Chapitre 8

Optimisation de Requêtes

Exercice A : Soit la base STATION DE SKI contenant les relations suivantes :

- `hotel (noms,nomh,categorie,adresse,tel,nb_chambres)` (**5** catégories différentes,**1000** pages, chaque page contient **10** tuples).
- `station (noms,gare)` (**100** pages, chaque page contient **10** tuples).
- `activite (type_activite,noms)` (**100** pages, chaque page contient **10** tuples).

On suppose de plus l'existence des index suivants (arbres B+) :

- Index sur le couple `(noms,nomh)` de la relation `hotel` (**26 pages**).
- Index sur l'attribut `categorie` de `hotel` (**33 pages**).

Hypothèses :

- Chaque catégorie comprend un nombre égal d'hôtels.
- Chaque station comprend un nombre égal d'hôtels.
- Le nombre d'E/S lors du parcours de l'arborescence des index est *négligeable*.

Requête 1 : adresse, numéro de téléphone et nombre de chambres des hôtels de catégorie 3' dans la station de nom 'pesey' .

```
SELECT adresse, tel, nb_chambres
FROM   hotel
WHERE  noms='pesey' AND categorie=3;
```

1. Calculer le nombre d'E/S effectuées lors de l'évaluation de cette requête de sélection par un parcours séquentiel de la relation `hotel`.
2. Calculer le nombre d'E/S effectuées lors de l'évaluation de cette requête en utilisant l'index sur `categorie`.
3. Calculer le nombre d'E/S effectuées lors de l'évaluation de cette requête en utilisant l'index sur `noms,nomh`.
4. Conclure

Solution :

(1) Balayage séquentiel *On évalue séquentiellement la condition sur tous les tuples de la relation, ceux-ci étant chargés page par page. Le nombre d'E/S est donc le nombre de pages nécessaires pour stocker la relation* `hotel` *soit* 1000 *:*

$$N_{E/S} = 1000.$$

(2) Index sur `categorie`

Le coût de l'évaluation du prédicat dans ce cas est donné par la formule :

$$\frac{1}{nb\ de\ catégories}(NPI_{cat}(hotel) + NPT(hotel))$$

– NPI est le nombre des pages feuilles de l'index `categorie` *de la relation* `hotel`*.*
– NPT(R) est le nombre de tuples de la relation `hotel`*.*
En effet, l'index `categorie` *contenant* $NPI(hotel)$ *pages, il faut lire* $\frac{NPI_{cat}(hotel)}{nb\ de\ catégories}$ *pages de l'index pour récupérer les identifiants des pages contenant les tuples de* `hotel` *ayant comme catégorie* 3*. Ce nombre d'identifiants est* $\frac{NPT(hotel)}{nb\ de\ catégories}$ *(1 page pour chacun des* $\frac{NPT(hotel)}{nb\ de\ catégories}$ *hôtels de catégorie* 3*).*
Application numérique :

$$N_{E/S} = \frac{1}{5} \times 33 + \frac{1}{5} \times 10000 \simeq 2007$$

(3) Index sur `noms, nomh` : *Il serait également possible d'utiliser l'index sur* `noms, nomh` *de* `hotels`*. La stratégie consiste alors à sélectionner dans l'index les feuilles ayant pour* `noms` *la valeur* `pesey` *(Un index sur* `noms, nomh` *est aussi un index sur* `noms`*). Ainsi,* $\frac{NPI_{noms,nomh}(hotel)}{nb\ de\ stations\ différentes}$ *pages seront sélectionnées. Dans ces pages de l'index se trouveront des liens vers* $\frac{NPT(hotel)}{nb\ de\ stations\ différentes}$ *pages contenant chacune un article répondant à la sélection sur l'attribut* `noms`*. Ces articles sont ensuite chargés et filtrés suivant la deuxième condition sur* `categorie`*. Le nombre d'E/S est donc au total :*

$$\frac{1}{nb\ de\ stations\ différentes}(NPI_{noms,nomh}(hotel) + NPT(hotel))$$

Le nombre de stations différentes est 1000*,* `noms` *étant clé primaire de la relation* `station` *qui contient* 1000 *tuples :*
Application numérique :

$$N_{E/S} = \frac{1}{1000}(26 + 10000) \simeq 10$$

(4) Conclusion : *L'utilisation systématique d'index lors de l'évaluation de requête de sélection n'apporte pas forcément un gain en terme de nombre d'E/S. Il faut prendre en compte la sélectivité du prédicat. Si celle-ci est faible (beaucoup de tuples partagent la même valeur pour l'attribut intervenant dans le prédicat), par exemple 1/5, l'utilisation de l'index peut engendrer une perte de performance. Cest le cas lors de l'utilisation de l'index sur* `categorie`*. En revanche, lorsque la sélectivité est forte (peu de tuples ont même valeur pour l'attribut en question), l'utilisation de l'index peut-être utile. Des informations sur les données présentes dans la base, telle que le nombre de valeurs différentes pour un attribut donné, apportent une information dont un optimiseur peut tirer profit pour améliorer les performances du SGBD.*

Requête 2 : Soit le schéma suivant :

```
CREATE TABLE FILM (                  CREATE TABLE VU (
       TITRE   VARCHAR2(32),                SPECTATEUR      VARCHAR2(32),
       REALISATEUR VARCHAR2(32),            TITRE           VARCHAR2(32)
       ACTEUR VARCHAR2(32)             );
);
```

Soit la requête SQL :

```
SELECT ACTEUR, REALISATEUR
  FROM FILM, VU
 WHERE FILM.TITRE=VU.TITRE
```

Dans chacun des cas suivants, donner l'algorithme de jointure de ORACLE (par EXPLAIN, un arbre d'exécution commenté, une explication textuelle ou tout autre moyen de votre choix) :

1. Il n'existe pas d'index sur TITRE ni dans FILM ni dans VU,

   Solution :
   Tri-fusion (voir cours) : Figure 8.1 et plan d'exécution Oracle.

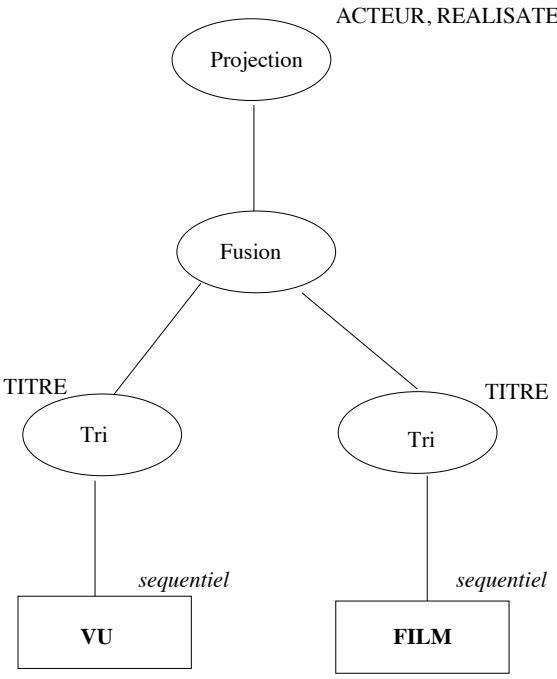

FIGURE 8.1 – Plan d'exécution

```
Plan d'exécution
----------------------------------------------------------------------
0 SELECT STATEMENT
  1 MERGE JOIN
    2 SORT JOIN
      3 TABLE ACCESS FULL VU
    4 SORT JOIN
      5 TABLE ACCESS FULL FILM
```

2. Il existe un index sur TITRE dans FILM seulement.

   Solution :
   La table VU est prise comme table directrice : pour chaque titre dans VU parcouru séquentiellement, on cherche à travers l'index le ROWID s'il existe d'un nuplet de FILM et on fait la jointure (comme la relation FILM n'est pas normalisée, il y a plusieurs nuplets de même titre) (Figure 8.2).

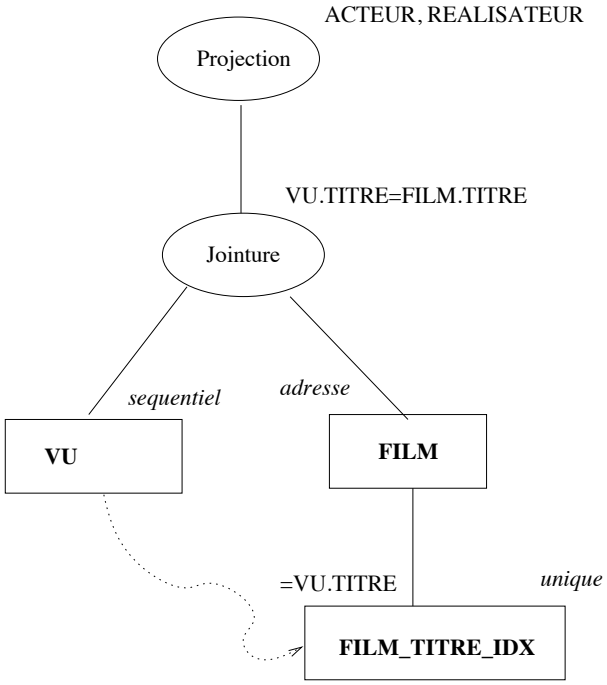

FIGURE 8.2 – Plan d'exécution

```
CREATE INDEX FILM_TITRE_idx on FILM (TITRE);

Plan d'exécution
------------------------------------------------------------------------
0 SELECT STATEMENT
  1 NESTED LOOPS
    2 TABLE ACCESS FULL VU
    3 TABLE ACCESS BY ROWID FILM
      4 INDEX RANGE SCAN FILM_TITRE_IDX
```

3. Il existe un index sur TITRE dans les deux relations.

   Solution :
   idem (l'optimiseur choisit la dernière table comme table directrice de la clause FROM quand il existe un index sur la colonne de jointure dans les deux tables)

```
CREATE INDEX FILM_TITRE_idx on FILM (TITRE);
CREATE INDEX VU_TITRE_idx on VU (TITRE);

Plan d'exécution
------------------------------------------------------------------------
0 SELECT STATEMENT
  1 NESTED LOOPS
    2 TABLE ACCESS FULL VU
    3 TABLE ACCESS BY ROWID FILM
      4 INDEX RANGE SCAN FILM_TITRE_IDX
```

Requête 3 : Soit la requête :

```
SELECT  e.enom, d.dnom
```

```
FROM    emp e, dept d
WHERE   e.dno = d.dno
AND     e.sal = 10000
```

sur la relation **EMP** de schéma *(EMPNO, SAL, MGR, DNO)*. Cette requête affiche le nom des employés dont le salaire (SAL) est égal à 10000, et celui de leur département. Indiquez le plan d'exécution dans chacune des hypothèses suivantes.

1. Index sur $DEPT(Dno)$ et sur $EMP(Sal)$

   Solution :
   Figure 8.3.

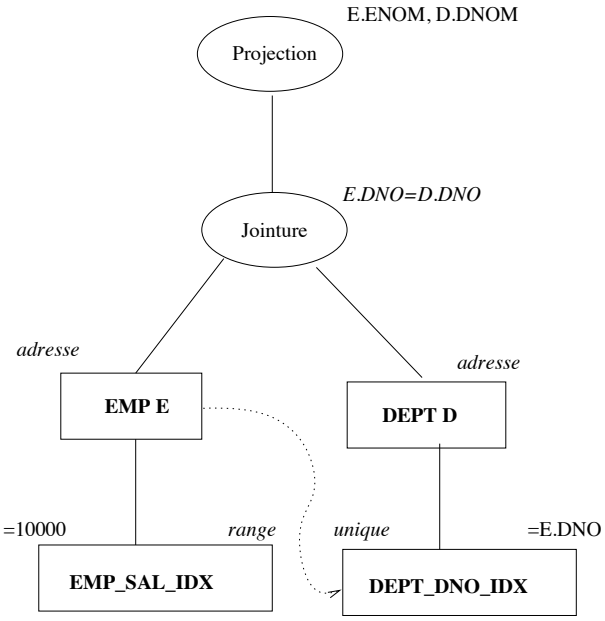

FIGURE 8.3 – Plan d'exécution

```
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
```

Boucles imbriquées (`NESTED LOOP`*) : on choisit de parcourir un sous-ensemble de EMP en utilisant l'index, puis on accède a DEPT avec l'index sur DEPTNO.*

2. Index sur $EMP(Sal)$ seulement.

   Solution :
   Figure 8.4.

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

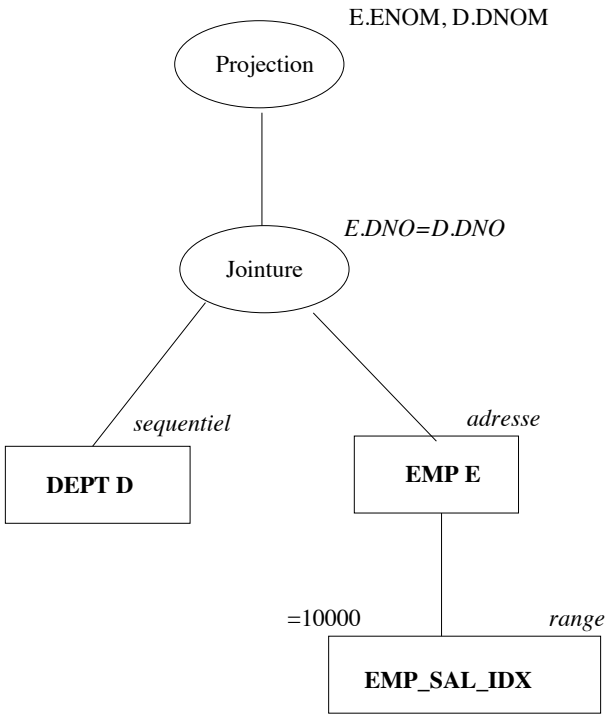

FIGURE 8.4 – Plan d'exécution

Algorithme de SCAN-INDEX. Equivalent à une jointure par NESTED-LOOP brutal. On pourrait (i) changer l'ordre des tables sans modifier la complexité, et (ii) faire un tri-fusion.

3. Index sur $EMP(Dno)$ et sur $EMP(Sal)$

   Solution :
   Figure 8.5.

```
Plan d'execution
----------------------------------------------------------------------------
0 SELECT STATEMENT
  1 NESTED LOOPS
    2 TABLE ACCESS FULL DEPT
    3 TABLE ACCESS BY ROWID EMP
      4 AND-EQUAL
        5 INDEX RANGE SCAN EMP_DNO
        6 INDEX RANGE SCAN EMP_SAL
```

   Comme l'index sur EMP(DNO) n'est pas unique, on a intérêt à limiter la liste des adresses de tuples en utilisant les deux index et en faisant l'intersection.

4. Voici une autre requête, légèrement différente. Plan d'exécution s'il n'y a pas d'index.

```
SELECT e.enom
FROM emp e, dept d
WHERE  e.dno = d.dno
AND    d.ville = 'Paris'
```

   Solution :

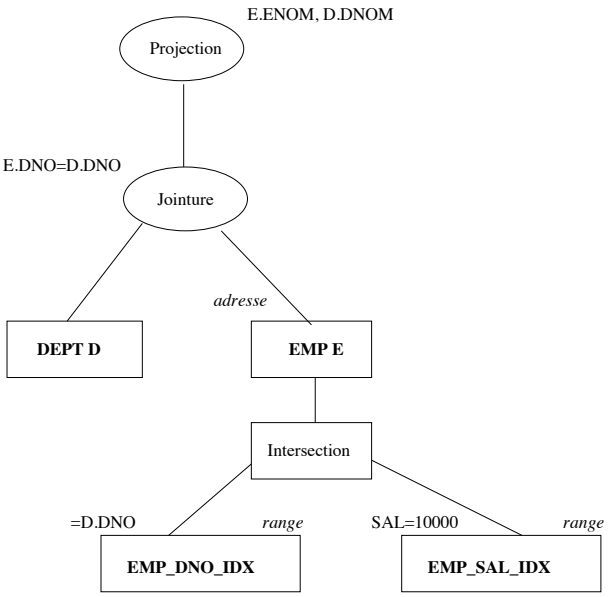

FIGURE 8.5 – Plan d'exécution

```
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
```

Algorithme de tri-fusion classique.

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

```
SELECT e.enom
FROM emp e
WHERE  e.dno IN (SELECT d.dno
                 FROM Dept d
                 WHERE d.Ville = '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 ?

Solution :

Création d'une table temporaire (`VIEW`*) contenant les numéros des départements à Paris. On a éliminé les doublons (*`SORT UNIQUE`*). Ensuite on fait un tri-fusion. Donc exécution différente pour une requête équivalente.*

Requête 4 : Sur le même schéma, voici maintenant la requête suivante.

```
SELECT *
FROM EMP X WHERE X.SAL IN (SELECT SAL
                             FROM EMP
                            WHERE EMP.EMPNO=X.MGR)
```

Cette requête cherche les employés dont le salaire (SAL) est égal à celui de leur patron (MGR). 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
```

Solution :

Figure 8.6. Boucles imbriquées (FILTER) : On parcourt la table EMP (ligne 2);

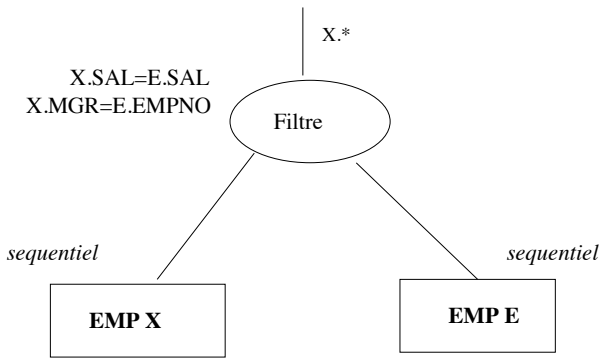

FIGURE 8.6 – Plan d'exécution

pour chaque employé, on regarde le salaire SAL et le numéro du patron MGR; on parcourt alors la table EMP (ligne 3) pour trouver l'employé dont le numéro est MGR et on compare le salaire à SAL. Donc c'est une boucle imbriquée brutale : on aurait pu faire un tri-fusion.

2. Index empno et index 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
```

Solution :

Figure 8.7. Boucles imbriquées (FILTER) : on parcourt la table EMP (ligne 2). Pour

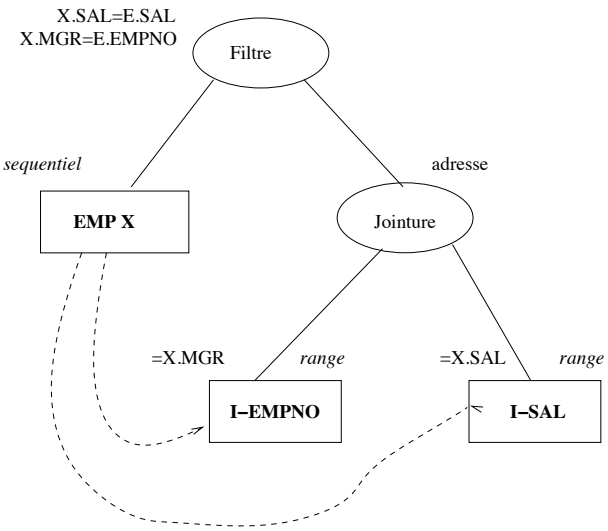

FIGURE 8.7 – Plan d'exécution

chaque employé, le salaire SAL et le numéro $EMPNO$, valeur de l'attribut MGR servent de clé pour accéder à l'index (lignes 4 et 5). L'intersection des listes de row-id (ligne 3) obtenues par les étapes 4 et 5 donne si elle est non vide un row-id de patron ayant même salaire que l'employé

3. Dans le cas où il y a les deux index (salaire et numéro d'employé) et où la requête est :

```
SELECT *
  FROM EMP X
 WHERE X.SAL = (SELECT SAL
                  FROM EMP
                 WHERE EMP.EMPNO=X.MGR)
```

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.

Solution :

Dans ce cas, seul l'index sur les numéros d'employés est utilisé. Boucles imbriquées (FILTER) ; on parcourt la table EMP : pour chaque employé, l'attribut MGR donne le numéro d'employé de son patron. On accède à son patron par l'index sur les numéros d'employé (lignes 4 puis 3) : on vérifie alors si son salaire est égal à celui de l'employé.

Requête 5 : On reprend le schéma CINEMA donné dans le cours, **mais on ne sais plus quels index existent**.

Questions :

1. Donner l'ordre SQL pour la requête : *Quels sont les films d'Hitchcock visibles après 20h00 ?*

   Solution :

```
SELECT F.TITRE
  FROM SEANCE S, FILM F, ARTISTE A
 WHERE A.NOM = 'Hitchcock'
   AND F.ID-REALISATEUR = A.ID-REALISATEUR
   AND S.ID-FILM = F.ID-FILM
   AND S.HEURE_DEBUT > '20:00'
```

2. Donner l'expression algébrique correspondante et proposez un arbre de requête qui vous paraît optimal.

   Solution :

   $\sigma_{HEURE\_DEBUT>'20:00'} SEANCE \bowtie (\sigma_{NOM='Hitchcock'} ARTISTE \bowtie_{ID-Artiste=ID-Realisateur} FILM)$

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 ?

   Solution :

   L'existence d'un tri-fusion indique qu'il manque un index. Ici on peut résoudre le pb en créant un index sur $SEANCE(id - Film)$.

Requête 6 : Soit le schéma suivant :

```
CREATE TABLE Artiste (
       ID-artiste     NUMBER(4),
       Nom            VARCHAR2(32),
       Adresse        VARCHAR2(32)
);

CREATE TABLE Film (
       ID-film        NUMBER(4),
       Titre          VARCHAR2(32),
       Année          NUMBER(4),
       ID-réalisateur NUMBER(4)
);

CREATE TABLE Joue (
       ID-artiste     NUMBER(4),
       ID-film        NUMBER(4)
);
```

Questions :

1. Donner l'ordre SQL pour la requête : *Afficher le nom des acteurs et le titre des films où ils ont joué.*
2. Donner l'expression algébrique correspondante.
3. Quel est à votre avis le plan d'exécution dans s'il n'existe que deux index, un sur FILM(ID-réalisateur), et un sur ARTISTE(ID-artiste)ÿ ?
4. Idem, avec un index sur $FILM(ID - Film)$, et un sur $JOUE(ID - Artiste)$.
5. Idem, avec un index sur $FILM(ID - Film)$, et un sur $JOUE(ID - Film)$.

Solution :

1.
```
SELECT nom, titre
FROM   artiste, film, joue
WHERE  artiste.ID-artiste = joue.ID-artiste
AND    film.ID-film = joue.ID-film
```

2. $Artiste \bowtie (Film \bowtie Joue))$

3.
```
0 SELECT STATEMENT
  1 MERGE JOIN
    2 SORT JOIN
      3 NESTED LOOPS
        4 TABLE ACCESS FULL JOUE
        5 TABLE ACCESS BY ROWID ARTISTE
          6 INDEX UNIQUE SCAN ARTISTE_IDX
    7 SORT JOIN
      8 TABLE ACCESS FULL FILM
```

Comme il n'y a qu'un seul index utilisable, on fait un parcours séquentiel sur Joue pour utiliser l'index le plus tôt possible, puis on fait un tri fusion. On peut aussi faire le parcours séquentiel initial sur Film. Petit piège : l'index sur ID_réalisateur n'est pas utilisable pour la jointure.

4.
```
0 SELECT STATEMENT
  1 NESTED LOOPS
    2 NESTED LOOPS
      3 TABLE ACCESS FULL ARTISTE
      4 TABLE ACCESS BY ROWID JOUE
        5 INDEX RANGE SCAN JOUE_ARTISTE
    6 TABLE ACCESS BY ROWID FILM
      7 INDEX UNIQUE SCAN FILM_IDX
```

Comme il y a une seule table (ARTISTE) sans index propre à la jointure, on la choisit pour effectuer le parcours séquentiel initial.

5.
```
0 SELECT STATEMENT
  1 MERGE JOIN
    2 SORT JOIN
      3 NESTED LOOPS
        4 TABLE ACCESS FULL JOUE
        5 TABLE ACCESS BY ROWID FILM
          6 INDEX UNIQUE SCAN FILM_IDX
    7 SORT JOIN
      8 TABLE ACCESS FULL ARTISTE
```

Les index existant ne peuvent servir qu'à une seule jointure : celle entre JOUE et FILM. Donc il ne reste que la solution de faire un TRI-FUSION pour la jointure avec ARTISTE. NB : on parcours JOUE puis on accède à FILM par l'index. On pourrait faire le contraire (parcourir FILM et accéder à JOUE). Quand il a des statistiques, le SGBD choisit la plus petite des tables pour le parcours séquentiel.