.salaries représente la table salaries dans le schéma 'hr'.

Tout utilisateur de base dispose d'un schéma par défaut dans lequel seront recherchés les objets non qualifiés.

Structure d'une table
Une table est composée des éléments suivants :

- colonnes ou champs ou attributs : les données qu'il sera possible de stocker dans la table
- lignes, enregistrements ou records (anglais) : ensemble des données affectées à plusieurs colonnes

Une colonne est définie par les éléments suivants :

- **un nom** : composé de caractères alpha-numériques et le caractère '_'. Pas de distinction minuscule/majuscule
- **un type** : nature des données autorisées
    - **INT** : données numériques entières (valeurs sans partie décimale) (taille de 4 octets : environ 4 milliards de valeurs différentes)
    - **BIGINT** : données numériques entières (taille de 8 octets : 9 milliards de milliards de valeurs différentes)
    - **NUMERIC(L,P)** : données numériques décimales (de l chiffres dont p chiffres pour la partie décimale)
    - **CHAR(L)** : texte de L caractères maximum. Une donnée de type CHAR(L) sera tjrs stockée sur L caractères (on complète éventuellement par des espaces).
    - **VARCHAR(L)** : texte de L caractères maximum. Ne seront stockés que les caractères de la données.
    - **NCHAR(L)** : caractères en codage UTF-8
    - **NVARCHAR(L)** : caractères en codage UTF-8 (multiplie par deux l'espace consommé)
    - **DATE**
    - **DATETIME, DATETIME2**
    - **TIME**
    - ...
- Contrainte
    - **NOT NULL** : non nullité -> impose la présence d'une valeur
    - **UNIQUE** : pas de valeurs en double
    - **PRIMARY KEY** : combine NOT NULL et UNIQUE (Une seule contrainte PRIMARY KEY par table, alors qu'on avoir plusieurs combinaisons NOT NULL UNIQUE.
    - **FOREIGN KEY** : Clé étrangère ou "contrainte d'intégrité référentielle"

Exemple d'une table : création de la table stagiaire

```
CREATE TABLE stagiaire (
id INT PRIMARY KEY IDENTITY(1,1),
prenom NVARCHAR(20) NOT NULL,
nom NVARCHAR(20) NOT NULL
);
```

Exemple n°2 : création de la table formation

```
CREATE TABLE formation (
id INT PRIMARY KEY IDENTITY(1,1),
titre NVARCHAR(100) NOT NULL UNIQUE,
nbjrs INT NOT NULL DEFAULT 3
);
```

Exemple n°3 : création de la table calendrier

```
CREATE TABLE calendrier (
id INT PRIMARY KEY IDENTITY(1,1),
id_f INT NOT NULL REFERENCES formation(id),
deb DATE NOT NULL
);
```

Exemple n°4 : création de la table inscription

```
CREATE TABLE inscription(
id INT PRIMARY KEY IDENTITY(1,1),
id_stagiaire INT NOT NULL REFERENCES stagiaire(id),
id_calendrier INT NOT NULL REFERENCES calendrier(id),
UNIQUE (id_stagiaire, id_calendrier)
);
```

Insertion, modification et la suppression des données

```
INSERT INTO stagiaire (prenom, nom) VALUES
```

- ('Ahmed-Reda','Mokhtari'),
- ('Alfoussein','Doucoure'),
- ('Ali-Haïdar','Atia'),
- ('Aser','Perou'),
- ('Denis','Kuçuk'),
- ('Jade','Da Silva Lima'),
- ('Nicolas','Potier'),

- ('Romain','Maury'),
- ('Sylvain','Janet'),
- ('Valentin','Nguyen');

```
INSERT INTO formation (titre, nbjrs) VALUES
```

- ('Ecriture de requêtes T-SQL',5),
- ('T-SQL avancé',3),
- ('Initiation au kayak',3),
- ('Perfectionnement kayak',2),
- ('Découvrir le tango vertical',2);

Peuplement de la table calendrier

La formation n°1 a lieu à partir d'aujourd'hui
La formation n°2 aura lieu à partir du 19 octobre 2020
La découverte du tango verticale se déroulera à partir du 17 octobre 2020.

Rem: les dates sont des chaînes de caractères au format YYYY-MM-DD ou YYYYMMDD

```
INSERT INTO calendrier (id_f, deb) VALUES
```

- (1,'2020-10-12'),
- (2,'2020-10-19'),
- (5,'2020-10-17');

Peuplement de la table inscription

Les 5 premiers suivent la programmation n°1
Les 4 premiers suivent la programmation n°2
Les 6 derniers la programmation n°3

(on ne peut pas déterminer l'ordre d'affichage d'un SELECT ==> utiliser **ORDER BY**)

```
INSERT INTO inscription (id_stagiaire, id_calendrier) VALUES
```

- (1,3),(2,3),(3,3),(4,3),(5,3),
- (1,4),(2,4),(3,4),(4,4),
- (5,5),(6,5),(7,5),(8,5),(9,5),(10,5);

Écrire des requêtes SELECT

Écrire des instructions SELECT simples

```
SELECT liste de colonnes
FROM table
WHERE critère(s)
```

Quelques types critères

- opérations logiques : =, <, >, >=, <=, <>, !=
- comparaison à des modèles : LIKE et NOT LIKE
    - % : signifie une chaîne quelconque de caractères, y compris la chaîne vide
    - _ : signifie un caractère quelconque
    - [...] : un caractère quelconque, compris dans la liste entre crochets
    - [^...] : un caractère quelconque non compris dans la liste entre crochets
- Combinaison de critères avec les opérateurs AND et OR
- La négation d'un critère est effectuée par l'opérateur NOT
- Opérateur IN pour une liste de valeurs possibles (identique à plusieurs OR)
- valeur BETWEEN deb AND fin (deb et fin sont comprises dans les bornes)
- IS NULL ou IS NOT NULL pour rechercher les NULL

Exos

- Afficher les prénoms ayant une voyelle en deuxième caractères
- Sur la base world
    - Liste des noms de ville ayant plus de 1 000 000 d'habitants
    - Liste des pays européens
    - Liste des pays dont le nom commence par A et se termine par A
    - Liste des langues de la Bolivie (BOL)

Éliminer les doublons avec DISTINCT

Exemple d'utilisation de la clause DISTINCT

```
SELECT DISTINCT Language
FROM countrylanguage
WHERE CountryCode IN ('BOL','COL','PER');
```

La clause DISTINCT figure obligatoirement immédiatement après l'ordre SELECT.
La clause DISTINCT supprime les lignes en doublon sur l'ensemble des colonnes.

Utiliser les alias de colonnes et de tables

Exemple de requête utilisant les alias

```
SELECT c.Name AS Ville, c.Population
FROM world.dbo.city c
WHERE c.Population > 1000000;
```

Écrire des expressions CASE

Première forme

```
SELECT Language AS Langue
, CASE Countrycode
WHEN 'BOL' THEN 'Bolivie'
WHEN 'COL' THEN 'Colombie'
WHEN 'PER' THEN 'Pérou'
ELSE 'Autre pays'
END AS Pays
FROM countrylanguage
WHERE CountryCode IN ('BOL','COL','PER');
```

Deuxième forme

```
SELECT name
, CASE
   • WHEN Population < 2000000 THEN 'Petit pays'
   • WHEN Population < 5000000 THEN 'Pas mal'
   • WHEN Population < 10000000 THEN 'Waouh !'
   • WHEN Continent = 'Asia' THEN 'Ah bon !'
   • ELSE 'Non ?'
END
FROM country;
```

Trier et filtrer les données

Trier des données : Clause ORDER BY

NB : On ne peut pas prédire l'ordre d'affichage des données.
Si on souhaite un ordre d'affichage précis il faut alors l'indiquer avec la clause ORDER BY.

La clause ***ORDER BY***, de par sa position dans le traitement, il est possible d'utiliser les alias dans cette clause.
Le tri est par défaut un tri croissant (***ASC***). Un tri décroissant impose la mention ***DESC***.
En cas d'aexquo sur les valeurs d'une clé, on ajouter une ou plusieurs autres clés pour affiner l'affichage.

Rem : on peut trier l'affichage selon une colonne non affichée.

Filtrer avec les options TOP et OFFSET-FETCH : Fenêtre d'affichage

La clause TOP permet de n'afficher que les X premières lignes ou X premiers pourcents.

```
SELECT TOP 10 name
FROM country
ORDER BY len(name);

SELECT TOP 10 WITH TIES name
FROM country
ORDER BY len(name);

SELECT TOP 10 PERCENT name
FROM country
ORDER BY len(name);
```

La clause OFFSET/FETCH dépend d'ORDER BY

```
SELECT name
FROM country
ORDER BY len(name)
OFFSET 10 ROWS
FETCH NEXT 5 ROWS ONLY;
```

Grouper et agréger des données

Utiliser les fonctions d'agrégation

Une fonction d'agrégation agrège l'ensemble des données d'une colonne en UNE SEULE VALEUR.
Une fonction scalaire (fonctions classiques telles que LEN(), DATALENGTH(), ...) affiche une valeur pour chacune des valeurs d'une colonne.

Les fonctions d'agrégation usuelles :

- **count()**
    - count(**col**) => compte les lignes pour lesquelles la colonne est non NULL
    - count(*****) => compte l'ensemble des lignes
- **sum(col)**
- **min(col)**
- **max(col)**
- **avg(col)**

Utiliser la clause GROUP BY

```
SELECT continent, sum(population)
FROM country
GROUP BY continent;
```

Il est possible de faire des regroupements sur plusieurs colonnes

```
SELECT continent, region, sum(population)
FROM country
GROUP BY continent, region
ORDER BY continent, region;
```

La clause **ROLLUP()** permet de dégrouper :

```
SELECT continent
, region
, sum(1.0 * population)
FROM country
GROUP BY ROLLUP(continent, region)
ORDER BY continent, region;
```

Filtrer les groupes avec HAVING

Une restriction sur le résultat d'une fonction d'agrégation issue d'un **GROUP BY** s'effectue avec la clause **HAVING** :

```
SELECT continent
, region
, sum(1.0 * population) AS PopTot
FROM country
```

```
GROUP BY continent, region
HAVING sum(population * 1.0) > 100000000
ORDER BY continent, region;


SELECT count(DISTINCT col.Language) AS nb_Langues
FROM countrylanguage col;

SELECT count(col.Language) AS nb_Langues, col.CountryCode
FROM countrylanguage col
GROUP BY col.CountryCode
ORDER BY nb_Langues DESC, col.CountryCode ASC;
```

Ecrire des sous-requêtes

Écrire des sous-requêtes

Une sous-requête est tout simplement une requête imbriquée dans une autre requête. Cela permet d'éviter de mettre des données en dur en utilisant plutôt les requêtes permettant de calculer ces valeurs. De la sorte la requête générale reste évolutive au sens de la mise à jour des données.

On peut une sous-requête partout où des données sont attendues :

- dans la liste des colonnes derrière le SELECT (ne doivent retourner qu'une seule valeur)
- dans la clause WHERE (peuvent retourner plusieurs lignes à condition d'utiliser l'opérateur IN)
- dans la clause ORDER BY
- ...
- derrière le FROM (à condition d'affecter obligatoirement un alias à la sous-requête)

Exemples

```
SELECT Name
FROM country
WHERE Population = (SELECT max(population) FROM country);

SELECT CountryCode
FROM countrylanguage
WHERE Language IN (SELECT Language FROM countrylanguage WHERE
CountryCode = 'BOL');
```

Sous-requêtes derrière le FROM

Une sous-requête derrière le FROM étant considérée comme une table, elle doit avoir un nom qui lui affecté par le biais d'un alias. En l'absence de cet alias, SQL-Server affichera un message d'erreur.

```
SELECT count(tbl.Language)
FROM (SELECT DISTINCT Language FROM countrylanguage) tbl
```

Écrire des sous-requêtes corrélées

Une sous-requête corrélée est une sous-requête dépend d'informations qui lui seront transmises par la requête externe. Une telle sous-requête n'est donc plus indépendante.

```
SELECT CountryCode ------------------------------|
V
, (SELECT name FROM country WHERE code = CountryCode) Pays
FROM countrylanguage
WHERE Language IN (SELECT Language FROM countrylanguage WHERE
CountryCode = 'BOL');
```

Utiliser le prédicat EXISTS avec les sous-requêtes

Ce prédicat s'utilise dans la clause **WHERE** et permet de tester si une requête retourne au moins une ligne.

```
SELECT cty.Name
FROM country cty
WHERE NOT EXISTS(SELECT 1
FROM city c
WHERE c.CountryCode = cty.Code);
```

Exercices sur les sous-requêtes

```
-- Pour la bolivie, calculer le nombre de locuteurs de chaque langue
SELECT (SELECT Population FROM country WHERE Code = CountryCode) *
(Percentage / 100) AS PopLoc
, Language
FROM countrylanguage
WHERE CountryCode = 'BOL';
```

```
-- Pour chaque pays, calculer le nombre de locuteurs de chaque langue
SELECT (SELECT Name FROM country WHERE Code = CountryCode) AS Pays
, Percentage * (SELECT Population FROM country WHERE Code = CountryCode)
/ 100 AS PopLoc
, Language
FROM countrylanguage
ORDER BY CountryCode;
```

Écrire des requêtes sur des tables multiples

Introduction
Il est possible et même fréquent d'utiliser plusieurs dans un même SELECT. La syntaxe de base est la suivante :

Produit cartésien

Le nombre de lignes résultant est le produit du nb de lignes de chaque table
Le produit cartésien de deux tables de seulement 1000 lignes chacune produira un résultat de 1 000 000 de lignes !!!

```
SELECT * FROM inscription, calendrier;
```

Comprendre les jointures

Voir document draw.io

Requêtes avec des jointures internes

-- Nombre de langues par pays

```
SELECT name, count(*) AS NbLg
FROM countrylanguage
JOIN country ON countrycode = code
GROUP BY name
ORDER BY NbLg DESC;
```

-- Exemple avec sous-requête

```
SELECT countrycode,(SELECT name
```

- FROM country
- WHERE code = countrycode) AS name, count(*) AS NbLg

```
FROM countrylanguage
GROUP BY countrycode
ORDER BY NbLg DESC;
```

Remarque : on peut tout à fait effectuer plusieurs jointures dans une requête

Requêtes avec des jointures externes

Requêtes avec des produits cartésiens et des auto-jointures

Exercices jointures

Nombre de villes par continent et par pays

```
SELECT CASE
WHEN continent IS NULL THEN ' Poptot mondiale'
ELSE continent
END AS Continent
,CASE
WHEN country.name IS NULL THEN 'Poptot ' + continent
ELSE country.name
END AS Pays
,count(*) AS "Nombre de villes"
FROM Country JOIN City ON country.code = city.CountryCode
GROUP BY ROLLUP(country.continent, country.name)
ORDER BY continent, country.name;
```

Afficher pour chaque stagiaire inscrit : La/Les formations prévues, la date de début, la durée, et la date de fin (optionnel)

```
SELECT s.nom,
```

- s.prenom,
- f.titre,
- c.deb AS Date_Debut,
- f.nbjrs AS Durée,
- DATEADD(day, f.nbjrs, c.deb) AS Date_Fin

- FROM inscription i
  - JOIN stagiaire s ON i.id_stagiaire = s.id
  - JOIN calendrier c ON i.id_calendrier = c.id
  - JOIN formation f ON c.id_f = f.id;

Afficher le titre des formations programmées mais sans inscrit

```
SELECT f.titre AS "Formations programmer sans inscrit"
```

```
FROM calendrier c
JOIN formation f ON c.id_f = f.id
LEFT JOIN inscription i ON c.id = i.id_calendrier
WHERE i.id_stagiaire IS NULL;
```

Utiliser des expressions de tables

Une expression de table est une source de données, figurant derrière la clause FROM.
Jusque là nous avons utilisé les tables et les sous-requêtes.

Tables temporaires

Une table temporaire est une table créée à la demande par l'utilisateur qui sera automatiquement détruite à la fin de la session et qui n'est visible, selon les cas, que :

- dans la session courante

ou

- pour toutes les sessions de l'utilisateur courant

Création d'une table temporaire limitée à la seule session courante :

```
CREATE TABLE #t1 (id int)
```

Création d'une table temporaire limitée à toutes les sessions de l'utilisateur courant :

```
CREATE TABLE ##t1 (id int)
```

Utiliser les vues

Une vue est une requête à laquelle on a donné un nom.
Intérêts d'une vue :

- donner un nom à une requête réutilisable (cela en masque la complexité)
- permet de n'afficher que certaines colonnes d'une table

```
CREATE VIEW pays AS SELECT name,continent,code,population FROM country
```

Selon sa définition, une vue peut être accessible en modification (INSERT, DELETE, UPDATE).
La clause de définition **WITH CHECK OPTION** limite les insertions aux valeurs respectant la restriction définie dans la définition de la vue.

```
CREATE VIEW v1 AS SELECT * FROM t2 WHERE id < 11 WITH CHECK OPTION;
```

Dès lors, toute insertion de valeurs ne correspondant à la restriction produit une erreur :

```
INSERT INTO v1 VALUES (30);
Msg 550, Niveau 16, État 1, Ligne 7
Les instructions INSERT ou UPDATE ont échoué parce que la vue cible
spécifiait WITH CHECK OPTION ou recouvrait une vue spécifiant WITH CHECK
OPTION, alors qu'une ou plusieurs lignes résultant de l'opération
n'étaient pas qualifiées sous la contrainte CHECK OPTION.
L'instruction a été arrêtée.
```

Utiliser les fonctions de table en ligne

Fonction de table en ligne est la traduction de TVF : Table Valued Function
Il s'agit de fonctions retournant, non pas une valeur, mais une table. Ces fonctions sont donc utilisables derrière la clause FROM.

Utiliser les tables dérivées

Exemple d'auto-jointure sur une table dérivée

```
SELECT al1.codepays, al1.année, al1.montanttot, al2.montanttot
FROM (SELECT codepays, year(jour) Année, sum(montant) MontantTot
FROM t1
GROUP BY year(jour), codepays) al1
LEFT JOIN
(SELECT codepays, year(jour) Année, sum(montant) MontantTot
FROM t1
GROUP BY year(jour), codepays) al2
ON al1.année = al2.année + 1 and al1.codepays = al2.codepays
ORDER BY codepays, année
```

Inconvénients d'une table dérivée

- rend la lecture/écriture peu aisée
- nécessité de ré-écriture dans le cas d'une auto-jointure

-> il y a tout intérêt à recourir aux CTE

Utiliser les expressions de tables "courantes"

Expression de table "courantes" est la traduction de CTE : Common Table Expression

```
WITH macte AS (
SELECT codepays, year(jour) Année, sum(montant) MontantTot
FROM t1
GROUP BY year(jour), codepays
) SELECT al1.codepays, al1.année, al1.montanttot, al2.montanttot
FROM macte al1
LEFT JOIN
macte al2
ON al1.année = al2.année + 1 and al1.codepays = al2.codepays
ORDER BY codepays, année
```

Avantages

- la sous-requête est mis introduction au SELECT : facilité de lecture/écriture
- la sous-requête n'est écrite qu'une seule fois et peut être utilisée plusieurs grâce au nom qui lui est attribué.
- Une CTE peut être récursive
- Plusieurs CTE sont possibles dans une même requête

Exemple simple d'une CTE récursive (affichage des entiers de 1 à 10)

```
WITH macte AS (
-- Requête d'ancrage
SELECT 1 as nb
UNION ALL
-- Requête récursive
SELECT nb + 1 FROM macte WHERE nb < 10
) SELECT * FROM macte;
```

-- Population urbaine de chaque pays

```
SELECT country.name AS Pays
, sum(city.Population) AS "Population urbaine"
FROM city
JOIN country ON city.CountryCode = country.Code
GROUP BY country.name
ORDER BY country.name
```

-- Calculer le pourcentage de population urbaine par pays

```
SELECT country.name AS Pays
```

- `, sum(city.Population) AS "Population urbaine"`
- `, sum(city.Population * 1.0)/ country.Population AS "Pourcentage de population urbaine"`

```
FROM city
JOIN country ON city.CountryCode = country.Code
GROUP BY country.name, country.Population
ORDER BY country.name
```

-- Avec une CTE

```
WITH macte AS (
SELECT countrycode, sum(population) urbanPop
FROM city
GROUP BY countrycode
)
SELECT p.name
,p.population
,c.urbanpop
,urbanpop * 100.0 / population
FROM country p
JOIN
macte c ON c.countrycode = p.code
ORDER BY name;
```

Employer des opérateurs de jeu

Notion de jeu

Un jeu est tout simplement le résultat d'une requête.

Opérateur de jeu

C'est un opérateur permettant de combiner le résultat de deux requêtes

- **UNION**
- **INTERSECT**
- **EXCEPT**

Les résultats des deux requêtes combinées par un opérateur de jeu doivent impérativement avoir la même structure, tant au niveau du nombre de colonnes qu'au niveau du type de chacun d'elle.

Écrire des requêtes avec l'opérateur UNION

L'opérateur UNION concatène les deux résultats.
Par défaut UNION supprime les doublons (tri implicite => coût)
UNION **ALL** conserve les éventuels doublons.

Si l'on a la garantie de l'absence de doublons, UNION ALL est plus performant de par l'absence du tri.

Utiliser INTERSECT

Permet de lister les lignes communes dans les deux jeux.

Utiliser EXCEPT

Afficher les lignes présentes dans le premier jeu et absentes du second.

NB: sur **Oracle** cet opérateur se nomme **MINUS**.

Exercices sur les opérateurs de jeux

```
-- Liste des pays parlant l'italien

SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language = 'italian';

-- Liste des pays parlant l'anglais

SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language = 'english';

-- Liste des pays parlant l'anglais ou l'italien

SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language IN ('italian','english');
```

```
-- Liste des pays parlant l'anglais et l'italien

SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language = 'italian'
INTERSECT
SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language = 'english';

-- Liste des pays parlant l'anglais ou l'italien (ou exclusif)

SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language IN ('italian','english');
EXCEPT
(
SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language = 'italian'
INTERSECT
SELECT name
FROM country
JOIN countrylanguage ON code = countrycode
WHERE language = 'english'
);
```

Utiliser **APPLY**

L'opérateur APPLY correspondant au principe des jointures latérales du standard SQL.

```
CREATE FUNCTION popprct_t (@code CHAR(3)) RETURNS @tbl TABLE(code
CHAR(3), prct INT)
BEGIN
DECLARE @percent INT;

SELECT @percent = sum(c.population) * 100.0 / (
```

- SELECT population FROM country WHERE code = **@code**)
- FROM city c

    - WHERE countrycode = **@code**;
- INSERT INTO **@tbl** VALUES (**@code**, **@percent**);
- RETURN

```
END
GO

-- Pourcentage de population urbaine de chaque pays ayant des villes

SELECT DISTINCT p.name, prct
FROM country p
LEFT JOIN city ON code = countrycode
CROSS APPLY dbo.popprct_t(code)
WHERE countrycode IS NOT NULL;
```

Utiliser des fonctions de fenêtrage

Rappel : revoir la définition d'une fonction scalaire et d'une fonction d'agrégation.

```
-- Fonction scalaire
SELECT len(name) FROM country;

-- Fonction d'agrégation
SELECT count(name) FROM country;

-- Ce qui est impossible
SELECT name, count(name) FROM country -- Inimaginable en l'état
```

Une fonction de fenêtrage s'apparente à une fonction d'agrégation s'applique sur un ensemble de lignes appelée "**fenêtre**" et défini par la clause **OVER**().
Ce qui est à remarquer est que les lignes conservent leur individualité tout en étant associées à une fonction genre "**agrégation**".

Utiliser la clause OVER

La clause OVER() permet de définir l'étendue de la fenêtre, son éventuelle ouverture progressive ainsi que le nombre de lignes avant et après la ligne courante.
OVER() : définit une fenêtre égale à la totalité de la table
OVER(PARTITION BY colonne) : définit une fenêtre comprenant toutes les lignes ayant la même valeur que la ligne courante dans la colonne mentionnée.

```
SELECT t.INTITULE,
i.jour,
i.NOTE,
avg(i.NOTE) OVER() AS MoyenTot,
```

```
avg(i.NOTE) OVER(PARTITION BY t.INTITULE) AS MoyenAtel
FROM atelier t
JOIN activite a ON t.NO_ATEL = a.NO_ATEL
JOIN inscription i ON i.NO_ATEL = a.NO_ATEL AND a.JOUR = i.JOUR
ORDER BY t.INTITULE, i.jour;
```

OVER(ORDER BY col) : la fenêtre s'ouvre peu à peu en prenant les lignes précédentes, la ligne courante ainsi que les lignes suivantes si elles ont la même valeur que la ligne courante.

OVER(ROWS BETWEEN X PRECEDING AND Y FOLLOWING) : la fenêtre est constituée des X lignes précédentes et des Y lignes suivantes.
OVER(ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)

OVER(RANGE BETWEEN X PRECEDING AND Y FOLLOWING)

Explorer des fonctions de fenêtrage

```
count(*)
sum(col)
min(col)
max(col)
avg(col)
std(col)

row_number() -- nécessite un OVER(... ORDER BY...)
rank() -- idem
dense_rank() -- idem
ntile(nb) -- idem

first_value(col) -- idem
last_value(col) -- idem
lag(col) -- idem
lead(col) -- idem
```

Implémenter et exécuter des procédures stockées

Fonction et procédure stockées

Stockées : cela signifie que les fonctions et procédures sont des objets stockés dans la base -> ils sont donc disponibles pour tout client.

Une fonction se distingue d'une procédure en ce qu'elle renvoie une valeur, ou une table, et qu'elle s'utilise au sein d'une requête.

```
-- Exemple
CREATE FUNCTION popprct_t (@code CHAR(3)) RETURNS @tbl TABLE(code
CHAR(3), prct INT)
BEGIN
DECLARE @percent INT;
SELECT @percent = sum(c.population) * 100.0 / (SELECT population
FROM country WHERE code = @code)
FROM city c
WHERE countrycode = @code;
INSERT INTO @tbl VALUES (@code, @percent);
RETURN
END

SELECT * FROM popprct_t('CUB');
```

A l'inverse une procédure ne renvoie pas nécessaire un résultat et ne peut être utilisée directement au sein d'une requête.

Interroger les données avec les procédures stockées
Passer des paramètres aux procédures stockées
Créer des procédures stockées simples

Travailler avec SQL Dynamique

Permet de créer des requêtes de manière dynamique. Une requête sera construire, peu à peu, en tant que chaîne de caractères pour être ensuite exécutée à l'aide de la procédure stockée ***sys.sp_executesql***.

```
--Création
CREATE PROCEDURE montanttot_2 (@table NVARCHAR(10),
                                                    • @code CHAR(2),
                                                    • @annee INT,
                                                    • @total INT OUTPUT)
AS
DECLARE @req NVARCHAR(200);
SET @req = 'SELECT @tot = sum(montant) FROM '+ @table + ' WHERE
year(jour) = '+ CAST(@annee
                                                    • AS NCHAR(4)) + ' AND
                                                      codepays='''+ @code +
                                                      '''';
PRINT @req;
EXECUTE sys.sp_executesql @statement=@req, @params=N'@tot INT OUTPUT',
@tot = @total OUTPUT;
GO
```

```
--Exécution
DECLARE @tot INT
EXECT montanttot_2 @table = 't1', @code='LU', @annee=2000, @total=@tot
OUTPUT;
SELECT @tot;
```

Programmer avec T-SQL

Éléments de programmation T-SQL
Contrôler le flux des programmes

Gérer des transactions

Les transactions et les moteurs de base de données

Une transaction fait passer une base d'un état cohérent à un autre état cohérent.

Contrôler les transactions

Il est possible de créer manuellement une transaction.
Une transaction se définit par :
BEGIN TRAN[SACTION]
Ordre 1
Ordre 2
...
SAVE TRAN Name_tran
...
ROLLBACK TRAN Name_tran
...
COMMIT (validation) ou ***ROLLBACK*** (annulation)

A ce moment, toutes les instructions exécutées dans la transactions auront été toutes validées ou toutes annulées.

Niveaux d'isolation

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED -- Pas d'isolation
SET TRANSACTION ISOLATION LEVEL READ COMMITED -- Niveau par défaut
(verrouillage sur SQLServer) (Pour éviter le verrouillage activer
l'option 'Read Commited Snapshot Isolation' dans la BD
```

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
SET TRANSACTION ISOLATION LEVEL SNAPSHOT -- Tous les select au sein
d'une transaction seront identiques, à condition d'avoir activé l'option
de base de données 'Autoriser l'isolement d'instantané
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Atelier récapitulatif

```
-- Liste des ateliers et leur animateur dont le genre est ‘SCIENCES’

SELECT t.intitule, t.genre, a.nom, a.prenom
FROM Atelier t
JOIN Animateur a ON t.NO_ANIM = a.NO_ANIM
WHERE t.genre = 'SCIENCES';

-- Liste des inscriptions pour les adhérents de la ville de Nantes

SELECT intitule, jour, nom, prenom , i.DATE_INSCRIPTION
FROM inscription i
JOIN adherent a ON i.NO_ADHER = a.NO_ADHER
JOIN atelier at ON at.NO_ATEL = i.NO_ATEL
WHERE VILLE = 'Nantes'
ORDER BY INTITULE, JOUR, NOM,PRENOM, DATE_INSCRIPTION;

-- Moyenne des notes par atelier
SELECT a.INTITULE, avg(i.NOTE) OVER(PARTITION BY a.NO_ATEL) as MoyByAt
FROM atelier a
JOIN inscription i ON a.NO_ATEL = i.NO_ATEL

SELECT a.INTITULE, avg(note) as MoyByAt
FROM atelier a
JOIN inscription i ON a.NO_ATEL = i.NO_ATEL
GROUP BY a.INTITULE, a.NO_ATEL
-- Le recours à l'attribut no_atel permet l'optimisation de la requête
par l'utilisation de la clé primaire

-- Nombre des inscriptions par ville et par atelier

SELECT ville, INTITULE, count(*) as Inscrits
FROM inscription i
JOIN adherent ad ON ad.NO_ADHER = i.NO_ADHER
JOIN atelier atl ON atl.NO_ATEL = i.NO_ATEL
GROUP BY ville, INTITULE
```

```
-- La liste des animateurs n'animant aucun d’atelier

SELECT nom, prenom, tel as telephone
FROM animateur a
LEFT JOIN atelier [at] ON a.NO_ANIM = at.NO_ANIM
WHERE genre IS NULL

-- La liste des adhérents inscrits à la fois dans un atelier de SCIENCES
et un atelier de TNIC

WITH adherentsSciences AS
(
SELECT nom AS Nom
,prenom AS Prénom
,ville AS Ville
FROM adherent adh
JOIN inscription i ON i.NO_ADHER = adh.NO_ADHER
JOIN atelier ate ON ate.NO_ATEL = i.NO_ATEL
WHERE genre = 'SCIENCES'
), adherentsTNIC AS
(
SELECT nom AS Nom
,prenom AS Prénom
,ville AS Ville
FROM adherent adh
JOIN inscription i ON i.NO_ADHER = adh.NO_ADHER
JOIN atelier ate ON ate.NO_ATEL = i.NO_ATEL
WHERE genre = 'TNIC'
)
SELECT *
FROM adherentsSciences
INTERSECT
SELECT *
FROM adherentsTNIC;
```

OU

```
SELECT ad.NOM, ad.PRENOM, VILLE
FROM adherent ad
JOIN inscription i ON ad.NO_ADHER = i.NO_ADHER
JOIN atelier a ON i.NO_ATEL = a.NO_ATEL
WHERE GENRE = 'SCIENCES'
INTERSECT
SELECT ad.NOM, ad.PRENOM, VILLE
```

```
FROM adherent ad
JOIN inscription i ON ad.NO_ADHER = i.NO_ADHER
JOIN atelier a ON i.NO_ATEL = a.NO_ATEL
WHERE GENRE = 'TNIC';
```

OU

```
SELECT nom, prenom, ville
FROM adherent ad
JOIN inscription i ON ad.NO_ADHER = i.NO_ADHER
JOIN atelier a ON i.NO_ATEL = a.NO_ATEL
WHERE a.genre = 'TNIC'

INTERSECT

SELECT nom, prenom, ville
FROM adherent ad
JOIN inscription i ON ad.NO_ADHER = i.NO_ADHER
JOIN atelier a ON i.NO_ATEL = a.NO_ATEL
WHERE a.genre = 'SCIENCES';

-- La liste des adhérents qui sont dans l’atelier « Bureautique » ou
l’atelier «Dentelle» ou les deux

select nom, prenom, intitule
from adherent ad
join inscription i on ad.NO_ADHER = i.NO_ADHER
join atelier ate on i.NO_ATEL = ate.NO_ATEL
where intitule IN ('Bureautique','Dentelle')

-- La liste des adhérents qui sont dans l’atelier « Bureautique » ou
l’atelier «Dentelle» mais pas dans les deux
select nom, prenom, intitule
from adherent ad
join inscription i on ad.NO_ADHER = i.NO_ADHER
join atelier ate on i.NO_ATEL = ate.NO_ATEL
where intitule IN ('Bureautique','Dentelle')
except
(
select nom, prenom, intitule
from adherent ad
join inscription i on ad.NO_ADHER = i.NO_ADHER
join atelier ate on i.NO_ATEL = ate.NO_ATEL
where intitule = 'Bureautique'
intersect
```

```
select nom, prenom, intitule
from adherent ad
join inscription i on ad.NO_ADHER = i.NO_ADHER
join atelier a on i.NO_ATEL = a.NO_ATEL
where intitule = 'Dentelle'
);

-- La liste des ateliers qui ont plus de 2 inscrits
SELECT t.intitule, t.NO_ATEL, t.genre, count(i.NO_INSC) AS TotalInscrit
FROM atelier t
JOIN inscription i ON i.NO_ATEL = t.NO_ATEL
GROUP BY t.intitule, t.NO_ATEL, t.genre
HAVING count(i.NO_INSC) > 2
```

- - La liste des ateliers dont l'âge moyen des inscrits est supérieur à 52 ans

```
WITH adherent_age AS
(
SELECT *
, DATEDIFF(year, date_naissance, GETDATE()) as Age
FROM adherent
), atelAgeMoy AS
(
SELECT ate.no_ATEL
,intitule AS Intitulé
,avg(Age) AS "Age moyen"
FROM atelier ate
JOIN inscription i ON ate.NO_ATEL = i.NO_ATEL
JOIn adherent_age adh ON adh.NO_ADHER = i.NO_ADHER
GROUP BY ate.NO_ATEL, intitule
)
SELECT *
FROM atelAgeMoy
WHERE "Age moyen" > 52


SELECT ate.INTITULE, avg(DATEDIFF(year, DATE_NAISSANCE, GETDATE())) as
Age
FROM adherent ad
JOIN inscription i on ad.NO_ADHER = i.NO_ADHER
JOIN atelier ate on i.NO_ATEL = ate.NO_ATEL
GROUP BY INTITULE
HAVING avg(DATEDIFF(year, DATE_NAISSANCE, GETDATE())) > 52

-- Adhérent le plus jeune inscrit dans le genre « SPORT » (considérer le
```

```
cas où il y a plusieurs adhérents de même âge

-- Solution avec TOP
SELECT TOP 1 WITH TIES
i.no_adher AS No_adher
,nom AS Nom
,prenom AS Prénom
,date_naissance AS Date_naissance
FROM inscription i
JOIN adherent adh ON adh.NO_ADHER = i.NO_ADHER
JOIN atelier ate ON ate.NO_ATE L= i.NO_ATEL
WHERE GENRE = 'SPORT'
GROUP BY i.no_adher, nom, prenom, date_naissance
ORDER BY date_naissance DESC

-- Autres solutions

SELECT i.no_adher AS No_adher
,nom AS Nom
,prenom AS Prénom
,date_naissance AS Date_naissance
FROM inscription i
JOIN adherent adh ON adh.NO_ADHER = i.NO_ADHER
JOIN atelier ate ON ate.NO_ATEL = i.NO_ATEL
WHERE genre = 'SPORT'
AND date_naissance = (SELECT max(date_naissance)
FROM adherent a
JOIN inscription i ON a.no_adher = i.no_adher
JOIN atelier t ON i.no_atel = t.no_atel
WHERE t.genre = 'SPORT');



WITH inscritSPORTageranked AS
(
SELECT DISTINCT i.no_adher, nom, prenom, date_naissance, rank()
OVER(ORDER BY
date_naissance DESC) as rang
FROM inscription i
JOIN adherent adh ON adh.NO_ADHER = i.NO_ADHER
JOIN atelier ate ON ate.NO_ATEL = i.NO_ATEL
WHERE ate.genre = 'SPORT'
)
SELECT no_adher AS No_adher
,nom AS Nom
```

```
,prenom AS Prénom
,date_naissance AS Date_Naissance
FROM inscritSPORTageranked
WHERE rang = 1;

-- Liste des personnes habitant dans la même ville que M. Germain

WITH villeGermais AS
(
SELECT ville as v
FROM adherent
WHERE NOM like 'germain' and sexe = 'm'
) SELECT nom, prenom, ville FROM adherent WHERE ville = (select * from
villeGermais);

WITH villeGermain AS
(
SELECT ville
FROM adherent
WHERE NOM like 'germain' and sexe = 'm'
)
SELECT nom, prenom, a.ville
FROM adherent a
JOIN villeGermain v ON v.ville = a.ville

-- Liste des adhérents qui ont le même animateur : M. POIRIER

SELECT ad.NO_ADHER, ad.NOM, ad.PRENOM
FROM adherent ad
JOIN inscription i ON ad.NO_ADHER = i.NO_ADHER
JOIN atelier a ON i.NO_ATEL = a.NO_ATEL
JOIN animateur anim ON a.NO_ANIM = anim.NO_ANIM
WHERE anim.NOM = 'POIRIER'
--GROUP BY ad.NO_ADHER, ad.NOM, ad.PRENOM;

-- Liste des adhérents les plus âgés de chaque ville

WITH adherent_age AS
(
SELECT *
, DATEDIFF(year, date_naissance, GETDATE()) as Age
FROM adherent
), adhVilleAgeRanked AS
(
SELECT nom as Nom
```

```
,prenom as Prénom
,ville as Ville
,date_naissance AS Date_naissance
,rank() OVER(PARTITION BY ville ORDER BY Age DESC) AS rang
FROM adherent_age
)
SELECT Nom, Prénom, Ville, Date_naissance
FROM adhVilleAgeRanked
WHERE rang = 1
ORDER BY ville;

WITH adhVilleAgeRanked AS
(
SELECT nom as Nom
,prenom as Prénom
,ville as Ville
,date_naissance AS Date_naissance
,rank() OVER(PARTITION BY ville ORDER BY date_naissance) AS rang
FROM adherent_age
)
SELECT Nom, Prénom, Ville, Date_naissance
FROM adhVilleAgeRanked
WHERE rang = 1
ORDER BY ville;


SELECT nom as Nom
,prenom as Prénom
,ville as Ville
,date_naissance AS Date_naissance
FROM adherent a
WHERE date_naissance = (SELECT min(date_naissance)
FROM adherent
WHERE ville = a.ville)
ORDER BY ville;

-- Liste de la meilleure note pour chaque atelier

SELECT ad.no_adher
,ad.nom
,ad.prenom
,i.note
,i.no_atel
FROM adherent ad
JOIN inscription i ON i.NO_ADHER = ad.NO_ADHER
```

```
WHERE i.note = (SELECT max(note) FROM inscription WHERE no_atel =
i.no_atel)
ORDER BY i.no_atel;

-- Liste de la meilleure note de chaque adhérent

SELECT ad.no_adher,
       t.intitule,
       i.note,
       i.no_atel
FROM adherent ad
JOIN inscription i ON ad.NO_ADHER = i.NO_ADHER
JOIN atelier t ON i.NO_ATEL = t.NO_ATEL
WHERE i.note = (SELECT max(i.note)
FROM adherent
JOIN inscription i ON ad.NO_ADHER = i.NO_ADHER)
ORDER BY ad.no_adher;

-- Pourcentage des adhérents de chaque ville

WITH nbrAdhVille AS
(
SELECT ville AS Ville
, count(*) AS nbrAdh
FROM adherent
GROUP BY ville
)
SELECT Ville
, nbrAdh * 100.0 / (SELECT sum(nbrAdh) FROM nbrAdhVille) AS "%
adhérent / ville"
FROM NbrAdhVille;

-- Liste des ateliers par rang selon leur fréquentation (avec calcul du
rang)

SELECT t.intitule
,count(i.NO_INSC) Frequentation
,dense_rank() OVER(ORDER BY count(i.NO_INSC) DESC) Rang
FROM inscription i
JOIN atelier t ON i.NO_ATEL = t.NO_ATEL
GROUP BY t.intitule;

-- Marge par atelier (attention à la durée)

select distinct ate.intitule, sum((ate.VENTE_HEURE - COUT_HEURE) *
```

```
atl.duree)
from atelier ate
join animateur ani ON ate.NO_ANIM = ani.NO_ANIM
JOIN activite atl ON ate.NO_ATEL = atl.NO_ATEL
group by ate.intitule;

-- Cotisation hebdomadaire de chaque adhérent

WITH atelierNbrHeures AS
(
SELECT ate.no_atel AS no_atel
,intitule AS Intitulé
,sum(duree) AS nbrHeures
,VENTE_HEURE AS prixparheure
FROM atelier ate
JOIN activite act ON ate.NO_ATEL = act.NO_ATEL
GROUP BY ate.no_atel, intitule, VENTE_HEURE
), atelierCout AS
(
SELECT no_atel, Intitulé
,nbrHeures * prixparheure AS Cout
FROM atelierNbrHeures
)
SELECT adh.no_adher AS No_adher
,nom AS Nom
,prenom AS Prénom
,COALESCE(sum(Cout), 0) AS Cotisation
FROM adherent adh
LEFT JOIN inscription i ON adh.NO_ADHER = i.NO_ADHER
LEFT JOIN atelierCout aC ON aC.no_atel = i.NO_ATEL
GROUP BY adh.no_adher, nom, prenom
ORDER BY no_adher;
```

Annexes

Liste des fonctions utilisées

Fonctions scalaires

- **LEN**(...) : nombre de caractères d'une chaîne
- **DATALENGTH**(...) : nombre d'octets occupés par une chaîne

- **GETDATE**() : affiche la date et l'heure courantes
- **SUBSTRING**(chaîne, pos, long) : extraire la chaîne de longueur 'long' à partir de la position 'pos'.
- **DATEADD**(DAY, nbjrs, c.datedebut) AS DateFin
- **DATEDIFF**(YEAR,deb,fin)
- **YEAR**(date)
- **MONTH**(date)
- **DAY**(date)
- **DATEPART**(weekday, date)
- **DATENAME**(weekday, date)
- **COALESCE**(A,B,C,D)

Fonctions d'agrégation

- **count**()
- **max**(col)
- **min**(col)
- **sum**(col)
- **avg**(col)
- **stdev**(col) -- écart-type

```
ALTER TABLE City ADD FOREIGN KEY (countrycode) REFERENCES country(code);
ALTER TABLE CountryLanguage ADD FOREIGN KEY (countrycode) REFERENCES
country(code);

SET STATISTICS TIME ON : pour voir le temps d'exécution d'une requête
```