

Rapports et requêtes 2

Vous avez suivi la formation Rapport et requêtes 1 ?

Continuons d'explorer le SQL et la base de données Santia. Les exemples concernent le dépistage, les IO et les paiements.

La version Santia 6/5/2010 ou ultérieure est requise pour les travaux pratiques

formation 5/7/2010

2/18

Champs de saisie 1/2

Depuis la formation 1, je sais obtenir le nombre de retraits de tests de dépistage pour un mois donné.

Mais j'ai un souci:

On ne peut pas modifier cette période sans le mot de passe de paramétrage. Or je voudrais que les utilisateurs lancent le rapport sur la période de leur choix, sans me solliciter à chaque fois !

Ok. On va ajouter une partie `#### saisie ####` qui définit des champs de saisie pour l'utilisateur.

! Copier/coller le texte de paramétrage et supprimer le saut de ligne éventuel dans la requête SQL.

```
#### saisie ####
Début de la période|debut_période|date|
Fin de la période|fin_période|date|
#### sql ####
nb_depi<-SELECT COUNT(*) FROM Depi WHERE TestRetr BETWEEN debut_période
AND fin_période
#### affichage ####
Période|debut_période|fin_période
Nb de dépistages retirés:
nb_depi
```

Ah, on voit maintenant apparaître un petit formulaire à remplir avant de lancer le rapport.

Oui, et ces champs sont obligatoires.

3/18

Champs de saisie 2/2

Comment faire ce formulaire de saisie ?

Et bien regardons la partie `#### saisie ####` en détail:

Chaque ligne ajoute un champ de saisie.

On a d'abord l'intitulé du champ: "Début de la période"

Je peux intituler le champ de saisie comme je veux ?

Oui.

Ensuite, on a le mot `debut_période`. Ce mot va prendre la valeur saisie par l'utilisateur pour la placer plus bas dans la partie `#### sql ####`.

Ce mot est libre mais ne doit pas contenir d'espaces.

En effet, je remarque qu'on retrouve le mot `debut_période` après `BETWEEN` dans la partie `#### sql ####`.

Voilà, `debut_période` peut prendre la valeur '2010-01-01', '2010-02-25' etc. selon ce que l'utilisateur saisit.

On n'a plus besoin de modifier à chaque fois la partie `#### sql ####` comme dans la formation 1.

Super.

Sur cette ligne on trouve ensuite "date" qui indique le type de valeur à saisir. 3 types de valeur sont possibles: date, nombre et texte.

Tout cela doit être dans l'ordre, et séparé par le signe |.

On peut encore ajouter une valeur par défaut dans le formulaire:

Ok, j'ai modifié.

Mais à quoi peut servir la saisie d'un nombre ou d'un texte pour lancer un rapport ?

Avec un nombre, on peut faire une requête SQL sur des tranches d'âge par exemple.

Avec un texte, on peut compter les patients qui habitent telle localité ou qui ont telle profession.

4/18



Je veux ajouter un formulaire de saisie pour lancer le rapport.

5/18

Rapport Dépistage 1/3

Je voudrais faire un rapport Dépistage qui affiche le nombre de conseils pré-test, de tests effectués et de test retirés, sur une période au choix.

Ok. On a toutes ces informations dans la table `Depi`:

`Dat_` est la date du conseil pré-test

`TestDat_` est la date de prélèvement du test

`TestRetr` est la date de retrait du test

Mais comment je pouvais savoir ça ?

La documentation Annexes > Structure de la base de données donne ces informations. On verra plus tard qu'il y a aussi un autre moyen de le savoir.

Ok. Il faut commencer par le formulaire de saisie, non ?

Oui, on peut reprendre la partie `#### saisie ####` qu'on a déjà utilisée:

```
#### saisie ####
Début de la période|debut_période|date|
Fin de la période|fin_période|date|
```

6/18

Rapport Dépistage 2/3

Ensuite, c'est la partie `#### sql ####` et ça se complique...

Procédons par étape.

Le rapport doit faire 3 calculs et afficher les 3 résultats.

Il faut définir un mot pour stocker chaque résultat.

Au choix ?

Je dirais `nb_conseils` pour le nombre de conseils, `nb_tests` pour le nombre de test et `nb_retraits` pour le nombre de retraits.

Faisons le 1er calcul progressivement.

On commence par mettre à gauche le mot qui va stocker le résultat, avec le signe qui ressemble à une flèche:

```
nb_conseils<-
```

Pour compter tous les dépistages, on a vu dans la Formation 1 que c'est:

```
nb_conseils<-SELECT COUNT(*) FROM Depi
```

Mais pour filter ceux qui ont une date de conseil pré-test dans la période:

```
nb_conseils<-SELECT COUNT(*) FROM Depi WHERE Dat_ BETWEEN debut_période
AND fin_période
```

Ah, je vois qu'on filtre sur `Dat_` après le mot `WHERE`.

Oui, c'est la date du conseil pré-test. Pour les 2 autres calculs:

```
nb_tests<-SELECT COUNT(*) FROM Depi WHERE TestDat_ BETWEEN debut_période
AND fin_période
nb_retraits<-SELECT COUNT(*) FROM Depi WHERE TestRetr BETWEEN
debut_période AND fin_période
```

Ok, c'est juste les mots `Dat_`, `TestDat_` et `TestRetr` qui changent finalement.

Oui, on compte les dépistages en filtrant sur chaque type de date.

7/18

Rapport Dépistage 3/3

Je crois que je n'aurai pas de souci avec la partie `####` affichage `####`.

Il suffit d'afficher les 3 mots qui stockent les résultats:

```
#### affichage ####
Période|debut_période|fin_période
Nb de conseils pré-test:
nb_conseils
Nb de dépistages effectués:
nb_tests
Nb de tests retirés:
nb_retraits
```

On peut compliquer en distinguant les femmes et les hommes ?

Oui, il suffit d'ajouter une condition sur la colonne `Sexe` qui prend la valeur 1 pour les hommes et 2 pour les femmes:

```
nb_conseils_hm<-SELECT COUNT(*) FROM Depi WHERE ( Dat_ BETWEEN
debut_période AND fin_période ) AND Sexe=1
nb_conseils_fm<-SELECT COUNT(*) FROM Depi WHERE ( Dat_ BETWEEN
debut_période AND fin_période ) AND Sexe=2
```

Encore une fois, les hommes (valeur 1) passent avant les femmes (valeur 2). Vous n'êtes pas militant de la condition féminine ?

8/18



Je sais compter le nombre d'éléments d'une table en filtrant sur des dates.

La documentation me renseigne sur les mots à utiliser pour la structure de la base de données.

9/18

Rapport Infections opportunistes 1/4

Je cherche une autre information. Il me faudrait toutes les infections opportunistes (IO) sur une période.

Dans Analyse > Requêtes sur les dossiers ou Analyse > Rapport d'activité, je ne trouve qu'une sélection partielle d'IO.

Ok. Il faut d'abord savoir où cette information est saisie dans les écrans Santia.

C'est dans les consultations médicales.

C'est ça. On va donc compter les consultations médicales qui diagnostiquent chaque IO.

Je vois dans la documentation que la table (le "classeur") des consultations médicales s'appelle `MediCons`. C'est pas très poli ?

C'est Medi pour médical et Cons pour consultations.

On va faire `SELECT COUNT(*) FROM MediCons` alors ?

Oui. Mais il faut filtrer, sinon on a juste le nombre total de consultations.

10/18

Rapport Infections opportunistes 2/4

Je sais filtrer seulement sur une date dans une période.

Imaginons la liste des consultations comme une feuille Excel.

Il y a une ligne par consultations.

Il y a plusieurs colonnes: dossier du patient, date de la consultation, poids du patient, médicaments prescrits, etc.

Ok, dans Excel je peux trier une colonne.

En SQL aussi, on filtre chaque colonne.

C'est ce qu'on a fait pour les dépistages:

SQL a compté le nombre de lignes (=le nombre de dépistages) où la colonne Date appartient à la période.

Comment SQL peut compter les consultations où la colonne Zona est cochée par exemple ?

Les cases de diagnostic IO que le médecin coche sont dans la colonne `ConcCase`.

La documentation dit: "ConcCase: Valeur des cases à cocher du diagnostic".

"Conc" comme conclusion de diagnostic.

C'est cette colonne qui nous intéresse.

11/18

Rapport Infections opportunistes 3/4

Alors on va avoir quelque chose comme `SELECT COUNT(*) FROM MediCons WHERE ConcCase = ... ?`

Oui, mais ça va être un peu spécial.

Pour filtrer les consultations qui ont la case IO n°1 cochée (case n°1= "Patient asymptomatique" dans l'écran Santia), il faut faire :

```
nb_patient_asymptomatique<-SELECT COUNT(*) FROM MediCons WHERE SUBSTR
(ConcCase,1,1)=1
```

Pour la case n°2 ("Lymphadénopathie persistante généralisée"):

```
nb_lymphadenopathie<-SELECT COUNT(*) FROM MediCons WHERE SUBSTR
(ConcCase,2,1)=1
```

Ok, le numéro change après `ConcCase`, selon la case qu'on cherche.

Oui, ce numéro est le n° de la case à cocher. On expliquera `SUBSTR` une autre fois.

C'est la méthode à suivre et c'est long à écrire pour toutes les IO.

En plus, il faut compter l'ordre des cases dans l'écran Santia sans se tromper.

On compte d'abord la partie gauche: Patient asymptomatique=1... Candidose buccale=11

Puis la partie droite: Alité plus de 50%=17... Tuberculose extra-pulmonaire=27

Et on finit par la partie inférieure: Paludisme=35, Neuropathie périphérique=36, Rhumatismes inflammatoires=37, Manifestations cutanées majeures=38.

12/18

Rapport Infections opportunistes 4/4

Il faut aussi filtrer la date des consultations sur la période, non ?

Oui, c'est vrai. On obtient donc:

```
nb_patient_asymptomatique<-SELECT COUNT(*) FROM MediCons WHERE SUBSTR
(ConcCase,1,1)=1 AND Dat_ BETWEEN debut_période AND fin_période
```

Le mot `AND` rajoute une condition.

Pour finir, il ne faut pas oublier d'écrire la partie `####` saisie `####`. Sinon SQL ne peut pas connaître la valeur de `debut_période` et `fin_période`.



Les tables sont comme des feuilles Excel, avec des colonnes qu'on peut filtrer.

Rapport Paiements médicaments 1/5

Un dernier rapport: les paiements d'ARV et de médicaments courants.

Ok. Dans quelle écran sont-ils saisis?

Dans Dispensation, si on a activé Paramètres > Application: Dispensation: afficher le paiement.

Et que dit la documentation ?

La table des dispensations s'appelle `Item`.

Quelles sont les colonnes à filtrer dans la table `Item` ?

Il y a une colonne `Paim` pour le paiement.

Ok.

Mais cette fois-ci, on veut le total des paiements et non pas le nombre de paiements.

Il y a un mot SQL pour calculer le total d'une colonne: `SUM`.

```
total_paiement<-SELECT SUM(Paim) FROM Item
```

On peut faire ainsi le total de n'importe quelle colonne qui contient des chiffres, comme dans Excel. J'ajoute la condition `Paim>0` pour faire économiser le temps de calcul:

```
total_paiement<-SELECT SUM(Paim) FROM Item WHERE Paim>0
```

Rapport Paiements médicaments 2/5

Mais comment distinguer les paiements d'ARV et ceux de médicaments courants ?

Ça se complique:

Dans la table `Item`, on a une colonne `Prod` qui désigne le médicament délivré. Mais il n'est pas indiqué si c'est un ARV ou pas.

Pour le savoir, il faut lire une autre table, celle des produits pharmaceutiques. Elle s'appelle `Prod` également et contient toutes les informations sur les médicaments.

En fait, il faut que SQL lise les 2 tables pour établir l'information.

La documentation mentionne que la table `Prod` des produits possède une colonne `Typ_` qui indique si le médicament est un ARV. La valeur dans la colonne = 3 pour les ARV et = 1 pour les médicaments courants.

Oui, on va utiliser cela.

Attention, on va manipuler 2 tables dans la même ligne SQL: La table `Item` (dispensations) et la table `Prod` (produits pharmaceutiques).

On y va progressivement.

J'ajoute d'abord la table `Prod` qu'il faut que SQL lise aussi:

```
total_paiement_arv<-SELECT SUM(Paim) FROM Item,Prod WHERE Paim>0
```

Ok, il suffit de l'ajouter à coté de l'autre table, séparée par une virgule.

Oui. Mais ce n'est pas toute la table `Prod` qui nous intéresse, il faut chercher le produit qui a été délivré.

Pour cela, j'ajoute ce filtrage:

```
total_paiement_arv<-SELECT SUM(Paim) FROM Item,Prod WHERE Paim>0 AND
Item.Prod=Prod.Nume
```

Rapport Paiements médicaments 3/5

Que veut dire cette condition supplémentaire "AND Item.Prod=Prod.Nume" ?

Décomposons:

Item.Prod c'est le médicament délivré dans la table **Item** des dispensations

Prod.Nume ce sont tous les médicaments de la table **Prod** des produits pharmaceutiques. C'est la liste de tous les produits possibles.

Avec "=" entre les deux, on demande à SQL d'aller chercher le médicament délivré dans la table des produits.

Une fois que SQL a trouvé dans la table des produits le médicament qui a été délivré, il peut lire quel est le type de médicament. C'est la 3ème condition qu'on ajoute:

```
total_paiement_arv<-SELECT SUM(Paim) FROM Item,Prod WHERE Paim>0 AND
Item.Prod=Prod.Nume AND Prod.Typ_=3
```

On vérifie ainsi que le type du médicament est un ARV.

Je suis perdu.

Pourquoi "Item.Prod", "Prod.Nume" et "Prod.Typ_" ? On mélange des noms de table et des noms de colonnes, non ?

On évite justement de les mélanger par cette formulation.

Le "." veut dire "qui appartient à".

Item.Prod veut dire: la colonne **Prod** de la table **Item**

Prod.Nume veut dire: la colonne **Nume** de la table **Prod**

Prod.Typ_ veut dire: la colonne **Typ_** de la table **Prod**

Ça évite de s'y perdre.

Quelle est cette colonne "Nume" ?

La colonne **Nume** se trouve dans chaque table et contient un identifiant unique numérique, un peu comme le numéro de dossier dans Santia ou les numéros de lignes dans une feuille Excel.

Ces identifiants sont différents d'une table à l'autre.

Par exemple, chaque médicament de la table **Prod** a son propre identifiant.

C'est cet identifiant qui est copié dans la table **Item** des dispensations pour stocker la référence du médicament qui est délivré. On ne recopie pas toutes les informations du médicament (désignation, dosage, etc.) à chaque délivrance.

Rapport Paiements médicaments 4/5

Je vois. Vous pouvez me répéter cela encore une fois ?

Les données sont stockées dans différentes tables, avec des lignes et des colonnes comme dans Excel.

Par exemple la table **Prod** contient une ligne par médicament, et une colonne qui indique le type de médicament.

Chaque ligne a un identifiant (la colonne **Nume**).

Dans certaines tables, on trouve une colonne qui porte le nom d'une autre table (ex. **Prod**) et qui contient l'identifiant d'une ligne dans cette autre table.

Ce lien évite de répéter les informations.

Voilà notre exemple:

Item	Prod
5/2009	021
3/2009	467

Nume	Typ_
003	1
006	1
021	3
389	3
412	1

A gauche, la table des dispensations **Item**. Sa colonne **Prod** contient l'identifiant du produit délivré (n°21).

Cet identifiant correspond à droite à un médicament (une ligne) dans la table **Prod**. Cette table nous renseigne sur le type de produit (colonne **Typ_**).

La condition **Item.Prod=Prod.Nume** ("colonne **Prod** de table **Item** = colonne **Nume** de table **Prod**") permet d'aller chercher une information dans une autre table.

Rapport Paiements médicaments 5/5

Ok, c'est fini ?

Presque.

Pour les médicaments courants, on remplace simplement "Typ_=3" par "Typ_=1".

Il faut encore ajouter un filtrage sur la date dans une période:

```
total_paiement_arv<-SELECT SUM(Paim) FROM Item,Prod WHERE Paim>0 AND
Item.Prod=Prod.Nume AND Prod.Typ_=3 AND ( Dat_ BETWEEN debut_période AND
fin_période )
```

Je reconnais le filtrage sur la date, mais pourquoi ces parenthèses en plus ?

C'est plus clair. On peut même écrire ainsi :

```
total_paiement_arv<-SELECT SUM ( Paim ) FROM Item,Prod WHERE ( Paim>0 )
AND ( Item.Prod=Prod.Nume ) AND ( Prod.Typ_=3 ) AND ( Dat_ BETWEEN
debut_période AND fin_période )
```

! Attention les espaces autour des mots debu_période et fin_périod sont obligatoires, il ne faut pas écrire "fin_période)".

Voilà, nous avons terminé.

C'était un peu difficile mais nous avons étudié des points importants du SQL et de la structure de la base Santia.

Le SQL peut s'utiliser aussi sur d'autres bases de données comme Access de Microsoft.

Seuls les noms de tables et de colonnes changent selon les données.

Merci de votre attention,

A bientôt !
