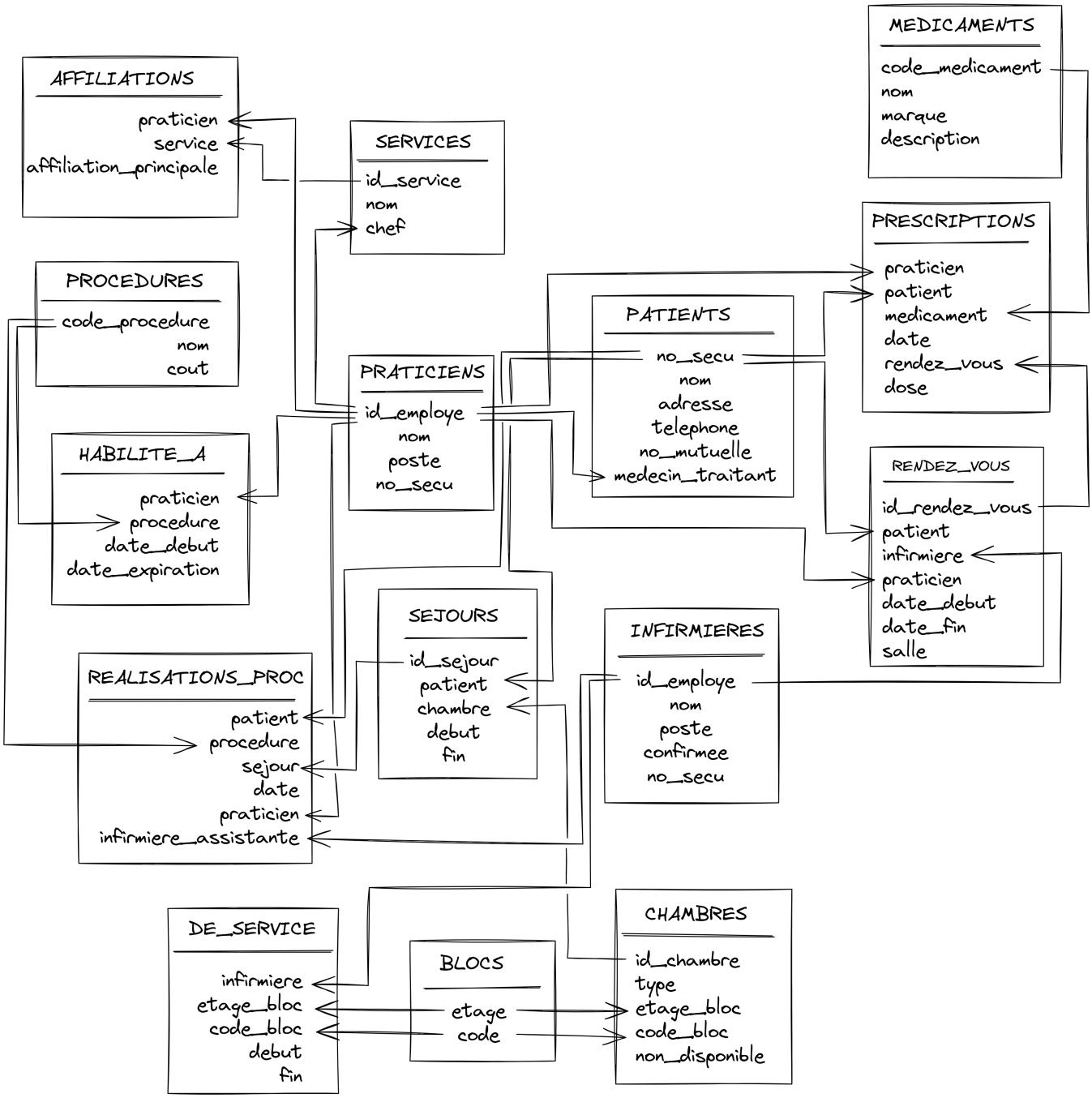
Enseignant :	Louis RAYNAL	**Établissement :**	ICES
Contact :	l-raynal@ices.fr	**Année :**	2025-2026
Public :	L3 Maths		
Ressources :	https://github.com/LouisRaynal/coursSQL		

Cours-TP 6 : Sous-requêtes

Pour ce cours-TP, nous allons nous intéresser à la base de données d'un hôpital : nommée `hopital`. Son schéma relationnel est le suivant :



Pratique

Commencez par télécharger le fichier `hopital.db` sur votre bureau, et connectez-vous-y depuis SQLiteStudio. Pour le moment, nous n'utiliserons qu'une partie de cette base.

1. Sous-requêtes

Les sous-requêtes sont l'un des outils les plus puissants de SQL. Une **sous-requête** est simplement une **requête qui est utilisée à l'intérieur d'une autre requête**. Dans la suite, on appellera **requête mère** la requête qui utilise et contient la sous-requête. La requête mère peut être de type `SELECT`, mais aussi `DELETE`, `UPDATE`, `INSERT INTO ... VALUES`. Dans ce cours nous nous concentrerons sur le premier type.

Une sous-requête est toujours encadrée par des **parenthèses**, et en pratique elle est **exécutée avant la requête mère**. Comme n'importe quelle requête, une sous-requête peut retourner une ou plusieurs lignes de données, et une ou plusieurs colonnes. **Le format du résultat de la sous-requête va déterminer comment ce résultat peut être utilisé pour interagir avec la requête mère.** La sous-requête va donc agir comme une table *temporaire* dont le résultat sera utilisé par la requête mère.

Pratique

A titre d'illustration, nous allons interroger la table des rendez-vous médicaux `RENDEZ_VOUS`. Ses colonnes sont

- `id_rendez_vous` : l'identifiant unique de chaque rendez-vous ;
- `patient` : l'identifiant du patient ayant assisté au rendez-vous ;
- `infirmiere` : l'identifiant de l'infirmière (si présente) ayant assisté le praticien ;
- `praticien` : l'identifiant du praticien ayant mené le rendez-vous ;
- `date_debut` : la date et l'heure de début du rendez-vous ;
- `date_fin` : la date et l'heure de fin du rendez-vous ;
- `salle` : le numéro de la salle où a eu lieu le rendez-vous.

Imaginez que vous voulez écrire une requête de type `SELECT`, qui retourne les lignes de `RENDEZ_VOUS` avec la plus faible valeur de l'identifiant infirmière. Naïvement, on serait tenté d'écrire

Code 1 – Exemple naïf (qui ne fonctionne pas)

```
SELECT *
FROM RENDEZ_VOUS
WHERE infirmiere = MIN(infirmiere)
;
```

Cela ne fonctionne pas comme ça en SQL. Il faut en fait écrire une sous-requête, qui récupérera la valeur minimale de la colonne `infirmiere`, et remplacera ainsi `MIN(infirmiere)` ci-dessus. Cela donne :

Code 2 – Exemple de sous-requête qui retourne une valeur numérique

```
SELECT *
FROM RENDEZ_VOUS
WHERE infirmiere = ( SELECT MIN(infirmiere)
                     FROM RENDEZ_VOUS )
;
```

> Notez que la sous-requête retourne une seule valeur, `101`, il est donc possible d'utiliser cette valeur avec l'opérateur `=` de la requête mère. Si la sous-requête avait renvoyé plusieurs colonnes, ou lignes, cela n'aurait pas été possible.

2. Types de sous-requêtes

Le comportement d'une sous-requête va d'une part dépendre du **type de résultats retournés (une ligne et une colonne, plusieurs lignes et une colonne, plusieurs lignes et plusieurs colonnes)**.

D'autre part, nous allons faire la différence entre deux types de requêtes.

- Les **sous-requêtes autonomes (ou non-corrélées)**, qui ne dépendent pas de colonnes de la requête mère. Elles peuvent être exécutées indépendamment de la requête mère.
- Les **sous-requêtes corrélées**, qui font appel à des colonnes de la requête mère.

2.1. Sous-requêtes autonomes scalaires

L'exemple que nous avons vu auparavant était une **sous-requête autonome**, car elle pouvait être exécutée toute seule, et ne faisait référence à aucun élément de la requête mère.

La sous-requête précédente retournait aussi une seule ligne et une seule colonne, donc une valeur scalaire, on parle alors de **sous-requête scalaire**. Ce genre de sous-requêtes peut notamment s'utiliser dans une condition, grâces aux opérateurs de comparaison tels que `=`, `<>`, `<`, `>`, `<=`, `>=`.

Pratique

Voici un autre exemple de sous-requête autonome scalaire, qui utilise l'opérateur de différence `<>` afin de sélectionner les rendez-vous qui n'ont pas été effectuées par (le seul) "Staff Internist" de l'hôpital.

Code 3 – Exemple de sous-requête autonome scalaire

```sql
SELECT *
FROM RENDEZ_VOUS
WHERE praticien <> ( SELECT id_employe
                     FROM PRATICIENS
                     WHERE poste = 'Staff Internist' )
;
```

Dans la plupart des variantes de SQL, s'il y avait eu plusieurs "Staff Internist" dans l'hôpital, nous aurions eu une erreur car l'opérateur `<>` nécessite une valeur scalaire à sa droite, alors que la sous-requête aurait renvoyé plusieurs lignes pour une colonne.

Attention

> En SQLite la requête se lancera, mais cela ne retournera pas le résultat attendu, car la comparaison se fera uniquement sur la valeur de la première ligne du résultat de la sous-requête.

2.2. Sous-requêtes autonomes à plusieurs lignes et une colonne

Lorsqu'une sous-requête retourne **plusieurs lignes et une seule colonne**, il y a certains opérateurs que l'on peut utiliser pour construire des conditions avec ce type de sous-requêtes.

Les opérateurs `IN` et `NOT IN` peuvent être utilisés pour construire des conditions d'appartenance à liste de valeurs.

Pratique

Reprenons la requête précédente, mais cette fois nous souhaitons récupérer tous les rendez-vous réalisés par un praticien dont le poste est "Staff Internist" ou "Attending Physician". La sous-requête retournera au moins deux identifiants d'employé `id_employe`, on utilise alors l'opérateur `IN`.

Code 4 – Exemple d'utilisation de `IN` avec sous-requête autonome

```
SELECT *
FROM RENDEZ_VOUS
WHERE praticien IN ( SELECT id_employe
                     FROM PRATICIENS
                     WHERE poste = 'Staff Internist'
                         OR poste = 'Attending Physician' )
;
```

Si l'on veut obtenir tous les rendez-vous non effectués par des praticiens avec ces postes, on utilise alors l'opérateur `NOT IN`.

Code 5 – Exemple d'utilisation de `NOT IN` avec sous-requête autonome

```
SELECT *
FROM RENDEZ_VOUS
WHERE praticien NOT IN ( SELECT id_employe
                         FROM PRATICIENS
                         WHERE poste = 'Staff Internist'
                             OR poste = 'Attending Physician' )
;
```

2.3. Sous-requêtes autonomes à plusieurs colonnes

Enfin, il arrive que l'on ait besoin de sous-requêtes renvoyant **plusieurs colonnes**. Prenons l'exemple ci-dessous.

Pratique

On vous demande de trouver tous les rendez-vous de patients ayant eu une opération (peu importe quand) et dont le médecin du rendez-vous est celui de l'opération. Vous devez donc chercher dans la table des opérations `REALISATIONS_PROC` la liste des patients opérés et la liste des praticiens associés.

Pour résoudre ce problème, il faut d'abord écrire une sous-requête sur la table `REALISATIONS_PROC`, pour récupérer tous les binômes (`patient, praticien`). On utilise ensuite l'opérateur `IN` pour faire la comparaison avec les binômes issus de `RENDEZ_VOUS` et les binômes issus de la sous-requête.

Code 6 – Exemple de sous-requête autonome multi-lignes et multi-colonnes

```
SELECT *
FROM RENDEZ_VOUS
WHERE (patient, praticien) IN ( SELECT patient, praticien
                                FROM REALISATIONS_PROC )
;
```

Attention

Pour utiliser `IN` (ou `NOT IN`) sur plusieurs colonnes, assurez vous que l'ordre des colonnes à gauche est le même que celui à droite.

2.4. Sous-requêtes corrélées

Une **sous-requête corrélée est dépendante de la requête mère**, au travers d'une ou plusieurs colonnes de cette dernière qui seront appelées dans la sous-requête. Contrairement à une sous-requête autonome, vous ne pourrez pas la lancer avant de l'intégrer à la requête mère.

La syntaxe classique d'une sous-requête corrélée est la suivante :

Code 7 – Syntaxe d'une sous-requête corrélée

```
SELECT col1_tab1, col2_tab1, ....
FROM tab1
WHERE col1_tab1 opérateur ( SELECT col1_tab2
                            FROM tab2
                            WHERE tab2.col = tab1.col )
;
```

Notez que la sous-requête fait référence dans sa clause `WHERE` à une colonne d'une table de la requête mère. Cela définit le caractère corrélé de la sous-requête.

Une sous-requête corrélée va être exécutée une fois par ligne de la table mère.

Pratique

La requête ci-dessous donne un exemple de sous-requête corrélée.

Code 8 – Exemple de sous-requête corrélée

```
SELECT *
FROM RENDEZ_VOUS
WHERE 3 <> ( SELECT COUNT(*)
             FROM REALISATIONS_PROC
             WHERE REALISATIONS_PROC.praticien = RENDEZ_VOUS.praticien )
;
```

La sous-requête est exécutée pour chaque ligne de la table `RENDEZ_VOUS`. Pour chacune de ces lignes candidates, la valeur de la colonne `praticien` est injectée dans la sous-requête, à la place de `RENDEZ_VOUS.praticien`. La sous-requête va donc ici compter combien de fois un praticien donné a effectué d'opérations. La ligne candidate de `RENDEZ_VOUS` va être affichée, uniquement si le praticien a effectué un nombre d'opérations différent de 3. Ici, seul le praticien avec pour identifiant `3`, a effectué 3 opérations. Toutes les lignes excluant se praticien sont donc renvoyées par la sous-requête.

Afin de mieux comprendre ce qui est fait dans une sous-requête, vous pouvez lancer la sous-requête, en spécifiant vous même une valeur scalaire, par exemple

```
SELECT COUNT(*)
FROM REALISATIONS_PROC
WHERE REALISATIONS_PROC.praticien = 2
;
```

Remarque

La requête

```
SELECT *
FROM RENDEZ_VOUS
WHERE ( SELECT COUNT(*)
        FROM REALISATIONS_PROC
        WHERE REALISATIONS_PROC.praticien = RENDEZ_VOUS.praticien ) <> 3
;
```

aurait aussi été valide. Peu importe la place de la sous-requête par rapport à l'opérateur de comparaison.

Les sous-requêtes corrélées ne sont pas limitées à des conditions d'égalité ou d'inégalité. Vous pouvez par exemple utiliser une sous-requête corrélée en association avec le comparateur d'étendue `BETWEEN`.

Pratique

La requête ci-dessous renvoie les lignes de la table `RENDEZ_VOUS`, pour lesquels le patient de la ligne a eu entre 1 et 20 opérations.

Code 9 – Exemple de sous-requête corrélée

```
1 SELECT *
2 FROM RENDEZ_VOUS
3 WHERE ( SELECT COUNT(*)
4         FROM REALISATIONS_PROC
5         WHERE REALISATIONS_PROC.patient = RENDEZ_VOUS.patient )
6     BETWEEN 1 AND 20
7 ;
```

Astuce

Nous verrons plus tard qu'une sous-requête ne s'utilise pas uniquement au niveau de la clause `WHERE`, ou même uniquement dans une requête mère de type `SELECT`.
Une sous-requête peut notamment s'utiliser pour obtenir de nouvelles colonnes. Afin de mieux comprendre une sous-requête, vous pouvez la placer au niveau du `SELECT` de la manière suivante :

```
SELECT *, ( SELECT COUNT(*)
            FROM REALISATIONS_PROC
            WHERE REALISATIONS_PROC.patient = RENDEZ_VOUS.patient ) AS res_sous_req
FROM RENDEZ_VOUS
--WHERE res_sous_req BETWEEN 1 AND 20
;
```

Vous pouvez ainsi voir pour chaque ligne, quel est le résultat de la sous-requête. Vous pouvez même décommenter la clause `WHERE` pour obtenir le même résultat qu'auparavant.

3. Existence avec `EXISTS`

Bien que vous verrez des sous-requêtes corrélées utilisées avec des conditions d'égalité ou d'étendue, l'opérateur le plus courant pour leur usage est celui d'existence : `EXISTS`. Cet opérateur permet d'identifier lorsqu'un lien existe entre deux choses, sans se soucier d'une quantité. Par exemple, vous voulez juste savoir si un patient a déjà été opéré, sans considération du nombre d'opérations subies.

Pratique

Si l'on veut identifier si un patient a déjà eu une opération, il faut simplement s'assurer qu'il existe une correspondance entre les patients de la table `RENDEZ_VOUS` et ceux de la table `REALISATIONS_PROC`.

Code 10 – Exemple de sous-requête corrélée avec `EXISTS`

```sql
SELECT *
FROM RENDEZ_VOUS
WHERE EXISTS ( SELECT 1
               FROM REALISATIONS_PROC
               WHERE REALISATIONS_PROC.patient = RENDEZ_VOUS.patient )
;
```

Pour chaque ligne de la table `RENDEZ_VOUS`, la valeur de l'identifiant du patient est injectée dans la sous-requête. Si la condition `REALISATIONS_PROC.patient = RENDEZ_VOUS.patient` est vérifiée, c'est-à-dire qu'il y a correspondance entre les identifiants, alors la sous-requête retourne 1. L'opérateur `EXISTS` va ensuite vérifier si quelque chose est retourné par la sous-requête ou non. Si c'est le cas, la ligne est conservée (car `EXISTS` retourne `1`). Si ce n'est pas le cas, la ligne n'est pas conservée (car `EXISTS` retourne `0`).

Remarque

Notez que la sous-requête associée à `EXISTS` spécifie `SELECT 1`. Vous pouvez mettre la valeur ou expression que vous voulez, peu importe puisque `EXISTS` vérifie juste si la sous-requête retourne quelque chose.

Il est aussi possible de vérifier la non existence avec l'opérateur `NOT EXISTS`.

Pratique

La requête ci-dessous récupère les lignes de la tables `RENDEZ_VOUS`, pour les patients n'ayant pas eu de prescriptions.

Code 11 – Exemple de sous-requête corrélée avec `EXISTS`

```sql
SELECT *
FROM RENDEZ_VOUS
WHERE NOT EXISTS ( SELECT 1
                   FROM PRESCRIPTIONS
                   WHERE PRESCRIPTIONS.patient = RENDEZ_VOUS.patient )
;
```

4. Quand utiliser des sous-requêtes ?

Jusqu'à présent nous avons uniquement utilisé des sous-requêtes dans la clause `WHERE` d'une requête de sélection. Nous allons voir dans les sections suivantes d'autres usages courants des sous-requêtes, tels que la création de tables personnalisées, la construction de conditions ou encore la génération de nouvelles colonnes.

4.1. Sous-requêtes comme sources de données

Les sous-requêtes peuvent très bien s'utiliser dans la clause `FROM` de la requête mère, comme nouvelle source de données. Il faut cependant que la sous-requête soit autonome. La sous-requête est alors exécutée en premier puis les données résultantes sont gardées en mémoire le temps que la requête mère s'exécute.

Pratique

Ci-dessous, nous calculons dans la sous-requête le nombre de rendez-vous effectués par praticien grâce à la clause `GROUP BY`. Dans le but d'ajouter à la table `PRATICIEN` ce nombre de rendez-vous, nous faisons une jointure (à gauche) entre la table `PRATICIEN` et la sous-requête.

Code 12 – Exemple de jointure avec sous-requête

```
SELECT PRATICIENS.*, rdvPrat.nbrRendezVous
FROM PRATICIENS
LEFT JOIN ( SELECT praticien, COUNT(id_rendez_vous) AS nbrRendezVous
            FROM RENDEZ_VOUS
            GROUP BY praticien ) AS rdvPrat
    ON rdvPrat.praticien = PRATICIENS.id_employe
;
```

Notez que nous avons utilisé un alias de table pour la sous-requête (`rdvPrat`). Cela est nécessaire lorsque des noms de colonne sont identiques dans les deux tables à joindre, pour pouvoir spécifier l'origine de chaque colonne.

4.2. Sous-requêtes pour combiner des données

Les sous-requête peuvent être utilisées pour créer de nouvelles tables temporaires à partir des opérateurs `UNION` ou `INTERSECT`.

• `UNION` concatène toutes les lignes du résultat de deux sous-requêtes. Il faut que les deux sous-requêtes retournent le même nombre de colonnes.

• `INTERSECT` compare toutes les lignes du résultat de deux sous-requêtes, et retourne les lignes communes en termes de valeurs.

Les noms des colonnes ne doivent pas forcément être identiques d'une requête à l'autre.

Pratique

On peut par exemple utiliser l'opérateur `UNION` pour obtenir la liste de tous les praticiens et infirmiers de l'hôpital.

Code 13 – Exemple d'usage de `UNION`

```
SELECT id_employe, nom, poste
FROM PRATICIENS
UNION
SELECT id_employe, nom, poste
```

```
FROM INFIRMIERES
;
```

Ci-dessous est un exemple d'usage d'`INTERSECT`.

Code 14 – Exemple d'usage de `INTERSECT`

```
SELECT id_employe, nom, poste
FROM PRATICIENS
INTERSECT
SELECT id_employe, nom, poste
FROM INFIRMIERES
;
```

Le résultat est vide car il n'y a pas de lignes communes entre les résultats des deux sous-requêtes.

Notez que vous pouvez utiliser plusieurs `UNION` ou `INTERSECT` à la suite.

4.3. Sous-requêtes pour créer de nouvelles colonnes

Une sous-requête peut s'utiliser après la clause `SELECT` afin de créer de nouvelles colonnes. Il faut alors que la sous-requête génère une valeur par exécution. C'est le cas des sous-requêtes corrélées scalaires.

Pratique

La requête ci-dessous ajoute deux colonnes calculées par des sous-requêtes. Il s'agit de sous-requêtes présentées un peu plus tôt.

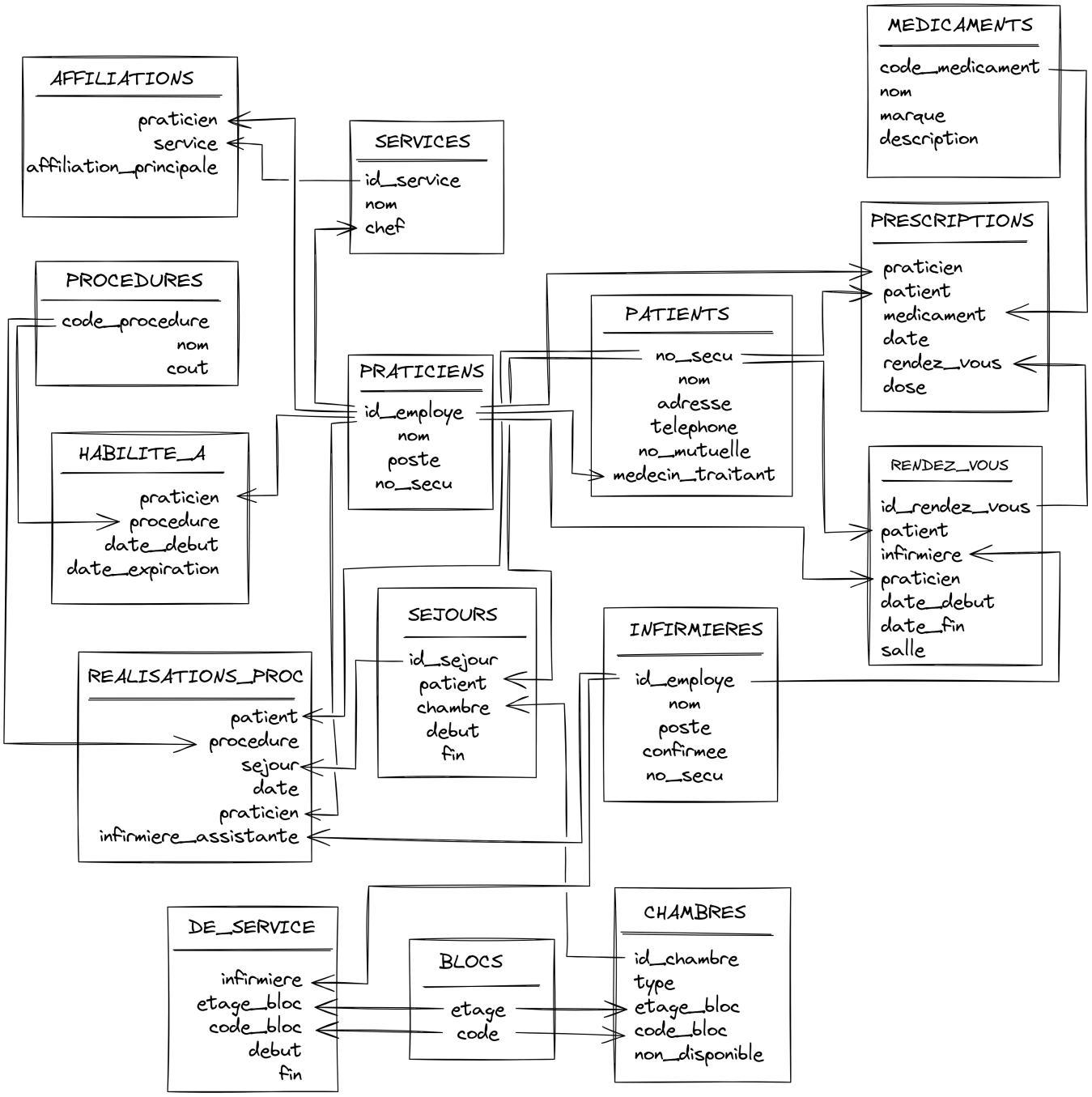
La première colonne affiche `1` si le patient a déjà subi une procédure, 0 sinon.
La deuxième calcule pour chaque patient, le nombre de procédures réalisées.

Code 15 – Exemple de sous-requête corrélée pour créer de nouvelles colonnes

```
SELECT
    *,
    EXISTS ( SELECT *
                FROM REALISATIONS_PROC
                WHERE REALISATIONS_PROC.patient = RENDEZ_VOUS.patient ) AS ProcReal,
    ( SELECT COUNT(*)
        FROM REALISATIONS_PROC
        WHERE REALISATIONS_PROC.patient = RENDEZ_VOUS.patient ) AS NbrProcReal
FROM RENDEZ_VOUS
;
```

5. Exercices

Pour ces exercices, vous utiliserez la base de données `hopital` pour laquelle le schéma relationnel est représenté ci-dessous.



1. Remarquez que la colonne `patient` de la table `REALISATION_PROC` est redondante avec la colonne `patient` de la table `SEJOUR`. En effet, ces deux tables sont liées via leur colonne respective `sejour` et `id_sejour`, on peut donc récupérer l'identifiant du patient via la colonne `patient` de `SEJOUR`. Cette redondance peut être source d'incohérence entre ces deux tables, au niveau de l'identité du patient. Pour mettre en évidence cette incohérence, **à l'aide d'une sous-requête corrélée**, affichez les lignes de la table `REALISATION_PROC` pour lesquelles les identités du patient ne correspondent pas avec `SEJOUR` alors qu'il s'agit du même séjour d'hospitalisation.
2. En vous basant sur la requête de la question précédente, affichez toutes les lignes de la table `REALISATION_PROC`, et ajoutez une colonne qui indique s'il y a une incohérence entre les identifiants des patients ou non.
3. Même question qu'au dessus mais **en utilisant une jointure** au lieu d'une sous-requête.
4. Affichez l'identifiant des infirmières qui ont été de service pour la chambre dont l'identifiant est `123`. Utilisez une sous-requête corrélée entre `DE_SERVICE` et `CHAMBRES`.
5. Même question que la précédente, mais affichez en plus le nom des infirmières.
6. Affichez le nom de tous les praticiens qui ont réalisés des procédures médicales. Utilisez une sous-requête corrélée.
7. Listez l'identifiant des praticiens ayant réalisé des procédures médicales **pour lesquelles ils n'ont jamais été habilités** (peu importe la date d'habilitation ou d'expiration). Modifiez ensuite la requête pour afficher également le nom du praticien.
8. Même question que la précédente, mais affichez les informations suivantes : nom du praticien, nom de la procédure, date de réalisation de la procédure, nom du patient sur lequel la procédure a été réalisée.
9. Affichez le nom de tous les praticiens qui ont réalisés des procédures médicales pour lesquelles ils ont déjà été habilités, mais où la date de réalisation n'entre pas dans les bornes de l'habilitation du praticien.
10. Même question que la précédente, mais listez les informations suivantes : nom du praticien, nom de la procédure, date de réalisation de la procédure, nom du patient sur lequel la procédure a été réalisée.
11. Affichez les lignes de `RENDEZ_VOUS` pour lesquelles le patient a rencontré un praticien qui n'est pas son médecin traitant. Utiliser dans un premier temps une jointure pour résoudre la question, puis faire une nouvelle requête en utilisant une sous-requête corrélée.
12. Même question que la précédente, mais affichez les informations suivantes : nom du patient, nom du praticien, nom de l'infirmière, date de début et fin du rendez-vous, salle du rendez-vous et le nom du médecin traitant.
13. Affichez le nom des patients pour lesquels le médecin traitant n'est pas chef de service. Utilisez une sous-requête autonome.
14. Affichez le nom des patients ayant subi une procédure dont le coût dépasse 5000 euros. Utilisez une sous-requête corrélée.
15. Affichez le nom des patients pour lesquels le médecin traitant a prescrit des médicaments. Utilisez une sous-requête corrélée.
16. Listez le nom des patients ayant eu au moins deux rendez-vous avec une infirmière confirmée (modalité `1` de la colonne `confirmee`). Utilisez une sous-requête corrélée.