# Fiche

## BD réparties

- Réutiliser les fragementation précédente comme filtre avec les jointures
- Fragmentation horizontale : selection sur la table général ($garage_v$ TD6-exo1)
- Fragmentation horizontale dérivé : réutilisation d'une fragmentation poru refaire une fragmentation
- Fragmentation disjointe : lorsqu'il y n'y a pas de répétition dans entre les fragments de cette table.

# TD6

## Exercice 1a

1.

Soit $v$ la ville du garage et $idgarage$ sont id

- $Garage_v = \sigma_{ville=v}(Garage)$
- $Habilite_v = Habilite \bowtie Garage_v$
- $Mecanicien_v = Mecanicien \bowtie Garage_v$
- $Personne_v = Personne \bowtie Mecanicien_v$
- $Reparation_v = Reparation \bowtie Mecanicien_v$
- $Client_v = Client \bowtie Réparation_v$
- $Possede_v = Possede \bowtie Client_v$
- ~~Tarif_v = Tarif \bowtie Reparation_v~~ $Tarif_v = Tarif$ répliqué dans toute les villes

2.

Pas disjoint si un client vas dans plusieurs ville ? Dans la correction de mathilde y'on dit que $possede_v$ et les fragment dérivé sont non disjoint ??

3.

Facile, on fait la même chose mais avec

- $Habilite_m = \sigma_m(Habilite)$
- $Garage_m = Garage \bowtie Habilite_m$

## Exercice 1b

1. On veut les marque des voiture ayant subit une réparation de moins de 100€ et avec une immatriculation < 6000
2. Version brutforce $\sigma_{prix<100 \& immat<6000}((Reparation \bowtie_{intervention} Tarif) \bowtie_{immat} Possede)$
   Version qui suit la consigne $\sigma_{prix<100}(Tarif) \bowtie \sigma_{imat<6000}(Possede) \bowtie Reparation$
3. $\sigma_{prix<100}(Tarif_{S1}) \bowtie \sigma_{imat<6000}(Possede_{S1}) \bowtie Reparation_{S1}$ Wouah compliqué compliqué la correction alors que c'était évident
4. C'était dire qu'on utilisait que le S1 car on a pas de donnée correspondant à nos filtre en S2
5. Aucune donnée transféré car on peut tout faire en S1. Efficace

# TD7

## Exercice 4 page 4 (ER2-19)

1. $P(R) = \frac{card(R)}{T_{page}/largeur(R)} = \frac{card(R)}{a} \Leftrightarrow card(R) = P(R) * T_{page}/largeur(R)$

Taille d'un tuple de $R_1$ : $t(R_1) = 2 * 10 = 20$ octet

- $card(R_1) = 10000 * 4000/20 = 10000 * 200 = 2 * 10^6$
- $card(R_1 \bowtie R_2) = card(R_2) * \frac{card(R_1)}{nbValDistinctDeADansR_1} = card(R_2) = 2 * 10^7$ car $R_2$ beaucoup plus grand que $R_1$ donc

2. $P(T) = card(R_1 \bowtie R_2) * \frac{largeur(T)}{T_{page}} = 2 * 10^7 * \frac{30}{4000} = 150000$ ;
   $P(Q) = card(Q) * \frac{40}{4000} = card(T) * 1/100 = 2 * 10^5$
3. Par Hashage
   1. Pas de transfert, jointure par hachage externe, avec $R_2$ et $R_1$ qui ne tienne pas en mémoire avec $K = 200, cout(R_1 \bowtie R_2) = 2 \lfloor \log_K(P(R_1)) \rfloor (P(R_1) + P(R_2)) + P(R) + P(R_2) = P(R_1) * t_{io} + P(R_2) * t_{io} = 110000t_{io}$
   2. Transfert de $P(T) = 2 * 10^7$ pages sur S3 : $cout = 150000t_s$
   3. Calcul de $Q$ sur S3 : $cout(Q) = (P(T) + P(R_3))t_{io} = 160000t_{io}$
   4. Transfert de $Q$ sur S1 : $cout = P(Q)t_s = 200000t_s$
   5. Finalement : $110000t_{io} + 160000t_{io} + 150000t_s + 200000t_s = 270000t_{io} + 350000t_s$

   => J'ai faux **je comprends rien à la correction**

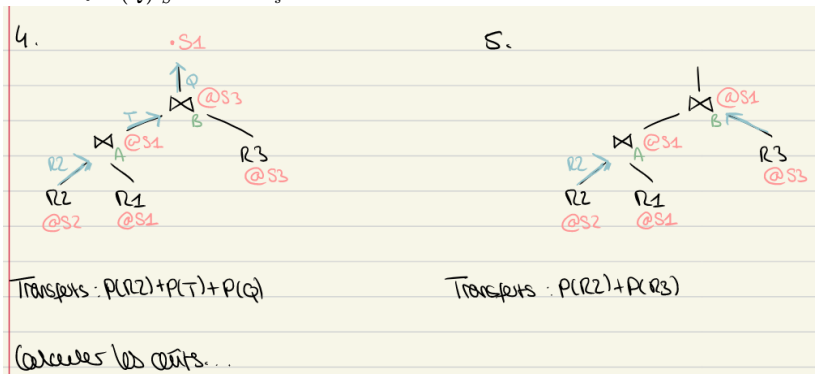
Pourquoi on doit se fix des blocs comme ça ? Pourquoi on divise par 50 pour R2 ? Pourquoi on utilise pas la formule du cout par hachage externe ? Qui ne peux mathématiquement pas donner un 3 pour le premier coût

Correction : Par hashage * On veut $Cout(T)$ par hachage mais ni R1 ni R2 ne tiennent en mémoire (mémoire de 201) * **Lire** R1 par blocs de 200 pages : on a $\frac{10000}{200} = 50$ blocs * On veut répartir les données de R1 en 50 paquets de 200 pages en utilisant une fonction de hachage $h \rightarrow P(R1)$ écritures pour répartir R1 * Lire R2 par blocs de 200 pages pour la répartir en fonction de $h$. Remarque : les paquets de T2 font $\frac{100000}{50} = 2000$ pages $\rightarrow$ P(R2) écritures * Jointure entre les paires de paquets ayant le numéro de paquet $\rightarrow P(R1) + P(R2)$ * Total : $cout(T) = 3P(R1) + 3P(R2)$

Correction : Par tri fusion * Lire et trier R1 en 50 blocs de 200 pages : $2P(R1)$ * Lire et trier R1 en $\frac{10000}{200} = 500$ blocs de 200 pages : $2P(R2)$ * On a $50 + 500 > 200$ donc il faut fusionner R2 avant de commencer la jointure oar fusion * Fusion des blocs de R2 en $\frac{500}{200} = 3$ blocs : $2P(R2)$ * Nombre de blocs : * R1 : 50 blocs * R2 : 3 blocs * $\rightarrow 52 < 200$ donc on peut calculer la jointure * Jointure par fusion : $P(1) + P(2)$ * Total : $Cout(T) = 3P(R1) + 5P(R2)$

- Transfert de T sur S3 : $P(T) * t_S = 150000t - S$
- $Cout(Q)$ ? Par hachage externe
  - Lire R3 et répartir en $\frac{10000}{200} = 50$ blocs : $2P(R3)$
  - Répartir T (provenant du site S2) en 50 blocs : $P(T)$
  - Jointure : $(P(R3) + P(T))t_{IO}$
  - Total : $(3P(R3) + P(T))t_{IO}$

- Transfert Q : $P(Q)t_S = 200000t_s$




## Exercice 1 page 12 : Requête réparties avec JDBC

1. 
   - `SELECT Etu.nom, Stage.lieu FROM Etu, Fait, Stage WHERE Etu.nE = Fait.nE AND Fait.nS = Stage.nS AND Fait.note > 10`
   - Sur S1 : `SELECT Etu.nom FROM Etu, Fait WHERE Etu.nE = Fait.nE AND Fait.note > 10`
   - Sur S2 : `SELECT lieu FROM Stage`

   ```
   s1 = c1.createStatement();
   res1 = s1.executeQuery("SELECT Etu.nom FROM Etu, Fait WHERE Etu.nE = Fait.nE

   p2 = c2.createStatement();
   res2 = p2.executeQuery("SELECT lieu FROM Stage s where s.nS=?")

   while (res1.next()){
      p2.setInt(1, res1.getInt("nS")); // on remplace le "?" par le Fait.nS
      res2 = p2.executeQuery();
      res2.next() // Faire avancer le curseur d'un tuple ; Résultat unique
      System.out.print(res1.getString("Etu.nom") + res2.getString("lieu"))
   }
   ```

2. 
   ```
   SELECT s.lieu, COUNT(*) AS nbEtu FROM Etu e, Stage s
   WHERE e.résidence = s.lieu
      AND s.durée = 6
   GROUP BY s.lieu
   HAVING COUNT(*) > 10
   ORDER BY lieu ASC
   ```

   - Sur S1 :
     `SELECT e.résidence AS nbEtu FROM Etu e`
   - Sur S2 :

     ```
     SELECT s.lieu FROM Stage s
     WHERE s.lieu = ?
        AND s.durée = 6
     ```

   ```
   s1 = c1.createStatement();
   res1 = s1.executeQuery("SELECT résidence AS nbEtu FROM Etu");

   p2 = c2.createStatement();
   res2 = p2.executeQuery("SELECT s.lieu AS nbEtu FROM Stage s WHERE s.lieu =

   while (res1.next()){
      p2.setString(1, res1.getString("résidence"));
      res2 = p2.executeQuery();
      res2.next() // Résultat non unique
      if (res2.len() > 10) {
         System.out.print(res1.getString("résidence") + res2.len())
      }
   }
   ```

=> Pratiquement une bonne solution, le `len()` est fait directement dans la requete SQL

**Correction** :

```
SELECT s.lieu, COUNT(*) AS nbEtu FROM Etu e, Stage s
WHERE e.résidence = s.lieu
   AND s.durée = 6
GROUP BY s.lieu
HAVING COUNT(*) > 10
ORDER BY lieu ASC
```

- Sur S1 :
  `SELECT count(*) AS nbEtu FROM Etu WHERE residence = ?`
- Sur S2 :
  `SELECT * FROM Stage s WHERE s.durée = 6ORDER BY lieu ASC`

```
s1 = c1.createStatement();
res1 = s1.executeQuery("SELECT count(*) AS nbEtu FROM Etu WHERE residence =

p2 = c2.createStatement();
res2 = p2.executeQuery("SELECT * FROM Stage s WHERE s.durée = 6 ORDER BY li

while (res1.next()){
   r1.setString(1, res2.getString("lieu"));
   res1 = R1.executeQuery();
   res1.next(); // Résultat non unique
   if (res1.getInt("nbEtu") > 10) {
      System.out.print(res2.getString("lieu") + " " + res1.getInt("nbEtu"));
   }
}
```

Autre solution :

- `r1 = "SELECT * FROM (SELECT count(*) AS nbEtu FROM Etu WHERE residence = ?) WHERE nbEtu > 10" pui test sur le if(){}`

- Si on commence par S1

```
SELECT e.residence, count(*) AS nbEtu
FROM Etu
GROUP BY residence
HAVING count(*) > 10
ORDER BY residence ASC

SELECT 1 FROM Stage WHERE Lieu = ? AND durée =6
```

3. 
```
SELECT s.durée, AVG(f.note) FROM Stage s, Fait f
WHERE s.nS = f.nS
   AND s.lieu = "Paris"
GROUP BY s.durée
```

- Sur S1 : `SELECT SUM(note) as n, COUNT(*) AS c FROM Fait WHERE nS = ?`
- Sur S2 : `SELECT DISTINCT durée FROM Stage WHERE s.lieu = "Paris"`
  et
  `SELECT nS FROM Stage WHERE durée = ?`
- Obtenir les durée distinct pour Paris
- Pour chaque durée trouver les nS associé
- Pour chaque nS obtenir toute ces notes leurs compte
- Dans l'application fait la moyenne pour chaque durée

```
for r1 in R1 :
   Ts = 0
   T_c = 0
   for r2 in R2(R1.durée)
      s,c = R3(r2.nS)
      Ts += s
      Tc += c
   print(r1.durée, Ts/Tc)
```

4. 
```
SELECT DISTINCT nE FROM Visite v, Fait f, Stage s
WHERE v.nE = f.nE
   AND f.nE = s.nE
   AND v.ville = "Aix"
   AND s.durée >= 3
```

- Sur S1 :

```
SELECT nS FROM Fait
WHERE nE = ?
```

- Sur S2 :

```
SELECT "oui" FROM Stage
WHERE durée = 3 // Toujours le cas
AND nS = ?
```

- Sur S3 : `SELECT nE FROM Visite WHERE ville = "Aix"`
- Pour chaque nE ayant visité Aix
  - Récuperer sa liste de nS à partir de son nE avec la query sur S1
  - Pour chaque nS :
    - Vérifier si c'est un stage de 3 mois à partir de son nS avec la query sur S2 (if == 'oui')
    - Si oui print son nE et break (évite les doublons)

# TD8

## Exercice 2 page 12 : Requêtes réparties avec JDBC

1. 
- Sur S1 : `SELECT numA, a1 FROM TA WHERE a2 < 10`
- Sur S2 : `SELECT 1 FROM TB WHERE b3 = 9 AND numA = ?`

2. 
- Sur S1 : (=r1) `SELECT a2 FROM TA`
- Sur S2 : (=r2) `SELECT numB, b3, b4 FROM TB WHERE b4 <> ?`
- Sur S3 : (=r3) `SELECT c5 FROM TB WHERE numB = ? AND ? < c5`

```
for a2 in r1:
   for numB, b3, b4 in r2(a2):
      for c5 in r3(numB, b3):
         print(b4, c5)
```

=> Je sais pas si c'est bon perso ça me semble correct, la **correction** inverse les deux première boucle en gros

- Sur S1 : `SELECT "ok" FROM TA WHERE a2 = ?`
- Sur S2 : `SELECT numB, b3, b4 FROM TB`
- Sur S3 : `SELECT c5 FROM TB WHERE numB = ? AND ? < c5`

```
for r2 in R2:
   if R1(r2.b4) != 'ok':
      for r2 in R3(r2.numB, r2.b3):
         print(r2.b4, r3.c5)
```

3. 
- Méthode de traitement par semi-jointure : ?
- **Ce qu'il ne faut pas faire**
  - R1 : `SELECT num1, a1 FROM TA where a2 = 0`
  - R2 : `SELECT b3 FROM TB WHERE b4 = 0 AND numA = ?`

```
for r1 in R1:
   for r2 in R2(r1.numA):
      print(r1.numA, r1.a1, r2.b3)
```

  - **Inconvénient** : L'application lit tous les tuples de r1, y compris ceux qui ne sont pas dans le résultat. Il y a lecture de a1
- Correction : On vas aller chercher a1 uniquement si nécéssaire
  - R1 : `SELECT num1 FROM TA where a2 = 0`
  - R2 : `SELECT b3 FROM TB WHERE b4 = 0 AND numA = ?`
  - R3 : `SELECT a1 FROM TA where numA = ?`

```
    for num1 in R1:
        for r2 in R2(numA):
            print(r1.numA, R3(numA), r2.b3)
```

4. 
   - Tri fusion : ?
   - R2 : `SELECT b4, numB FROM TB ORDER BY b4`
   - R3 : `SELECT c6, numB FROM TC ORDER BY c6`
   - Algo

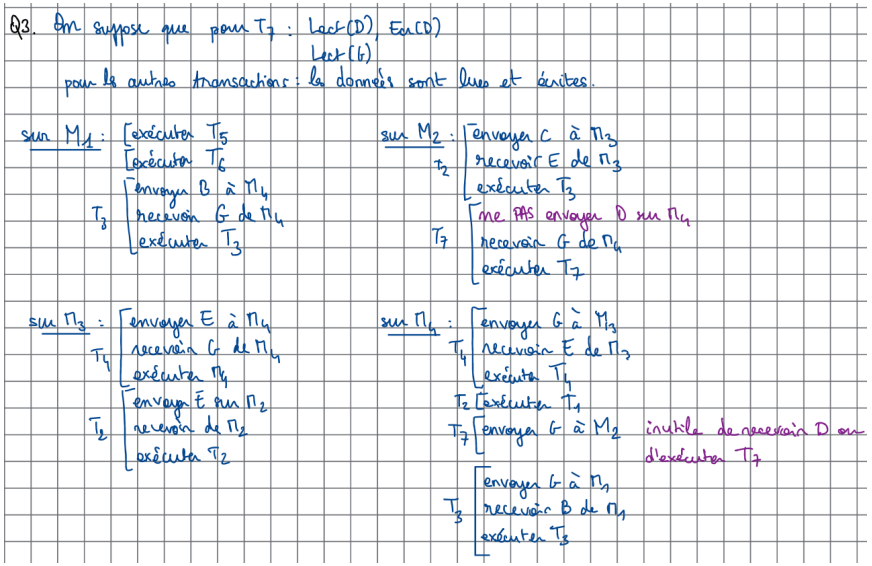
```
for b4, numB in R2:
    while (b4 > c6):
        r2.next()
    while (b4 == c6):
        print()
        r2.next()
```

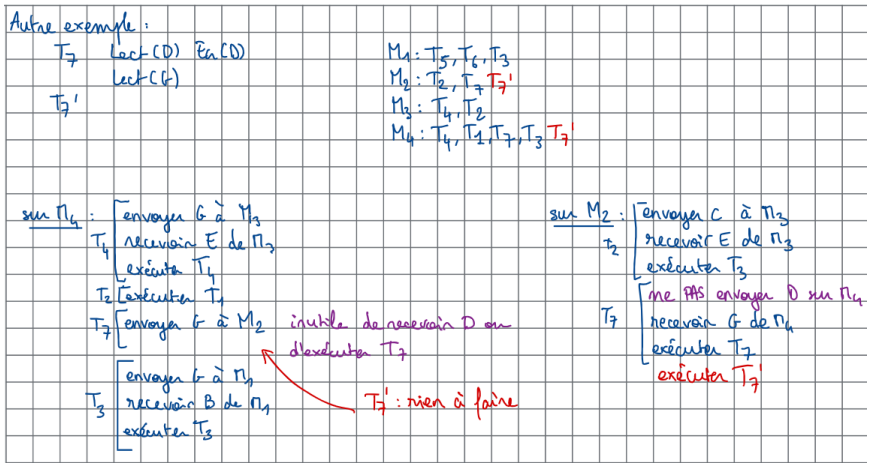
# TD9

## Exercice 1 page 18

1. 
   - Indépendant de la machine qui reçoit la transaction
   - Transaction locale : $T_1, T_5, T_6$ (Transaction pouvant être traité en une machine)
   - Transaction globale : $T_2, T_3, T_4, T_7$
2. 
   1. 
      - On considère les transactions reçues sur M1 puis sur M2, M3, M4
        - Sur M1 :
          - T4 sera traité par M3 et M4
          - T5 sera traité par M1
        - Sur M2 :
          - T2 sera traité par M2 et M3
          - T6 sera traité par M1
        - Sur M3 :
          - T1 sera traité par M4
          - T7 sera traité par M2 et M4
        - Sur M4 :
          - T3 sera traité par M1 et M4
      - Finalement on réécrit toutes les transaction sur toutes les machine (Ti, Mj) = (Ti venant de Mj)
        - M1 : (T5, M1), (T6, M2), (T3, M4)
        - M2 : (T2, M2), (T7, M3)
        - M3 : (T4, M1), (T2, M2)
        - M4 : (T4, M1), (T1, M3), (T7, M3), (T3, M4)
      - Je pense que c'est ordonée comme demandé dans la consigne, d'abord par rapport à la machine d'origine, si égalité par rapport au numéro de la requete
   2. Pas d'impact dance qu'on a fait avant. Dans M4 on lit juste G. pour T7 sur M4 : lire G et l'envoyer à M2 mais inutile de traiter (lire le code de la transaction) T7 sur M4. (**pas compris**)
3. 

**Correction** : PAS COMPRIS




## Exercice 10 page 10

Les joueurs sont répartis sur 15 machines en "round"robin" (a tour de rôle)

1. 
   - 

- Sur Mi : `SELECT * FROM Ji WHERE salaire = (SELECT max(salaire) FROM Ji)` envoyer le résultat sur M0
- Sur M0 : Union des résultats reçus des Mi et selectionner le joueur avec le max(salaire)