M1 MIAGE FC/FA – 2010/2011

Bases de Données Avancées

novembre 2010

Tuning

1 Plan d'exécution d'une requête

Pour commencer, sous SQLPlus, créez une table `T` avec une centaine de lignes :

```
create table T(
a char(3) constraint pk_T primary key,
b char(97)
) ;
begin
  for i in 1..100 loop
    insert into T values (i,'voici la ligne '||i);
  end loop;
end ;
/
commit;
```

Dans JDeveloper, connectez-vous, sélectionnez votre connexion et ouvrez l'application `SQLWorksheet`. Cette application permet d'exécuter une requête SQL mais aussi de voir son plan d'exécution. Examinez et expliquez les plans d'exécution des instructions suivantes :

```
select * from T  ;
select * from T where a = '4 ' ;
select * from T where a=4 ;
select T.A from T  where T.A > '50';
select T.A, T.B from T where T.A > '50';
```

Jointure

Sous SQLPlus, créez une table `TT` qui fera référence à `T`, et remplissez-là grâce aux commandes suivantes :

```
create table TT(
a char(3) constraint PK_TT primary key,
t char(3) constraint FK_T references T,
b char(94)
) ;
```

```
begin
  for i in 1..100 loop
    for j in 0..2 loop
      if i < 90 then
        insert into TT values (j*100+i,i,'ligne ('||i||','||j||')');
      else
        insert into TT values (j*100+i,null,'ligne ('||i||','||j||')');
      end if ;
    end loop;
  end loop;
end ;
/
commit;
```

Examinez le plan d'exécution du calcul de la jointure entre `T` et `TT` :

```
select tt.a, t.a, tt.b
from t join tt on t.a = tt.t ;
```

Comparez avec les plans d'exécutions de ces requêtes (équivalentes) :

```
select tt.a, tt.t, tt.b from tt where tt.t in (select a from t) ;

select tt.a, tt.t, tt.b from tt where exists (select a from t where t.a = tt.t)

select tt.a, tt.t, tt.b from tt
where 1 = (select count(a) from t where t.a = tt.t)
```

Examinez le plan d'exécution du calcul de la jointure entre `T` et `TT`, avec un opérateur `distinct` :

```
select distinct(T.A) from T,TT where T.A = TT.T ;
```

Comparez avec cette requête équivalente, qui utilise une sous-requête au lieu d'un distinct :

```
select T.A from T where T.A in (select TT.T from TT) ;
```

group by

Comparez les plans de ces deux requêtes :

```
select a, count(*) from t group by a ;

select b, count(*) from t group by b;
```

Comparez également les plans de ces deux requêtes :

```
select t.a, count(*)
from t join tt on tt.t = t.a group by t.a ;

select t.b, count(*)
from t join tt on tt.t = t.a group by t.b ;
```

union

Comparez les plans de ces trois requêtes (équivalentes car `a` clé primaire) :

```
select * from t where a='5 ' or a='6 ' ;

select * from t where a='5 '
union
select * from t where a='6 '

select * from t where a='5 '
union all
select * from t where a='6 '
```

mode RULE et mode ALL_ROWS

Toutes les requêtes que vous avez testées utilisent le mode `ALL_ROWS` de l'optimiseur de requête. C'est le mode par défaut de traitement des requêtes. Dans les anciennes versions d'oracle, le mode par défaut était `RULE`. Vous pouvez changer de mode en utilisant la commande SQL :

```
ALTER SESSION SET OPTIMIZER_MODE = RULE
-- ou OPTIMIZER_MODE = ALL_ROWS
-- pour revenir au mode par défaut
```

Essayez maintenant la requête suivante, en examinant son plan d'exécution dans chacun des modes :

```
select t1.A, t1.B, t2.A, t2.B
from T t1, T t2
where t1.A > t2.A
```

Index sur plusieurs colonnes

Créez une dernière table `test` :

```
create table test(
a number(3),
b number(3),
c varchar2(15)
);

begin
  for i in 1..20 loop
    for j in 1..20 loop
      insert into test values (i,j,'ligne ('||i||','||j||')');
    end loop;
  end loop;
end ;
/
```

```
create index idxTest_a on test(a);

create index idxTest_b on test(b);
```

Comment est évaluée la requête suivante, dans les 2 modes d'optimisation ?

```
  select * from test
  where test.a = 15 and test.b = 20;
```

Et quand on crée un index sur les deux colonnes ?

```
create index idxTest_ab on test(a,b);
```

Revenez au mode `ALL_ROWS`. Supprimez les indexes `idxTest_a` et `idxTest_b`. Comment sont évaluées les requêtes :

```
select * from test where test.a = 15;

select * from test where test.b = 20;
```

Partitionnement

Pour optimiser les performances, l'administrateur d'une base de donnée peut être amené à partitionner ses tables ou indexes. Le partitionnement consiste à stocker séparément différentes parties d'une table (ou d'un index). Nous allons étudier ici le partionnement horizontal, où la localisation d'une donnée dépend des valeurs de ses attributs. Dans un partitionnemen vertical, ce sont les colonnes qui sont stockées à différents endroits.

Partitionnement horizontal

Dans l'exemple ci-dessous, on définit la partition selon des intervalles de valeurs d'une colonne. C'est la façon la plus simple de définir un partionnement horizontal.
Vous utiliserez les tables de l'utilisateur `CARON`

```
create table pays(
  ref_pays NUMBER constraint pays_pkey primary key,
  nom VARCHAR2(30) not null,
  nb_habitants NUMBER not null
) ;

create table ville(
  ref_ville NUMBER constraint ville_pkey primary key,
  nom VARCHAR2(30) not null,
  nb_habitants NUMBER not null,
  ref_pays NUMBER not null constraint ville_pays_fkey references pays
)
partition by range (ref_pays)
( partition ville1 values less than (5),
partition ville2 values less than (10),
partition ville3 values less than (MAXVALUE) );
```

Ainsi, une ville dont le pays a pour référence un entier entre 1 et 4 sera stockée dans la partition `ville1`, celle dont le pays a pour référence un entier entre 5 et 10 sera stockée dans `ville2` et celle dont le pays a pour référence un entier supérieur à 10 sera stockée dans `ville3`.
Regardez les plans d'exécution des requêtes suivantes :

```
select * from ville

select * from ville where ref_pays = 7

select nom, nb_habitants from ville where nom like 'c%';

select nom, nb_habitants from ville partition(ville1) where nom like 'c%';

select * from ville where ref_ville = 170
```

L'index `VILLE_PK` n'est pas partitionné. D'après-vous, quels types d'index pouvez-vous imaginer et quels sont leurs avantages ?

2 Le stockage des données

Supprimez toutes les tables créées pour les questions précédentes. Purgez votre schéma par la commande :

```
PURGE RECYCLEBIN
```

Nous allons étudier la gestion de la mémoire par le SGBD. Créez une nouvelle table sans contrainte déclarative :

```
create table T(
a char(3),
b char(97)
) ;
```

Que valent les paramètres de stockage `PCT_FREE`, `PCT_USED`, `INITIAL_EXTENT`, `NEXT_EXTENT`, `PCT_INCREASE` (vue `USER_TABLES`) pour cette table.
La commande

```
analyze table T compute statistics;
```

permet de remplir les informations suivantes :

- `NUM_ROWS` : nombre de lignes
- `BLOCKS` : nombre de blocs utilisés
- `EMPTY_BLOCKS` : nombre de blocs alloués mais non utilisés
- `AVG_SPACE` : espace moyen libre par bloc

- `AVG_ROW_LEN` : taille moyenne d'une ligne

1. Que valent ces nombres lorsque la table est vide ?
2. Même question après insertion de 50 lignes (de $a = 1$ à $a = 50$),
3. Même question après insertion de 50 lignes (de $a = 51$ à $a = 100$),
4. Même question après insertion de 100 lignes (de $a = 101$ à $a = 200$)

Supprimez une ligne sur 3 :

```
delete from T where mod(a,3)=0;
```

Recommencez l'analyse de la table et regardez les statistiques s'y rapportant.
Déduire de toutes ces expériences quelques éléments de la stratégie d'allocation de la mémoire.
On peut également examiner les paramètres de stockage d'un index, présents dans la vue `USER_INDEXES`.

- `BLEVEL` : nombre de niveaux dans le B-arbre
- `LEAF_BLOCKS` : nombre de blocs feuilles
- `DISTINCT_KEYS` : nombre de clefs différentes
- `AVG_LEAF_BLOCKS_PER_KEY` : nombre de blocs feuilles par clef
- `AVG_DATA_BLOCKS_PER_KEY` : nombre de blocs de données par clef.

1. Supprimez toutes les lignes de `T`.
2. Créez un index sur la colonne `A` de la table `T`.
3. insérez 100 lignes dans `T` (de $a = 1$ à $a = 100$)
4. Examinez les paramètres de stockage de l'index.
5. même chose après avoir inséré 800 lignes (de $a = 101$ à $a = 900$)
6. même chose après avoir inséré 100 lignes (à nouveau de $a = 1$ à $a = 100$)
7. même chose après avoir inséré 800 lignes (à nouveau de $a = 101$ à $a = 900$)
8. Que se passe-t-il si on supprime les lignes telles que $a > 100$?