Article

[Irène Mykhailova](#) · Mai 1, 2022 11m de lecture

La seule astuce de performance des requêtes que vous devez connaître ? Tune Table !

Un bon écrivain est censé vous attirer avec le titre et cacher la réponse quelque part dans l'article. Je suppose que cela fait de moi un mauvais écrivain - ne m'estimez pas, mon estime de moi vient de l'opinion des utilisateurs d'Internet ! Lorsque mon cher collègue Brendan a examiné les informations sur la Communauté des développeurs, il a remarqué que nous n'avions rien écrit sur Tune Table ! Cet outil est 2nd élément le plus important pour la performance des requêtes (le premier étant les index - sans index, pas de vitesse), et il vaut la peine de prendre le temps de le comprendre. Je vais utiliser beaucoup de demi-exemples ici, mais les détails peuvent être remplis trivialement. Dans tous les exemples, nous considérons que tous les champs sont indexés individuellement, sauf indication contraire.

Très bien, supposons que nous ayons la requête suivante :

```
SELECT *  
FROM People  
WHERE HomeState = 'MA'  
AND PersonId = '123-45-6789'
```

Quel index devons-nous utiliser ? Sérieusement, arrêtez de lire et trouvez une réponse. Répétez-la à haute voix. Ne vous inquiétez pas, vos collègues ne penseront probablement pas que vous êtes fou.

L'index correct à utiliser est, bien sûr, l'index de PersonId. Pourquoi est-ce vrai ? Eh bien, nous savons que PersonId est un identifiant numérique quasi unique, alors que des millions de personnes résident dans chaque État. Ce qu'il faut comprendre, c'est que nous le savons parce que nous connaissons certaines informations sur le schéma à partir des identifiants des colonnes. Pour un ordinateur, ces noms n'ont aucun sens. Pour illustrer ce point, regardons cette requête :

```
SELECT *  
FROM TheTable  
WHERE Field1 = 32  
AND Field2 = 0
```

Doit-on utiliser l'index pour f1 ou f2 ? La requête ci-dessus est similaire à celle avec laquelle l'optimiseur de requêtes travaille. Comment choisir les index dans ce cas ? Nous devons connaître les données afin de prendre les bonnes décisions. C'est là que TuneTable entre en jeu.

Tune Table examine un échantillon des données de votre tableau et stocke des statistiques sur votre tableau afin que notre optimiseur de requêtes puisse prendre de bonnes décisions. Si vous n'avez jamais exécuté Tune Table, vos requêtes choisissent effectivement les index au hasard (ce n'est pas strictement vrai, mais c'est tout comme). En fait, si vous appelez un jour le support technique à cause d'une requête lente, la première chose que nous vous

demandons sera "Avez-vous exécuté Tune Table ?". La réponse devrait être "oui".

Tableau de configuration Tune Table : lancement

Pour lancer Tune Table, vous pouvez aller dans un terminal et exécuter :

```
d $SYSTEM.SQL.TuneTable(<Table>)
```

Il y a beaucoup de paramètres que vous pouvez passer, mais je recommande ce qui suit :

```
d $SYSTEM.SQL.TuneTable(<Table>,1,1,.msg,1)
```

Ces paramètres mettent à jour la définition de la classe, affichent les nouvelles valeurs, et le tout dernier maintient la classe à jour. Vous devrez compiler toutes les requêtes intégrées à vos tableaux et purger les requêtes en cache pour que vos requêtes puissent bénéficier des nouvelles informations de Tune Table. Plus de renseignements sur les drapeaux sont disponibles [ici](#).

Ce que fait Tune Table

Tune Table comporte plusieurs étapes. Tout d'abord, il calcule le nombre de lignes du tableau en exécutant `SELECT COUNT(*)`. Avec ce nombre, il détermine le nombre de lignes qu'il veut utiliser comme échantillon (il est asymptotiquement similaire à la racine carrée du nombre de lignes). Ensuite, pour chaque ID défini pour le tableau, nous utilisons une pièce de monnaie électronique pondérée de manière appropriée et déterminons si cette ligne fera partie de l'échantillon. Nous utilisons cet échantillon pour calculer les statistiques que nous pouvons ensuite utiliser pour optimiser vos requêtes.

Il y a deux éléments importants ici. Le premier est que nous devons obtenir le nombre. Le second est que nous devons être capables d'obtenir chaque ID dans votre tableau. Le résultat est que le fait d'avoir un index étendu améliore considérablement le temps d'exécution de Tune Table. Si votre tableau est compatible avec les bitmaps (c'est-à-dire qu'il possède un nombre intégral positif IDKEY), tout index bitmap vous donnera automatiquement un bitmap étendu.

Que mesure le Tune Table ? Je vais le présenter de manière chronologique.

Les deux OG

Les deux métriques suivantes sont les plus anciennes métriques de Tune Table. Quel âge ont-ils ? Elles ont été ajoutées avant que j'aie au lycée (1999). Si vous utilisez un Caché datant d'avant Y2K, envoyez-moi un courriel, car j'aimerais beaucoup vous parler.

Extent Size – mesure le nombre de lignes de votre tableau.

Cette métrique est la plus facile à comprendre et est utile pour décider de l'ordre des JOIN. Supposons que nous ayons la requête suivante :

```
SELECT *  
FROM Table1 T1  
JOIN Table2 T2  
ON T1.Field1 = T2.Field1
```

Devons-nous lire T1 et ensuite JOIN dans les champs de T2 ou vice-versa ? ExtentSize nous dira quelle tableau a le moins de lignes et donc par quel tableau nous devrions commencer. Dans le cas extrême, vous pouvez imaginer que T1 a 10 milliards de lignes et T2 a 100 (pensez à quelque chose comme T1 est un tableau de toutes les visites à votre hôpital au fil du temps, et T2 est votre tableau des patients qui se trouvent actuellement dans votre hôpital). Dans ce cas, il est préférable de commencer par le petit tableau et d'effectuer un JOIN dans le grand tableau, car le JOIN limite le nombre de lignes que vous devez lire.

Sélectivité – mesure l'unicité d'une valeur pour un champ donné. Cette métrique nécessite un peu plus d'explications. Il existe deux façons d'y penser :

1. $1/x$ où x est le nombre de valeurs possibles que le champ peut prendre
2. Pourcentage moyen du tableau retourné pour la requête `SELECT * FROM T WHERE champ = ?`

Prenons quelques exemples pour éclaircir ce point, car aucune de ces explications ne semble logique.

Pour la requête ci-dessus, nous avons un champ HomeState. Ce champ devrait nécessairement (aux États-Unis) avoir une sélectivité de 2,0 %. Pourquoi ? Il y a 50 États, donc $1/50 = 0,02 = 2\%$. C'est assez simple. Mais que dire de la distribution ? Il est clair que les gens ne sont pas répartis uniformément entre les États ! C'est vrai, mais la sélectivité ignore cet aspect. La distribution uniforme finit par être une bonne estimation pour la plupart des requêtes et elle est facile à calculer, ce qui signifie que les requêtes sont compilées plus rapidement.

Un autre exemple, pensons à PersonId. Un PersonId peut, occasionnellement, être réutilisé au fur et à mesure que les gens meurent et naissent, donc le numéro n'est pas tout à fait unique. Cependant, il est très très unique. Pour cela, nous pourrions voir une sélectivité de 0,00001%. C'est une très bonne sélectivité ! La requête `SELECT * FROM People WHERE PersonId = ?` ramènera une ou deux lignes à chaque fois. En général, moins la sélectivité est élevée, mieux c'est. Il existe une exception...

Et si vous avez un champ unique - par exemple un ID. La sélectivité d'un champ unique est 1. Remarque : toutes les sélectivités que j'ai mentionnées jusqu'à présent sont des pourcentages. 1 n'est pas un pourcentage mais un nombre de lignes. Donc la requête : `SELECT * FROM People WHERE ID = ?` renverra toujours 1 ligne.

Cela constitue beaucoup d'informations sur un seul champ, mais c'est peut-être le plus important. La règle est la suivante : moins le pourcentage, mieux c'est, et 1 est la meilleure valeur.

Celui que nous avons "toujours" eu

La statistique suivante a toujours été estimée mais elle a commencé à être mesurée explicitement en 2013.1

Block Count – nombre de blocs - c'est une mesure du nombre de blocs que chaque carte (index et carte principale) occupe sur le disque.

Nous avons toujours estimé le nombre de blocs, mais nous avons décidé de le mesurer explicitement en 2013. Bien que cette mesure semble être redondante avec Extent Size (la taille de l'étendue), elle a une autre utilité. Voici une bonne requête pour l'expliquer : `SELECT COUNT(*) FROM MyTable` Si chaque champ est indexé, lequel devons-nous lire ? Eh bien, nous voulons lire celui qui prend le moins d'espace sur le disque. Cette métrique nous donne cette information d'une manière explicite. Elle apparaît également lorsqu'on essaie d'estimer le coût de la lecture de morceaux d'index dans des situations plus complexes. En général, elle est bonne pour mesurer la largeur de vos cartes, tandis que ExtentSize en mesure la longueur.

Le nombre de blocs est également utile si vous utilisez une relation parent/enfant. Cette relation stockera (par défaut) les deux tableaux dans le même globale, et sans nombre de blocs, l'optimiseur de requêtes n'aurait aucune idée qu'une carte lue dans un tableau parent potentiellement petit pourrait en fait être très grande à cause du grand nombre d'enfants. En fait, c'est l'une des raisons pour lesquelles nous ne recommandons pas l'utilisation des relations parent/enfant dans les nouveaux développements (JAMAIS ! Si vous pensez que vous voulez le faire, appelez le service d'assistance et laissez-nous en discuter avec vous !).

Note de Brendan/Trivia : Nous disposons en fait de ExtentSize, Selectivity et BlockCount depuis l'époque de FDBMS, donc depuis 1992 au moins, mais il fallait les configurer manuellement.

Celui qui a tout changé

En 2014.1, nous avons ajouté une métrique pour déterminer si une valeur d'un champ était surreprésentée. Cela a entraîné une grande différence dans la façon dont nous avons calculé la sélectivité :

Sélectivité des valeurs aberrantes - c'est la sélectivité d'un champ qui est surreprésenté dans le tableau.

Pour comprendre une situation avec une valeur aberrante, imaginons un hôpital. Les hôpitaux travaillent généralement pour leur communauté locale. Pour le champ HomeState, la plupart des valeurs seront identiques à celles de l'État dans lequel se trouve l'hôpital. Prenons l'exemple d'un hôpital situé dans le Massachusetts (car c'est là que je réside). 90% des patients de mon hôpital local sont originaires du Massachusetts, tandis que d'autres viennent d'autres États ou sont en visite. Nous voulons indiquer que la recherche de HomeState='MA' n'est pas sélective, alors que la recherche de tout autre HomeState EST sélective. C'est ce que la sélectivité des valeurs aberrantes nous apporte. Par exemple, Tune Table calculera la sélectivité pour le champ HomeState décrit ci-dessus comme étant de l'ordre de 0,04 %, et indiquera que la sélectivité aberrante est de 90 % et que la valeur aberrante est "MA". La sélectivité diminue généralement. Cette modification était importante, car elle a changé la façon dont les sélectivités étaient calculées et a donné lieu à des plans de requête qui ont surpris certaines personnes. Si vous effectuez une mise à niveau à la limite de la version 2014.1, veuillez garder cela à l'esprit.

Le dernier

Nous avons ajouté cette dernière mesure pour mieux être capables d'estimer la taille des fichiers temporaires en 2015.2 Taille moyenne du champ - c'est la taille moyenne d'un champ.

Cette métrique est également facile à comprendre. Pendant que nous prenons nos échantillons, il calcule également la taille moyenne de chaque champ du tableau. Ceci est utile pour déterminer la taille des fichiers temporaires, et permet à l'optimiseur de déterminer si nous pouvons construire des fichiers temporaires en mémoire (tableau local) ou si nous avons besoin d'une structure sauvegardée sur disque (processus global privé).

Considérations pratiques

La prochaine question que vous devriez vous poser est "à quelle fréquence dois-je l'exécuter ?". La réponse est un peu subtile. Vous devez (DEVEZ DEVEZ DEVEZ !!!!) l'exécuter au moins une fois, ou fournir vous-même certaines valeurs. Après cela, il est possible que vous n'ayez plus jamais à l'exécuter. La réponse typique à cette question est que si vous êtes satisfait de vos performances actuelles, il n'y a aucune raison de l'exécuter. En effet, si vous avez une base de données mature, et que vos données augmentent à un rythme régulier parmi vos tableaux, alors vous avez probablement de bonnes valeurs en perspective.

Cependant, si vos tableaux changent et si les performances de votre SQL se dégradent, vous devriez peut-être réexécuter Tune Table. Si vous recevez beaucoup de données et que vos tableaux changent par ordre de grandeur, il est peut-être temps d'exécuter Tune Table. Mais tout ceci est basé sur la situation suivante : "Toutes mes requêtes sont lentes". Si vous en arrivez à ce point, veuillez contacter le WRC qui voudra examiner certains des paramètres de votre système.

Si vous exécutez Tune Table, assurez-vous de l'exécuter sur tous les tableaux qui sont liés au tableau que vous voulez régler. Sinon, les tailles d'extension et les nombres de blocs deviennent incomparables d'un tableau à l'autre (et les performances de vos requêtes en SERONT affectées). Si vous utilisez une version plus récente que 2016.2, vous avez l'avantage de pouvoir geler vos plans de requête avant d'exécuter Tune Table. Je recommande fortement cette pratique, car elle vous permettra d'exécuter Tune Table et de ne profiter du nouveau plan que s'il est utile (vous pouvez tester votre requête avec %NOFPLAN pour voir si les nouveaux plans vous seront utiles).

Une autre note sur Frozen Query Plans - grâce à cette technologie, vous pouvez exécuter Tune Table sans affecter les requêtes que vous avez déjà exécutées sur votre système. C'est un moyen pour vous de lancer Tune Table sans craindre d'affecter vos performances actuelles ! PAS D'EXCUSES ! Si vous souhaitez en savoir plus sur Frozen Query Plans, vous pouvez visionner un webinaire que j'ai organisé à ce sujet [ici](#).

Pour conclure, lorsqu'il s'agit de performances de requêtes, vous devez être conscient des statistiques de votre Tune Table. Tune Table donne à notre optimiseur de requêtes les informations dont il a besoin pour prendre de bonnes décisions. En supposant que vous ayez déjà ajouté des index à vos tableaux, l'exécution de Tune Table est la prochaine étape pour rendre vos requêtes extrêmement rapides.

Je remercie tout particulièrement Aaron Bentley pour m'avoir montré ce formateur SQL que j'ai utilisé pour cet article : <http://dpriver.com/pp/sqlformat.htm>

Et un deuxième remerciement à Brendan Bannon qui rend mes articles compréhensibles.

[#Performances](#) [#SQL](#) [#Caché](#) [#InterSystems](#) [IRIS](#)

URL de la source: <https://fr.community.intersystems.com/post/la-seule-astuce-de-performance-des-requ%C3%AAtes-que-vous-devez-conna%C3%AAtre-tune-table>