

# TD6 - BDD

22-04-2021

## Partie 1 - Plans d'exécution de requêtes

### Préparation du TP

Vous devez la créer au préalable en exécutant le script `utlxplan.sql`.

```
--output: Table PLAN_TABLE créé(e).
```

Lancez le script `creerTablesPlanExec.sql` pour créer les tables dans votre schéma.

```
--output: Toutes les tables ont été créé et toutes les lignes insérées.
```

On peut aussi voir cela en affichant l'ensemble des tables, ou en comptant le nombre de ligne des tables.

### Examine un plan d'exécution de requête

Pour examiner un plan d'exécution de requête, vous allez procéder de la manière suivante :

#### R1

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R1'
FOR
SELECT count(*) FROM salaries ;
COMMIT ;
```

```
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R1'
ORDER BY id;
```

Voir fig1

- ::: pour connaître l'option d'optimisation

```
SELECT optimizer FROM plan_table
WHERE statement_id = 'R1';
```

Voir fig2

#### R2

```
CREATE INDEX idxIdSalarieSalarie ON salaries (idSalarie);
--output: Index IDXIDSALARIESALARIE créé(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R2'
FOR
SELECT count(*) FROM salaries ;
```

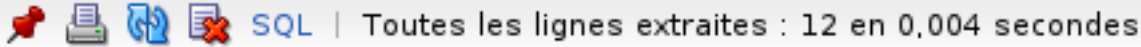
					
	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	SELECT STATEMENT	(null)	0	(null)	(null)
4	SELECT STATEMENT	(null)	0	(null)	(null)
5	SORT	AGGREGATE	1	0	(null)
6	SORT	AGGREGATE	1	0	(null)
7	SORT	AGGREGATE	1	0	(null)
8	SORT	AGGREGATE	1	0	(null)
9	TABLE ACCESS	FULL	2	1	SALARIES

Figure 1: fig1

	OPTIMIZER
1	ALL_ROWS
2	(null)
3	(null)
4	ALL_ROWS
5	(null)
6	(null)
7	ALL_ROWS
8	(null)
9	(null)

Figure 2: fig2

```
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R2'
ORDER BY id;
```

Voir fig3

	⇕ OPERATION	⇕ OPTIONS	⇕ ID	⇕ PARENT_ID	⇕ OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	SORT	AGGREGATE	1	0	(null)
4	SORT	AGGREGATE	1	0	(null)
5	TABLE ACCESS	FULL	2	1	SALARIES
6	TABLE ACCESS	FULL	2	1	SALARIES

Figure 3: fig3

- ::: pour connaître l'option d'optimisation

```
SELECT optimizer FROM plan_table
WHERE statement_id = 'R2';
```

Voir fig4

	⇕ OPTIMIZER
1	ALL_ROWS
2	(null)
3	(null)
4	ALL_ROWS
5	(null)
6	(null)

Figure 4: fig4

Supprimez l'index sur l'attribut idSalarie de la table salaries :

```
DROP INDEX idxIdSalarieSalarie;
--output: Index IDXIDSALARIESALARIE supprimé(e).
```

Ajoutez la contrainte de clé primaire pkSalarieIdSalarie sur l'attribut idSalarie de la table salaries (ie on indexe cet attribut avec un arbre B+) :

```
ALTER TABLE salaries ADD CONSTRAINT pkSalarieIdSalarie PRIMARY
KEY (idSalarie);
--output: Tables SALARIES modifié(e).
```

### R3

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R3'
FOR
SELECT count(*) FROM salaries ;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R3'
ORDER BY id;
```

Voir fig5

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SORT	AGGREGATE	1	0	(null)
3	INDEX	FAST FULL SCAN	2	1	PKSALARIEIDSALARIE

Figure 5: fig5

Passez l'optimiseur en mode RULE :

```
ALTER SESSION SET OPTIMIZER_MODE= RULE;
--output: Session modifié(e).
```

#### R4

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R4'
FOR
SELECT count(*) FROM salaries ;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R4'
ORDER BY id;
```

Voir fig5

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SORT	AGGREGATE	1	0	(null)
3	TABLE ACCESS	FULL	2	1	SALARIES

Figure 6: fig5

Remettez l'optimiseur en mode ALL\_ROWS :

```
ALTER SESSION SET OPTIMIZER_MODE= ALL_ROWS;
--output: Session modifié(e).
```

## R5

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R5'
FOR
SELECT * FROM salaries ;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R5'
ORDER BY id;
```

Voir fig6

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SORT	AGGREGATE	1	0	(null)
3	INDEX	FAST FULL SCAN	2	1	PKSALARIEIDSALARIE

Figure 7: fig6

## R6

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R6'
FOR
SELECT * FROM salaries WHERE iDSalarie = 5000;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R6'
ORDER BY id;
```

Voir fig7

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SORT	AGGREGATE	1	0	(null)
3	INDEX	FAST FULL SCAN	2	1	PKSALARIEIDSALARIE

Figure 8: fig7

## R7

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R7'
FOR
SELECT * FROM salaries WHERE ename = 'SCOTT';
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R7'
ORDER BY id;
```

Voir fig8

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	SORT	AGGREGATE	1	0	(null)
4	TABLE ACCESS	FULL	1	0	SALARIES
5	INDEX	FAST FULL SCAN	2	1	PKSALARIEIDSALARIE

Figure 9: fig8

Créez un index idxEnameSalaries sur l'attribut ename de la table salaries :

```
CREATE INDEX idxEnameSalaries ON salaries (ename);
--output: Index IDXENAMESALARIES créé(e).
```

Pour chacune des options de l'optimiseur : ALL\_ROWS, FIRST\_ROWS, RULE, donnez l'algorithme correspondant au plan d'exécution de la requête :

- version ALL\_ROWS

```
ALTER SESSION SET OPTIMIZER_MODE= ALL_ROWS
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R8'
FOR
SELECT * FROM salaries WHERE ename = 'SCOTT';
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R8'
ORDER BY id;
```

Voir fig9

- version FIRST\_ROWS

```
ALTER SESSION SET OPTIMIZER_MODE= FIRST_ROWS;
--output: Session modifié(e).
```

- ::: réalisation du plan

OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
SELECT STATEMENT		0		
TABLE ACCESS	FULL	1	0	SALARIES

Figure 10: fig9

```
EXPLAIN PLAN
SET statement_id = 'R8'
FOR
SELECT * FROM salaries WHERE ename = 'SCOTT';
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R8'
ORDER BY id;
```

Voir fig10

OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
SELECT STATEMENT		0		
TABLE ACCESS	BY INDEX ROWID BATCHED	1	0	SALARIES
INDEX	RANGE SCAN	2	1	IDXENAMESALARIES

Figure 11: fig10

- version RULE

```
ALTER SESSION SET OPTIMIZER_MODE= RULE
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R8'
FOR
SELECT * FROM salaries WHERE ename = 'SCOTT';
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R8'
ORDER BY id;
```

Voir fig11

## R9

Insérez un salarié de nom 'PETIT' dans la table salaries :

"sql INSERT INTO Salaries values (100000, 'PETIT', 'CLERK', 1000, 3); -output: 1 ligne inséré.

OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
SELECT STATEMENT		0		
TABLE ACCESS	BY INDEX ROWID	1	0	SALARIES
INDEX	RANGE SCAN	2	1	IDXENAMESALARIES

Figure 12: fig11

Pour l'option de l'optimiseur ALL\_ROWS : donnez l'algorithme correspondant au plan d'exécution de la requête

```
```sql
ALTER SESSION SET OPTIMIZER_MODE= ALL_ROWS
--output: Session modifié(e).

• ::: réalisation du plan

EXPLAIN PLAN
SET statement_id = 'R9'
FOR
SELECT * FROM salaries WHERE ename = 'PETIT';
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

```
• ::: consultation du résultat

SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R9'
ORDER BY id;
```

Voir fig12

	⚡ OPERATION	⚡ OPTIONS	⚡ ID	⚡ PARENT_ID	⚡ OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	TABLE ACCESS	BY INDEX ROWID BATCHED	1	0	SALARIES
4	TABLE ACCESS	FULL	1	0	SALARIES
5	INDEX	RANGE SCAN	2	1	IDXENAMESALARIES

Figure 13: fig12

## R10

Pour chacune des options de l'optimiseur ALL\_ROWS, FIRST\_ROWS, RULE : donnez l'algorithme correspondant au plan d'exécution de la requête :

```
• version ALL_ROWS

ALTER SESSION SET OPTIMIZER_MODE= ALL_ROWS
--output: Session modifié(e).

• ::: réalisation du plan

EXPLAIN PLAN
SET statement_id = 'R10'
FOR
SELECT * FROM salaries, services
WHERE salaries.idService=services.idService;
```



```
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R10'
ORDER BY id;
```

Voir fig13

	⚡ OPERATION	⚡ OPTIONS	⚡ ID	⚡ PARENT_ID	⚡ OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	HASH JOIN	(null)	1	0	(null)
3	TABLE ACCESS	FULL	2	1	SERVICES
4	TABLE ACCESS	FULL	3	1	SALARIES

Figure 14: fig13

- version FIRST\_ROWS

```
ALTER SESSION SET OPTIMIZER_MODE= FIRST_ROWS;
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R10'
FOR
SELECT *
FROM salaries, services
WHERE salaries.idService=services.idService;
COMMIT;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R10'
ORDER BY id;
```

Voir fig14, j ai des doublons, mais le résultat est là.

	⚡ OPERATION	⚡ OPTIONS	⚡ ID	⚡ PARENT_ID	⚡ OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	HASH JOIN	(null)	1	0	(null)
4	HASH JOIN	(null)	1	0	(null)
5	TABLE ACCESS	FULL	2	1	SERVICES
6	TABLE ACCESS	FULL	2	1	SERVICES
7	TABLE ACCESS	FULL	3	1	SALARIES
8	TABLE ACCESS	FULL	3	1	SALARIES

Figure 15: fig14

- version RULE

```
ALTER SESSION SET OPTIMIZER_MODE= RULE
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R10'
FOR
SELECT *
FROM salaries, services
WHERE salaries.idService=services.idService;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R10'
ORDER BY id;
```

Voir fig15, encore une fois j ai de plus en plus de doublons, mais le résultat est là.

	⚡ OPERATION	⚡ OPTIONS	⚡ ID	⚡ PARENT_ID	⚡ OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	SELECT STATEMENT	(null)	0	(null)	(null)
4	HASH JOIN	(null)	1	0	(null)
5	MERGE JOIN	(null)	1	0	(null)
6	HASH JOIN	(null)	1	0	(null)
7	TABLE ACCESS	FULL	2	1	SERVICES
8	TABLE ACCESS	FULL	2	1	SERVICES
9	SORT	JOIN	2	1	(null)
10	TABLE ACCESS	FULL	3	1	SALARIES
11	TABLE ACCESS	FULL	3	2	SERVICES
12	TABLE ACCESS	FULL	3	1	SALARIES
13	SORT	JOIN	4	1	(null)
14	TABLE ACCESS	FULL	5	4	SALARIES

Figure 16: fig15

## R11

Modifiez la table services de manière à ce que l'attribut idService soit clé (contrainte pkServicesIdService) :

```
ALTER TABLE services ADD CONSTRAINT pkServicesIdService PRIMARY KEY (idService);
--output: Table SERVICES modifié(e).
```

Pour chacune des options de l'optimiseur ALL\_ROWS, FIRST\_ROWS, RULE : donnez l'algorithme correspondant au plan d'exécution de la requête :

- version ALL\_ROWS

```
ALTER SESSION SET OPTIMIZER_MODE= ALL_ROWS
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R11'
FOR
SELECT * FROM salaries, services
WHERE salaries.idService=services.idService;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R11'
ORDER BY id;
```

Voir fig16

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	HASH JOIN	(null)	1	0	(null)
3	TABLE ACCESS	FULL	2	1	SERVICES
4	TABLE ACCESS	FULL	3	1	SALARIES

Figure 17: fig16

- version FIRST\_ROWS

```
ALTER SESSION SET OPTIMIZER_MODE= FIRST_ROWS;
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R11'
FOR
SELECT *
FROM salaries, services
WHERE salaries.idService=services.idService;
COMMIT;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R11'
ORDER BY id;
```

Voir fig17, j ai des doublons, mais le résultat est là.

- version RULE

```
ALTER SESSION SET OPTIMIZER_MODE= RULE;
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R11'
FOR
SELECT *
FROM salaries, services
WHERE salaries.idService=services.idService;
```

	⚡ OPERATION	⚡ OPTIONS	⚡ ID	⚡ PARENT_ID	⚡ OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	NESTED LOOPS	(null)	1	0	(null)
4	HASH JOIN	(null)	1	0	(null)
5	TABLE ACCESS	FULL	2	1	SERVICES
6	NESTED LOOPS	(null)	2	1	(null)
7	TABLE ACCESS	FULL	3	1	SALARIES
8	TABLE ACCESS	FULL	3	2	SALARIES
9	INDEX	UNIQUE SCAN	4	2	PKSERVICESIDSERVICE
10	TABLE ACCESS	BY INDEX ROWID	5	1	SERVICES

Figure 18: fig17

```
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R11'
ORDER BY id;
```

Voir fig18, encore une fois j ai de plus en plus de doublons, mais le résultat est là.

## R12

Créez un index idxIdServiceSalaries sur l attribut idService de la table salaries :

```
CREATE INDEX idxIdServiceSalaries ON salaries (idService);
--output: Index IDXIDSERVICESALARIES créé(e).
```

Pour chacune des options de l optimiseur ALL\_ROWS, FIRST\_ROWS, RULE : donnez l'algorithme correspondant au plan d exécution de la requête :

- version ALL\_ROWS

```
ALTER SESSION SET OPTIMIZER_MODE= ALL_ROWS;
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R12'
FOR
SELECT * FROM salaries,services
WHERE salaries.idService=services.idService;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R12'
ORDER BY id;
```

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	SELECT STATEMENT	(null)	0	(null)	(null)
4	NESTED LOOPS	(null)	1	0	(null)
5	NESTED LOOPS	(null)	1	0	(null)
6	HASH JOIN	(null)	1	0	(null)
7	NESTED LOOPS	(null)	2	1	(null)
8	NESTED LOOPS	(null)	2	1	(null)
9	TABLE ACCESS	FULL	2	1	SERVICES
10	TABLE ACCESS	FULL	3	1	SALARIES
11	TABLE ACCESS	FULL	3	2	SALARIES
12	TABLE ACCESS	FULL	3	2	SALARIES
13	INDEX	UNIQUE SCAN	4	2	PKSERVICESIDSERVICE
14	INDEX	UNIQUE SCAN	4	2	PKSERVICESIDSERVICE
15	TABLE ACCESS	BY INDEX ROWID	5	1	SERVICES
16	TABLE ACCESS	BY INDEX ROWID	5	1	SERVICES

Figure 19: fig18

Voir fig19

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	HASH JOIN	(null)	1	0	(null)
3	TABLE ACCESS	FULL	2	1	SERVICES
4	TABLE ACCESS	FULL	3	1	SALARIES

Figure 20: fig19

- version FIRST\_ROWS

```
ALTER SESSION SET OPTIMIZER_MODE= FIRST_ROWS;
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R12'
FOR
SELECT *
FROM salaries, services
WHERE salaries.idService=services.idService;
COMMIT;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R12'
ORDER BY id;
```

Voir fig20, j'ai des doublons, mais le résultat est là.

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	NESTED LOOPS	(null)	1	0	(null)
4	HASH JOIN	(null)	1	0	(null)
5	TABLE ACCESS	FULL	2	1	SERVICES
6	NESTED LOOPS	(null)	2	1	(null)
7	TABLE ACCESS	FULL	3	1	SALARIES
8	TABLE ACCESS	FULL	3	2	SERVICES
9	INDEX	RANGE SCAN	4	2	IDXIDSERVICESALARIES
10	TABLE ACCESS	BY INDEX ROWID	5	1	SALARIES

Figure 21: fig20

- version RULE

```
ALTER SESSION SET OPTIMIZER_MODE= RULE;
--output: Session modifié(e).
```

- ::: réalisation du plan

```
EXPLAIN PLAN
SET statement_id = 'R12'
FOR
SELECT *
FROM salaries, services
WHERE salaries.idService=services.idService;
COMMIT ;
--output: Explicité. Validation (commit) terminée.
```

- ::: consultation du résultat

```
SELECT operation, options, id, parent_id, object_name
FROM plan_table
WHERE statement_id = 'R12'
ORDER BY id;
```

Voir fig21, encore une fois j'ai de plus en plus de doublons, mais le résultat est là.

## Pseudo-code correspondant aux requêtes SQL données auparavant

- R1

```
résultat = 0
Pour tous les tuple tSalarié de la table salarie, faire :
    obtenir le tuple tSalarié
    résultat += 1
FinPour
```

- R2

De la même manière que R1

	OPERATION	OPTIONS	ID	PARENT_ID	OBJECT_NAME
1	SELECT STATEMENT	(null)	0	(null)	(null)
2	SELECT STATEMENT	(null)	0	(null)	(null)
3	SELECT STATEMENT	(null)	0	(null)	(null)
4	NESTED LOOPS	(null)	1	0	(null)
5	NESTED LOOPS	(null)	1	0	(null)
6	HASH JOIN	(null)	1	0	(null)
7	NESTED LOOPS	(null)	2	1	(null)
8	NESTED LOOPS	(null)	2	1	(null)
9	TABLE ACCESS	FULL	2	1	SERVICES
10	TABLE ACCESS	FULL	3	1	SALARIES
11	TABLE ACCESS	FULL	3	2	SERVICES
12	TABLE ACCESS	FULL	3	2	SERVICES
13	INDEX	RANGE SCAN	4	2	IDXIDSERVICESALARIES
14	INDEX	RANGE SCAN	4	2	IDXIDSERVICESALARIES
15	TABLE ACCESS	BY INDEX ROWID	5	1	SALARIES
16	TABLE ACCESS	BY INDEX ROWID	5	1	SALARIES

Figure 22: fig21

- R3

```

résultat = 0
Parcourt de l index primaryKey_Salarield_Salarie, position de l index sur le premier sous-élément :
Pour chaque sous-élément de cet index faire:
    résultat += 1
FinPour

```

- R4

De la même manière que R1

- R5

```

résultat = 0
Pour chaque tuple nommé tSalarie de la table salarie faire :
    obtenir le tuple tSalarie
    résultat += le_tuple_tSalarie
FinPour

```

- R6

```

résultat = ensembe_vide (ou None)

Parcourt de l index primary_key_Salarield_Salarie et position de l index sur la feuille, tel que salarie.i

lire le tuple tSalarie
obtenir le tuple tSalarie
résultat += tuple_tSalarie

```

- R7

```

resultat = none
pour chaque tuple tSalarie de la table salarie faire :

```

```

    obtenir tuple_tSalarie
    Si tSalarie.ename = 'SCOTT', alors :
        resultat += tuple_tSalarie
    FinSi
FinPour

```

- R8

pour ALL\_ROWS : de la même manière que R7  
 pour FIRST\_ROWS, RULE, faire :

```

resultat = None

```

parcourt de l index idxEnameSalaries, position de celui-ci sur la feuille telle que salarie.ename = 'SCOTT'

Pour chaque tuple tSalarie tel que salarie.ename='SCOTT' faire :

```

    lecture tuple_tSalarie
    obtention tuple_tSalarie
    resultat += tuple_tSalarie
FinPour

```

- R9

```

resultat = none/vide

```

parcourt de l index idxEnameSalaries, position de celui-ci sur la feuille telle que salarie.ename = 'PETIT'

Pour chaque tuple tSalarie tel que salarie.ename='PETIT' faire :

```

    lecture tuple_tSalarie
    obtention tuple_tSalarie
    resultat += tuple_tSalarie
FinPour

```

Faire de la même manière pour les autres options, FIRST\_ROWS et RULE.

- R10

Pour chaque tuple tService de la table services faire :

```

    obtenir tService
    numPaquet = h (tService.idService)
    insertion tService dans le paquet numPaquet
FinPour

```

```

resultat = none

```

Pour chaque tuple tSalarie de la table salaries faire :

```

    obtenir tSalarie
    numPaquet = h (tSalarie.idService)
    rechercher dans le paquet numPaquet le tuple tService/tService.idService=tSalarie.idService
    resultat += < tSalarie, tService >
FinPour

```

```

resultat = none

```

```

obtenir tService

```

```

obtenir tSalarie

```

TantQue existe(tService) et existe(tSalarie) répéter :

```

    Si tService.idService=tSalarie.idService, Alors :
        resultat += < tSalarie, tService >
    obtenir tSalarie
    Sinon
        obtenir tService
    FinSi
FintTantQue

```

- R11



```
resultat = none
```

Pour chaque tuple tSalaries de la table salaries faire :  
obtenir tSalarie

Parcourir de l'index primary\_key\_Services\_IdService, position de celui-ci sur la feuille telle que sera

```
lire tService  
obtenir tService  
resultat += < tSalarie, tService >
```

FinPour

- R12

pour FIRST\_ROWS, RULE :

```
resultat = none
```

Pour chaque tuple tService de la table services faire :  
obtenir tService

parcourt de l'index idxIdServiceSalaries, position de celui-ci sur la feuille telle que salaire.idServ

```
Pour chaque tuple tSalarie tel que salaire.idService=tService.idService faire :  
lire tSalarie  
obtenir tSalarie  
resultat += < tSalarie, tService >  
FinPour
```

FinPour

## Partie 2 - Optimisation des ressources de mémoire : calcul de ratios

- optimisation de la mémoire - shared pool area, zone LC : PIN RATIO

```
SELECT SUM(RELOADS)/SUM(PINS) FROM v$librarycache;
```

Voir figP2-1

	SUM(RELOADS)/SUM(PINS)
1	0,000883431246963205088563982508061310128539

Figure 23: figP2-1

- optimisation de la mémoire - shared pool area , zone DC : GET RATIO

```
SELECT SUM(GETMISSES)/SUM(GETS) FROM V$ROWCACHE;
```

Voir figP2-2

	SUM(GETMISSES)/SUM(GETS)
1	0,0136426499393056078397426477249165074655

Figure 24: figP2-2

- optimisation de la mémoire - buffer de données : HIT RATIO

```
SELECT 1-(A.VALUE/(B.VALUE+C.VALUE)) FROM V$SYSSTAT A, V$SYSSTAT B, V$SYSSTAT C  
WHERE A.NAME='physical reads'  
AND B.NAME='db block gets'  
AND C.NAME='consistent gets';
```

Voir figP2-3

	$1 - (A.VALUE / (B.VALUE + C.VALUE))$
1	0,9902764586022421925103301591940446327821

Figure 25: figP2-3

- optimisation de la mémoire - buffer de reprise : KEY RATIO

```
SELECT A.VALUE/B.VALUE FROM V$SYSSTAT A, V$SYSSTAT B
WHERE A.NAME='redo log space requests'
AND B.NAME='redo entries';
```

Voir figP2-4

	A.VALUE/B.VALUE
1	0,00000866449765408726015587431279702981020418

Figure 26: figP2-4