Rapport Travails Pratiques basse de donnees

Carlos Perez

3 décembre 2015

Table des matières

1 Interprétation du plan d'exécution

Cette section parle de l,evaluation des plans d'execution pour les requetes du point 1 du TP. **??**.

1.1 1.1

INDEX UNIQUE SCAN :`SELECT annee_naissance from artiste where id = 37;;`

Nombre de pages accedes en RAM 2.

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

```
| Id  | Operation                    | Name         | ----------------------------------------
|   0 | SELECT STATEMENT             |              | |   1 |  TABLE ACCESS BY INDEX ROWID| A
|   2 |   INDEX UNIQUE SCAN          | SYS_C0098771 | ----------------------------------------
```

1.2 1.2

— **INDEX UNIQUE SCAN :** La basse de données cherche la valeur dans l'index et arrete quand la valeur a ete trouve parce que ce ne pas possible d'avoir plus d'un valeur.

— **TABLE ACCESS INDEX BY ROWID :** On utilisse un index pour acceder directement au ressources de la basse de données.

1.3 1.3

Requete : `SELECT annee_naissance from artiste where id > -1;`

Nombre de pages accedes en RAM 7.

Plan d'execution ————————————- | Id | Operation | Name | ——————————————- | 0 | SELECT STATEMENT | | | 1 | TABLE ACCESS FULL| ARTISTE | ————————————-

1.4 1.4

— **TABLE ACCESS FULL :** La basse de données lisse tous les registres de la table. Dans ce cas le probleme c'est que la requete n'est pas selective, en d'autres mots, la requete contient la majoritè des valeurs de la table. C'est pour cette raison que l'optimisseur a realis'e un scan complet de la table meme si on utilise un indice dans la requete.

1.5 1.5

Il me semble raisonnable la choix du methode parce qu'utiliser l'index ne donnerai aucune avantage dans ce cas specifique.

1.6 1.6

Requete :

```
SELECT nom,prenom from artiste inner join (SELECT id_acteur from role inner join
(SELECT film.id AS idfilm from film inner join (SELECT id from artiste where nom = 'Tarantin
realisateur on film.id_realisateur = realisateur.id) realisateur on role.ID_FILM = realisate
```

```
---------------------------------------------------| Id  | Operation
---------------------------------------------------|   0 | SELECT STATEMENT            |
|   1 |  NESTED LOOPS                |              | |   2 |   NESTED LOOPS              |
|   3 |    NESTED LOOPS              |              | |   4 |     HASH JOIN               |
```

```
|   5 |      TABLE ACCESS FULL       | ARTISTE      ||   6 |      TABLE ACCESS FULL       |
|   7 |     INDEX RANGE SCAN         | SYS_C0098788 ||   8 |    INDEX UNIQUE SCAN         |
|   9 |   TABLE ACCESS BY INDEX ROWID| ARTISTE      |-------------------------------------
```

1.7 1.7

- **INDEX RANGE SCAN** La basse de données cherche les valeurs dans un rangue possible dans l'index. L'optimiseur utilise une structure de données en arbre pour trouver
- **INDEX UNIQUE SCAN** La basse de données cherche la valeur dans l'index et arrete quand la valeur a ete trouve parce que ce ne pas possible d'avoir plus d'un valeur.
- **HASH JOIN** L'optimiseur charge en memoire le resultat d'appliquer une fonction de hash a la plus petite de deux basse de données disponibles. Apres il compare les valeurs donnees par la function contre les valeurs de la table plus grande. Le resultat et la disminution des access innecessaires au disque.
- **NESTED LOOP** Cette methode cree deux boucles un pour la table grande et l'autre pour la table petite. Le boucle interne s'execute pour chaque file de la table exterieure.
- **TABLE ACCESS INDEX BY ROWID :** On utilisse un index pour acceder directement au ressources de la basse de données.

1.8 1.8

La solution choisie est trop efficace parce qu'elle fait seulement deux access complets dans une table et utilise des indices. L'access a la table artiste etait necessaire car

2 Vues

2.1 2.1

Vue : `CREATE view artisterole as SELECT nom,prenom from artiste  inner join (Select id_acteur, count(*) AS nomfilmes from role group by id_acteur) rolefilm ON artiste.`

2.2 2.2

Requete : `SSELECT * FROM artisterole;`

3 PL/SQL

Dans cette section on montre les declencheurs generees et la table genere

3.1 3.1

Requete : `create or replace TRIGGER trigger_artistes BEFORE INSERT`
`ON artiste  FOR EACH ROW`
`DECLARE`
`BEGIN`
`  :new.annee_naissance := :new.annee_naissance*10;`
`END;`

3.2 3.2

Requete : `CREATE TABLE nbupdates ( chaine VARCHAR2(50),  nombre_updates INT`
`);`

3.3 3.3

Declencheur : `create or replace TRIGGER trigger_nbupdates AFTER INSERT`
`ON artiste  FOR EACH ROW DECLARE  vupd integer; BEGIN`
`  SELECT nombre_updates INTO vupd FROM nbupdates where chaine =’ARTISTES’;`
`  UPDATE nbupdates  SET nombre_updates = vupd+1  WHERE chaine =’ARTISTES’;`
`EXCEPTION  WHEN NO_DATA_FOUND THEN  dbms_output.put_line(’La chaine ARTISTES n’’exi`
`WHEN OTHERS THEN  raise_application_error(-20001,’Un erreur a ete trouvé - ’||SQLCOD`
`END;`