Article
Irène Mykhailova · Mai 11, 2022 9m de lecture

Gestion des index

Partie 2 : Gestion des index

Vous avez maintenant une bonne idée du type d'index dont vous avez besoin pour votre classe et de la manière de les définir. Ensuite, comment les gérer ?

Plan de requête

(**L'AVERTISSEMENT**: Comme toute modification d'une classe, l'ajout d'index dans un système en direct comporte des risques - si des utilisateurs accèdent à des données ou les mettent à jour pendant qu'un index est rempli, ils peuvent rencontrer des résultats de requête vides ou incorrects, ou même corrompre les index en cours de construction. Veuillez noter qu'il y a des étapes supplémentaires pour définir et utiliser les index sur un système live - ces étapes seront abordées dans cette section et sont détaillées dans notre documentation.)

Une fois le nouvel index installé, nous pouvons vérifier si l'optimiseur SQL décidera que c'est le plus efficace à lire pour exécuter la requête. Il n'est pas nécessaire d'exécuter la requête pour vérifier le plan. Avec une requête, vous pouvez vérifier le plan de manière programmatique :

```
Set query = 1

Set query(1) =  “ SELECT SSN,Name FROM Sample.Person WHERE OfficeState = 'MA' ”

        D $system.SQL.ShowPlan(.query)
```

Ou en suivant l'interface dans le portail de gestion du système via System Explorer -> SQL.

À partir de là, vous pouvez voir quels index, le cas échéant, sont utilisés avant de charger les données du tableau ("carte principale" ou "master map"). Réfléchissez : Votre nouvelle requête se trouve-t-elle dans le plan comme prévu ? La logique du plan a-t-elle un sens ?

Une fois que vous savez que l'optimiseur SQL utilise vos index, vous pouvez vérifier que ces index fonctionnent correctement.

Construction des index

(Si vous n'en êtes qu'au stade de la planification et que vous n'avez pas de données pour le moment, les étapes détaillées ici ne seront pas nécessaires dans l'immédiat.)

En définissant un index, vous n'allez pas automatiquement le remplir, ou le "construire", avec les données de votre tableau. Si un plan de requête utilise un nouvel index qui n'a pas encore été construit, vous risquez d'obtenir des résultats de requête incorrects ou vides. Vous pouvez "désactiver" un index avant qu'il ne soit prêt à être utilisé en définissant sa capacité de sélection de carte à 0 - ce qui indique essentiellement à l'optimiseur SQL qu'il ne peut pas utiliser cet index pour exécuter des requêtes.

```
write $SYSTEM.SQL.SetMapSelectability("Sample.Person","QuickSearchIDX",0)

; Set selectability of index QuickSearchIDX false
```

Notez que vous pouvez utiliser l'appel ci-dessus avant même d'ajouter un nouvel index. L'optimiseur SQL

reconnaîtra le nom de ce nouvel index, saura qu'il est inactif et ne l'utilisera pas pour les requêtes.

Vous pouvez remplir un index à l'aide de la méthode %BuildIndices (##class(<class>).%BuildIndices($lb("MyIDX"))) ou dans la page SQL du Management Portal (sous le menu déroulant Actions).

Le temps nécessaire à la construction des index dépend de la quantité de lignes dans le tableau et des types d'index que vous construisez, les index bitlice étant généralement plus longs à construire.

Une fois ce processus est terminé, vous pouvez utiliser une nouvelle fois la méthode SetMapSelectivity (cette fois-ci à 1) pour réactiver votre index nouvellement rempli.

Notez que la construction d'index signifie essentiellement l'émission de commandes KILL et SET pour les remplir. Vous pouvez envisager de désactiver la journalisation pendant ce processus pour éviter de consommer de l'espace disque.

Vous trouverez des instructions plus détaillées sur la construction de nouveaux index et d'index existants, en particulier sur des systèmes en direct, dans notre documentation dont le lien figure ci-dessous :

https://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLOPTindices#GSQLOPTindicesbuildreadwrite

Maintenance des index

Nous pouvons maintenant passer à l'utilisation effective de nos index ! Certains points à prendre en compte sont l'efficacité d'une requête, la fréquence d'utilisation d'un index donné, et les mesures à prendre si vos index entrent dans un état incohérent.

Performance

Tout d'abord, nous pouvons examiner comment cet index a affecté les performances des requêtes. Si nous savons que l'optimiseur SQL utilise un nouvel index pour une requête, nous pouvons exécuter la requête pour recueillir ses statistiques de performance : nombre de références globales, nombre de lignes lues, temps de préparation et d'exécution de la requête, et temps passé sur le disque.

Reprenons un exemple précédent :

```
"SELECT SSN,Name,DOB FROM Sample.Person WHERE Name %STARTSWITH 'Smith,J' "
```

Nous disposons de l'index suivant pour améliorer les performances de cette requête :

```
Index QuickSearchIDX On Name [ Data = (SSN, DOB, Name) ];
```

Nous avons déjà un index NameIDX sur la propriété Name.

On peut dire intuitivement que la requête sera exécutée à l'aide de QuickSearchIDX ; NameIDX serait probablement un second choix parce qu'il est basé sur la propriété Name mais ne contient aucune des valeurs de données pour SSN ou DOB.

La vérification des performances de cette requête avec QuickSearchIDX est aussi simple que son exécution.

(A propos : j'ai purgé les requêtes en cache avant d'exécuter ces requêtes pour mieux montrer les différences de performance. Lorsqu'une requête SQL est exécutée, nous stockons le plan utilisé pour l'exécuter, ce qui permet d'améliorer les performances lors de sa prochaine exécution - les détails plus fins de ce point sortent du cadre de cet article, mais j'inclurai des ressources pour d'autres considérations relatives aux performances SQL à la fin de cet article.)

Query Plan

Relative cost = 1856

- Read index map Sample.Person.QuickSearchIDX, looping on %SQLUPPER(Name) (with a %STARTSWITH range condition) and ID.
- For each row:

  Output the row.

*Nombre de lignes : **31** Performance: **0.003** seconds **154** références globales **3264** lignes exécutées **1** latence de lecture du disque (ms)*

C'est simple : nous n'avons besoin que de QuickSearchIDX pour exécuter la requête.

Comparons les performances en utilisant NameIDX plutôt que QuickSearchIDX - nous pouvons le faire en ajoutant un mot clé de requête %IGNOREINDEX, qui empêchera l'optimiseur SQL de choisir certains index.

Nous réécrivons la requête comme suit :

```
"SELECT SSN,Name,DOB FROM %IGNOREINDEX QuickSearchIDX Sample.Person WHERE Name %STARTSWITH 'Smith,J' "
```

Le plan de requête utilise maintenant NameIDX, et nous voyons que nous devons lire à partir de la globale de données ("carte principale" ou "master map") de Sample.Person en utilisant les ID de ligne pertinents trouvés via l'index.

Query Plan

Relative cost = 24763

- Call module B, which populates bitmap temp-file A.
- Read bitmap temp-file A, looping on ID.
- For each row:

  Read master map Sample.Person.IDKEY, using the given idkey value.
  Output the row.

module B

- Read index map Sample.Person.NameIDX, looping on %SQLUPPER(Name) (with a %STARTSWITH range condition) and ID.
- For each row:

  Add ID bit to bitmap temp-file A.

*Nombre de lignes : **31** Performance: **0.020** seconds **137** références globales **3792** lignes exécutées **17** latence de lecture du disque (ms)*

Nous constatons que le temps d'exécution est plus long, que le nombre de lignes de code exécutées est plus élevé et que la latence du disque est plus importante.

Ensuite, considérons l'exécution de cette requête sans aucun index. Nous ajustons notre requête comme suit :

```
"SELECT SSN,Name,DOB FROM %IGNOREINDEX * Sample.Person WHERE Name %STARTSWITH 'Smith,J' "
```

Sans l'utilisation des index, nous devons vérifier les lignes de données pour notre condition.

Query Plan

Relative cost = 3622510

- Read master map Sample.Person.IDKEY, looping on ID.
- For each row:

  Output the row.

*Nombre de lignes : **31** Performance: **0.765** seconds **149999** références globales **1202681** lignes exécutées **517** latence de lecture du disque (ms)*

Il montre qu'un index spécialisé, QuickSearchIDX, permet à notre requête de s'exécuter plus de 100 fois plus vite que sans index et près de 10 fois plus vite qu'avec l'index NameIDX plus général - et l'utilisation de NameIDX est au moins 30 fois plus rapide que sans index du tout.

Pour cet exemple particulier, la différence de performance entre QuickSearchIDX et NameIDX peut être négligeable, mais pour des requêtes exécutées des centaines de fois par jour avec des millions de lignes à requérir, nous verrions un gain de temps précieux jour après jour.

Analyse des index existants à l'aide de SQLUtilites

%SYS.PTools.SQLUtilities contient des procédures, telles que IndexUsage, JoinIndices, TablesScans, et TempIndices. Elles analysent les requêtes existantes dans un espace de noms donné et fournissent des informations sur la fréquence d'utilisation d'un index donné, sur les requêtes qui choisissent d'itérer sur chaque ligne d'une table et sur les requêtes qui génèrent des fichiers temporaires pour simuler des index.

Vous pouvez utiliser ces procédures pour déterminer les lacunes qu'un index pourrait résoudre et tout index que vous pourriez envisager de supprimer en raison d'un manque d'utilisation ou d'inefficacité.

Vous trouverez des détails sur ces procédures et des exemples de leur utilisation dans la documentation de nos classes ci-dessous :

https://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLOPToptqueryindexanalysis

Des problèmes d'index ?

La validation des index confirme que chaque index existe et est défini correctement pour chaque ligne d'une classe. Aucune classe ne devrait se retrouver dans un état où les index sont corrompus, mais si vous constatez que les requêtes renvoient des ensembles de résultats vides ou incorrects, vous pouvez envisager de vérifier si les index existants des classes sont actuellement valables.

Vous pouvez valider les index de manière programmatique comme suit :

```
Set status = ##class(<class>).%ValidateIndices(indices,autoCorrect,lockOption,multiProcess)
```

Ici, le paramètre des index est une chaîne vide par défaut, ce qui signifie que nous validons tous les index, ou un objet $listbuild contenant les noms des index.

Notez que la valeur par défaut d'autoCorrect est 0. Si elle est 1, toutes les erreurs rencontrées pendant le processus de validation seront corrigées. Bien que cela soit fonctionnellement identique à la reconstruction des index, les performances de ValidateIndices sont plus lentes en comparaison.

Veuillez vous référer à la documentation de la classe %Library.Storage pour trouver plus d'info :

https://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?PAGE=CLASS&LIBRARY=%25SYS&CLASSNAME=%25Library.Storage#%ValidateIndices

Suppression des index

Si vous n'avez plus besoin d'un index - ou si vous prévoyez d'apporter un grand nombre de modifications à un tableau et que vous souhaitez conserver l'impact sur les performances de la construction des index pertinents pour plus tard - vous pouvez simplement supprimer la définition de l'index de la classe dans Studio et supprimer le nœud global d'index approprié. Vous pouvez également exécuter une commande DROP INDEX via DDL, qui effacera également la définition et les données de l'index. À partir de là, purgez les requêtes en cache pour vous assurer qu'aucun plan existant n'utilisera l'index désormais supprimé.

Et maintenant, quoi ?

Les index ne sont qu'une partie des performances de SQL. Dans le même ordre d'idées, il existe d'autres options pour surveiller les performances et l'utilisation de vos index. Pour comprendre les performances de votre SQL, vous pouvez également envisager de vous renseigner sur les sujets suivants :

Tune Tables - Un utilitaire que nous exécutons une fois qu'un tableau est rempli de données représentatives ou si la distribution des données prend un virage massif. Il remplit votre définition de classe avec des métadonnées, par exemple, la longueur attendue d'un champ ou le nombre de valeurs uniques dans un champ, ce qui aide l'optimiseur SQL à choisir un plan de requête pour une exécution efficace.

Kyle Baxter a publié un article à ce sujet ici.

Plans de requête - La représentation logique de la façon selon laquelle notre code sous-jacent exécute les requêtes SQL. Si votre requête est lente, nous pouvons examiner quel plan de requête est généré, s'il a un sens pour votre requête et ce qui peut être fait pour optimiser ce plan.

Requêtes mises en cache - Instructions SQL dynamiques préparées - les requêtes mises en cache sont essentiellement le code sous-jacent aux plans de requête.

Lecture supplémentaire

Documentation sur la définition et la construction des index comprend des étapes supplémentaires à prendre en compte sur les systèmes de lecture-écriture en direct.
https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLOPTindices

Commandes SQL ISC - voir CREATE INDEX et DROP INDEX pour les références syntaxiques de la manipulation des index via DDL. Comprend les autorisations d'utilisateur appropriées pour l'exécution de ces commandes.
https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=RSQLCOMMANDS

Détails sur la collation SQL dans les classes ISC. Par défaut, les valeurs de chaîne sont stockées dans les globales d'index comme SQLUPPER (" STRING ") :

https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLcollation

#Indexation #Performances #SQL #Caché #InterSystems IRIS

URL de la source:https://fr.community.intersystems.com/post/gestion-des-index