Blin Sébastien
Collin Pierre-Henri

ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

École supérieure d'ingénieurs de Rennes

1ère Année
Parcours Informatique

BDD

Compte-Rendu TP1,2,3

Sous l'encadrement de :
Zoltan Miklos

1 TP1

1.1 Exercice 1

Création d'une base de données de trois tables : **etudiant** (avec les attributs etuId, nom, prenom, adress), **professeur** (avec les attributs profId, nom, prenom) et **enseignement** (avec les attributs ensId, sujet, etuId, profId).

```
sqlite> CREATE TABLE etudiant (etuId INT PRIMARY KEY, nom varchar(255),
prenom varchar(255), adress varchar(255));
sqlite> CREATE TABLE professeur (profId INT PRIMARY KEY, nom varchar(255),
prenom varchar(255));
sqlite> CREATE TABLE enseignement (ensId INT PRIMARY KEY, sujet varchar(255),
etuId INT REFERENCES etudiant, profId INT REFERENCES professeur);
```

Insertion de quelques enregistrements dans les tables :

```
INSERT INTO etudiant VALUES(1, "Martin", "Georges",
 "45 avenue de la Victoire 35000 Rennes");
INSERT INTO etudiant VALUES(2, "Durand", "Laura",
 "9 rue des Abeilles 44000 Nantes");
INSERT INTO etudiant VALUES(3, "Roux", "Bertrand",
 "16 rue Leon Blum 35000 Rennes");
INSERT INTO etudiant VALUES(4, "Dubois", "Valerie",
 "178 Boulevard Saint-Germain 75000 Paris";
INSERT INTO etudiant VALUES(5, "Fontaine", "Patrice",
 "56 allee des Camelias 29200 Brest");
INSERT INTO professeur VALUES(1, "Martinez", "Fabrice");
INSERT INTO professeur VALUES(2, "Lambert", "Andre");
INSERT INTO enseignement VALUES(1, "Base de donnees", 1, 1);
INSERT INTO enseignement VALUES(1, "Base de donnees", 2, 1);
INSERT INTO enseignement VALUES(3, "Base de donnees", 3, 1);
INSERT INTO enseignement VALUES(4, "Systeme", 4, 2);
INSERT INTO enseignement VALUES(5, "Systeme", 5, 2);
```

Quelques exemples de requêtes SQL :

Liste des noms des étudiants qui suivent le cours **Base de données**

```
SELECT nom
FROM etudiant etu, enseignement ens
WHERE ens.sujet="Base de donnees" AND etu.etuId=ens.etuId;
Martin
Durand
Roux
```

Liste des prénoms des professeurs de l'étudiant **Georges**

```
SELECT prof.prenom
FROM etudiant etu, enseignement ens, professeur prof
WHERE etu.prenom="Georges" AND etu.etuId=ens.etuId AND ens.profId=prof.profId;
Fabrice
```

1.2 Exercice 3

Vérification des réponses du TD1 :

Question 1 :

```
SELECT nom_p
FROM Produit
WHERE origine="Dijon";
```

Question 2 :

```
SELECT F.f, F.nom_f
FROM Fournisseur F, Produit P, Fourniture Ft
WHERE F.f=Ft.f AND P.p=Ft.p AND P.nom_p="Salade";
```

Question 3 :

```
SELECT P.p
FROM Produit P, Fournisseur F, Fourniture Ft
WHERE F.f=Ft.f AND P.p=Ft.p AND F.ville="Paris";
```

Question 4 :

```
SELECT F.nom_f, F.remise
FROM Fournisseur F
EXCEPT
SELECT F.nom_f, F.remise
FROM Produit P, Fournisseur F, Fourniture Ft
WHERE F.f=Ft.f AND P.p=Ft.p AND P.origine="Dijon";
```

Question 5 :

```
SELECT nom_p
FROM Fournisseur F, Produit P, Fourniture Ft
WHERE F.f=Ft.f AND P.p=Ft.p AND qte<5 AND nom_f="Bornibus";
```

Question 6 :

```
SELECT nom_f, ville
FROM Fournisseur A, Fournisseur B
WHERE F.nom_f="Bornibus" AND A.remise > B.remise;
```

Utilisation de la commande **.read**

```
sqlite> .read td1.sql
cassis
moutarde
cornichon
p1
p4
p5
p6
p2
p4
```

```
Dupont|0
Tanguy|10
cassis
moutarde
cornichon
Mercier|Paris
Bossuet|Dijon
Tanguy|Riec
```

1.3 Exercice 4

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

```
Bornibus|cassis
Bornibus|moutarde
Bornibus|salade
Bornibus|cornichon
Mercier|champagne
Mercier|moutarde
Colbert|champagne
Colbert|moutarde
Bossuet|moutarde
Bossuet|salade
Bossuet|cornichon
Tanguy|muscadet
```

Cette requête permet de connaitre les produits proposés par les fournisseurs

```
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;
```

```
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=?)
```

2 TP2

2.1 But du TP

Une entreprise enregistre ses transactions dans un fichier Excel qui est très dur à maintenir. On doit donc proposer une solution pour rendre la maintenance plus facile et éviter des pertes de données.

2.2 Solution proposée

Nous sommes donc partis sur une base de données comprenant 3 tables :

— Une base Client contenant un nom de client, un numéro de téléphone, et une adresse. Ce qui évite la perte de client avec la suppression d'une facture.

— Une base Produit contenant un nom de produit, un prix d'achat et de vente (pour calculer les rabais) et un prix de TVA.

— Une base Facture contenant un numéro de facture, un client, un produit, une quantité et une date.

Ce qui nous donne en SQL :

```
sqlite TP2.db
CREATE TABLE Client (numC INT PRIMARY KEY, nomC TEXT,
tel TEXT, address TEXT);
INSERT INTO Client VALUES(1, "Alpha CO", "1231231231",
"3, rue du thabor, 35000 Rennes");
INSERT INTO Client VALUES(2, "Beta CO", "3213213213",
"43, place de la republique, 35000 Rennes");
INSERT INTO Client VALUES(3, "Gamma CO", "7417417417",
"32, rue d'orleans, 44000 Nantes");
CREATE TABLE Produit (numP INT PRIMARY KEY, nomP TEXT,
prixVente FLOAT, prixAchat FLOAT, tva FLOAT);
INSERT INTO Produit VALUES(1, "pomme", 1, 0.2, 1);
INSERT INTO Produit VALUES(2, "tomate", 0.5, 0.1, 2);
INSERT INTO Produit VALUES(3, "orange", 1, 0.2, 2);
CREATE TABLE Facture (num INT PRIMARY KEY, numC INT
REFERENCES Client, numP INT REFERENCES Produit, qte INT,
date TEXT);
INSERT INTO Facture VALUES(123, 1, 1, 2, "2.3.2013");
INSERT INTO Facture VALUES(124, 2, 2, 22, "3.4.2013");
INSERT INTO Facture VALUES(125, 3, 3, 13, "4.4.2013");
SELECT * FROM Facture f, Client c, Produit p WHERE
f.numC = c.numC AND f.numP = p.numP;
123|1|1|2|2.3.2013|1|Alpha CO|1231231231|3, rue du
thabor, 35000 Rennes|1|pomme|1|1
124|2|2|22|3.4.2013|2|Beta CO|3213213213|43, place de
la republique, 35000 Rennes|2|tomate|0.5|2
125|3|3|13|4.4.2013|3|Gamma CO|7417417417|32, rue
d'orleans, 44000 Nantes|3|orange|1|2
```

2.3 Quelques requêtes

Ces 3 tables nous permettent de facilement trouver les clients habitant à Rennes :

```
SELECT nomC
FROM Client
WHERE address LIKE '%Rennes';
Alpha CO
Beta CO
```

Nous pouvons obtenir le prix total de chaque facture avec :

```
SELECT prix*qte+tva
FROM Facture f, Client c, Produit p
WHERE f.numC = c.numC AND f.numP = p.numP;
3
13
15
```

Retrouver le plus gros client, avec une requête imbriquée qui recherche le nom du client avec la plus grosse commande (On aurait pu faire un GROUP BY Client, mais on a choisi de chercher la plus grosse facture).

```
SELECT C.nomC
FROM Facture f, Client c, Produit p
WHERE f.numC = c.numC AND f.numP = p.numP AND prix*qte+tva =
  (SELECT MAX(prix*qte+tva)
  FROM Facture f, Client c, Produit p
  WHERE f.numC = c.numC AND f.numP = p.numP);
Gamma CO
```

Nous pouvons aussi facilement modifier les prix des produits (ce qui change le prix des factures passées, vu que nous avons décidé de ne pas en garder la trace. Il aurait fallu rajouter un prix total calculé automatiquement dès l'arrivée de la facture (TRIGGER)). Ainsi que modifier les tables facilement (ajout d'un client, d'un produit, d'une facture ou suppression) :

```
Le prix des oranges a baisse de 10%
sqlite> SELECT * FROM Produit;
1|pomme|1|0.2|1
2|tomate|0.5|0.1|2
3|orange|1|0.2|2
sqlite> UPDATE Produit SET prixVente = prixVente -0.1*prixVente WHERE
nomP = "orange";
sqlite> SELECT * FROM Produit;
1|pomme|1|0.2|1
2|tomate|0.5|0.1|2
3|orange|0.9|0.2|2

INSERT INTO Produit VALUES(4, "Kiwi", 1, 0.2, 2);
DELETE FROM Produit WHERE nomP = "Kiwi";
```

Rabais maximal par facture

```
SELECT num, qte*prixVente - qte*prixAchat
FROM Facture f, Produit p
WHERE f.numP = p.numP;
123|1.6
124|8.8
125|9.1
```

Le rabais maximum possible pour le client qui a payé la plus grosse facture.

```
sqlite> SELECT c.nomC, num, qte*prixVente - qte*prixAchat
FROM Facture f, Produit p, Client c
WHERE f.numP = p.numP AND f.numC = c.numC AND f.numC =
  (SELECT c.numC
  FROM Facture f, Client c, Produit p
  WHERE f.numC = c.numC AND f.numP = p.numP AND prixVente*qte+tva =
    (SELECT MAX(prixVente*qte+tva)
    FROM Facture f, Client c, Produit p
    WHERE f.numC = c.numC AND f.numP = p.numP));
Gamma CO|125|9.1
```

3 TP3

3.1 But du TP

Le but de ce TP est de montrer à quel point les index sont utiles et améliorent la vitesse des requêtes lorsqu'ils sont bien choisis.

3.2 Les index

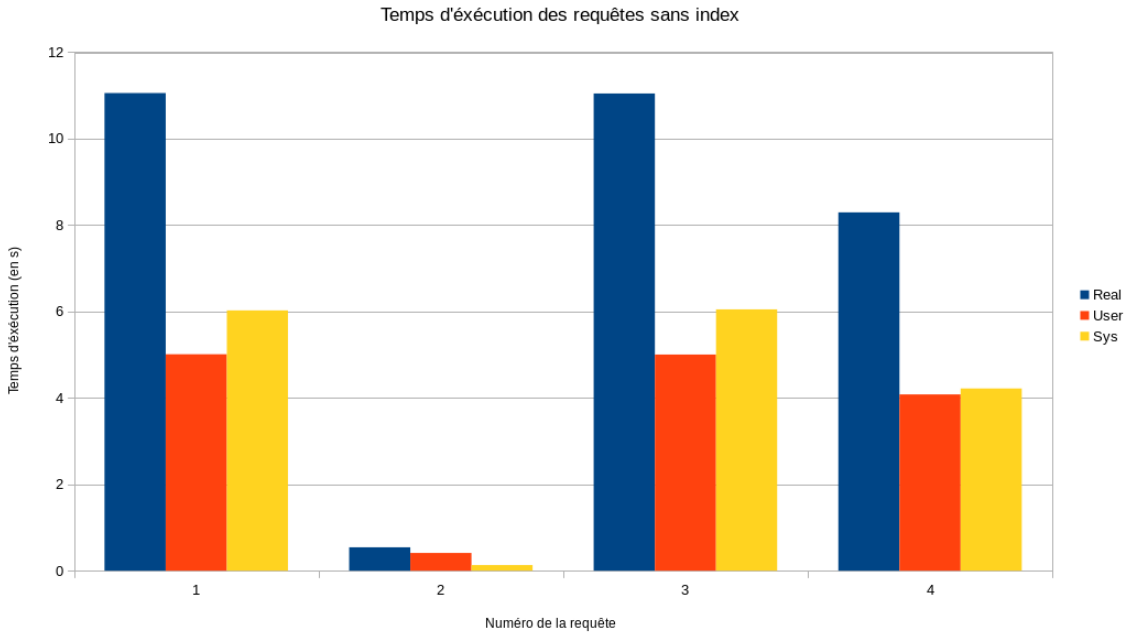
Dans cette partie du TP nous avons dû regarder le temps mis à exécuter 2 requêtes avec et sans index. Sans index, les 2 requêtes s'exécutaient en environ 1.5 secondes. Avec index, le temps mis à calculer les requêtes est quasiment nul.

3.3 Optimisation

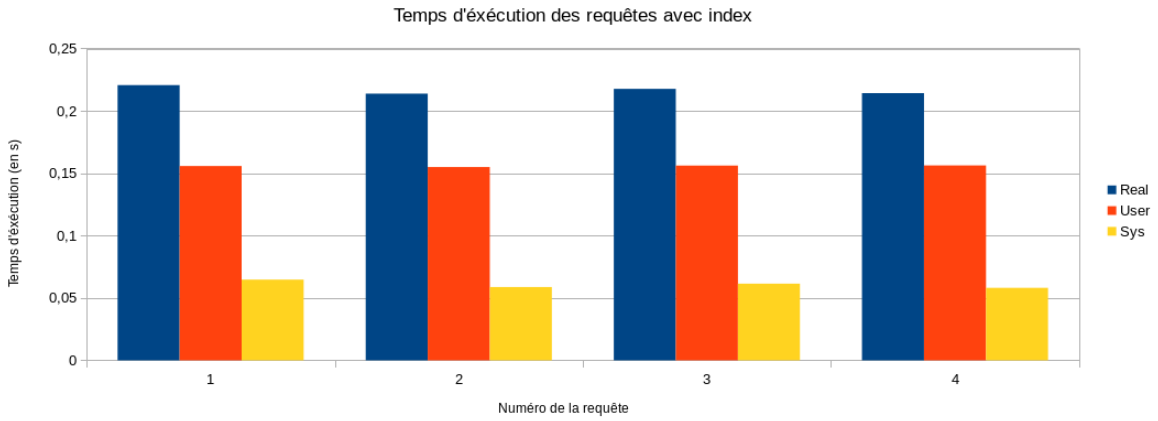
Dans cette partie, nous avons remarqué que les jointures (naturelles et à l'aide de FROM A,B) étaient plus longues à évaluer que les requêtes imbriquées.
Á l'aide de la commande CREATE INDEX, nous avons alors mis un index sur l'attribut id de la table customer.
Le temps a été considérablement réduit.

	Sans Index	Avec Index
1	11,0552	0,2206
2	0,542	0,2138
3	11,0448	0,2176
4	8,2936	0,2142

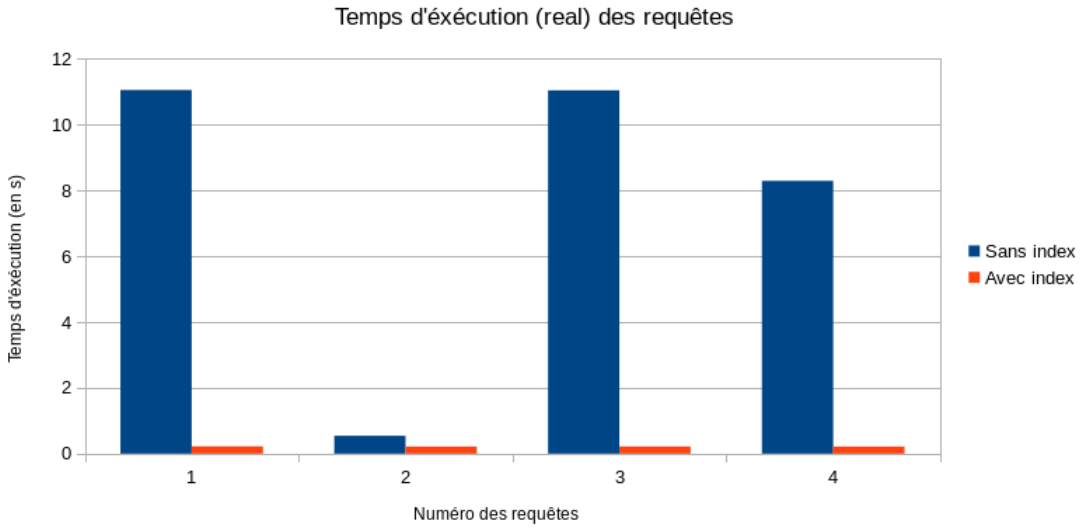
Nous avons pu réaliser quelques graphiques pour montrer l'amélioration apportée par les index. Ainsi que les optimisations apportées par ceux-ci (le temps d'exécution des 4 requêtes est identique).



Temps d'éxécution des requêtes sans index



Temps d'éxécution des requêtes avec index



Temps d'éxécution avec et sans index