09/11/2017

Système de Gestion de Base de Données

Compte-Rendu de TP

Jérémy BAUDSON & Paul CHAPELON
MASTER 1 - INFORMATIQUE

I. Seuil d'utilisation des indexes secondaires

Ce premier test devra mettre en avant l'existence d'un seuil d'utilisation des indexes secondaires, c'est-à-dire un pourcentage de tuples cherchés à partir duquel Oracle utilise un index secondaire pour effectuer cette recherche.

Pour cela, nous avons mis en place au sein de notre base de données une table ETUDIANT composées de 100 000 tuples. Chaque étudiant possède différentes informations, comme leur numéro d'étudiant qui est unique (clé primaire) ou une ville qui sera l'attribut qui nous intéresse ici. En effet, cette dernière table est composée de différents pourcentages d'étudiant, qui vont nous permettre d'analyser le seuil d'utilisation des indexes secondaires. De plus, un index secondaire a été créé sur les villes afin de voir quand est-ce que ce dernier sera utilisé.

Nous utilisons donc des requêtes de ce type pour effectuer nos analyses :

```
SELECT COUNT(*) INTO vnombre FROM ETUDIANT WHERE VILLE = 'BRESSEY' AND
COMMENTAIRE LIKE '%' ;
```

Dans cette requête, le « count(*) » permet de parcourir tous les enregistrements de la table. Le « commentaire like '%' » ne change pas le résultat de la reqûete, en revanche, il force Oracle a analysé la table en elle-même au lieu de juste compter les tuples dans l'index secondaire. Il suffit donc de changer le nom de la ville pour analyser les différents pourcentages.

Afin de calculer le seuil optimal de l'utilisation de l'index secondaire, on force Oracle à utiliser l'index secondaire, ou faire un balayage séquentiel de la table étudiant. Notons que ce forçage s'appelle un « hint ». Pour cela on utilise les deux requêtes suivantes :

```
SELECT /*+ FULL(ETUDIANT)*/ COUNT(*) into vnombre FROM ETUDIANT WHERE VILLE =
'BRESSEY' AND COMMENTAIRE LIKE '%';
```

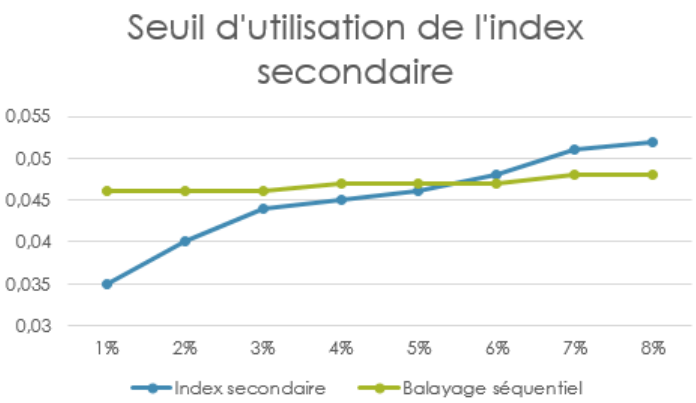
```
SELECT /*+ INDEX (A, I_Ville) */ COUNT(*) into vnombre FROM ETUDIANT A WHERE
VILLE = 'SEMUR' AND COMMENTAIRE LIKE '%';
```

La première requête force l'utilisation du balayage séquentiel, et la seconde celle de l'index secondaire sur les villes.

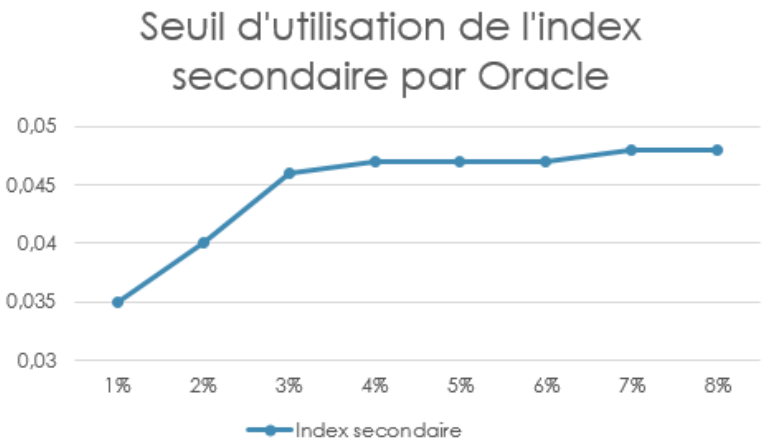
Pourcentage	Index secondaire	Balayage séquentiel
1%	0,035	0,046
2%	0,04	0,046
3%	0,044	0,046
4%	0,045	0,047
5%	0,046	0,047
6%	0,048	0,047
7%	0,051	0,048
8%	0,052	0,048

A l'aide d'un chronomètre, il nous est possible de calculer le temps d'exécution d'une requête. En effet, nous pouvons voir que le seuil optimal pour utiliser l'index secondaire se situe à l'intersection des deux courbes dans le graphique ci-dessous, c'est-à-dire environ 5.5%. Si nous interprétons ces résultats, cela veut dire que si Oracle devait utiliser un seuil optimal

d'utilisation de l'index secondaire, il devrait prendre 5%. C'est-à-dire que si 5% ou moins des tuples de la relation vérifient la condition sur l'index secondaire, alors ce dernier est plus rapide que le balayage séquentiel. Dans l'autre cas, le balayage séquentiel est plus rapide que l'index secondaire.



Cependant, on remarque qu'Oracle n'utilise pas ce seuil optimal. En effet, si l'on ne force pas l'utilisation de l'index secondaire ou du balayage séquentiel, on observe une courbe du temps comme ci-dessous. Il faut donc noter qu'ici, Oracle utilise l'index secondaire que si les tuples recherchés sont en proportion inférieur ou égale à 2%. Au-delà, il utilise le balayage séquentiel, on pourrait presque dire, à tort.



On notera donc qu'Oracle n'utilise pas le seuil optimal pour utiliser l'index secondaire à tous les coups, il utilise un seuil qu'il l'est un peu moins. Néanmoins, Oracle reste très « intelligent » pour prendre l'initiative lui-même de l'utilisation de l'index secondaire.

Nous ferons remarquer une dernière chose ; en effet, nous pouvons voir que lorsqu'Oracle utilise le balayage séquentiel, le temps continue de croître proportionnellement au pourcentage de tuples. Cette faible croissance peut s'expliquer par le fait qu'Oracle doit incrémenter le compteur « vnombre » dans la requête, et que plus le pourcentage est élevé, plus il y a d'incrémentation.

II. Variation du seul d'utilisation des indexes secondaires en fonction de la taille des enregistrements

Ce second test est inspiré du précédent. En effet, nous devons mettre en évidence que le seuil d'utilisation de l'index secondaire choisi par Oracle peut varier en fonction de la taille des tuples.

Pour cela, nous disposons de la même table ETUDIANT remplie avec les mêmes proportions d'étudiant par ville. Cependant, l'attribut commentaire a été « UPDATE » pour que le commentaire ne fasse plus 50 caractères, mais 2000 (VARCHAR2). Bien entendu, un index secondaire créé sur la Ville est toujours présent.

Pour parvenir à nos fins, nous effectuerons les mêmes requêtes que dans les précédents, en l'occurrence les requêtes du modèle suivant :

SELECT COUNT(*) into vnombre FROM ETUDIANT WHERE VILLE = 'OUGES' AND COMMENTAIRE LIKE '%';

Afin d'analyser les différents pourcentages, nous changerons les villes recherchées.

En résultat de ce test, nous n'obtenons pas le même seuil d'utilisation de l'index secondaire. Précédemment 2%, il passe désormais à 3% quand la taille des tuples est plus importante.

Ce résultat peut s'expliquer de la manière suivante : étant donné que la taille d'un enregistrement de la relation est beaucoup plus importante, le nombre d'enregistrement contenu dans un bloc de mémoire est donc réduit, ce qui augmente le nombre de bloc permettant de stocker l'intégralité des enregistrements. On peut aussi penser que l'update créer des liens de chaînage vers d'autres blocs puisque les enregistrements qui étaient auparavant dans les blocs ne peuvent plus tenir dedans, même malgré le « pctfree ». Cela explique donc qu'Oracle préfère légèrement augmenter le seuil d'utilisation des indexes secondaires car le balayage séquentiel est devenu légèrement plus coûteux.

III. Comparaison de l'utilisation d'un index secondaire et d'un index primaire sur un intervalle

Ce troisième test doit mettre en avant la différence entre un index secondaire et un index primaire, les deux construits sur le même attribut. La recherche d'enregistrements via une requête sélective sur l'attribut indexé devrait être plus rapide si elle s'effectue via un index primaire plutôt qu'un index secondaire, puisque que ce premier est trié.

Pour cela, nous travaillons toujours avec la même table ETUDIANT. Nous allons nous intéresser à l'attribut « nEtu » représentant un numéro d'étudiant sous forme d'entier unique, et qui sera donc la clé primaire de la relation. Dans un premier temps, l'index primaire sera créé sur la table grâce à la clause « ORGANIZATION INDEX » pour effectuer les deux tests suivants :

SELECT COUNT(*) FROM ETUDIANT WHERE nEtu <= 10000 AND commentaire like '%';

SELECT COUNT(*) FROM ETUDIANT WHERE MOD(nEtu,10)= 0 AND commentaire like '%';

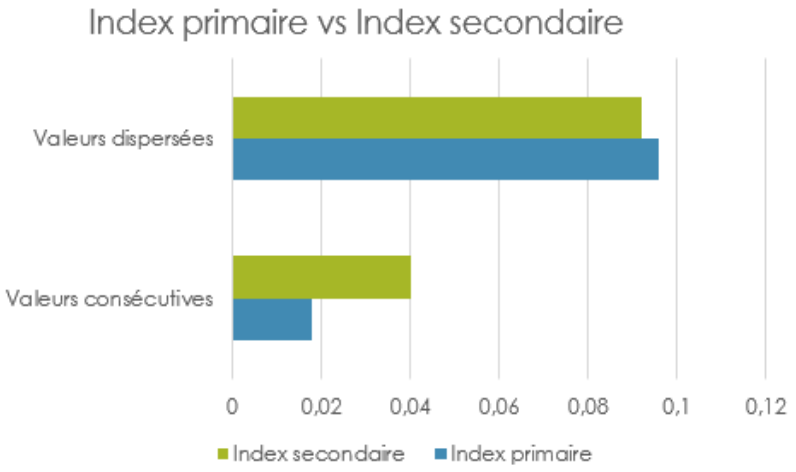
Par la suite, on applique deux nouvelles requêtes à une nouvelle table ETUDIANT, qui sera quant à elle créée sans index primaire, mais avec un index secondaire sur le même attribut que précédemment :

SELECT COUNT(*) FROM ETUDIANT WHERE nEtu <= 10000 AND commentaire like '%' ORDER BY DBMS_RANDOM.VALUE;

SELECT COUNT(*) FROM ETUDIANT WHERE MOD(nEtu,10)= 0 AND commentaire like '%' ORDER BY DBMS_RANDOM.VALUE;

Ici, la clause « ORDER BY » permet de trier aléatoirement l'index secondaire afin de ne pas récupérer les valeurs dans l'ordre croissant. Les critères de sélection permettent dans le premier cas, de récupérer les 10000 premiers enregistrements de la table, et dans le second cas, de récupérer ce même nombre d'enregistrements mais dispatchés dans la table.

Nous obtenons les résultats illustrés par le graphique ci-contre. On observe que la recherche de valeurs consécutives via un index primaire est nettement plus rapide, et nécessite moins de temps de lecture des blocs qui contiennent les enregistrements. On remarque aussi que le facteur « physicals reads » qui représente les lectures sur le disque de l'index primaire est inférieur à celui de l'index secondaire, ce qui signifie qu'Oracle lit



une quantité de bloc supérieure quand l'index n'est pas trié. Tout ceci s'explique par la consécutivité des blocs contenant les enregistrements successifs créé par l'index primaire ; étant donné un enregistrement et son suivant sur l'index primaire, ils seront dans le même bloc ou dans les blocs voisins, ce qui diminuera le temps de lecture.

Cependant, la recherche de tuples à travers l'index primaire ou l'index secondaire prend approximativement le même temps d'exécution. Désormais, les enregistrements recherchés sont dispersés dans les différents blocs, il faudra donc beaucoup plus de temps qu'avant à la tête de lecture pour accéder et lire tous ces blocs.

IV. Comparaison entre la recherche via un index secondaire et une table stockée dans un cluster de type « hash-code »

V. Comparaison de la taille d'un index secondaire et d'un index bitmap

Le test suivant a pour but d'analyser et comparer les tailles des indexes secondaires et des indexes bitmap. Ces derniers sont normalement plus efficaces quand les valeurs sur l'attribut indexés ont peu de valeurs différentes, comme un genre par exemple (Homme, Femme), puisqu'ils stockent soit 0, soit 1 pour chacune de ces valeurs.

Pour effectuer nos recherches, nous utilisons toujours la table ETUDIANT. La création d'indexes est la partie la plus concrète. En effet, dans un premier temps nous allons créer des indexes secondaires sur différents attributs grâce à ce type de commande :

```
CREATE INDEX I_VILLE ON ETUDIANT3 (ville);
```

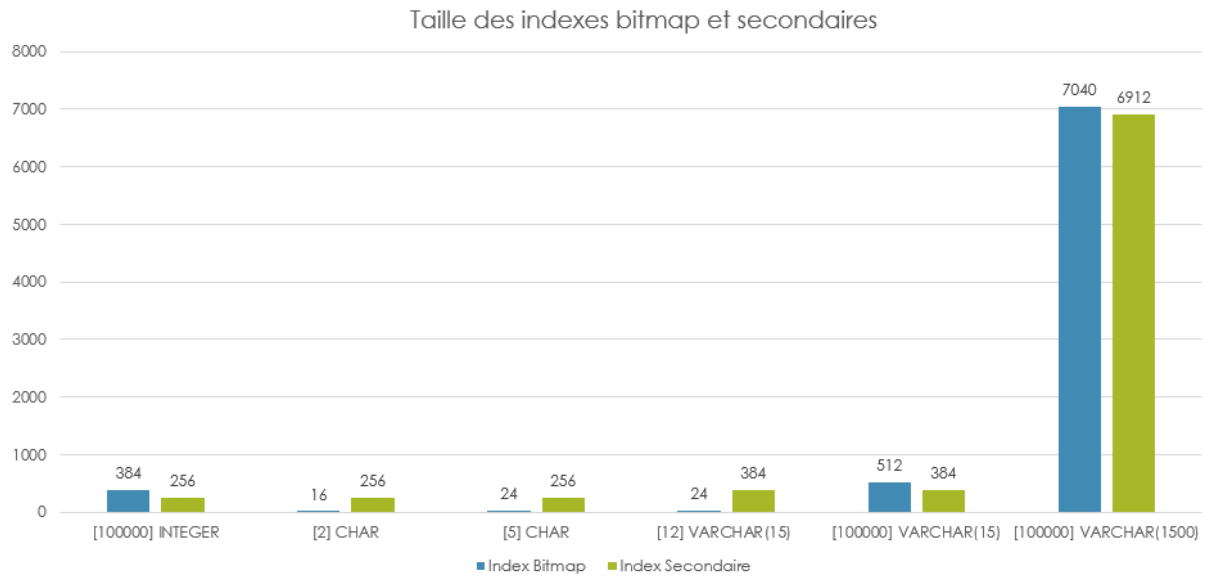
Une fois les valeurs relevées, nous créons des indexes bitmap sur les mêmes attributs grâce à une commande similaire :

```
CREATE BITMAP INDEX IB_VILLE ON ETUDIANT (ville);
```

Et pour chacun de ces indexes, on exécute une requête pour récupérer le nombre de blocs qu'ils occupent.

```
SELECT BLOCKS FROM USER_SEGMENTS WERE SEGMENT_NAME = 'I_VILLE';
```

A la suite de ces requêtes, on obtient la taille de chaque index, représenté dans le graphique ci-dessous.



Il faut noter que l'unité de l'axe des ordonnées est le nombre de blocs représentant la taille d'un index, et la valeur entre crochet à gauche des types correspond au nombre de valeurs distinctes de l'attribut. On remarque donc que les indexes bitmap sont plus petits que les indexes secondaire quand le nombre de valeurs distinctes est faible, ce qui s'explique par le fait qu'il y a un nombre de bits (0 ou 1) égal au nombre de valeurs distinctes multiplié par le nombre d'enregistrements. Dès lors que le nombre de valeurs distinctes explose, comme pour une clé primaire par exemple, les indexes bitmap perdent de leur intérêt. On notera par ailleurs que la taille des indexes secondaires dépend uniquement de la taille de l'attribut sur lequel il est construit, et du nombre d'enregistrements de la relation. Enfin, on remarque les attributs de taille élevé comme un « VARCHAR(1500) » ne sont pas des bons clients pour les indexes.

VI. Impact de la clause « size » sur la taille et l'efficacité d'un cluster de type indexé

VII. Résultats des plans d'exécution de requêtes multicritères

VIII. Utilisation des algorithmes de jointure