Projet
Base de données
IN206

Øystein Hovind, Gabriel Petry

6 janvier 2017

École nationale supérieure des techniques avancées

MCD

Le MCD a été créé en considérant que chaque prescription et chaque achat ne concernent qu'un médicament. On peut voir dans la figure 1 que dans ce modèle les fournisseurs sont lié aux médicaments seulement par les achats, donc c'est possible qu'un médicament quelconque soit achetée des différents fournisseurs.

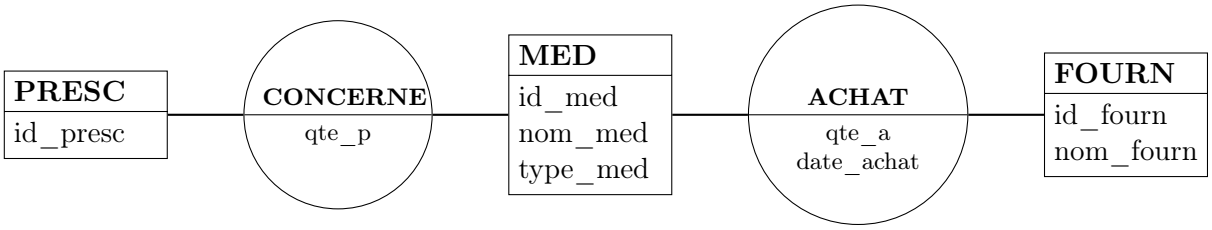

Figure 1: MCD basé sur le text de partie 1.

MLD

Dans ce modèle tous les entités provenant du MCD ont été transformées en tables. En plus, la propriété 'quantité' présente dans la relation entre les médicaments et les prescriptions a été ajoutée à la table des prescriptions, vue que la cardinalité de l'association avec ce entité était unitaire.

Par contre, la relation entre les médicaments et ses fournisseurs a devenu une nouvelle table car les cardinalités n'étaient pas unitaires. Le MLD résultant est le suivant:

- MED (**id_med**, type_med, nom_med)
- FOURN (**id_fourn**, nom_fourn)
- PRESC (**id_presc**, qte_p, *id_med*)
- ACHAT (**id_achat**, qte_a, *id_med*, *id_fourn*, date_achat)

Les champs en gras constituent les clés primaires de chaque table, pendant que ceux en italique constituent les clés étrangères, qui ont été ajoutées afin de représenter les relations entre les entités.

Implémentation en SQL

La création des tables en SQL c'est simplement une traduction du modèle logique présenté au dessus, et le jeu de données a été créé afin qu'on puisse tester le fonctionnement des questions SQL de la partie suivante.

```sql
-- création de la base

DROP TABLE PRESC;
DROP TABLE ACHAT;
DROP TABLE MED;
DROP TABLE FOURN;
DROP TABLE LOGINS;

CREATE TABLE MED (
        id_med number(5) Constraint PK_MED PRIMARY KEY,
        type_med char(20) Constraint TYPE_MED_N_NULL NOT NULL,
        nom_med char(20) Constraint NOM_MED_N_NULL NOT NULL
);

CREATE TABLE FOURN (
        id_fourn number(5) Constraint PK_FOURN PRIMARY KEY,
        nom_fourn char(30) Constraint NOM_FOURN_N_NULL NOT NULL
);

CREATE TABLE PRESC (
        id_presc number(5) Constraint PK_PRESC PRIMARY KEY,
        qte_p number(5) Constraint QTE_P_N_NULL NOT NULL,
        id_med number(5) Constraint PRESC_REF_MED REFERENCES MED
);

CREATE TABLE ACHAT (
        id_achat number(5) Constraint PK_ACHAT PRIMARY KEY,
        qte_a number(5) Constraint QTE_A_N_NULL NOT NULL,
        id_med number(5) Constraint ACHAT_REF_MED REFERENCES MED,
        id_fourn number(5) Constraint ACHAT_REF_FOURN REFERENCES FOURN,
        date_achat date
);

CREATE TABLE LOGINS (
        login char(50) Constraint LOGIN_N_NULL NOT NULL,
        password char(50) Constraint PW_N_NULL NOT NULL
);
```

```
-- insértion des données

INSERT INTO MED VALUES(1,'antibiotique','peniciline');
INSERT INTO MED VALUES(2,'antibiotique','amoxiciline');
INSERT INTO MED VALUES(3,'antibiotique','cefalexine');
INSERT INTO MED VALUES(4,'antibiotique','ampiciline');
INSERT INTO MED VALUES(5,'analgesiques','paracetamol');
INSERT INTO MED VALUES(6,'analgesiques','aspirine');
INSERT INTO MED VALUES(7,'analgesiques','ibuprofene');

INSERT INTO FOURN VALUES(1,'Alpha');
INSERT INTO FOURN VALUES(2,'Beta');

INSERT INTO ACHAT VALUES(1, 100, 1, 1, '14-APR-16');
INSERT INTO ACHAT VALUES(2, 200, 1, 1, '20-MAY-16');
INSERT INTO ACHAT VALUES(3, 150, 5, 2, '10-JAN-17');
INSERT INTO ACHAT VALUES(4, 300, 2, 2, '15-JUL-15');
INSERT INTO ACHAT VALUES(5, 200, 4, 2, '27-SEP-16');
INSERT INTO ACHAT VALUES(6, 150, 7, 1, '04-DEC-15');

INSERT INTO PRESC VALUES(1, 50, 1);
INSERT INTO PRESC VALUES(2, 60, 5);
INSERT INTO PRESC VALUES(3, 100, 2);
INSERT INTO PRESC VALUES(4, 50, 4);
INSERT INTO PRESC VALUES(5, 20, 7);
INSERT INTO PRESC VALUES(6, 50, 1);
INSERT INTO PRESC VALUES(7, 60, 5);

INSERT INTO LOGINS VALUES('gabriel', 'petry');
INSERT INTO LOGINS VALUES('oystein', 'hovind');
INSERT INTO LOGINS VALUES('admin', 'admin');
```

Requêtes SQL

Les questions pour lesquelles on devait créer une requête SQL étaient celles-ci:

1. Liste des médicaments de type antibiotique triés par ordre alphabétique.
2. Liste des médicaments fournis par le fournisseur "Alpha".
3. Liste des achats de médicaments fabriqués par le fournisseur "Beta" triés par date.

4. Liste des médicaments jamais prescrits pour chaque médicament, somme des quantités de médicaments prescrits.

5. Liste des médicaments à réapprovisionner, c'est-à-dire pour lesquels la somme des quantités achetées est inférieure à la somme des quantités prescrites + 100 unités.

Puis, le code implémenté pour résoudre cela est donné ci-dessous:

```
-- Q1
PROMPT Question 1:;
SELECT nom_med
FROM MED
WHERE type_med='antibiotique'
ORDER BY nom_med ASC;

-- Q2
PROMPT Question 2:;
SELECT distinct nom_med
FROM MED, ACHAT, FOURN
WHERE MED.id_med = ACHAT.id_med
AND FOURN.id_fourn = ACHAT.id_fourn
AND FOURN.nom_fourn = 'Alpha';

-- Q3
PROMPT Question 3:;
SELECT id_achat, nom_med, date_achat
FROM MED, ACHAT, FOURN
WHERE MED.id_med = ACHAT.id_med
AND FOURN.id_fourn = ACHAT.id_fourn
AND FOURN.nom_fourn = 'Beta'
ORDER BY date_achat ASC;

-- Q4
PROMPT Question 4:;
SELECT nom_med
FROM MED
WHERE MED.id_med
NOT IN (SELECT id_med
        FROM PRESC);

-- Q5
PROMPT Question 5:;
SELECT nom_med, sum(qte_p) as qte_prescrit
FROM MED, PRESC
```

```
WHERE MED.id_med = PRESC.id_med
GROUP BY nom_med;

-- Q6
PROMPT Question 6:;
SELECT nom1, qte_achetee, qte_prescrit
FROM (SELECT nom_med as nom1, sum(PRESC.qte_p) as qte_prescrit
        FROM MED, PRESC
        WHERE MED.id_med = PRESC.id_med
        GROUP BY nom_med),
     (SELECT nom_med as nom2, sum(ACHAT.qte_a) as qte_achetee
        FROM MED, ACHAT
        WHERE MED.id_med = ACHAT.id_med
        GROUP BY nom_med)
WHERE nom1 = nom2 AND qte_achetee < (qte_prescrit + 100);
```

Injection SQL

La dernière activité demandée consistait en réaliser deux attaques par injection dans le programme C qui se connecte avec la base de données: un pour contourner le mécanisme d'authentification créé et l'autre pour récupérer le nom et le mot de passe de tous les utilisateurs.

Injection 1

Dans le programme mis en oeuvre, on demande au utilisateur de taper un login et un mot de passe. Après on récupère ses information et puis on exécute la requête suivante:

```
SELECT * FROM LOGINS
WHERE login = '<login tapé>'
AND password = '<mot de passe tapé>';
```

Cela donne la ligne de la table LOGINS qui correspond aux informations saisis par l'utilisateur. Le but de l'injection est donc de garantir que la condition WHERE soit toujours vrai, ce qui est abouti si on tape, par exemple, `' or 'y'='y` comme login et aussi comme mot de passe. La requête résultante est la suivante:

```
SELECT * FROM LOGINS
WHERE login = '' or 'y'='y' AND password = '' or 'y'='y';
```

En exécutant cela on a réussi à sélectionner tous les informations contenus dans la table des logins, vue que l'expression [n'importe] or 'y'='y' est toujours vrai.

Injection 2

On a effectué la deuxième injection dans la partie du code où on demande la saisie d'un nom de fournisseur afin d'exécuter la requête ci-dessous, qui sélectionne les informations sur des médicaments fournis par le fournisseur désiré

```
SELECT distinct nom_med, type_med
FROM MED, ACHAT, FOURN
WHERE MED.id_med = ACHAT.id_med
AND FOURN.id_fourn = ACHAT.if_fourn
AND FOURN.nom_fourn = '<fournisseur tapé>';
```

Si, au lieu de taper un vrai nom, on tapait `' union select * from logins where 'y'='y` la fin de la requête deviendrait ceci:

```
AND FOURN.nom_fourn = '' union select * from logins where 'y'='y';
```

Le résultat c'est qu'on a fait l'union de la requête original (dont le résultat sera nul, vue qu'on n'a pas envoyé un nom_fourn valide) avec une nouvelle requête, qui demande les informations contenues dans la table des logins. Cette union est possible parce que les deux requêtes imbriquées donnent le même type de retour: deux colonnes d'éléments du type char. Pour réaliser ce genre d'injection il faut, donc, avoir des connaissances à priori sur la basé de données en question.

Mesures de protection

Pour se protéger contre les injection SQL, il faut empêcher l'utilisateur de saisir un login ou un nom de fournisseur qui puisse contenir du code. Afin de protéger le base de données contre une injection, on a modifié la fonction qui enregistre l'entrée d'utilisateur de façon qu'elle n'enregistre pas le caractère qui indique le début et la fin d'une chaîne de caractères.

Enfin, on a testé les injections implémentés avec le nouveau code et on s'est certifié du bon fonctionnement du mécanisme de sécurité.