Cours de bases de données

http://deptinfo.cnam.fr/rigaux/

# SGBD relationnels

# Aspects Système - Exercices

Philippe Rigaux

## Table des matières

# 1 Techniques de stockage

| Caractéristique | Performance |
|---|---|
| Taille d'un secteur | 512 octets |
| Nbre de plateaux | 5 |
| Nbre de têtes | 10 |
| Nombre de cylindres | 100 000 |
| Nombre de secteurs par piste | 4 000 |
| Temps de positionnement moyen | 10 ms |
| Vitesse de rotation | 7 400 rpm |
| Déplacement de piste à piste | 0,5 ms |
| Taux de transfert max. | 120 Mo/s |

TABLE 1 – Spécifications d'un disque magnétique

**Exercice 1.** *Le tableau 1 donne les spécifications partielles d'un disque magnétique. Répondez aux questions suivantes.*

1. *Quelle est la capacité d'une piste ?, d'un cylindre ? d'une surface ? du disque ?*
2. *Quel est le temps de latence maximal ? Quel est le temps de latence moyen ?*
3. *Quel serait le taux de transfert (en Mo/sec) nécessaire pour pouvoir transmettre le contenu d'une piste en une seule rotation ? Est-ce possible ?*

**Exercice 2.** *Étant donné un disque contenant $C$ cylindres, un temps de déplacement entre deux pistes adjacentes de $sms$ ms, donner la formule exprimant le temps de positionnement moyen. On décrira une demande de déplacement par une paire $[d, a]$ où $d$ est la piste de départ et $a$ la piste d'arrivée, et on supposera que toutes les demandes possibles sont équiprobables.*

*Attention, on ne peut pas considérer que les têtes se déplacent en moyenne de $C/2$ pistes. C'est vrai si la tête de lecture est au bord des plateaux ou de l'axe, mais pas si elle est au milieu du plateau. Il faut commencer par exprimer le temps de positionnement moyen en fonction d'une position de départ donnée, puis généraliser à l'ensemble des positions de départ possibles.*

*On pourra utiliser les deux formules suivantes :*

1. $\Sigma_{i=1}^{n}(i) = \frac{n\times(n+1)}{2}$
2. $\Sigma_{i=1}^{n}(i^2) = \frac{n\times(n+1)\times(2n+1)}{6}$

*et commencer par exprimer le nombre moyen de déplacements en supposant que les têtes de lecture sont en position $p$. Il restera alors à effectuer la moyenne de l'expression obtenue, pour l'ensemble des valeurs possibles de $p$.*

**Exercice 3.** *Soit un disque de 5 000 cylindres tournant à 12 000 rpm, avec un temps de déplacement entre deux pistes adjacentes égal à 0,2 ms et 500 secteurs de 512 octets par piste. Quel est le temps moyen de lecture d'un bloc de 4 096 ?*

**Exercice 4.** *On considère un fichier de 1 Go et un buffer de 100 Mo.*

- *quel est le hit ratio en supposant que la probabilité de lire les blocs est uniforme ?*
- *même question, en supposant que 80 % des lectures concernent 200 Mo, les 20 % restant étant répartis uniformément sur 800 Mo ?*
- *avec l'hypothèse précédente, jusqu'à quelle taille de buffer peut-on espérer une amélioration significative du hit ratio ?*

**Exercice 5.** *Soit l'ordre SQL suivant :*

```
DELETE FROM Film
WHERE condition
```

*La table est stockée dans un fichier de taille $n$ blocs, sans index. Donner le nombre moyen de blocs à lire, en fonction de $n$, pour l'exécution de l'ordre SQL dans les cas suivants :*

- *aucun enregistrement ne satisfait la condition ;*
- *la condition porte sur une clé unique et affecte donc un seul enregistrement ;*
- *la condition ne porte pas sur une clé unique (et on ne connaît donc pas à priori le nombre d'enregistrements concernés).*

**Exercice 6.** *Soit la table de schéma suivant :*

```
CREATE TABLE Personne (id     INT NOT NULL,
                  nom    VARCHAR(40) NOT NULL,
                  prenom VARCHAR(40) NOT NULL,
                  adresse VARCHAR(70),
                  dateNaissance DATE)
```

*Cettte table contient 300 000 enregistrements, stockés dans des blocs de taille 4 096 octets. Un enregistrement ne peut pas chevaucher deux blocs, et chaque bloc comprend un entête de 200 octets.*

1. *Donner la taille maximale et la taille minimale d'un enregistrement. On suppose par la suite que tous les enregistrements ont une taille maximale.*
2. *Quel est le nombre maximal d'enregistrements par bloc ?*
3. *Quelle est la taille du fichier ?*
4. *En supposant que le fichier est stocké sur le disque de l'exercice 1, combien de cylindres faut-il pour stocker le fichier ?*
5. *Quel est le temps moyen de recherche d'un enregistrement dans les deux cas suivants : (a) les blocs sont stockés le plus contigument possible et (b) les blocs sont distribués au hasard sur le disque.*
6. *On suppose que le fichier est trié sur le nom. Quel est le temps d'une recherche dichotomique pour chercher une personne avec un nom donné ?*

**Exercice 7.** *On considère un disque composé de $C$ cylindres qui doit satisfaire un flux de demandes d'entrées-sorties de la forme $[t, a]$ où $t$ est l'instant de soumission de la demande et $a$ l'adresse. On suppose que le SGBD applique de deux techniques de séquencement des entrées-sorties. La première, nommée* PDPS *(premier demandé premier servi) est classique. La second,* balayage *se décrit comme suit :*

- *les demandes d'entrées-sorties sont stockées au fur et à mesure de leur arrivée dans un tableau $T_{es}$ à $C$ entrées ; chaque entrée est une liste chaînée stockant, dans l'ordre d'arrivée, les demandes concernant le cylindre courant ;*
- *les entrées du tableau $T_{es}$ sont balayées séquentiellement de 1 à $C$, puis de $C$ à 1, et ainsi de suite ; quand on arrive sur une entrée $T_{io}[c]$, on place les têtes de lecture sur le cylindre $c$, et on effectue les entrées/sorties en attente dans l'ordre de la liste chaînée.*
  *On supposera qu'on traite les demandes connues au moment où on arrive sur le cylindre,*

*L'idée de* balayage *est bien entendu de limiter les déplacements importants des têtes de lecture. On suppose que le disque a 250 cylindres, effectue une rotation en 6 ms, et qu'un déplacement entre deux pistes adjacentes prend 0,2 ms.*

1. *On suppose que le flux des entrées/sorties est le suivant :*

$$d_1 = [1, 100], d_2 = [15, 130], d_3 = [17, 130], d_4 = [25, 15],$$
$$d_5 = [42, 130], d_6 = [27, 155], d_7 = [29, 155], d_8 = [70, 250]$$

   *Indiquez à quel moment chaque entrée/sortie est effectuée (attention : une E/S ne peut être traitée avant d'être soumise !). L'adresse est ici simplement le numéro de cylindre et les instants sont exprimés en valeur absolue et en millisecondes. On suppose que les têtes de lectures sont à l'instant 1 sur le cylindre 1.*
2. *Donnez les instants où sont effectuées les entrées/sorties avec l'algorithme classique PDPS (contraitement à balayage, on traite donc les demandes dans l'ordre où elles sont soumises).*
3. *Mêmes questions avec la liste suivante :*

$$[1, 100], [10, 130], [15, 10], [20, 180], [25, 50], [30, 250]$$

4. *Soit la liste d'E/S suivante :*

$$[1, c_1], [2, c_2], [3, c_3], \ldots, [100, c_{100}]$$

   *Supposons que le disque vienne de traiter la demande $d_1 = [1, c_1]$. Donner le temps maximal pour que la demande $d_2 = [2, c_2]$ soit traitée, avec les deux approches (balayage et premier demandé-premier servi).*
5. *Soit $n > 250$ le nombre de demandes en attente. On suppose qu'elles sont également réparties sur tous les 250 cylindres. Quel est le temps moyen d'accès à un bloc avec l'agorithme de balayage, exprimé en fonction de $n$ ?*

# 2 Indexation

Organisation Physique

**Exercice 8.**

*On prend ici comme exemple la relation* **Directeur** *(*nom_directeur, nom_film*).*

1. *Organisation Séquentielle : Expliquer l'organisation séquentielle et représenter un exemple sur la relation Directeur. Montrer le fichier aprés une insertion et aprés quelques suppressions d'articles.*

2. *Organisation Indexée : Montrer des exemples d'index non dense (primaire) et dense (secondaire) sur la relation Directeur.*

**Exercice 9.** *Soit l'arbre B+ de la figure 1, avec 4 entrées par bloc au maximum. Expliquez les opérations suivantes, et donnez le nombre d'entrées-sorties de blocs nécessaires.*

1. *Recherche de l'enregistrement 41.*
2. *Recherche des enregistrements 17 à 30.*
3. *Insertion des enregistrements 1 et 4.*
4. *Destruction de l'enregistrement 23.*

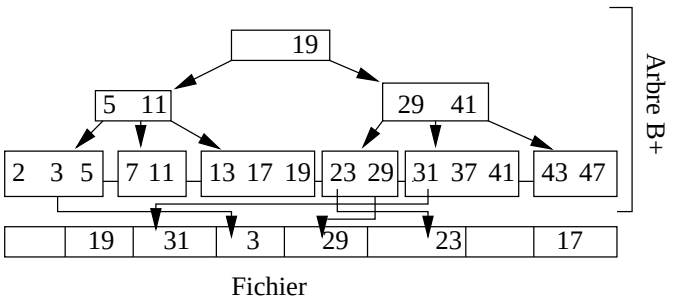


FIGURE 1 – Un arbre B+

**Exercice 10.** *Soit la liste des départements suivants.*

```
3  Allier
36 Indre
18 Cher
75 Paris
39 Jura
9  Ariège
81 Tarn
11 Aude
12 Aveyron
25 Doubs
73 Savoie
55 Meuse
15 Cantal
51 Marne
42 Loire
40 Landes
14 Calvados
30 Gard
84 Vaucluse
7 Ardèche
```

1. *Construire, en prenant comme clé le numéro de département, un index dense à deux niveaux sur le fichier contenant les enregistrements dans l'ordre donné ci-dessus, en supposant 2 enregistrements par bloc pour les données, et 8 par bloc pour l'index.*
2. *Construire un index non-dense sur le fichier trié par numéro, avec les mêmes hypothèses.*
3. *Construire un arbre-B sur les noms de département, en supposant qu'il y a au plus 4 enregistrements par bloc, et en prenant les enregistrements dans l'ordre donné ci-dessus.*
4. *On suppose que les départements sont stockés dans un fichier séquentiel, dans l'ordre donné, et que le fichier de données contient deux enregistrements par bloc. Construisez un arbre-B+ sur le nom de département avec au plus 4 entrées par bloc.*
5. *Quel est le nombre de lectures nécessaires, pour l'arbre-B puis pour l'arbre-B+, pour les opérations suivantes, :*
   (a) *rechercher le Cher ;*
   (b) *rechercher les départements entre le Cantal et les Landes (donner le nombre de lectures dans le pire des cas).*

**Exercice 11** (Index dense et non-dense)**.** *Soit un fichier de données tel que chaque bloc peut contenir 10 enregistrements. On indexe ce fichier avec un niveau d'index, et on suppose qu'un bloc d'index contient 100 entrées* $[valeur, adresse]$.

*Si* $n$ *est le nombre d'enregistrements, donnez la fonction de* $n$ *permettant d'obtenir le nombre minimum de blocs pour les structures suivantes :*

1. *Un fichier avec un index dense.*
2. *Un fichier trié sur la clé avec un index non-dense.*

**Exercice 12** (Arbre-B+)**.** *On reprend les hypothèses précédentes, et on indexe maintenant le fichier avec un arbre-B+. Les feuilles de l'arbre contiennent donc des pointeurs vers le fichier, et les nœuds internes des pointeurs vers d'autres nœuds. On suppose qu'un bloc d'arbre B+ est plein à 70 % (69 clés, 70 pointeurs).*

1. *Le fichier est indexé par un arbre-B+ dense. Donnez (1) le nombre de niveaux de l'arbre pour un fichier de 1 000 000 articles, (2) le nombre de blocs utilisés (y compris l'index), et (3) le nombre de lectures pour rechercher un article par sa clé.*
2. *On effectue maintenant une recherche par intervalle ramenant 1 000 articles. Décrivez la recherche et donnez le nombre de lectures dans le pire des cas.*
3. *Mêmes questions que (1), mais le fichier est trié sur la clé, et l'arbre-B+ est non dense.*

**Exercice 13.** *Soit un fichier de 1 000 000 enregistrements répartis en pages de 4 096 octets. Chaque enregistrement fait 45 octets et il n'y a pas de chevauchement de blocs. Répondez aux questions suivantes en justifiant vos réponses (on suppose que les blocs sont pleins).*

1. *Combien faut-il de blocs ? Quelle est la taille du fichier ?*
2. *Quelle est la taille d'un index de type arbre-B+ si la clé fait 32 octets et un pointeur 8 octets ? Même question si la clé fait 4 octets.*

3. *Si on suppose qu'un E/S coûte 10 ms, quel est le coût moyen d'une recherche d'un enregistrement par clé unique, avec index et sans index ?*

**Exercice 14.** *Un arbre B+ indexe un fichier de 300 enregistrements. On suppose que chaque nœud stocke au plus 10 clés et 10 pointeurs. Quelle est la hauteur minimale de l'arbre et sa hauteur maximale ? (Un arbre constitué uniquement de la racine a pour hauteur 0).*

*Inversement, on constate que l'arbre B+ a pour hauteur 1. Quel est le nombre minimal de pointeurs par bloc ? Le nombre maximal ?*

**Exercice 15.** *On indexe une table par un arbre B+ sur un identifiant dont les valeurs sont fournies par une* séquence*. À chaque insertion un compteur est incrémenté et fournit la valeur de clé de l'enregistrement inséré.*

*On suppose qu'il n'y a que des insertions dans la table. Montrez que tous les nœuds de l'index qui ont un frère droit sont exactement à moitié pleins.*

**Exercice 16.** *Soit un fichier non trié contenant $N$ enregistrements de 81 octets chacun. Il est indexé par un arbre-B+, comprenant $3$ niveaux, chaque entrée dans l'index occupant 20 octets. On utilise des blocs de 4 096 octets, sans entête, et on suppose qu'ils sont utilisés à 100 % pour le fichier et à 70 % pour l'index.*

*On veut effectuer une recherche par intervalle dont on estime qu'elle va ramener $m$ enregistrements. On suppose que tous les blocs sont lus sur le disque pour un coût uniforme.*

1. *Donnez la fonction exprimant le nombre de lectures à effectuer pour cette recherche avec un parcours séquentiel.*
2. *Donnez la fonction exprimant le le nombre de lectures à effectuer en utilisant l'index.*
3. *Pour quelle valeur de $m$ la recherche séquentielle est-elle préférable à l'utilisation de l'index, en supposant un temps d'accès uniforme pour chaque bloc ?*

   *En déduire le pourcentage d'enregistrements concernés par la recherche à partir duquel le parcours séquentiel est préférable.*

   *On pourra simplifier les équations en éliminant les facteurs qui deviennent négligeables pour des grandes valeurs de $N$ et de $m$.*

**Exercice 17.** *Soit les deux tables suivantes :*

```
CREATE TABLE R (idR VARCHAR(20) NOT NULL,
                  PRIMARY KEY (idR));

CREATE TABLE S (idS INT NOT NULL,
                idR VARCHAR(20) NOT NULL,
                  PRIMARY KEY (idS),
                 FOREIGN KEY idR REFERENCES R);
```

*Indiquez, pour les ordres SQL suivants, quels index peuvent améliorer les performances ou optimiser la vérification des contraintes* `PRIMARY KEY` *et* `FOREIGN KEY`*.*

1. `SELECT * FROM R WHERE idR = 'Bou'`
2. `SELECT * FROM R WHERE idR LIKE 'B%'`
3. `SELECT * FROM R WHERE LENGTH(idR) = 3`

4. `SELECT * FROM R WHERE idR LIKE '_ou'`
5. `INSERT INTO S VALUES (1, 'Bou')`
6. `SELECT * FROM S WHERE idS BETWEEN 10 AND 20`
7. `DELETE FROM R WHERE idR LIKE 'Z%'`

**Exercice 18.** *Écrire l'algorithme de recherche par intervalle dans un arbre-B.*

**Exercice 19.** *Soit la liste des départements données dans l'exercice 10, la clé étant le numéro de département. On suppose qu'un bloc contient 5 enregistrements.*

1. *Proposez une fonction de hachage et le nombre d'entrées de la table, puis construisez la structure en prenant les enregistrements dans l'ordre indiqué.*
2. *Insérez un ou plusieurs autres départements de manière à provoquer un chaînage pour l'une des entrées.*

**Exercice 20.** *Même exercice, mais avec une structure basée sur le hachage extensible.*

*On prendra une fonction de hachage sur le nom, définie de la manière suivante :* $f(nom) = i_1 i_2 \cdots i_4$ *avec* $i_j = 1$ *si la lettre* $nom[i_j]$*est en position impaire dans l'alphabet, et 0 sinon. Donc* $f(Aude) = 1101$*. Voici la liste des valeurs de hachage, en ne prenant que les 4 premiers bits.*

```
Allier  1001    Indre   1000    Cher    1010    Paris   0101
Jura    0101    Ariège          1011    Tarn    0100    Aude    1101
Aveyron 1011    Doubs   0110    Savoie  1101    Meuse   1111
Cantal  1100    Marne   1100    Loire   0110    Landes  0100
Calvados 1100   Gard    1100    Vaucluse 0111   Ardèche 1001
```

*Le hachage extensible consiste à considérer les* $n$ *derniers bits de la valeur de hachage (en d'autres termes, on prend* $h(v) mod 2^n$*, le reste de la division de la valeur de hachage par* $2^n$*). Au départ on prend* $n = 1$*, puis* $n = 2$ *quand une extension est nécessaire, etc. Montrez l'évolution de la structure de hachage extensible en insérant les départements dans l'ordre indiqué (de gauche à droite, puis de haut en bas).*

**Exercice 21.** *On indexe l'ensemble des pistes de ski du domaine skiable alpin français. Chaque piste a une couleur qui peut prendre les valeurs suivantes :* `Verte`*,* `Rouge`*,* `Bleue`*,* `Noire`*. On suppose qu'il y a le même nombre de pistes de chaque couleur, et que la taille d'un bloc est de 4 096 octets.*

1. *Donnez la taille d'un index bitmap en fonction du nombre de pistes indexées.*
2. *Combien de blocs faut-il lire, dans le pire des cas, pour rechercher les pistes vertes ?*
3. *Combien de blocs pour compter le nombre de pistes vertes ?*
4. *Que se passerait-il si on utilisait un arbre B+ ?*

**Exercice 22.** *Soit un arbre-R dont chaque nœud a une capacité maximale de 4 entrées. Quelle est à votre avis la meilleure distribution possible au moment de l'éclatement du nœud de la figure 2 ?*

**Exercice 23.** *Même hypothèse que précédemment : on insère le rectangle 8 dans le nœud de la figure 3.*

1. *Quel est le résultat après réorganisation ?*

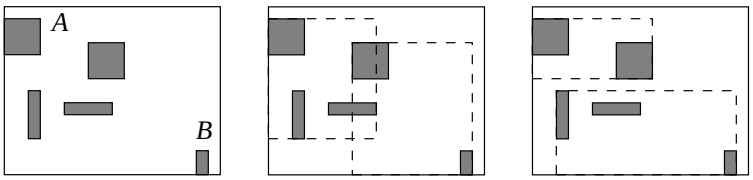


FIGURE 2 – Éclatement d'un nœud d'arbre-R

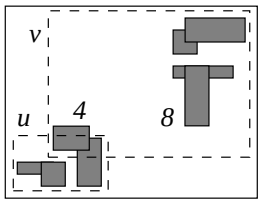


FIGURE 3 – Éclatement avec réinsertion

2. *Coment pourrait-on faire mieux, en s'autorisant la* réinsertion dans l'arbre *de un ou plusieurs rectangles d'un nœud avant éclatement.*

**Exercice 24.** *Soit un rectangle $R$ de dimension $[h, l]$ dans un espace de dimension $[d, d]$.*

1. *Quelle est la probabilité qu'un pointé $P(x, y)$ ramène le rectangle, en supposant une distribution uniforme ?*
2. *Quelle est la probablité qu'un fenêtrage $W(w_h, w_l)$ ramène le rectangle ?*

**Exercice 25.** *Écrire l'algorithme de pointé avec un arbre-R.*

# 3 Évaluation de requêtes

Les premiers exercices sont à faire en TD, et adoptent le modèle d'exécution par itération/pipelinage. Les exercices suivants consistent à expérimenter les concepts proposés avc le SGBD ORACLE, en créant une base de données, en l'alimentant avec un nombre choisi d'enregistrements, et en évaluant l'effet de manipulations diverses sur les performances du système.

## 3.1 Opérateurs d'accès aux données

**Exercice 26.** *Soit la liste des départements de l'exercice 10, page 5. On suppose qu'on peut stocker deux enregistrements par bloc. Décrire l'algorithme de tri-fusion sur le numéro de département appliqué à ce fichier dans les cas suivants :*

1. $M = 4$ *blocs ;*

2. $M = 3$ *blocs.*

**Exercice 27.** *Soit un fichier de 10 000 blocs et un buffer en mémoire centrale de 3 blocs. On trie ce fichier avec l'algorithme de tri-fusion.*

- *Combien de fragments sont produits pendant la première passe ?*
- *Combien de passes faut-il pour trier complètement le fichier ?*
- *Quel est le coût total en entrées/sorties ?*
- *Combien faut-il de blocs en mémoire faut-il pour trier le fichier en une fusion seulement.*

*Répondre aux mêmes questions en prenant un fichier de 20 000 blocs et 5 blocs de buffer, puis un fichier de 2 000 000 de blocs et 17 blocs de buffer.*

**Exercice 28.** *Donnez une spécification en pseudo-langage orienté-objet des méthodes* `constructeur`, `open`, `next` *et* `close` *de l'itérateur* PARCOURSSÉQUENTIEL *qui lit séquentiellement un fichier.*

**Exercice 29.** *Donner des plans d'exécution pour les requêtes suivantes :*

- `SELECT titre FROM Film ORDER BY annee`
- `SELECT MIN(annee) FROM Film`
  *Il faut définir un itérateur* `MIN`*.*
- `SELECT DISTINCT genre FROM Film`
  *Il faut définir un itérateur* `DISTINCT`*, en supposant que ce dernier s'appuie sur une source de données (un autre itérateur) triée.*
- `SELECT idMES, COUNT(*) FROM Film GROUP BY idMES`
  *Il faut définir un itérateur* `GROUP BY`*, en supposant encore qu'il s'appuie sur une source de données (un autre itérateur) triée.*

**Exercice 30.** *Soit deux relations $R$ et $S$, de tailles respectives $|R|$ et $|S|$ (en nombre de blocs). On dispose d'une mémoire $mem$ de taille $M$, dont les blocs sont dénotés $mem[1], mem[2], \ldots, mem[M]$.*

1. *Donnez les formules exprimant le coût d'une jointure $R \bowtie S$, en nombre d'entrées/sorties, pour l'algorithme suivant :*

```
posR = 1
Tant que posR <= |R| Faire
  Lire  R[posR] dans mem[1]
  posS = 1
  Tant que posS <= |S| faire
      Lire  S[posS] dans mem[2]
      Pour chaque enregistrement r de mem[1]
        Pour chaque enregistrement s de mem[2]
           Si r.a = s.b alors retourner [r, s]
        Fin pour
     Fin pour
     posS = posS + 1
   Fait
   posR = posR + 1
Fait
```

2. *Même question avec l'algorithme suivant :*

```
posR = 1
Tant que posR <= |R| Faire
```

```
  Lire  R[posR..(posR+M-1)] dans mem[1..M-1]
  posS = 1
  Tant que posS <= |S| faire
      Lire  S[posS] dans mem[M]
      Pour chaque enregistrement r de mem[1..M-1]
        Pour chaque enregistrement s de mem[M]
           Si r.a = s.b alors retourner [r, s]
        Fin pour
     Fin pour
     posS = posS + 1
   Fait
   posR = posR + M
Fait
```

3. *Quelle table faut-il prendre pour la boucle extérieure ? La plus petite ou la plus grande ?*

**Exercice 31** (Jointures)**.** *On suppose que* $|R| = 10\,000$*,* $|S| = 1\,000$ *et* $M = 51$*. On a 10 enregistrements par bloc,* `b` *est la clé primaire de* $S$ *et on suppose que que pour chaque valeur de* `a` *on trouve en moyenne 5 enregistrements dans* $R$*. On veut calculer* $\pi_{R.c}(R \bowtie_{a=b} S)$*,*

- *Donnez le nombre d'entrée-sorties* dans le pire des cas *pour les algorithmes de l'exercice 30.*
- *Même question en supposant (a) qu'on a un index sur* $R$*, (b) qu'on a un index sur* $S$*, (c) qu'on a deux index, sachant que dans tous les cas l'index a 3 niveaux.*
- *Même question pour une jointure par hachage.*
- *Même question avec un algorithme de tri-fusion.*

**Exercice 32.** *Donner le meilleur plan d'exécution pour les requêtes suivantes, en supposant qu'il existe un index sur* `idFilm`*.*

- `SELECT * FROM Film WHERE idFilm = 20 AND titre = 'Vertigo'`
- `SELECT * FROM Film WHERE idFilm = 20 OR titre = 'Vertigo'`
- `SELECT COUNT(*) FROM Film`
- `SELECT MAX(idFilm) FROM Film`

*Lesquelles de ces requêtes peuvent s'évaluer uniquement avec l'index ?*

**Exercice 33.** *Soit les tables relationnelles suivantes (les attributs qui forment une clé sont en gras) :*

- *Produit (**code**, nom, marque, prix)*
- *Client (**id**, nom, prénom)*
- *Achat (**codeProduit**, **idClient**, date, quantité)*

*On donne ci-dessous une requête SQL et le plan d'exécution fourni par Oracle :*

```
select quantité
from Produit p, Client c, Achat a
where p.code = a.codeProduit
and   c.id = a.idClient
and   prix > 50
```

*Plan d'exécution :*

```
0 SELECT STATEMENT
  1 MERGE JOIN
```

```
2 SORT JOIN
  3 NESTED LOOPS
    4 TABLE ACCESS FULL ACHAT
    5 TABLE ACCESS BY INDEX ROWID PRODUIT
      6 INDEX UNIQUE SCAN A34561
7 SORT JOIN
  8 TABLE ACCESS FULL Client
```

1. *Que peut-on déduire de ce plan : peut-on savoir s'il existe un index sur la table Client ? Sur la table Produit ? Sur la table Achat ? Si vous pensez qu'un index existe, donnez les attributs indexés. Justifiez vos réponses.*

2. *Algorithme de jointure : Expliquer en détail le plan d'exécution appliqué par ORACLE (accès aux tables, sélections, jointure, projections)*

3. *Quel(s) index peut-on ajouter pour obtenir les meilleures performances possibles ? Donnez, sous la forme que vous souhaitez, le plan d'exécution après ajout d'index.*

4. *On fait une jointure entre les tables Produit et Achat. Produit occupe 500 blocs, Achat 10 000 blocs. Vaut-il mieux appliquer un algorithme de boucles imbriquées ou de hachage si la mémoire $M$ disponibe est de $600$ blocs ? Et si elle est de 101 blocs ? Indiquez dans chaque cas le nombre d'entrées/sorties (sans prendre en compte l'écriture du résultat final).*

**Exercice 34.** *Soit le schéma relationnel :*

```
Journaliste (jid, nom, prénom)
Journal (titre, rédaction, id_rédacteur)
```

*La table* `Journaliste` *stocke les informations (nom, prénom) sur les journalistes (*`jid` *est le numéro d'identification du journaliste). La table* `Journal` *stocke pour chaque rédaction d'un journal le titre du journal (titre), le nom de la rédaction (redaction) et l'id de son rédacteur (*`rédacteur_id`*). Le titre du journal est une clé . On a un index dense sur la table* `Journaliste` *sur l'attribut* `jid`*, nommé* `Idx-Journaliste-jid`*.*

*On considère la requête suivante :*

```
SELECT nom
FROM Journal, Journaliste
WHERE titre='Le Monde'
AND jid=id_redacteur
AND prenom='Jean'
```

1. *Voici deux expressions algébriques :*

   *(a)* $\pi_{nom}(\sigma_{titre='Le\,Monde' \wedge prenom='Jean'}(Journaliste \bowtie_{jid=redacteur\_id} Journal))$

   *(b)* $\pi_{nom}(\sigma_{prenom='Jean'}(Journaliste) \bowtie_{jid=redacteur\_id} \sigma_{titre='Le\,Monde'}(Journal))$

   *Les deux expressions retournent-elles le même résultat (sont-elles équivalentes) ? Une expression est-elle meilleure que l'autre si on les considère comme des plans d'exécution ?*

2. *Donner le meilleur plan d'exécution physique sous forme arborescente ou sous forme d'une expression EXPLAIN, et expliquez en détail ce plan.*

**Exercice 35.** *Soit la base d'une société d'informatique décrivant les clients, les logiciels vendus, et les licences indiquant qu'un client a acquis un logiciel.*

```
Société (id, intitulé)
Logiciel (id, nom)
Licence (idLogiciel, idSociété, durée)
```

*Bien entendu un index unique est créé sur les clés primaires. Pour chacune des requêtes suivantes, donner le plan d'exécution qui vous semble le meilleur.*

- ```
  SELECT intitulé
  FROM  Société, Licence
  WHERE durée = 15
  AND   id = idSociete
  ```
- ```
  SELECT intitule
  FROM  Société, Licence, Logiciel
  WHERE nom='EKIP'
  AND   Société.id = idSociete
  AND   Logiciel.id = idLogiciel
  ```
- ```
  SELECT intitule
  FROM  Société, Licence
  WHERE Société.id = idSociete
  AND   idLogiciel IN (SELECT id FROM Logiciel WHERE nom='EKIP')
  ```
- ```
  SELECT intitule
  FROM  Société s, Licence c
  WHERE s.id = c.idSociete
  AND   EXISTS (SELECT * FROM Logiciel l
                WHERE nom='EKIP' AND c.idLogiciel=l.idLogiciel)
  ```

## 3.2 Plans d'exécution ORACLE

**Exercice 36.** *On prend les tables suivantes, abondamment utilisées par ORACLE dans sa documentation :*

- `Emp (empno, ename, sal, mgr, deptno)`
- `Dept (deptno, dname, loc)`

*La table* `Emp` *stocke des employés, la table* `Dept` *stocke les départements d'une entreprise. La requête suivante affiche le nom des employés dont le salaire est égal à 10000, et celui de leur département.*

```
SELECT  e.ename, d.dname
FROM    emp e, dept d
WHERE   e.deptno = d.deptno
AND     e.sal = 10000
```

*Voici des plans d'exécution donnés par ORACLE, qui varient en fonction de l'existence ou non de certains index. Dans chaque cas expliquez ce plan.*

1. *Index sur* `Dept(deptno)` *et sur* `Emp(Sal)`.

```
Plan d'execution
----------------------------------------------------------------------
```

```
0 SELECT STATEMENT
  1 NESTED LOOPS
    2 TABLE ACCESS BY ROWID EMP
      3 INDEX RANGE SCAN EMP_SAL
    4 TABLE ACCESS BY ROWID DEPT
      5 INDEX UNIQUE SCAN DEPT_DNO
```

2. *Index sur* `Emp(sal)` *seulement.*

```
Plan d'exécution
----------------------------------------------------------------------------
0 SELECT STATEMENT
  1 NESTED LOOPS
    2 TABLE ACCESS FULL DEPT
    3 TABLE ACCESS BY ROWID EMP
      4 INDEX RANGE SCAN EMP_SAL
```

3. *Index sur* `Emp(deptno)` *et sur* `Emp(sal)`.

4. *Voici une requête légèrement différente.*

```
SELECT e.ename
FROM emp e, dept d
WHERE  e.deptno = d.deptno
AND    d.loc = 'Paris'
```

*On suppose qu'il n'y a pas d'index. Voici le plan donné par ORACLE.*

```
Plan d'execution
----------------------------------------------------------------------------
0 SELECT STATEMENT
  1 MERGE JOIN
    2 SORT JOIN
      3 TABLE ACCESS FULL DEPT
    4 SORT JOIN
      5 TABLE ACCESS FULL EMP
```

*Indiquer quel(s) index on peut créer pour obtenir de meilleures performances (donner le plan d'exécution correspondant).*

5. *Que pensez-vous de la requête suivante par rapport à la précédente ?*

```
SELECT e.ename
FROM emp e
WHERE  e.deptno IN (SELECT d.deptno
                    FROM Dept d
                    WHERE d.loc = 'Paris')
```

*Voici le plan d'exécution donné par ORACLE :*

```
0 SELECT STATEMENT
  1 MERGE JOIN
    2 SORT JOIN
```

```
      3 TABLE ACCESS FULL EMP
    4 SORT JOIN
      5 VIEW
        6 SORT UNIQUE
          7 TABLE ACCESS FULL DEPT
```

*Qu'en dites vous ?*

*Sur le même schéma, voici maintenant la requête suivante.*

```
SELECT *
FROM Emp e1 WHERE sal IN (SELECT salL
                          FROM Emp e2
                          WHERE e2.empno=e1.mgr)
```

*Cette requête cherche les employés dont le salaire est égal à celui de leur patron. On donne le plan d'exécution avec Oracle (outil EXPLAIN) pour cette requête dans deux cas : (i) pas d'index, (ii) un index sur le salaire et un index sur le numéro d'employé.*

*Expliquez dans les deux cas ce plan d'exécution (éventuellement en vous aidant d'une représentation arborescente de ce plan d'exécution).*

1. *Pas d'index.*

   ```
   Plan d'exécution
   ----------------------------------------------------------------------
   0 FILTER
     1 TABLE ACCESS FULL EMP
     2 TABLE ACCESS FULL EMP
   ```

2. *Index sur* `empno` *et index sur* `sal`.

   ```
   Plan d'exécution
   ----------------------------------------------------------------------
   0 FILTER
     1 TABLE ACCESS FULL EMP
     2 AND-EQUAL
       3 INDEX RANGE SCAN I-EMPNO
       4 INDEX RANGE SCAN I-SAL
   ```

3. *Dans le cas où il y a les deux index (salaire et numéro d'employé), on a le plan d'exécution suivant :*

   ```
   Plan d'exécution
   ----------------------------------------------------------------------
   0 FILTER
     1 TABLE ACCESS FULL EMP
     2 TABLE ACCESS ROWID EMP
       3 INDEX RANGE SCAN I-EMPNO
   ```

   *Expliquez-le.*

**Exercice 37.** *Soit le schéma suivant :*

```
CREATE TABLE Artiste  (id_artiste INT NOT NULL,
                       nom VARCHAR (60),
                       prénom VARCHAR (60),
                       année_naissance INT,
                       PRIMARY KEY (id_artiste));

CREATE TABLE film  (id_film  INT NOT NULL,
                    titre    VARCHAR(60),
                    année     INT,
                    id_réalisateur INT,
                    PRIMARY KEY (id_film),
                    FOREIGN KEY (id_réalisateur) REFERENCES Artiste);

CREATE TABLE seance  (nom_cinema   VARCHAR (60) NOT NULL,
                      no_salle     NUMBER(2) NOT NULL,
                      no_séance    NUMBER(2) NOT NULL,
                      heure_debut  INT,
                      heure_fin    INT,
                      id_film      INT  NOT NULL,
                      PRIMARY KEY (nom_cinema, no_salle, no_séance),
                      FOREIGN KEY (id_film) REFERENCES Film
                             ON DELETE CASCADE);
```

*Questions :*

1. *Donner l'ordre SQL pour la requête :* Quels sont les films d'Hitchcock visibles après 20h00 ?
2. *Donner l'expression algébrique correspondante et proposez un arbre de requête qui vous paraît optimal.*
3. *Sous ORACLE, l'outil EXPLAIN donne le plan d'exécution suivant :*

```
0 SELECT STATEMENT
  1 MERGE JOIN
    2 SORT JOIN
      3 NESTED LOOPS
        4 TABLE ACCESS FULL ARTISTE
        5 TABLE ACCESS BY ROWID FILM
          6 INDEX RANGE SCAN IDX-ARTISTE-ID
    7 SORT JOIN
     8 TABLE ACCESS FULL SEANCE
```

*Commentez le plan donné par EXPLAIN. Pourrait-on améliorer les performances de cette requête ?*

**Exercice 38.** *Soit le schéma, la requête et le plan d'exécution ORACLE suivants :*

```
CREATE TABLE TGV (
        NumTGV integer,
        NomTGV varchar(32),
        GareTerm varchar(32));

CREATE TABLE Arret (
        NumTGV integer,
```

```
        NumArr integer,
        GareArr varchar(32),
        HeureArr varchar(32));


EXPLAIN PLAN
    SET statement_id = 'eds0'
    FOR select NomTGV
  from TGV, Arret
 where TGV.NumTGV = Arret.NumTGV
   and GareTerm = 'Aix';

@exbdb;

/* sans index */
0 SELECT STATEMENT
  1 MERGE JOIN
    2 SORT JOIN
      3 TABLE ACCESS FULL ARRET
    4 SORT JOIN
      5 TABLE ACCESS FULL TGV
```

- *Que calcule la requête ?*
- *Que pouvez-vous dire sur l'existence d'index pour les tables "TGV" et "Arret" ? Décrivez en détail le plan d'exécution : quel algorithme de jointure a été choisi, quelles opérations sont effectuées et dans quel ordre ?*
- *On fait la création d'index suivante :*
  `CREATE INDEX  index arret_numtgv ON Arret(numtgv);`
  *L'index créé est-il dense ? unique ? Quel est le plan d'exécution choisi par Oracle ? Vous pouvez donner le plan avec la syntaxe ou sous forme arborescente. Expliquez en détail le plan choisi.*
- *On rajoute encore un index :*
  `CREATE INDEX tgv_gareterm on tgv(gareterm);`
- *Quel est le plan d'exécution choisi par Oracle ? Expliquez le en détail.*

  *C'est presque le même plan, sauf que au lieu de balayer tous les TGV (toute la table TGV), on accède directement à ceux dont le terminus est 'Aix' en traversant l'index sur les gares terminus*

## 3.3 Utilisation de EXPLAIN et de TKPROF

Le site *Films* propose les fichiers suivants pour créer et alimenter une base (sur des films bien sûr...).

1. Le fichier *Schema.sql* contient les commandes de création du schéma sous ORACLE.
2. Le fichier *CrFilms.pc* est un programme PRO*C pour alimenter cette base. L'entête du fichier contient les instructions pour le compiler et l'exécuter.

Avec ces deux fichiers vous pouvez créer une base de taille quelconque, et tester l'évaluation des requêtes avec EXPLAIN et TKPROF.

**Utilisation de EXPLAIN**

Voici le mode d'utilisation de EXPLAIN.

– Les plans d'exécution sont stockés dans une table *PLAN_TABLE* dont le script de création se trouve, en principe, dans *$ORACLE_HOME/rdbms/admin/utlxplan.sql*. Vous pouvez également la récupérer sur le site
– Ensuite on stocke le plan d'exécution d'une requête dans cette table avec la commande `EXPLAINJ PLAN`. Voici un exemple :

```
EXPLAIN PLAN
        SET statement_id = 'plan0'
        FOR SELECT a.nom
            FROM   film f, artiste a
            WHERE  f.idMES = s.idArtiste
            AND    f.titre = 'Vertigo';
```

La clause '`statement_id = 'plan0''` attribue un identifiant au plan d'exécution de cette requête dans la table *PLAN_TABLE*. Bien entendu vous devez donner à chaque requête stockée un identifiant spécifique.

– Pour connaître le plan d'exécution, on interroge la table `PLAN_TABLE`. L'information est un peu difficile à interpréter : le plus simple est de faire tourner le fichier *Explain.sql* (à récupérer sur le site) dont voici le code :

```
undef query;

SELECT LPAD(' ',2*(LEVEL-1))||operation||' '||options
   ||' '||object_name
   "Plan d'exécution"
   FROM plan_table
   START WITH id = 0 AND statement_id = '&&query'
   CONNECT BY PRIOR id = parent_id AND statement_id ='&&query';
```

Quand on exécute ce fichier, il demande (2 fois) le nom du plan d'exécution à afficher. Dans le cas de l'exemple ci-dessus, on répondrait deux fois `'plan0'`. On obtient l'affichage suivant qui présente de manière relativement claire le plan d'exécution.

```
Plan d'exécution
----------------------------------------------------------
0 SELECT STATEMENT
  1 NESTED LOOPS
    2 TABLE ACCESS FULL Film
    3 TABLE ACCESS BY ROWID Artiste
      4 INDEX UNIQUE SCAN SYS_C004709
```

Ici, le plan d'exécution est le suivant : on parcourt en séquence la table *Film* (ligne 2) ; pour chaque séance, on accède à la table *Artiste* par l'index [1] (ligne 4), puis pour chaque `ROWID` provenant de l'index, on accède à la table elle-même (ligne 3). Le tout est effectué dans une boucle imbriquée (ligne 1).

1. Cet index a été automatiquement créé en association avec la commande `PRIMARY KEY`.

**Utilisation de TKPROF**

TKPROF est complémentaire de EXPLAIN : il donne les différents coûts constatés pour l'exécution d'une requête. Le principe d'utilisation est le suivant : on passe en mode TRACE avec la commande suivante (sous SQLPLUS)

```
ALTER SESSION SET SQL_TRACE = TRUE;
```

À partir de ce moment-là, toutes les requêtes exécutées sous SQLPLUS entraîneront l'insertion dans un fichier des informations sur le coût d'exécution. Quand on a exécuté toutes les requêtes à analyser, on stoppe le mode TRACE avec :

```
ALTER SESSION SET SQL_TRACE = FALSE;
```

Il reste à récupérer le fichier de trace et à l'analyser avec l'utilitaire TKPROF. Pour localiser le fichier, on peut utiliser la requête suivante :

```
SELECT value
FROM   v$parameter
WHERE  name = 'user_dump_dest';
```

On obtient le répertoire où se trouve le fichier, dont le nom suit le format *oraSID_ora_noProcessus*. Il reste à lui appliquer le programme TKPROF :

`tkprof` *`nomFichier`*

Vous devez bien entendu avoir les droits de lecture sur le fichier, ce qui peut poser problème si vous n'êtes pas l'administrateur. Il reste une troisième solution, peut-être la plus simple, avec l'option AUTOTRACE de SQLPLUS.

**Cumuler EXPLAIN et TKPROF avec AUTOTRACE**

En passant en mode AUTOTRACE sous SQLPLUS, on obtient les principales informations fournies par EXPLAIN et TKPROF sans avoir besoin d'effectuer les manipulations détaillées précédemment. Avant de pouvoir utiliser cette option, il faut avoir effectuer la petite installation suivante :

1. créer le rôle PLUSTRACE avec le script *$ORACLE_HOME/sqlplus/admin/plustrce.sql* (il faut exécuter ce script sous le compte SYS) ;
2. donner ce rôle avec la commande GRANT à tout utilisateur souhaitant faire appel à AUTOTRACE.

Si, maintenant, on est un de ces utilisateurs, on se place en mode AUTOTRACE avec la commande suivante :

```
ALTER SESSION SET AUTOTRACE {OFF | ON | TRACEONLY} [EXPLAIN] [STATISTICS]
```

Le mode TRACEONLY permet de ne pas afficher le résultat de la requête, ce qui est le but recherché en général.

### 3.4 Exercices d'application

**Exercice 39.** *Effectuer des requêtes en mode TRACE, afin d'étudier les plans d'exécution. Pour l'instant, se mettre en mode « règles ».*

1. *Des sélections sur des champs indexés ou non indexés (faire une sélection sur le genre par exemple)*
2. *Des jointures (penser à inverser l'ordre des tables dans le* `FROM`*)*
3. *Des requêtes avec* `NOT IN` *ou* `NOT EXISTS`

**Exercice 40.** *Maintenant, analyser les tables, les index et les distributions avec* `ANALYSE`*, et regarder les modifications des plans d'exécution en optimisation par les coûts. En particulier*

- *Chercher des films par le genre en mode règles*
- *Chercher des films par le genre en mode coûts*

*Le genre étant peu sélectif, l'optimiseur ne devrait pas utiliser l'index dans le second cas.*

**Exercice 41.** *Reprendre les requêtes du polycopié cherchant le film paru en 1958 avec James Stewart. Analyser les différentes versions (sans imbrication, avec imbrication mais sans corrélation (*`IN`*), avec imbrication et corrélation (*`EXISTS`*), etc). Comparer les plans d'exécution obtenus dans chaque cas.*

**Exercice 42.** *Supprimer quelques index, et regarder le changement dans les plans d'exécutions des requêtes précédentes. NB : vous pouvez obtenir la liste des index existant sur vos tables avec la commande :*

```
SELECT table_name, index_name FROM user_indexes;
```

**Exercice 43.** *Maintenant appliquez des fonctions de groupe, des* `GROUP BY` *et des* `HAVING`*, regardez les plans d'exécution et interprétez-les.*

**Exercice 44.** *Essayer d'appliquer des fonctions dans les conditions de jointure. Que constate-t-on au niveau du plan d'exécution ?*

**Exercice 45.** *Essayer d'appliquer l'opérateur* `LIKE` *dans des sélections sur des attributs indexés. Que constate-t-on ?*

## 4 Concurrence

**Exercice 46.** *On prend l'hypothèse qu'un système sans gestion de concurrence (une seule version des données, les lectures et écritures sont faites directement sur cette version). Donnez deux exécutions concurrente du programme* RÉSERVATION*, pour le même client et deux spectacles différents, telles qu'on aboutisse à une incohérence de la base.*

**Exercice 47.** *On crée une table* Compteur *pour attribuer des identifiants uniques*

```
CREATE TABLE Compteur (val INT NOT NULL);
INSERT INTO Compteur VALUE (0);
```

*Voici le pseudo-code d'une procédure qui génère et renvoie un nouvel identifiant.*

```
Procédure génère_id()
début
  my_val INTEGER;
  SELECT val INTO :my_val FROM Compteur;
  my_val = my_val+1;
  UPDATE Compteur SET val=:my_val;
  RETURN my_val;
fin
```

- *Pour chaque niveau d'isolation de la norme SQL, indiquer si deux processus exécutant indépendamment la procédure* `génère_id()` *peuvent obtenir le même numéro d'identifiant.*
- *En mode* `READ UNCOMMITTED`*, donnez un exemple d'exécution concurrente, comprenant des* `commit` *ou* `rollback`*, aboutissant à un "trou" dans la séquence des valeurs dans la table* Compteur.
- *Cette anomalie est-elle possible en* `READ COMMITTED` *?*
- *Comment modifier la procédure pour assurer que chaque numéro généré est unique, et ce quel que soit le niveau d'isolation ?*
- *Quel problème peut quand même survenir ?*

**Exercice 48.** *Voici le schéma d'une base de données pour un marchand de voitures.*

- `Modèle (nomModèle, marque, prix)`
- `Véhicule (idVéhicule, nomModèle, couleur)`

*Donnez le code (en PL/SQL par exemple), des procédures suivantes. À chaque fois indiquez, si besoin est, l'emplacement des* `commit` *ou des* `rollback`*.*

1. *Pour un modèle donné (désigné par son nom), afficher une ligne avec la marque et le prix, puis autant de lignes que de véhicules avec leur couleur.*
2. *Détruire tout ce qui concerne un modèle donné ;*
3. *On ajoute les tables* `Option (nomOption, tarif)` *et* `OptionsVéhicule(idVéhicule, nomOption)`*. On veut tenir compte de ces options pour le prix du véhicule.*
   - *Ajouter un attribut* `prix` *à* Véhicule*, et écrire la procédure qui affecte au prix de chaque véhicule le prix du modèle, plus le cumul du tarif de ses options.*
   - *Définir une vue* `VéhiculePrix(idVéhicule, nomModèle, couleur, prix)`*, où* `prix` *est calculé comme précédemment*

   *Discutez des avantages et inconvénients des deux solutions.*
4. *On reçoit un véhicule, avec une marque, un modèle, un prix et une couleur : écrire la procédure qui insère ce véhicule, ainsi que le modèle s'il n'existe pas déjà.*

**Exercice 49.** *Soit le programme suivant qui transfère (certes de manière un peu baroque) un montant d'un compte à un autre, en s'assurant qu'aucun ne descend en dessous de 0 :*

```
Transfert (int id_c1, int id_c2, double montant)
Début
  UPDATE Compte SET solde=solde + :montant
   WHERE id=:id_c2;

  SELECT solde INTO :solde_c1
```

```
  FROM Compte WHERE id = :id_c1;

  Si (solde_c1 < montant)
    UPDATE Compte SET solde=solde - :montant
     WHERE id=:id_c2;
  Sinon
    UPDATE Compte SET solde=solde - :montant
     WHERE id=:id_c1;
  Fin
Fin
```

*On suppose qu'on a trois comptes : A , B, et C, avec respectivement 100, 200 et 300 Euros. On exécute $T_1$ pour transférer 150 euros de A à B, et $T_2$ pour transférer 250 euros de B à C.*

*Décrire le déroulement de l'exécution concurrente suivante, et le résultat final obtenu, en supposant que le système n'applique pas de contrôle de concurrence (mode `READ UNCOMMITTED`) :*

$$w_2[C]w_1[B]r_2[B]r_1[A]w_2[B]w_1[B]$$

**Exercice 50.** *On considère un programme de copie entre deux comptes. Ce programme est donné de manière simplifiée ci-dessous.*

```
Copie (compte1, compte2)
début
  SELECT solde INTO :v_solde
   FROM Compte WHERE nom='compte1';

  UPDATE Compte SET solde = :v_solde
  WHERE nom='compte2';

  Commit;
fin
```

*On dispose de deux comptes $A$ et $B$ et on lance à peu près simultanément deux exécutions $Copie(A, B)$ et $Copie(B, A)$.*

1. *Quel état final de la base caractérise le mode sérialisable de ces deux exécutions ?*
2. *On se place en mode `READ COMMITTED` (le système ne pose pas de verrou en lecture). Donnez une exécution concurrente aboutissant à un résultat incorrect.*
3. *On se place en mode `SERIALIZABLE`. Donnez une exécution concurrente aboutissant à un interblocage.*
4. *On se place en mode `READ COMMITTED` et on essaie d'éviter les deux problèmes précédents en ajoutant un verrouillage explicite dans le programme, comme suit (la clause `FOR UPDATE` pose un verrou exclusif).*

   ```
   Copie (compte1, compte2)
   début
     SELECT *  FROM Compte WHERE nom='compte2' FOR UPDATE;
     SELECT solde INTO v_solde FROM Compte WHERE nom='compte1';
     UPDATE Compte SET solde = v_solde WHERE nom='compte2';
     Commit;
   fin
   ```

*Évite-t-on une exécution incorrecte ? Évite-t-on les interblocages ?*

**Exercice 51.** *On considère maintenant le problème (délicat) de la réservation des places pour un match. Pour cela on dispose des tables* Stade*(id_stade, nb_places),* Match *(id_match, id_stade, nb_places_prises) et* Client *(id_client, nb_places_réservées). Les opérateurs disposent du petit programme suivant pour réserver des places.*

```
Places (id_client, id_match, nb_places)
début
  Lire le match m                  // Donne le nbre de places prises
  Lire le stade s                  // Donne le nbre total de places

  if ( (s.nb_places - m.nb_places_prises) > nb_places)
         // Il reste assez de places
  début
    Lire le client c
    m.nb_places_prises += nb_places;
    c.nb_places_réservées += nb_places;
    Ecrire le match m
    Ecrire le client c
  fin
  commit
fin
```

*Les organisateurs s'aperçoivent rapidement qu'il n'y a pas assez de place dans les stades en entreprennent des travaux d'agrandissement. On a le deuxième programme :*

```
Augmenter (id_stade, nb_places)
début
  Lire s
  s.places += nb_places
  Ecrire s
  commit
fin
```

1. *On lance simultanément deux exécutions du programme* Augmenter (SF, 1000) *pour augmenter la capacité du Stade de France et on suppose que le système applique un mode d'isolation* `READ UNCOMMITTED`*.*

   (a) *Donnez un exemple d'une exécution concurrente non sérialisable.*

   (b) *Donnez un exemple d'une exécution concurrente non recouvrable (en plaçant un ordre* `commit`*).*

2. *On a maintenant une exécution concurrente de* Places (C, M, 1500) *et de* Augmenter (SF, 1000)*, $M$ étant un match qui se déroule au Stade de France. Voici le début de l'exécution* [2] *:*

$$\mathbf{H} = r_P[M] r_P[SF] r_A[SF] w_A[SF] c_A \ldots$$

2. l'indice $P$ désigne $Places$, $A$ $Augmenter$.

(a) *Donner la fin de l'exécution concurrente en supposant qu'il y a 2 000 places libres au début de l'exécution. Donnez également le nombre de places libres pour le match à la fin de l'exécution.*

(b) *Cette exécution concurrente est-elle sérialisable ?*

(c) *Est-il possible d'obtenir une exécution non sérialisable pour ces deux transactions*

(d) *On suppose qu'on a ajouté des clauses* `FOR UPDATE` *sur chaque lecture. Le* `FOR UPDATE` *pose un verrou exclusif qui est libéré au moment du* `commit` *ou du* `rollback`.
*Donner l'exécution concurrente qui en résulte en reprenant l'ordre des opérations donné ci-dessus. Conclusion, les* `FOR UPDATE` *étaient-ils nécessaires ? Quels sont les lectures pour lesquelles ils sont vraiment utiles ?*

3. *Écrire une procédure qui compare le nombre de places réservées pour un match avec la somme des places des clients ayant réservé (pour simplifier on supposera que tous les clients ont réservé pour le même match). Donner un exemple d'exécution concurrente avec la procédure* `Augmenter`*, en mode* `READ UNCOMMITTED`*, qui fausse le résultat obtenu. Quel est le bon niveau d'isolation ?*

4. *Soit l'exécution concurrente du programme* Places *:* $T_1$ = Places (C, M, 100) *et* $T_2$ = Places (C, M, 200) *pour le même match et le même client.*

$$\mathbf{H} = r_1[S]r_2[S]r_1[M]r_1[C]w_1[M]r_2[M]r_2[C]w_2[M]w_1[C]c_1w_2[C]c_2$$

(a) *Montrer que* **H** *n'est pas sérialisable.*

(b) *On suppose que le système de gère pas la concurrence (mode* `READ UNCOMMITED`*) et qu'il y a 1 000 places prises dans* $Match$ *au début de l'exécution. Combien y en a-t-il à la fin de l'exécution de* **H** *? Quel est le nombre de places réservées par* $C$*, en supposant qu'il n'avait rien réservé au départ ?*

(c) *Choisir le niveau d'isolation approprié pour que cette exécution se déroule correctement.*

(d) *Appliquer un verrouillage à deux phases sur* **H** *et donner l'exécution* **H'** *résultante. Les verrous sont relâchés après le* `commit`.

**Exercice 52.** *Soit* $T_1$ *et* $T_2$ *deux transactions et* $x$*,* $y$ *deux items de la base. L'ordonnancement suivant est-il sérialisable ? Est-il accepté par un verrouillage en deux phases ?*

| **Opération** | $T_1$ | $T_2$ |
|---|---|---|
| (1) | $r_1(x)$ | |
| (2) | $w_1(x)$ | |
| (3) | | $r_2(x)$ |
| (4) | $w_1(y)$ | |
| (5) | $r_1(y)$ | |
| (6) | | $r2(y)$ |

**Exercice 53** (Verrouillage à 2 phases)**.** *Un contrôleur avec verrouillage à 2 phases reçoit la séquence d'opérations ci-dessous.*

$$r_1[x]\ r_2[y]\ w_3[x]\ w_1[y]\ w_1[x]\ w_2[y]\ c_2\ r_3[y]\ r_1[y]\ c_1\ w_3[y]\ c_3$$

*Indiquez l'ordre d'exécution établi par le contrôleur, en considérant qu'une opération bloquée en attente d'un verrou est exécutée en priorité dès que le verrou devient disponible. On suppose que les verrous d'une transaction sont relâchés au moment du* `Commit`.

**Exercice 54** (Graphe de sérialisabilité et équivalence des exécutions)**.** *Construisez les graphes de sérialisabilité pour les exécutions (histoires) suivantes. Indiquez les exécutions sérialisables et vérifiez s'il y a des exécutions équivalentes.*

1. $H_1 : w_2[x]\ w_3[z]\ w_2[y]\ c_2\ r_1[x]\ w_1[z]\ c_1\ r_3[y]\ c_3$
2. $H_2 : r_1[x]\ w_2[y]\ r_3[y]\ w_3[z]\ c_3\ w_1[z]\ c_1\ w_2[x]\ c_2$
3. $H_3 : w_3[z]\ w_1[z]\ w_2[y]\ w_2[x]\ c_2\ r_3[y]\ c_3\ r_1[x]\ c_1$

**Exercice 55** (Recouvrabilité)**.** *Parmi les exécutions concurrentes suivantes, lesquelles sont recouvrables (impossibilité d'assurer correctement un* `commit` *ou un* `rollback`*), lesquelles évitent les annulations en cascade (l'annulation d'une transaction entraîne l'annulation d'une ou plusieurs autres) ?*

*Indiquez s'il y a des exécutions sérialisables.*

1. $H_1 : r_1[x]\ w_2[y]\ r_1[y]\ w_1[x]\ c_1\ r_2[x]\ w_2[x]\ c_2$
2. $H_2 : r_1[x]\ w_1[y]\ r_2[y]\ c_1\ w_2[x]\ c_2$
3. $H_3 : r_1[y]\ w_2[x]\ r_2[y]\ w_1[x]\ c_2\ r_1[x]\ c_1$

**Exercice 56.** *Le programme suivant s'exécute dans un système de gestion de commandes pour les produits d'une entreprise. Il permet de commander une quantité donnée d'un produit qui se trouve en stock. Les paramètres du programme représentent respectivement la référence de la commande ($c$), la référence du produit ($p$) et la quantité commandée ($q$).*

```
Commander (c, p, q)
début
  Lecture prix produit p
  Lecture du stock de p
  si (q > stock de p) alors
    rollback
  sinon
     Mise à jour stock de p;
     Enregistrement dans c du total de facturation
     commit
  fin si
fin
```

*Notez que le prix et la quantité de produit en stock sont gardés dans des enregistrements différents.*

1. *Lesquelles des transactions suivantes peuvent être obtenues par l'exécution du programme ci-dessus ? Justifiez votre réponse.*
   *(a)* $T_1 : r[x]r[y]R$
   *(b)* $T_2 : r[x]R$
   *(c)* $T_3 : r[x]w[y]w[z]C$
   *(d)* $T_4 : r[x]r[y]w[y]w[z]C$
2. *Dans le système s'exécutent en même temps trois transactions : deux commandes d'un même produit et le changement du prix de ce même produit. Montrez que l'histoire ci-dessous est une exécution concurrente de ces trois transactions et expliquez la signification des enregistrements qui y interviennent.*
   **H** *:* $r_1[x]\ r_1[y]\ w_2[x]\ w_1[y]\ c_2\ r_3[x]\ r_3[y]\ w_1[z]\ c_1\ w_3[y]\ w_3[u]\ c_3$
3. *Vérifiez si* **H** *est sérialisable en identifiant les conflits et en construisant le graphe de sérialisation.*
4. *Quelle est l'exécution obtenue par verrouillage à deux phases à partir de* **H** *? Quel prix sera appliqué pour la seconde commande, le même que pour la première ou le prix modifié ?*
   *On considère que le relâchement des verrous d'une transaction se fait au* Commit *et qu'à ce moment on exécute en priorité les opérations bloquées en attente de verrou, dans l'ordre de leur blocage.*

**Exercice 57** (Concurrence : Gestion Bancaire)**.** *Les trois programmes suivants peuvent s'exécuter dans un système de gestion bancaire.* Débit *diminue le solde d'un compte* `c` *avec un montant donné* `m`*. Pour simplifier, tout débit est permis (on accepte des découverts).* Crédit *augmente le solde d'un compte* `c` *avec un montant donné* `m`*.* Transfert *transfère un montant* `m` *à partir d'un compte source* `s` *vers un compte destination* `d`*. L'exécution de chaque programme démarre par un* **Start** *et se termine par un* **Commit** *(non montrés ci-dessous).*

```
Débit (c:Compte;   |  Crédit (c:Compte;   |  Transfert (s,d:Compte;
       m:Montant)  |          m:Montant)  |              m:Montant)
begin              |  begin               |  begin
  t := Read(c);    |    t := Read(c);     |    Débit(s,m);
  Write(c,t-m);    |    Write(c,t+m);     |    Crédit(d,m);
end                |  end                 |  end
```

*Le système exécute en même temps les trois opérations suivantes :*
*(1) un transfert de montant 100 du compte A vers le compte B*
*(2) un crédit de 200 pour le compte A*
*(3) un débit de 50 pour le compte B*

1. *Écrire les transactions* $T_1$*,* $T_2$ *et* $T_3$ *qui correspondent à ces opérations. Montrer que l'histoire* **H** *:* $r_1[A]\ r_3[B]\ w_1[A]\ r_2[A]\ w_3[B]\ r_1[B]\ c_3\ w_2[A]\ c_2\ w_1[B]\ c_1$ *est une exécution concurrente de* $T_1$*,* $T_2$ *et* $T_3$*.*
2. *Mettre en évidence les conflits dans* **H** *et construire le graphe de sérialisation de cette histoire.* **H** *est-elle sérialisable ?* **H** *est-elle recouvrable ?*

3. *Quelle est l'exécution* **H'** *obtenue à partir de* **H** *par verrouillage à deux phases ? On suppose que les verrous d'une transaction sont relâchés après le Commit de celle-ci. Une opération bloquée en attente d'un verrou bloque le reste de sa transaction. Au moment du relâchement des verrous, les opérations en attente sont exécutées en priorité.*
   *Si au début le compte A avait un solde de 100 et B de 50, quel sera le solde des deux comptes après la reprise si une panne intervient après l'exécution de* $w_1[B]$ *?*

**Exercice 58.** *Dans un contrôleur multi-versions, on met en attente une transaction* $T$ *qui essaye d'effectuer une mise à jour sur un tuple* $a$ *qui a verrouillé en écriture. Pourquoi ne pas rejeter directement cette transaction ? Donner un cas où cette transaction finit par s'exécuter.*

**Exercice 59.** *Soit* $T_1$*,* $T_2$*,* $T_3$ *et* $T_4$ *quatre transactions et* $x$*,* $y$*,* $z$ *trois enregistrements. On considère l'ordonnancement suivant :*

| **Instant** | $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|---|
| (15) | | $r_2[x]$ | | |
| (16) | | | $r_3[x]$ | |
| (17) | | $w_2[y]$ | | |
| (18) | | | $w_3[x]$ | |
| (19) | | $r_2[z]$ | | |
| (20) | $r_1[y]$ | | | |
| (21) | | | | $r_4[y]$ |
| (22) | $r_1[x]$ | | | |
| (23) | $w_1[z]$ | | | |
| (24) | | | | $w_4[x]$ |

1. *Cet ordonnancement est-il sérialisable ? Si oui, donner l'ordre séquentiel des transactions équivalent.*

2. *Donnez l'ordonnancement obtenu par un verrouillage à deux phases ?*

3. *On suppose qu'avant le début des transactions les estampilles de lecture et d'écriture de* $x$*,* $y$ *et* $z$ *sont toutes égales à 10. Les opérations s'exécutent dans l'ordre indiqué, l'instant de la demande d'exécution étant indiqué en première colonne.*
   *Appliquer l'algorithme de contrôle multi-version sur cette exécution concurrente. Que se passe-t-il si* $T_3$ *choisit finalement de valider ? Et si* $T_3$ *choisit d'annuler ?*

**Exercice 60.** *L'exécution suivante est reçue par un SGBD :*

$$H : r_1[x]r_2[z]r_1[y]w_1[x]r_3[x]r_2[y]w_2[z]w_2[y]c_2r_3[y]r_3[z]c_3w_1[y]c_1$$

1. *Parmi les transactions* $T_1$*,* $T_2$*,* $T_3$*, certaines correspondent au virement d'un compte vers un autre : on lit les valeurs des deux comptes, puis on les modifie. De quelles transactions s'agit-il ?*

2. *Existe-t-il des "lectures sales" dans* $H$ *? Indiquez-les, ainsi que les conséquences possibles.*

3. *Vérifiez si $H$ est sérialisable en identifiant les conflits et en construisant le graphe de sérialisation.*

4. *Qu'obtient-on en appliquant un verrouillage à deux phases à partir de $H$ (indiquer le déroulement de l'exécution, et les blocages éventuels) ?*

5. *On suppose maintenant que toutes les transactions placent un* `FOR UPDATE` *pour toute lecture d'une donnée* qui va être modifiée ensuite. *Qu'obient-on si on applique à $H$ une variante du verrouillage à deux phases qui pose un verrou exclusif (au lieu d'un verrou partagé) pour les lectures comprenant la clause* `FOR UPDATE` *?*

6. *Quelle est l'exécution obtenue par l'algorithme de contrôle de concurrence avec versionnement ? Indiquez la (ou les) transaction(s) rejetée(s) et expliquez pourquoi. Quel danger a-t-on évité ?*

   *On suppose que toutes les transactions débutent au même moment.*

# A Schémas

L'exemple 1 donne le schéma de la base de données illustrant la plupart des exemples.

**Exemple 1.** *SchemaFilms.sql : Le schéma de la base Films*

```
/*
   Commandes de crion de la base Films,  testvec MySQL et PostgreSQL.
   Pour Oracle, il suffit de remplacer le type TEXT par le type LONG dans
   la table Film.
   Philippe Rigaux, 2004
*/

/* Destruction eventuelle des tables existantes */

DROP TABLE  Notation;
DROP TABLE  Role;
DROP TABLE  Film;
DROP TABLE Artiste;
DROP TABLE  Internaute;
DROP TABLE  Pays;
DROP TABLE  Genre;

/* Creation des tables                          */

CREATE TABLE Internaute (email VARCHAR (40) NOT NULL,
                         nom VARCHAR (30) NOT NULL ,
                         prenom VARCHAR (30) NOT NULL,
                         region VARCHAR (30),
                         CONSTRAINT PKInternaute PRIMARY KEY (email));

CREATE TABLE Pays (code    VARCHAR(4) NOT NULL,
                   nom  VARCHAR (30) DEFAULT 'Inconnu' NOT NULL,
                   langue VARCHAR (30) NOT NULL,
                   CONSTRAINT PKPays PRIMARY KEY (code));
```

```
CREATE TABLE Artiste  (idArtiste INTEGER NOT NULL,
                       nom VARCHAR (30) NOT NULL,
                       prenom VARCHAR (30) NOT NULL,
                       anneeNaiss INTEGER,
                       CONSTRAINT PKArtiste PRIMARY KEY (idArtiste),
                       CONSTRAINT UniqueNomArtiste UNIQUE (nom, prenom));

CREATE TABLE Film  (idFilm INTEGER NOT NULL,
                    titre    VARCHAR (50) NOT NULL,
                    annee    INTEGER NOT NULL,
                    idMES    INTEGER,
                    genre VARCHAR (20) NOT NULL,
                    /* Remplacer TEXT par LONG pour ORACLE */
                    resume      TEXT,
                    codePays    VARCHAR (4),
                    CONSTRAINT PKFilm PRIMARY KEY (idFilm),
                    FOREIGN KEY (idMES) REFERENCES Artiste,
                    FOREIGN KEY (codePays) REFERENCES Pays);

CREATE TABLE Notation (idFilm INTEGER NOT NULL,
                       email  VARCHAR (40) NOT NULL,
                       note  INTEGER NOT NULL,
                       CONSTRAINT PKNotation PRIMARY KEY (idFilm, email));

CREATE TABLE Role (idFilm  INTEGER NOT NULL,
                   idActeur INTEGER NOT NULL,
                   nomRole  VARCHAR(30),
                   CONSTRAINT PKRole PRIMARY KEY (idActeur,idFilm),
                   FOREIGN KEY (idFilm) REFERENCES Film,
                   FOREIGN KEY (idActeur) REFERENCES Artiste);

CREATE TABLE Genre (code    VARCHAR (20) NOT NULL,
                    CONSTRAINT PKGenre PRIMARY KEY (code));
```