SCI6306 Bases de données documentaires (Automne 2023)

Christine Dufour, EBSI, UdeM

A2023 6 octobre 2023

Cours 5 : SQL (partie 3 de 3)

SCI6306

Christine Dufour, EBSI, UdeM

Table des matières

Cours 5 - SQL (partie 3 de 3)

- + Au programme aujourd'hui
- SQL (niveau 3)
    - Sous-requêtes
        - Scénario 1 : Utilisation d'une sous-requête dans une clause WHERE
        - Scénario 2 : Utilisation d'une sous-requête dans la liste des champs de résultats (après SELECT)
        - Scénario 3 : Utilisation d'une sous-requête comme table de données (clause FROM)
        - Exercices sur les sous-requêtes
            - Exercice 1 : Que fait cette requête?
            - Exercice 2 : Que fait cette requête?
    - Recherche plein texte
    - Requête Ajout d'enregistrements
    - Requête Mise à jour d'enregistrements
    - Requête Suppression d'enregistrements
- Ressources en lien avec le cours

1. + Au programme aujourd'hui

- **SQL – partie 3 de 3** (*niveau 3*)
    - Sous-requêtes
    - Recherche plein texte
    - Autres types de requêtes
        - Requête Ajout d'enregistrements
        - Requête Mise à jour d'enregistrements
        - Requête Suppression d'enregistrements
- Laboratoire : **TP Requêtes SQL** – niveau 3 + Partie B

2. SQL (niveau 3)

2.1. Sous-requêtes

Une requête retournant comme résultat une **table de données**, voire une valeur unique, elle peut donc être utilisée comme **sous-requête** dans une autre requête (appelée *requête principale*). On retrouve deux types de sous-requête :

- Sous-requête **corrélée**, c'est-à-dire liée par au moins un champ à la requête principale
    - Son résultat peut changer à chacun des enregistrements de la requête principale
- Sous-requête **non corrélée**, c'est-à-dire indépendante de la requête principale
    - Son résultat est le même pour tous les enregistrements de la requête principale

On peut identifier trois scénarios principaux d'utilisation de sous-requêtes, ces scénarios pouvant être cumulés :

- Sous-requête utilisée dans une **condition** (clause WHERE)
- Sous-requête utilisée pour afficher une **valeur dans les résultats** (dans la liste des champs affichés après SELECT)
- Sous-requête utilisée comme **table de données** (clause FROM)

Chacun de ces scénarios sera détaillé dans les sections qui suivent.

a) Scénario 1 : Utilisation d'une sous-requête dans une clause WHERE

***Exemple* : Liste des étudiant.e.s non inscrit.e.s à des cours**

Si la table SUIT nous permet d'identifier les étudiant.e.s inscrit.e.s dans des cours, à elle seule, elle ne peut réussir à identifier les étudiant.e.s qui, au contraire, ne sont inscrit.e.s à aucun cours. En fait, cette liste d'étudiant.e.s sans inscription correspond aux étudiant.e.s qui n'appartiennent pas à la liste des inscrit.e.s! On peut ainsi utiliser la requête qui permet d'identifier la liste des étudiant.e.s inscrit.e.s comme sous-requête de la manière suivante :

```
1 SELECT no_etud, nom
2 FROM etud
3 WHERE no_etud NOT IN (SELECT no_etud FROM suit);
```

La sous-requête `SELECT no_etud FROM suit` permet d'obtenir la liste de tous les **numéros des étudiant.e.s qui suivent au moins un cours**. C'est l'utilisation dans la requête principale du `NOT IN` qui permet de ne retenir de la liste de tous les étudiant.e.s (`FROM etud`) que celles et ceux qui ne sont pas dans cette liste d'inscrit.e.s. Cette sous-requête est dite **non corrélée** comme elle s'exécute sans l'apport du reste de la requête principale. Elle est autonome et indépendante.

Une manière simple de savoir si une sous-requête est corrélée ou non corrélée est d'essayer de l'exécuter toute seule. Si cela fonctionne, elle est clairement indépendante et autonome, donc non corrélée. Si cela ne fonctionne pas car il manque certains éléments de la requête principale, elle est corrélée.

b) Scénario 2 : Utilisation d'une sous-requête dans la liste des champs de résultats (après SELECT)

***Exemple* : Liste des étudiant.e.s avec le nombre de cours suivis**

L'utilisation de la clause GROUP BY permet de compter le nombre de cours suivis pour les étudiant.e.s inscrit.e.s à des cours, mais sans pouvoir identifier les étudiant.e.s inscrit.e.s à aucun cours. Une requête UNION pourrait nous permettre d'y ajouter la liste des étudiant.e.s non inscrit.e.s, certes, mais l'utilisation d'une sous-requête permet de le faire de manière encore plus simple :

```
1 SELECT no_etud, nom, (SELECT count(*) FROM suit WHERE etud.no_etud = suit.no_etud) AS 'Nb
  cours suivis'
2 FROM etud
3 ORDER BY no_etud;
```

Comme on le voit dans cette requête, la sous-requête se retrouve dans la liste des éléments affichés dans la table des résultats. Chacune des lignes de la table des résultats présentera le numéro de l'étudiant.e, le nom de l'étudiant.e et le nombre de cours suivis calculé grâce à la sous-requête :

no_etud	nom	Nb cours suivis
10001	Wagner, Richard	1
10002	Bretécher, Claire	3
10003	Tremblay, Michel	2
10004	Asimov, Isaac	3
10005	Simenon, Georges	0
10006	Smithwick, Dale	2

Table des résultats

Examinons plus avant la sous-requête :

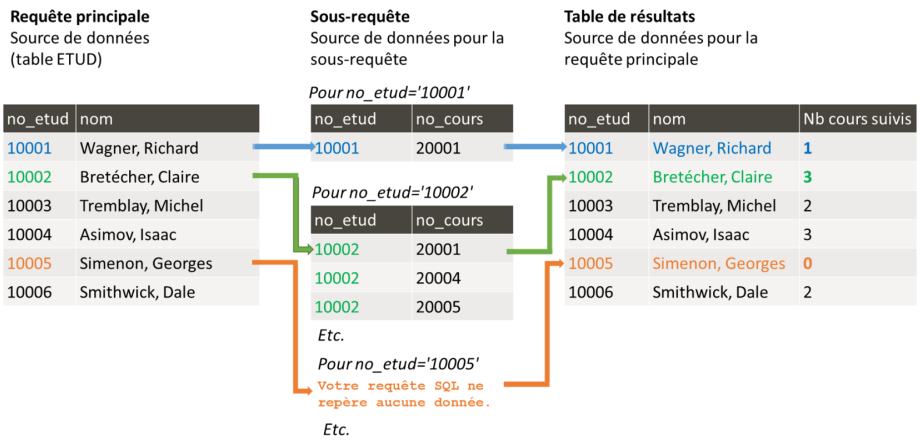
```
1 SELECT count(*) FROM suit WHERE etud.no_etud = suit.no_etud
```

- Cette sous-requête **affiche uniquement un chiffre** calculé par la fonction `COUNT()`. Pour pouvoir utiliser une sous-requête dans la liste des éléments affichés, elle doit **nécessairement** ne retourner qu'une et une seule valeur (il ne faut pas qu'elle retourne une table avec plusieurs lignes et/ou colonnes).
- Cette sous-requête puise ses données dans la **table SUIT** (clause FROM).
- Cette sous-requête possède une **condition**, soit que le numéro de l'étudiant.e dans la table SUIT soit le même que le numéro de l'étudiant.e dans la table ETUD. Il est légitime ici de se demander d'où cette table ETUD provient. On ne la retrouve en effet pas dans la clause FROM de la sous-requête... Il s'agit en fait de la **table ETUD de la clause FROM de la requête principale**! La sous-requête est de ce fait **corrélée**, c'est-à-dire qu'elle est dépendante de la requête principale. Essayez d'exécuter la sous-requête toute seule et vous verrez que le système vous retournera un message d'erreur à l'effet que le champ *etud.no_etud* que l'on retrouve dans la clause WHERE est inconnu.

Rappelons qu'une requête s'exécute de manière linéaire. Chacune des lignes de la table ETUD de la requête principale sera traitée l'une à la suite de l'autre :

- La requête débutera par l'étudiant 10001, pour afficher son numéro (*10001*), son nom (*Wagner, Richard*) et ensuite calculer le nombre de lignes dans la table SUIT qui correspond à son numéro (*1 cours*).
- Elle passera par la suite à la deuxième ligne de la table ETUD, pour afficher le numéro de l'étudiante (*10002*), son nom (*Bretécher, Claire*) et ensuite calculer le nombre de lignes dans la table SUIT qui correspond à son numéro (*3 cours*).
- Et ainsi de suite jusqu'à la fin de la table ETUD.

On voit que le résultat de la sous-requête **va varier** à **chacune des lignes** comme elle **s'adapte** au numéro d'étudiant.e de la **requête principale**.



Schématisation du processus linéaire d'exécution de la requête SQL

c) Scénario 3 : Utilisation d'une sous-requête comme table de données (clause FROM)

***Exemple* : Liste des étudiant.e.s suivant des cours sans local attitré**

Parfois notre logique nous amène vers des solutions impliquant des sous-requêtes, parfois plutôt vers des solutions avec des jointures. Par exemple, afin d'identifier les étudiant.e.s qui suivent des cours sans local attitré, vous pourriez vous dire qu'une manière de procéder est de premièrement identifier les numéros des cours n'ayant pas de local pour ensuite vérifier dans cet ensemble les inscriptions à ces cours. Cette logique est clairement une logique de sous-requête qui donnerait la requête qui suit :

```
SELECT etud.no_etud, nom
FROM (SELECT cours.no_cours FROM cours WHERE isnull(local)) as sanslocal, etud, suit
WHERE etud.no_etud=suit.no_etud AND suit.no_cours=sanslocal.no_cours;
```

Dans cette requête, la sous-requête se trouve dans la clause FROM et sert à construire la table de données qui servira de source de données. La table de résultats de la sous-requête comprendra l'ensemble des numéros de cours pour les cours n'ayant pas de local attitré. Lorsque l'on utilise une sous-requête dans la clause FROM il faut la nommer en utilisant un **alias** afin de pouvoir l'utiliser par la suite dans le reste de la requête. Dans ce cas, elle a été nommée SANSLOCAL et sert à faire une jointure avec la table des inscriptions SUIT (`suit.no_cours=sanslocal.no_cours`) afin d'aller extraire les informations sur les étudiant.e.s inscrit.e.s à ces cours sans local attitré.

Il y a manière de répondre à ce besoin autrement, sans exploiter de sous-requête pour plutôt utiliser des jointures :

```
SELECT etud.no_etud, nom
FROM cours, etud, suit
WHERE cours.no_cours=suit.no_cours AND etud.no_etud=suit.no_etud
AND isnull(local);
```

d) Exercices sur les sous-requêtes

i) Exercice 1 : Que fait cette requête?

***Simulation* :**

Examinez la requête ci-dessous pour identifier, d'une part, les **caractéristiques** de la sous-requête et, d'autre part, le **besoin sous-jacent**.

```
SELECT no_cours, titre
FROM cours
WHERE no_cours IN (SELECT no_cours FROM suit WHERE NOT note_p)
ORDER BY cours.no_cours;
```

Caractéristique de la sous-requête :

- Où apparaît-elle?
- Est-elle corrélée ou non corrélée?
- Que fait-elle?

Besoin de la requête?

ii) Exercice 2 : Que fait cette requête?

***Simulation* :**

Examinez la requête ci-dessous pour identifier, d'une part, les **caractéristiques** des deux sous-requêtes qu'on y retrouve et, d'autre part, le **besoin sous-jacent**.

```
SELECT no_prof, nom, (SELECT COUNT(*) FROM etud WHERE no_etud IN (SELECT no_etud FROM
suit, cours WHERE suit.no_cours = cours.no_cours AND cours.no_prof = prof.no_prof))
FROM prof;
```

Caractéristique de la sous-requête 1 :

- Où apparaît-elle?
- Est-elle corrélée ou non corrélée?
- Que fait-elle?

Caractéristique de la sous-requête 2 :

- Où apparaît-elle?
- Est-elle corrélée ou non corrélée?
- Que fait-elle?

Besoin de la requête?

2.2. Recherche plein texte

On retrouve dans SQL la fonction `MATCH ... AGAINST ...` qui permet de faire une **recherche dite plein texte**. Bien que l'on puisse chercher dans du texte avec ce que nous avons vu jusqu'à présent (par exemple l'opérateur LIKE et la troncature), ces outils deviennent rapidement lourds pour des besoins de recherche d'information textuelle plus complexes.

Par exemple, si on cherchait à identifier tous les cours de sciences, avec une préférence pour les mathématiques, mais pas les cours d'histoire ou de philosophie des sciences, il faudrait construire une série de conditions qui deviendrait rapidement assez longue afin de prévoir trouver les mots-clés pertinents (par exemple, math*, chimi*, physi*) dans les champs pertinents (le titre et la description du cours), ainsi que pour pouvoir enlever des résultats les cours où l'on retrouve dans le titre ou la description des mots-clés comme histo* et philosoph* :

```
1 SELECT no_cours, titre, descr
2 FROM cours
3 WHERE ((titre LIKE '%math%' OR titre LIKE '%chimi%' OR titre LIKE '%physi%') OR (descr
  LIKE '%math%' OR descr LIKE '%chimi%' OR descr LIKE '%physi%')) AND NOT ((titre LIKE
  '%histo%' OR titre LIKE '%philo%') OR (descr LIKE '%histo%' OR descr LIKE '%philo%'));
```

La recherche plein texte permet de **simplifier ce type de recherche** en proposant une syntaxe et des opérateurs particuliers. Elle fait partie des fonctions plus avancées et probablement moins connues de SQL. Elle demeure toutefois fort pertinente pour la recherche en texte intégral. Sa syntaxe est la suivante :

`MATCH(`*`champ(s) à chercher`*`) AGAINST ('`*`expression recherchée`*`' IN BOOLEAN MODE)`

Plus précisément :

- Cette fonction est une **condition** que l'on retrouvera dans la clause WHERE. Elle retournera 'Vrai' si l'enregistrement de la ligne de la table servant de source de données en cours de traitement correspond aux critères recherchés.
- Le(s) champ(s) à chercher doit(vent) se retrouver dans un **index de type "fulltext" obligatoirement**.
  - Un index "fulltext" peut couvrir plus d'un champ, mais nécessairement de la même table.
  - On retrouve un antidictionnaire que l'on peut modifier.
  - Les mots trop courts ne sont pas indexés (sur le serveur EBSI, il s'agit des mots de 3 lettres et moins).
- La recherche est **insensible à la casse**.
- La recherche peut être **insensible aux accents** en fonction de la collation définie.
- Le paramètre **IN BOOLEAN MODE** est facultatif. Il permet l'utilisation des opérateurs booléens dans l'expression recherchée.

- En **mode booléen**, on peut retrouver les opérateurs ci-dessous dans l'expression recherchée :
  - **+mot** : ET (mot doit être présent)
  - **-mot** : SAUF (mot doit être absent)
  - **Rien devant un mot** : OU (mot peut être présent, si c'est le cas, plus haute pertinence)
  - ***** : troncature à droite seulement
  - **"expression"** : recherche d'expression exacte
  - **( )** : pour faire des sous-expressions et entre autres s'assurer de l'ordre d'exécution des opérateurs
  - **>mot** ou **<mot** : pour changer l'importance d'un mot dans le calcul de pertinence, **>** donnant plus d'importance
  - **~mot** : pour diminuer au maximum l'impact du mot dans le calcul de pertinence sans toutefois l'éliminer

Exemple : Liste des cours de sciences, avec une préférence pour les mathématiques, mais pas les cours d'histoire ou de philosophie des sciences

Si on exploite la fonction `MATCH ... AGAINST ...` pour répondre au besoin initialement décrit, la requête ressemblera à la suivante :

```
1 SELECT no_cours, titre, descr
2 FROM cours
3 WHERE MATCH (titre, descr) against ('>math* chimi* physi* -histo* -philosoph*' IN BOOLEAN
  MODE);
```

Si vous êtes curieux ou curieuse de voir la pertinence calculée, vous pouvez ajouter la condition `MATCH ... AGAINST` dans la liste des champs affichés :

```
1 SELECT no_cours, titre, descr, MATCH (titre, descr) against ('>math* chimi* physi* -
  histo* -philosoph*' IN BOOLEAN MODE) as Pertinence
2 FROM cours
3 WHERE MATCH (titre, descr) against ('>math* chimi* physi* -histo* -philosoph*' IN BOOLEAN
  MODE);
```

Ce qui donnera les résultats qui suivent :

no_cours	titre	descr	Pertinence
20005	Mathématique 307	Liens entre les mathématiques et la technologie (informatique, cryptographie, traitement d'images, etc.). Culture mathématique générale. Théorie des représentations	1.592020034790039
20007	Chimie 007	Structure des atomes. Fonctions et liaisons chimiques. Chimie des solutions. Équilibre. Problématiques contemporaines (chimie de l'eau, pollution de l'air et de l'eau, plastiques et polymères, etc.). Regard critique	0.8880301117897034

Table des résultats

Plus précisément :

- La recherche plein texte se fait dans un **index plein texte** qui regroupe les champs TITRE et DESCR.
- On cherche dans cet index des mots qui **débutent** par *math* ou *chimi* ou *physi* mais où **on ne retrouve pas** de mots débutant par *histo* ou *philosoph*.
  - En effet, comme le mode de recherche booléenne est activé, l'**espace** entre les mots représente un **OU** et le **moins** (-) représente le **SAUF**.
  - L'**astérisque** représente la **troncature** à droite.
- Le plus grand que **(>)** devant *math** signifie que **plus de poids** sera donné aux enregistrements où des mots débutant par *math* se trouvent dans le calcul de pertinence.

2.3. Requête Ajout d'enregistrements

La forme générique d'une requête permettant d'**ajouter des enregistrements** dans une table est la suivante :

```
INSERT INTO nom de la table (nom(s) de champ)
VALUES (valeur(s));
```

Toutes les clauses sont **obligatoires**. Des explications détaillées sur ce type de requête se retrouvent dans la documentation complémentaire du cours[1].

Exemple : Insertion d'un nouvel étudiant

Si on veut ajouter un nouvel étudiant ou une nouvelle étudiante à la table ETUD, il faut minimalement y saisir les informations obligatoires qui ne sont pas générés automatiquement (NO_ETUD, NOM et PROG) :

```
INSERT INTO etud (no_etud, nom, prog)
VALUES ('10009','Bauer, Jack', 'Musique');
```

2.4. Requête Mise à jour d'enregistrements

La forme générique d'une requête permettant de **modifier des données** dans un enregistrement est la suivante :

```
UPDATE nom de la table
SET valeur(s)
WHERE critère(s);
```

Les **deux premières lignes** sont **obligatoires**, tandis que la clause **WHERE** est **facultative**. Des explications détaillées sur ce type de requête se retrouvent dans la documentation complémentaire du cours[2].

Exemple : Changement de programme d'un étudiant et ajout de sa date de naissance

Si on veut changer M. Bauer de programme ainsi qu'ajouter sa date de naissance dans la table ETUD, la requête sera la suivante :

```
UPDATE etud
SET etud.prog = 'Littérature', dat_nais = '1966-02-18'
WHERE no_etud = '10009';
```

À noter : Afin de faire la modification, une clause WHERE est nécessaire pour identifier l'enregistrement correspondant à la bonne personne. Il est important pour cette condition d'utiliser la clé primaire, soit NO_ETUD, et non le nom de l'étudiant.e (qui n'est pas nécessairement unique). Cela présuppose ainsi qu'il faille connaître le numéro de l'étudiant.e.

2.5. Requête Suppression d'enregistrements

La forme générique d'une requête permettant d'**effacer des enregistrements** dans une table est la suivante :

```
DELETE FROM nom de la table
WHERE critère(s);
```

Tous les éléments sont **obligatoires**. Des explications détaillées sur ce type de requête se retrouvent dans la documentation complémentaire du cours[3].

Exemple : Suppression d'un étudiant suite à l'abandon des études

Finalement, M. Bauer a été recruté par la Delta Force et décide d'abandonner ses études. La requête suivante permet de supprimer son enregistrement dans la table ETUD :

```
DELETE FROM etud
WHERE no_etud='10009';
```

[1] https://cours.ebsi.umontreal.ca/sci6306/co/structure_requete_ajout.html
[2] https://cours.ebsi.umontreal.ca/sci6306/co/structure_requete_majour.html
[3] https://cours.ebsi.umontreal.ca/sci6306/co/structure_requete_suppression.html

Tout comme pour la requête précédente, on utilise le numéro de l'étudiant pour faire la suppression. Utiliser le nom effacerait en effet toutes les personnes portant ce nom.

3. Ressources en lien avec le cours

Matériel de cours

- *Notes de cours [cf. sci6306_cours5_notes.pdf]*