Guilpain Léo Esir1
Legris Thomas

Base de Données :
Compte Rendu TP
Séquence n°1

''J'atteste que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée''

TP n°1 :

Le TP1 consiste à prendre en main Sqlite. Tout d'abord, nous avons installé Sqlite sur les machines et nous avons pris en main le shell en s'habituant aux différentes commandes de linux et des bases de données, notamment le '.read' pour compiler et exécuter les Fichier de type sql.

Exercice 1 :

Premier Point : création de 3 nouvelles Tables :

```
CREATE TABLE etudiant(
        etudId INT PRIMARY KEY,
        nom TEXT,
        prenom TEXT,
        adress TEXT);

CREATE TABLE professeur(
        profId INT PRIMARY KEY,
        nom TEXT,
        prenom TEXT);

CREATE TABLE enseignement(
        ensId INT PRIMARY KEY,
        sujet TEXT,
        etudId INT,
        profId INT);
```

Nous avons ensuite inséré des relations dans les 3 tables avec le mot clé « INSERT »

```
INSERT INTO etudiant
VALUES(1,'Legris','Thomas','1 rue de la liberation');

INSERT INTO etudiant
VALUES(2,'Robin','Theo','66 rue des Monts Faie Danjou');

INSERT INTO etudiant
VALUES(3,'Greco','Vincent','35 avenue du professeur Charles Foulon');

INSERT INTO etudiant
VALUES(4,'Guilpain','Leo','112 rue dantrain');

INSERT INTO professeur
VALUES(1,'Ridoux','Olivier');

INSERT INTO professeur
VALUES(2,'Colombel','Franck');

INSERT INTO enseignement
VALUES(1,'BDD',1,1);

INSERT INTO enseignement
VALUES(2,'BDD',3,1);

INSERT INTO enseignement
VALUES(3,'BDD',4,1);

INSERT INTO enseignement
VALUES(4,'Elec',1,2);

INSERT INTO enseignement
VALUES(5,'Elec',2,2);

INSERT INTO enseignement
VALUES(6,'Elec',4,2);
```

Dans la suite de l'exercice nous devions faire des requêtes simples sur les tables. Voici un rendu avec l'invite de commande linux montrant le résultat des 2 requêtes.

Ex1.sql

```
.print -> Requête n°1
SELECT nom
FROM professeur
WHERE prenom = 'Franck';

.print -> Requête n°2

SELECT etudiant.nom
FROM etudiant,enseignement,professeur
WHERE sujet = 'BDD'
AND enseignement.etudId = etudiant.etudId
AND enseignement.profId = professeur.profId;
```

thomaslegris@thomaslegris-Inspiron-5558: ~/Bureau/Tp1

Fichier Édition Affichage Rechercher Terminal Aide

```
thomaslegris@thomaslegris-Inspiron-5558:~$ cd Bureau
thomaslegris@thomaslegris-Inspiron-5558:~/Bureau$ cd Tp1
thomaslegris@thomaslegris-Inspiron-5558:~/Bureau/Tp1$ sqlite3
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .read Ex1.sql

sqlite> .read Ex11.sql
-> Requête n°1
Colombel
-> Requête n°2
Legris
Greco
Guilpain

sqlite>
```

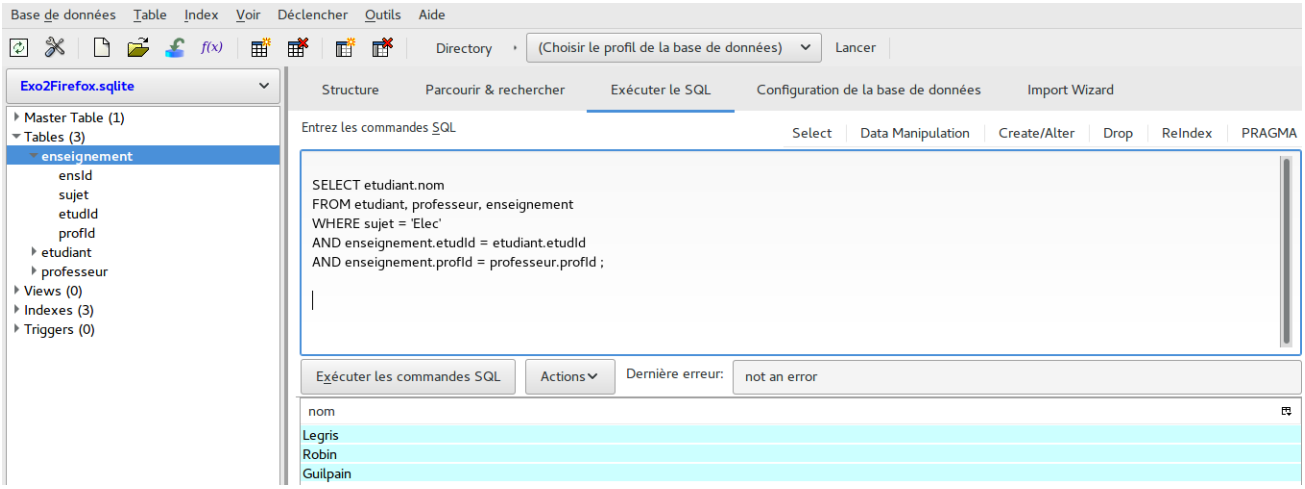
La 1ère est simple, cette requête renvoie le(s) nom(s) associée au prénom du professeur nommé ‘Franck’ .

La 2ème requête nous permet de trouver les noms des étudiants qui assistent au cours de base de donnée.

Exercice 2 :

Nous avons installer le plugin SQLite Manager sur Mozilla Firefox, on a crée une nouvelle base de donnée puis nous avons importé les tables de l'exercice 1.

Après la prise en main du nouveau logiciel nous avons essayé quelques requêtes. En voici une :



Dans cette requête nous voulons obtenir les noms des étudiants qui suivent le cours d'électricité. Nous obtenons bien Legris, Robin et Guilpain comme rentré dans la table.

Exercice 3 :

Voici la nouvelle table que l'on va utiliser dans cet exercice, elle correspond à la table utilisé dans le td1

```
CREATE TABLE Produit(p TEXT, nom_p TEXT, couleur TEXT, origine TEXT);
CREATE TABLE Fourniture(f TEXT, p TEXT, qte INTEGER);

INSERT INTO Fournisseur VALUES ('f1','Bornibus',5,'Paris');
INSERT INTO Fournisseur VALUES ('f2','Mercier',7,'Paris');
INSERT INTO Fournisseur VALUES ('f3','Colbert',3,'Reims');
INSERT INTO Fournisseur VALUES ('f4','Bossuet',6,'Dijon');
INSERT INTO Fournisseur VALUES ('f5','Tanguy',10,'Riec');
INSERT INTO Fournisseur VALUES ('f6','Dupont',0,'Paris');

INSERT INTO Produit VALUES ('p1','cassis','rouge','Dijon');
INSERT INTO Produit VALUES ('p2','champagne','blanc','Reims');
INSERT INTO Produit VALUES ('p3','huitre','vert','Riec');
INSERT INTO Produit VALUES ('p4','moutarde','jaune','Dijon');
INSERT INTO Produit VALUES ('p5','salade','vert','Nice');
INSERT INTO Produit VALUES ('p6','cornichon','vert','Dijon');
INSERT INTO Produit VALUES ('p7','muscadet','blanc','Nantes');

INSERT INTO Fourniture VALUES ('f1','p1',1);
INSERT INTO Fourniture VALUES ('f1','p4',1);
INSERT INTO Fourniture VALUES ('f1','p5',8);
INSERT INTO Fourniture VALUES ('f1','p6',2);
INSERT INTO Fourniture VALUES ('f2','p2',1);
INSERT INTO Fourniture VALUES ('f2','p4',1);
INSERT INTO Fourniture VALUES ('f3','p2',5);
INSERT INTO Fourniture VALUES ('f3','p4',1);
INSERT INTO Fourniture VALUES ('f4','p4',2);
INSERT INTO Fourniture VALUES ('f4','p5',7);
INSERT INTO Fourniture VALUES ('f4','p6',3);
INSERT INTO Fourniture VALUES ('f5','p7',10);
```

Nous avons refait les 5 premières questions correspondant au TD n°1.

Voici maintenant les requêtes utilisés à chaque question ainsi que les réponses à ces requêtes.

```
.print 'Q1 :'
SELECT Produit.nom_p
FROM Produit
WHERE origine = 'Dijon';

.print 'Q2 :'
SELECT Fr.f,Fr.nom_f
FROM Fournisseur Fr,Produit P,Fourniture Ft
WHERE P.nom_p = 'salade' AND Fr.f = Ft.f AND P.p = Ft.p;

.print 'Q3 :'
SELECT P.p
FROM Fournisseur Fr,Produit P,Fourniture Ft
WHERE ville ='Paris' AND P.p = Ft.p AND Fr.f = Ft.f;

.print 'Q4 :'
SELECT DISTINCT nom_f, remise
FROM Fournisseur Fr,Produit P,Fourniture Ft
WHERE Ft.f = Fr.f AND Ft.p = P.p AND origine = 'Dijon';

.print 'Q5 :'
SELECT P.nom_p
FROM Fournisseur Fr,Produit P,Fourniture Ft
WHERE Ft.p = P.p AND Ft.f = Fr.f AND nom_f = 'Bornibus' AND qte < 5;
```

```
thomaslegris@thomaslegris-Inspiron-5558:~$ cd Bureau
thomaslegris@thomaslegris-Inspiron-5558:~/Bureau$ cd Tp1
thomaslegris@thomaslegris-Inspiron-5558:~/Bureau/Tp1$ sqlite3
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .read epicerie.sql

sqlite> .read td1.sql
Q1 :
cassis
moutarde
cornichon
Q2 :
f1|Bornibus
f4|Bossuet
Q3 :
p1
p4
p5
p6
p2
p4
Q4 :
Bornibus|5
Mercier|7
Colbert|3
Bossuet|6
Q5 :
cassis
moutarde
cornichon

sqlite>
```

- A la question 1, nous voulons obtenir le nom des produits provenant de la ville de Dijon.

- Ensuite, à la question 2, l'objectif est de donner le nom des fournisseurs (ainsi que leurs numéros) qui vendent des salades.

- La question 3 nous permet de connaître les produits (leurs numéro produits : leurs clés) qui viennent d'un fournisseurs parisiens.

On peut voir que sur l'exemple, p4 apparaît deux fois, il est vendu par 2 fournisseurs parisiens. Pour régler ce problème nous avons juste inséré un « DISTINCT » après le select, car nous ne voulons

pas vraiment de doublons, par besoin de savoir que p4 viens 2 fois de paris dans cet exemple.

- Puis, la question 4 permet d'avoir les remises de chaque fournisseur vendant au moins un produit de la ville de Dijon, Bornibus, Mercier, Colbert et Bossuet sont concernés par cette requête.

- Enfin, la question 5 s'intéresse à Bornibus, nous faisons une requête pour avoir le nom des produits de ce fournisseur dont la quantité est inférieur à 5.

Exercice 4 :

SELECT nom f,nom p

FROM Fournisseur fs, Produit p, Fourniture ft

WHERE fs.f=ft.f AND p.p=ft.p;

Cette requête permet d'avoir les noms des fournisseurs et de leurs produits qu'ils fournissent.

Nous allons utiliser le QUERY PLAN, qui sert à résoudre les problèmes suivant : le plan d'action pour exécuter une requête est parfois éloigner de la requête en elle même, cette commande permet d'avoir le plan détaillé.

```
EXPLAIN QUERY PLAN SELECT nom_f,nom_p FROM Fournisseur
fs, Produit p, Fourniture ft WHERE fs.f=ft.f AND p.p=ft.p;
```

```
thomaslegris@thomaslegris-Inspiron-5558: ~/Bureau/Tp1
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
thomaslegris@thomaslegris-Inspiron-5558:~$ cd Bureau
thomaslegris@thomaslegris-Inspiron-5558:~/Bureau$ cd Tp1
thomaslegris@thomaslegris-Inspiron-5558:~/Bureau/Tp1$ sqlite3
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .read epicerie.sql

sqlite> .read Exo4Queryplan.sql
0|0|0|SCAN TABLE Fournisseur AS fs
0|1|2|SEARCH TABLE Fourniture AS ft USING AUTOMATIC COVERING INDEX (f=?)
0|2|1|SEARCH TABLE Produit AS p USING AUTOMATIC COVERING INDEX (p=?)

sqlite>
```

TP n°2 :

La table créée par l'entreprise n'était pas assez précise pour répondre aux attentes formulées. C'est pourquoi nous avons dû mettre en place une nouvelle table afin de faciliter nos requêtes.

Création d'une nouvelle table :

```
CREATE TABLE client(
        num_c INT PRIMARY KEY,
        nom_c TEXT,
        adresse TEXT,
        telephone TEXT,
        ville TEXT);

CREATE TABLE facture(
        num_f INT PRIMARY KEY,
        num_c INT,
        num_p INT,
        qte INT,
        remise INT,
        date_f TEXT);

CREATE TABLE produit(
        num_p INT PRIMARY KEY,
        nom_p TEXT,
        prix INT,
        TVA INT);
```

Insertion de valeurs dans les différentes tables :

```
INSERT INTO client
VALUES(1, 'Theo&Co' , '66 rue des Monts' , '0658743689' , 'Faye Danjou');

INSERT INTO client
VALUES(2, 'Vincent Corporation' , '35 avenue du professeur Charles Foulon', '0648742360', 'Rennes') ;

INSERT INTO facture
VALUES(3 , 2 , 5 , 100 , 0 , '14/02/17');

INSERT INTO facture
VALUES(6 , 1 , 4 , 200 , 0 , '14/02/17');

INSERT INTO produit
VALUES(4, 'salade' , 150 , 300);

INSERT INTO produit
VALUES(5 , 'radis' , 40 ,200);
```

Nous avons ensuite tenté de répondre aux problèmes posés :

- *« Quand un représentant de commerce veut visiter une certaine région du pays, la secrétaire doit travailler dur pour retrouver toutes les adresses »*

```
.print 'Dans quelle ville habite le client Vincent Corporation ? :'
SELECT ville
FROM client
WHERE nom_c = 'Vincent Corporation';
```

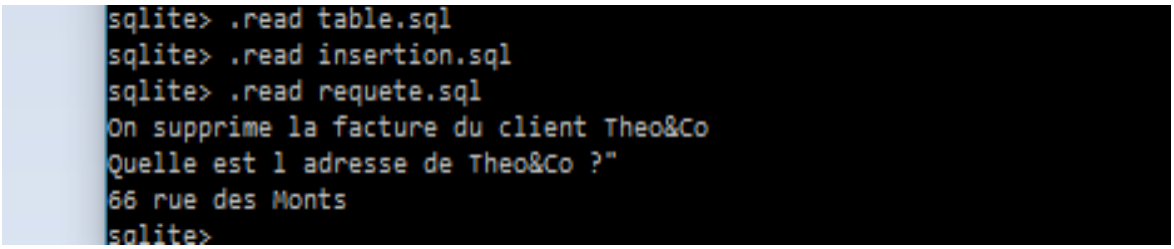
```
sqlite> .read table.sql
sqlite> .read requete.sql
Dans quelle ville habite le client Vincent Corporation ? :
Rennes
sqlite> _
```

Le problème a donc été résolu puisque la secrétaire peut directement faire ses recherches avec les noms de ville.

- *« Il s'est produit plusieurs fois que l'effacement des enregistrements correspondant aux factures annulées a conduit à la perte des adresses de clients importants »*

```
.print 'On supprime la facture du client Theo&Co'
DELETE FROM facture
WHERE num_f = 6;

.print 'Quelle est l'adresse de Theo&Co ?'
SELECT adresse
FROM client
WHERE nom_c = 'Theo&Co';
```



On supprime dans un premier temps la facture correspondant à l'entreprise « Theo&Co ». Ensuite on exécute une requête qui nous permet de savoir l'adresse de Theo&Co. On peut voir qu'il est quand même possible par la suite de récupérer l'adresse de cette entreprise.

- *Parfois, le montant total d'une facture ne correspond pas à la somme des prix unitaires des produits factures (augmentés de la TVA), car on saisit des chiffres (prix unitaire, prix total) manuellement, en se basant sur les factures papier.*

```
.print 'Quelle est le prix total de Theo&Co lorsqu'il achète ses salades'

SELECT (prix*qte) + (qte*prix*TVA*0.01) AS total

FROM facture Fa,produit P,client Cl
WHERE nom_p = 'salade' AND nom_c = 'Theo&Co' AND Fa.num_p = P.num_p AND Fa.num_c = Cl.num_c ;
```

```
sqlite> .read requete.sql
Quelle est le prix total de Theo&Co lorsqu il achète ses salades ?
120000.0
```

Cette requête permet de connaître le prix total de l'entreprise Theo&Co. Il faut donc multiplier le prix unitaire de la salade par la quantité de produits demandée par l'entreprise. Enfin, il faut ajouter la TVA propre au produit.

– *« Si on veut accorder une ristourne (pour les meilleurs clients), il est très difficile de déterminer le prix qui est encore rentable pour la société. »*

```
.print 'on met à jour la table.'
UPDATE  facture
SET ristourne = 30
WHERE num-f = 6;

.print 'Quelle est le prix total de Theo&Co lorsqu il achète ses salades ?'
SELECT (prix*qte) + (qte*prix*TVA*0.01) - remise AS total
FROM facture Fa,produit P,client Cl
WHERE nom_p = 'salade' AND nom_c = 'Theo&Co' AND Fa.num_p = P.num_p AND Fa.num_c = Cl.num_c ;
```

```
sqlite> .read requete.sql
on met à jour la table.
Quelle est le prix total de Theo&Co lorsqu il achète ses salades ?
119970.0
```

La première requête permet de mettre à jour la table, on met à jour la facture n°6 en lui ajoutant une remise de 30 centimes.

On peut voir qu'avec la remise le prix de fin est différent. Il a bien diminuer de 30 centimes.

Cependant, il y a problème si la remise est trop importante, cela ne devient plus rentable pour l'entreprise. Il faut donc calculer le prix de revient d'un produit lors de sa fabrication ou son achat pour pouvoir fixer la remise correctement.

```
CREATE TABLE produit(
        num_p INT PRIMARY KEY,
        nom_p TEXT,
        prix INT,
        TVA INT
        PrixRevient INT);

.print 'on met à jour la table.'
UPDATE facture
SET ristourne = 120
WHERE num_f = 6;

.print 'Quelle est le prix total de Theo&Co lorsqu il achète ses salades ?'
SELECT (prix*qte) + (qte*prix*TVA*0.01) - ristourne AS total
FROM facture Fa,produit P,client Cl
WHERE nom_p = 'salade' AND nom_c = 'Theo&Co' AND Fa.num_p = P.num_p AND Fa.num_c = Cl.num_c AND ristourne < PrixRevient;
```

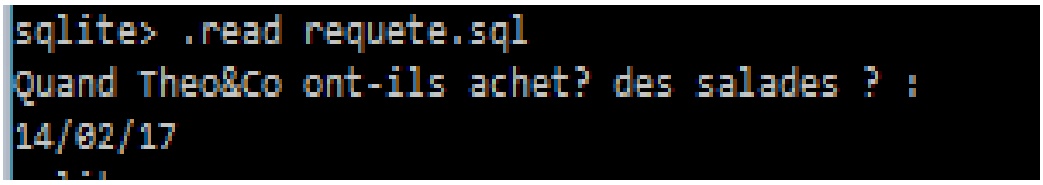
```
sqlite> .read requete.sql
on met à jour la table.
Quelle est le prix total de Theo&Co lorsqu il achète ses salades ?

sqlite>
```

Rien n’est retourné, cela signifie que le prix de revient du produit est supérieure à la remise et donc qu’il n’est pas intéressant d’effectuer une remise sur ce produit.

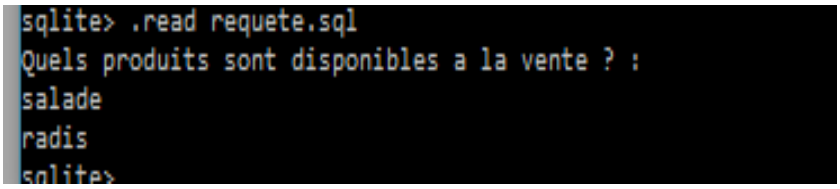
On va par la suite effectuer quelques requêtes supplémentaires afin de vérifier le bon fonctionnement de nos tables.

```
.print 'Quand Theo&Co ont-ils acheté des salades ? : '
SELECT date_f
FROM facture Fa,produit P,client Cl
WHERE nom_p = 'salade' AND nom_c = 'Theo&Co' AND Fa.num_p = P.num_p AND Fa.num_c = Cl.num_c;
```



Grâce à cette requête, on peut voir que nos tables nous permettent de déterminer la date d'une facture.

```
.print 'Quels produits sont disponibles a la vente ? : '
SELECT nom_p
FROM produit P ;
```



On a ainsi tous les produits possibles.

TP n°3 :

La première partie consiste à la mise en place de la base de donnée « DBgenerate », pour cela nous nous plaçons dans le répertoire /tmp qui permet de dépasser le quota d'espace disque qu'utilise la Base de donnée.

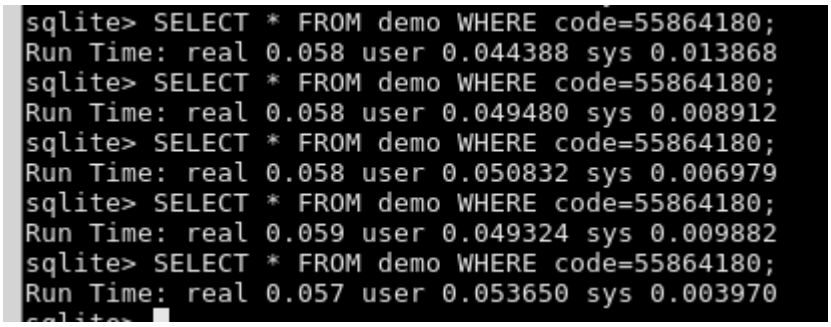
Exercice 1 : Index

Q1. Nous utilisons la commande « .timer ON » dans SqLite pour que l'invite de commande nous retourne le temps des requêtes que l'on applique sur la table « demo » (la commande « .tables » sert à connaître la ou les table(s) du fichier test.sql que l'on a ouvert.
Nous avons cherché un code dans la table et nous avons pris : 51457707.

```
Run Time: real 0.099 user 0.088668 sys 0.010114
sqlite> SELECT * FROM demo WHERE code=51457707;
44349|51457707
412341|51457707
Run Time: real 0.094 user 0.082336 sys 0.011941
sqlite> SELECT * FROM demo WHERE code=51457707;
44349|51457707
412341|51457707
Run Time: real 0.091 user 0.082424 sys 0.008979
sqlite> SELECT * FROM demo WHERE code=51457707;
44349|51457707
412341|51457707
Run Time: real 0.098 user 0.086764 sys 0.010986
sqlite> SELECT * FROM demo WHERE code=51457707;
44349|51457707
412341|51457707
Run Time: real 0.098 user 0.086766 sys 0.010980
sqlite> SELECT * FROM demo WHERE code=51457707;
```

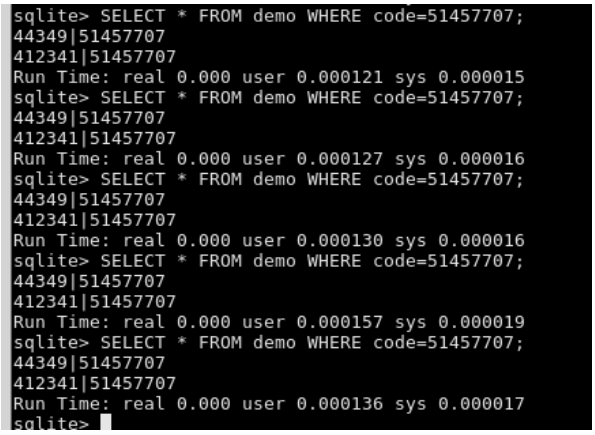
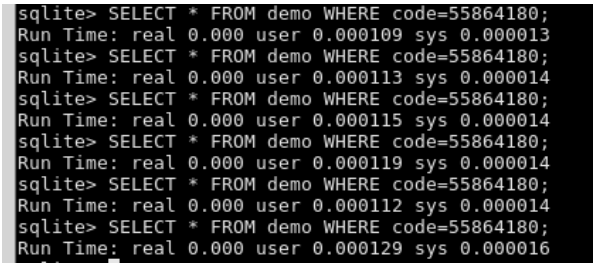
Après avoir créer la base de donnée avec la commande « sqlite3 test < database.sql »nous avons mesurer le temps d'exécution de la requête affiché dans la capture d'écran ci-dessus, on mesure le temps réel : autour de 0.09 s pour cette requête.

Q2. Nous avons cherché un code qui n'est pas dans la table et nous avons pris : 555864180.



Nous recommençons l'opération avec la requête ci dessus.
Le temps mis par l'ordinateur pour exécuter cette nouvelle requête est légèrement inférieure, proche de 0.06 s.

Q3&4. Nous allons créer un index sur l'attribut « code » grâce à la commande ci dessous, ce qui permet un accès direct aux données.
CREATE INDEX demoIDX ON demo(code) ;
Voici le résultat de la question 1&2 avec l'index.

Question 1 avec index	Question 2 avec index
	

Le temps réel est très faible, en arrondissant il est égale à 0.

Nous pouvons donc constater un temps beaucoup plus court avec l'utilisation de l'index, ce qui est beaucoup plus performant pour exécuter des requêtes.

Exercice 2 : Comprendre le temps d'exécution d'une requête, optimisation.

Nous étudions une autre BDD automatiquement généré, elle est constitué de 2 tables, « facture » et « customer ».

1. SELECT c.name
FROM customer c, facture f
WHERE f.customerId=c.customerId AND f.amount>1000;

2. SELECT name
FROM customer
WHERE customerID IN (SELECT f.customerId FROM facture f WHERE amount>1000);

3. SELECT name
FROM (customer NATURAL JOIN facture)
WHERE amount>1000;

4. SELECT name
FROM customer
WHERE customerId IN (SELECT c.customerId FROM customer c, facture f WHERE c.customerId=f.customerId AND f.amount>1000);

On pense que la n°2 est la requête la plus rapide puisqu'il n y a qu'une seule table. Ensuite la 1 semble rapide car elle ne possède pas de jointure, nous pensons que la requête n°3 se place en position 3 car elle n'a pas de requête complexe (imbriqué) malgré

la jointure, enfin la requête n°4 devrait se positionné en dernière selon notre point de vue, en effet, c'est un requête qui semble complexe.

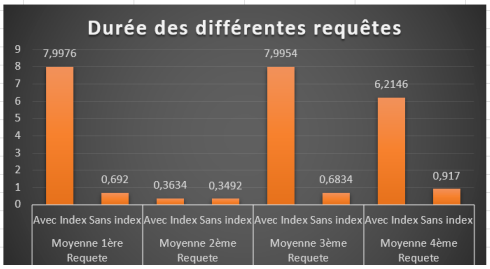
```
sqlite> SELECT c.name FROM customer c, facture f WHERE f.customerID=c.customerID AND f.amount>1000;
f51abc03-fb45-4f1e-bfd9-77471fe5b564
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
f51abc03-fb45-4f1e-bfd9-77471fe5b564
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
Run Time: real 17.517 user 7.748000 sys 9.768000
sqlite> SELECT name FROM customer WHERE customerID IN (SELECT f.customerID FROM facture f WHERE amount>1000);
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
Run Time: real 0.526 user 0.528000 sys 0.000000
sqlite> SELECT name FROM (customer NATURAL JOIN facture) WHERE amount>1000;
f51abc03-fb45-4f1e-bfd9-77471fe5b564
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
f51abc03-fb45-4f1e-bfd9-77471fe5b564
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
Run Time: real 17.464 user 8.132000 sys 9.332000
sqlite> SELECT name FROM customer WHERE customerId IN ( SELECT c.customerId FROM customer c, facture f WHERE c.customerId = f.customerId AND f.amount
> 1000);
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
f51abc03-fb45-4f1e-bfd9-77471fe5b564
7ba3467f-06c2-47c5-82cf-4e87f77a7e88
0b3152d1-bce8-4b80-8a5d-eb469e9d4cb6
Run Time: real 13.014 user 6.364000 sys 6.648000
sqlite> PRA
```

Nos estimations n'étaient pas corrects. En effet, la requête 2 est la plus rapide suivie de la 3 puis de la 4 et enfin de la 1.
Pour nous faciliter les commandes, nous avons créé plusieurs fichiers .sql avec les différentes requêtes.

```
1 .print '-> premier temps : '
2 SELECT name FROM customer WHERE customerID IN (SELECT f.customerID FROM facture f WHERE amount>1000);
3
4 .print '-> deuxieme temps : '
5 SELECT name FROM customer WHERE customerID IN (SELECT f.customerID FROM facture f WHERE amount>1000);
6
7 .print '-> troisieme temps : '
8 SELECT name FROM customer WHERE customerID IN (SELECT f.customerID FROM facture f WHERE amount>1000);
9
10 .print '-> quatrieme temps : '
11 SELECT name FROM customer WHERE customerID IN (SELECT f.customerID FROM facture f WHERE amount>1000);
12
13 .print '-> cinquieme temps : '
14 SELECT name FROM customer WHERE customerID IN (SELECT f.customerID FROM facture f WHERE amount>1000);
```

On nous demandait ensuite de créer un tableau avec les différentes mesures de temps.

1ère Requete		2ème Requete		3ème Requete		4ème Requete	
Avec Index	Sans Index	Avec Index	Sans index	Avec Index	Sans Index	Avec Index	Sans index
7.981	0.687	0.342	0.348	8.083	0.689	6.237	0.920
7.934	0.696	0.331	0.334	7.847	0.681	6.147	0.933
7.994	0.706	0.347	0.335	7.903	0.680	6.169	0.886
8.089	0.680	0.346	0.356	8.026	0.685	6.117	0.893
7.999	0.691	0.361	0.373	8.118	0.682	6.403	0.953
Moyenne 1ère Requete		Moyenne 2ème Requete		Moyenne 3ème Requete		Moyenne 4ème Requete	
Avec Index	Sans index	Avec Index	Sans index	Avec Index	Sans index	Avec Index	Sans index
7,9976	0,692	0,3634	0,3492	7,9954	0,6834	6,2146	0,917



Après avoir rentré les valeurs des durées d'exécutions de toutes les requêtes, nous avons calculé les moyennes afin de tracer les graphiques.

On peut voir que, sauf dans la 2ème requête, l'index ralentit l'exécution de la requête.

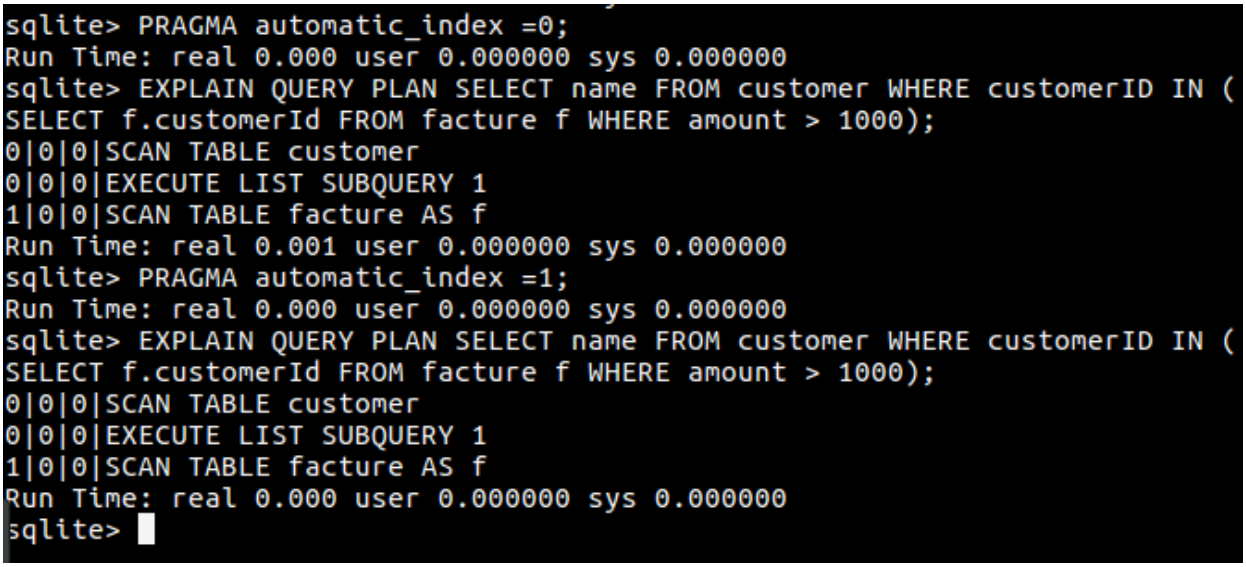
Ensuite, pour comprendre le schéma d'exécution de chaque requête nous avons utilisé « explain query plan » et « .stats ON. »

→ Voici un exemple sur la requête n°4

```
sqlite> EXPLAIN QUERY PLAN SELECT name FROM customer WHERE customerId IN (
 SELECT c.customerId FROM customer c, facture f WHERE c.customerId = f.cus
tomerId AND f.amount > 1000);
0|0|0|SCAN TABLE customer
0|0|0|EXECUTE LIST SUBQUERY 1
1|0|1|SCAN TABLE facture AS f
1|1|0|SEARCH TABLE customer AS c USING AUTOMATIC COVERING INDEX (customerI
d=?)
Run Time: real 0.001 user 0.000000 sys 0.000000
sqlite> PRAGMA automatic_index =0;
Run Time: real 0.000 user 0.000000 sys 0.000000
sqlite> EXPLAIN QUERY PLAN SELECT name FROM customer WHERE customerId IN (
 SELECT c.customerId FROM customer c, facture f WHERE c.customerId = f.cus
tomerId AND f.amount > 1000);
0|0|0|SCAN TABLE customer
0|0|0|EXECUTE LIST SUBQUERY 1
1|0|1|SCAN TABLE facture AS f
```

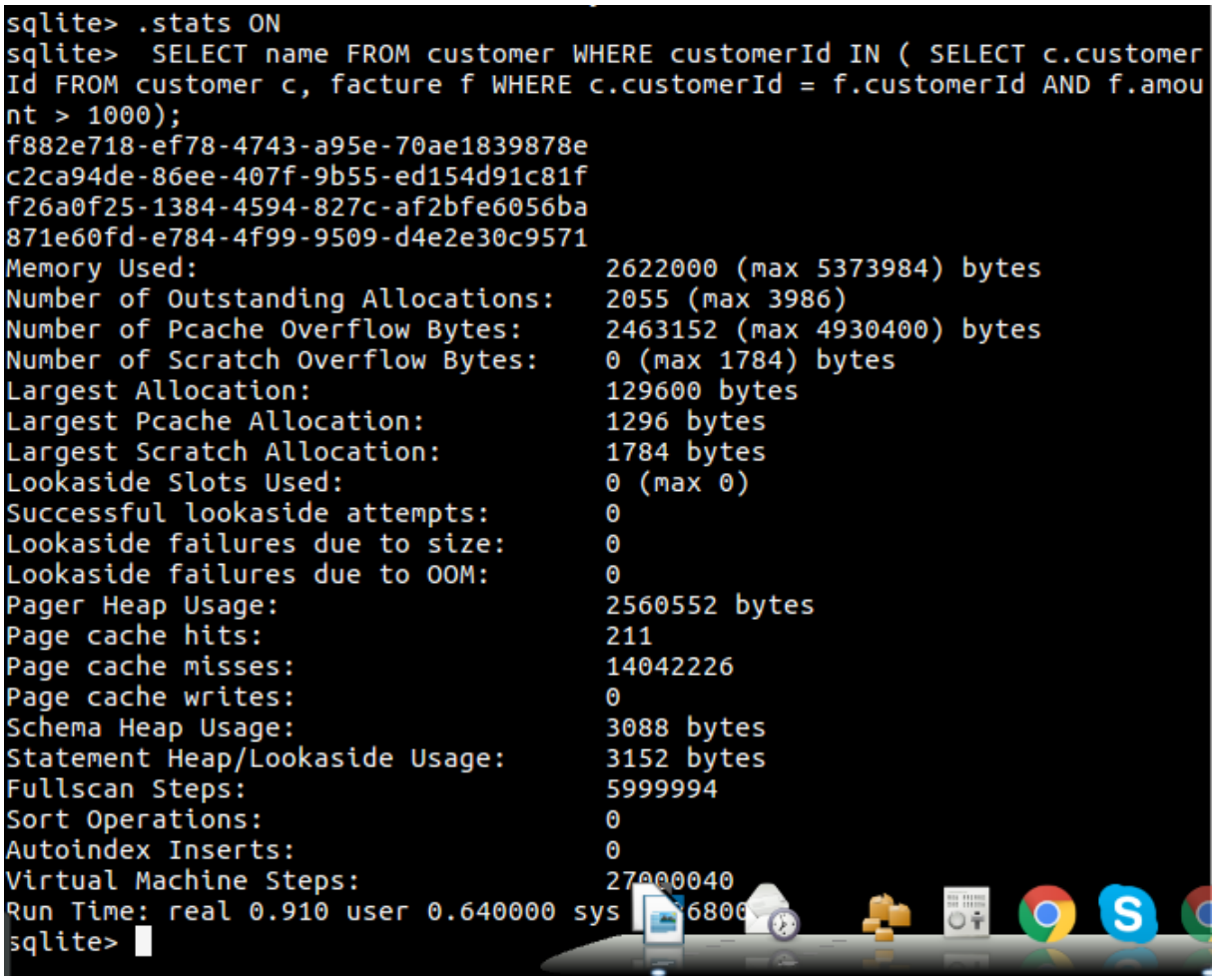
On remarque que le query plan nous indique la présence de l'index et de son utilisation lorsque « PRAGMA automatic_index = 1 » et dans l'exécution de cette requête on a constaté un temps plus long (voir table ci dessus) que sans l'index. C'est à dire que la création de l'index met du temps, on peut donc dire que ce n'est pas bénéfique pour notre requête.

Comme vous avez pu le voir dans le tableau, la requête n°2 ne change pas de temps d'exécution lorsqu'on utilise l'index. On a donc utilisé le query plan pour comprendre.



Comme on peut le voir, avec ou sans l'index, le chemin suivi dans le schéma est le même donc aucun index n'est utilisé. Cela explique donc les deux temps identiques.

Utilisons .stats ON, ici sans l'index, le temps d'exécution est proche de 0.9 s. On accède à plusieurs informations dont des informations sur la taille de la requête



En utilisant « .stats ON » sur différentes requêtes on se rend compte que la mémoire utilisées n'est pas la même. En effet certaines requêtes demandent plus de mémoire que d'autres.