Artículo
Ricardo Paiva · Mar 26, 2020 Lectura de 14 min

Conoce tus índices

Este es el primero de dos artículos sobre los índices SQL.

Parte 1 - Conoce tus índices

¿ Qué es un índice?

Recuerda la última vez que fuiste a una biblioteca. Normalmente, los libros están ordenados por temática (y luego autor y título) y cada repisa tiene un cartel en el extremo con un código que describe la temática de los libros. Si necesitaras libros de un cierto tema, en lugar de caminar por cada pasillo y leer la descripción en la parte interior de cada libro, podrías dirigirte directamente al estante cuyo cartel describa la temática que buscas y elegir tus libros de allí. Sin esos carteles, el proceso de encontrar los libros que quieres, habría sido muy lento.

Un índice SQL tiene la misma función general: mejorar el rendimiento, al ofrecer una referencia rápida del valor de los campos para cada fila de una tabla.

Configurar índices es uno de los pasos más importantes a la hora de preparar tus clases para un rendimiento óptimo de SQL.

En este artículo veremos:

1. ¿Qué es un índice y por qué/cuando se deben usar?
2. ¿Qué tipo de índices existen y en qué situaciones son perfectos?
3. ¿A qué se parece un índice?
4. ¿Cómo se puede crear uno?
5. Y cuando tengo índices, ¿qué hago con ellos?

Me referiré a las clases de nuestro esquema Sample, que se incluye en el namespace Samples en instalaciones de Caché y Ensemble. Puedes descargar estas clases de nuestro repositorio en Github:

https://github.com/intersystems/Samples-Data

Lo fundamental

Puedes indexar cualquier propiedad persistente y cualquier propiedad que puede ser calculada de forma fiable a partir de datos persistentes.

Supongamos que queremos indexar la propiedad TaxID (identificador impositivo) en Sample.Company. En Studio o Atelier, añadiríamos lo siguiente a la definición de la clase:

```
Index TaxIDIdx On TaxID [ Type = index, Unique ];
```

La sentencia DDL SQL equivalente se vería como algo así:

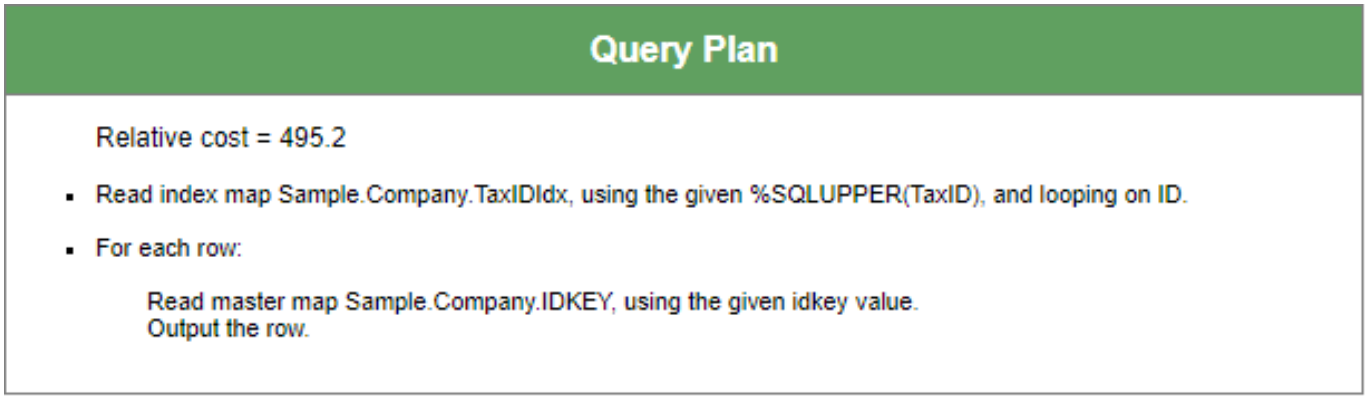
```
CREATE UNIQUE INDEX TaxIDIdx ON Sample.Company (TaxID);
```

La estructura del índice global predeterminado es la siguiente:

```
^Sample.CompanyI("TaxIDIdx ",<TaxIDValueAtRowID>,<RowID>) = ""
```

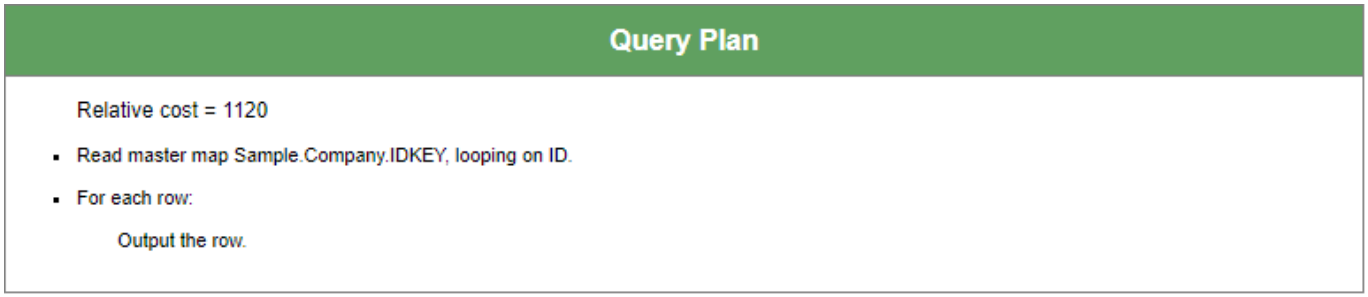
Ten en cuenta que hay menos subíndices para leer que campos en un global de datos típicos.

Mira la consulta "`SELECT Name,TaxID FROM Sample.Company WHERE TaxID = 'J7349'`". Es lógica y simple, y el plan de consulta para ejecutar esta consulta lo refleja:



Este plan dice, básicamente, que buscamos columnas con el valor dado de TaxID en el global del índice, y luego volvemos a buscar en el global de datos ("master map") para recuperar la fila coincidente.

Ahora mira la misma consulta *sin* un índice en TaxIDX. El plan de consulta resultante es, como se esperaría, menos eficiente:



Sin índices, la ejecución de consultas subyacente de Caché se basa en leer en la memoria y aplicar la condición de la cláusula WHERE a cada fila de la tabla. Y como TaxID es único, ¡estamos haciendo todo este trabajo tan solo para una fila!

Por supuesto, tener índices significa tener datos de índices *y* filas en disco. Dependiendo de en qué tengamos una condición y cuántos datos contenga nuestra tabla, esto puede generar sus propios desafíos al crear y poblar un índice.

Entonces, ¿cuándo agregamos un índice a una propiedad?

El caso general es cuando condicionamos con frecuencia en base a una propiedad. Algunos ejemplos

son identificar información como el número de seguridad social (SSN) de una persona o su número de cuenta bancaria. También puede pensar en fechas de nacimiento o en los fondos de una cuenta. Volviendo a Sample.Company, quizás la clase se vería beneficiada de indexar la propiedad Revenue (ingresos) si quisiéramos recopilar datos sobre organizaciones con altos ingresos. Por otra parte, las propiedades que es poco probable que condicionemos son menos adecuadas para indexar, como por ejemplo el eslogan o descripción de una empresa.

Simple, ¡excepto que también debemos considerar qué tipo de índice es el mejor!

Tipos de índices

Hay seis tipos principales de índices que describiré aquí: estándar, bitmap, compuesto, recopilación, bitslice y datos. También describiré brevemente los índices iFind, que se basan en flujos. Aquí hay posibles solapamientos, y ya hemos visto los índices estándar con el ejemplo anterior.

Compartiré ejemplos de cómo crear índices en tu definición de clase, pero añadir nuevos índices a una clase es más complejo que tan solo añadir una línea a tu definición de clase. En la próxima parte analizaremos todas las consideraciones adicionales.

Usemos Sample.Person como ejemplo. Ten en cuenta que Person tiene la subclase Employee (empleado), que será relevante para entender algunos ejemplos. Employee comparte su almacenamiento de global de datos con Person, y todos los índices de Person son heredados por Employee. Esto significa que Employee usa el global de índices de Person para estos índices heredados.

Si no estás familiarizado con ellas, esta es una descripción general de las clases: Person tiene propiedades SSN (número de seguridad social), DOB (fecha de nacimiento), Name (nombre), Home (un objeto embebido de dirección tipo Address que contiene State (estado) y City (ciudad)), Office (oficina, también de tipo Address) y la colección de listas FavoriteColors (colores favoritos). Employee tiene la propiedad adicional Salary (salario, que definí yo misma).

Estándar

```
Index DateIDX On DOB;
```

Aquí uso "estándar" de forma de forma poco precisa, para referirme a índices que almacenan el valor sencillo de una propiedad (a diferencia de una representación binaria). Si el valor es una cadena, se almacenará bajo alguna compilación (collation) – SQLUPPER por defecto.

En comparación con índices bitmap o bitslice, los índices estándar son mucho más fáciles de leer por una persona y su mantenimiento es bastante sencillo. Tenemos un nodo global para cada fila de la tabla.

A continuación se muestra cómo se almacena DateIDX a nivel global.

```
^Sample.PersonI("DateIDX",51274,100115)="~Sample.Employee~" ; Date is
05/20/81
```

Ten en cuenta que el primer subscript después del nombre del índice es el valor de la fecha, el último subscript es el ID de Person con esa DOB (fecha de nacimiento) y el valor almacenado en este nodo global indica que esta persona también es miembro de la subclase Sample.Employee. Si esa persona no fuera miembro de ninguna subclase, el valor en el nodo sería una cadena vacía.

Esta estructura base será consistente con la mayoría de los índices que no sean bits, en los cuales los índices en más de una propiedad crean más subscripts en el global y tener más de un valor almacenado en el nodo genera un objeto $listbuild, por ejemplo:

```
        ^Package.ClassI(IndexName,IndexValue1,IndexValue2,IndexValue3,RowID)
= $lb(SubClass,DataValue1,DataValue2)
```

Bitmap – Una representación bit a bit (bitwise) del conjunto de IDs que corresponden al valor de una propiedad.

```
   Index HomeStateIDX On Home.State [ Type = bitmap];
```

Los índices de bitmap se guardan por valor único, a diferencia de los índices estándar, que se almacenan por fila.

Continuando con el ejemplo anterior, digamos que la persona con el ID 1 vive en Massachusetts, el ID 2 en Nueva York, ID 3 en Massachusetts e ID 4 en Rhode Island. HomeStateIDX básicamente se almacena así:

(...)	0	0	0	0	-
MA	1	0	1	0	-
NY	0	1	0	0	-
RI	0	0	0	1	-
(...)	0	0	0	0	-

Si quisiéramos que una consulta devuelva datos de las personas que viven en New England, el sistema realizaría un OR bit a bit (bitwise) sobre las filas relevantes del índice bitmap. Se puede ver rápidamente que debemos cargar en memoria los objetos Person con ID 1, 3 y 4 como mínimo.

Los bitmaps pueden ser eficientes para operadores AND, RANGE y OR en tus claúsulas WHERE.

Si bien no hay un límite oficial sobre la cantidad de valores únicos que puede tener para una propiedad antes de que un índice tipo bitmap sea menos eficiente que un índice estándar, la regla general es de hasta unos 10.000 valores distintos. Entonces, mientras un índice tipo bitmap podría ser efectivo en un estado de los EE. UU., un índice bitmap para una ciudad o condado no sería tan útil.

Otro concepto a tener en cuenta es la eficiencia de almacenamiento. Si piensas añadir o eliminar filas de tu tabla con frecuencia, el almacenamiento de tu índice tipo bitmap podría volverse menos eficiente. Veamos el ejemplo anterior: si eliminámaramos muchas filas por algún motivo y ya no tenemos a nadie en

nuestra tabla que viva en estados menos poblados como Wyoming o Dakota del Norte, el bitmap entonces tendría varias filas solo con ceros. Por otra parte, crear nuevas filas en tablas grandes al final puede volverse más lento, ya que el almacenamiento de bitmaps grandes debe alojar más valores únicos.

En estos ejemplos tengo unas 150.000 filas en Sample.Person. Cada nodo global almacena hasta 64.000 ID's, por lo que el global del índice bitmap en el valor MA está dividido en tres partes:

```
^Sample.PersonI("HomeStateIDX"," MA",1)=$zwc(135,7992)_$c(0,(...))
^Sample.PersonI("HomeStateIDX"," MA",2)=$zwc(404,7990,(…))
^Sample.PersonI("HomeStateIDX"," MA",3)=$zwc(132,2744)_$c(0,(…))
```

Caso especial: Extent Bitmap

Un bitmap extent, a menudo llamado $<ClassName>, es un índice tipo bitmap sobre los ID de una clase. Esto brinda a Caché una forma rápida de saber si una fila existe y puede ser útil para consultas COUNT o consultas sobre subclases. Estos índices se generan cuando un índice tipo bitmap se añade a la clase. También puedes crear manualmente un índice extent bitmap en una definición de clase de la siguiente forma:

```
Index Company [ Extent, SqlName = "$Company", Type = bitmap ];
```

O mediante la DDL keyword BITMAPEXTENT:

```
CREATE BITMAPEXTENT INDEX "$Company" ON TABLE Sample.Company
```

Compuesto – Índices basados en dos o más propiedades

```
Index OfficeAddrIDX On (Office.City, Office.State);
```

El caso de uso general de los índices compuestos es tener consultas frecuentes con condiciones sobre dos o más propiedades.

El orden de las propiedades en un índice compuesto importa, debido a la forma en que se almacena el índice en un nivel global. Tener primero la propiedad más selectiva es más eficiente para el rendimiento, ya que ahorrará lecturas de disco iniciales del global de índices. En este ejemplo. Office.City está primero debido a que hay más ciudades únicas que estados en los EE. UU.

Tener primero una propiedad menos selectiva es más eficiente para el espacio. En términos de estructura global, el árbol de índices estaría mejor equilibrado si el estado (State) estuviera primero. Piénsalo: cada estado contiene varias ciudades, pero algunos nombres de ciudades pertenecen a un único estado.

También puedes considerar si esperas ejecutar consultas frecuentes que condicionan solo una de las propiedades. Esto puede ahorrarte definir otro índice más.

Este es un ejemplo de la estructura global del índice compuesto:

```
   ^Sample.PersonI("OfficeAddrIDX"," BOSTON","
MA",100115)="~Sample.Employee~"
```

¿Índice compuesto o índices de bitmap?

Para consultas con condiciones sobre múltiples propiedades, puede que también quieras evaluar si índices tipo bitmap separados serían más efectivos que un único índice compuesto.

Las operaciones bit a bit en dos índices distintos podrían ser más eficientes, considerando que los índices bitmap se adecuan bien a cada propiedad.

También puedes tener índices tipo bitmap compuestos: son índices tipo bitmap en los que el valor único es la intersección de las múltiples propiedades sobre las que estás indexando. Por ejemplo, la tabla de la sección anterior, pero si en lugar de estados tenemos cada par posible de estado y ciudad (p. ej. Boston, MA, Cambridge, MA, incluso Los Angeles, MA, etc.) y las celdas reciben valores 1 por las filas que cumplen ambos valores.

Recopilación – Índices basados en propiedades de recopilación

Aquí tenemos la propiedad FavoriteColors definida de la siguiente forma:

```
Property FavoriteColors As list Of %String(JAVATYPE =
"java.util.List", POPSPEC =
"ValueList("",Red,Orange,Yellow,Green,Blue,Purple,Black,White""):2");
```

Con cada uno de los siguientes índices definidos con fines de demostración:

```
Index fcIDX1 On FavoriteColors(ELEMENTS);

Index fcIDX2 On FavoriteColors(KEYS);
```

Aquí uso "recopilación" para referirme de forma más amplia a propiedades de una celda que contienen más de un valor. Las propiedades List Of y Array Of son relevantes aquí e incluso las cadenas delimitadas.

Las propiedades de recopilación se analizan automáticamente para construir sus índices. Para propiedades delimitadas, como un número telefónico, deberá definir este método, <PropertyName>BuildValueArray(value, .valueArray), de forma explícita.

Dado el ejemplo anterior para FavoriteColors, fcIDX1 se vería como algo así para una persona cuyos colores favoritos fueran azul y blanco:

```
^Sample.PersonI("fcIDX1"," BLUE",100115)="~Sample.Employee~"

(...)

^Sample.PersonI("fcIDX1"," WHITE",100115)="~Sample.Employee~"
```

fcIDX2 se vería así:

```
^Sample.PersonI("fcIDX2",1,100115)="~Sample.Employee~"

^Sample.PersonI("fcIDX2",2,100115)="~Sample.Employee~"
```

En este caso, como FavoriteColors es una colección de List, un índice basado en sus claves es menos útil que un índice basado en sus elementos.

Consulta nuestra documentación para cuestiones más específicas sobre crear y gestionar índices basados en propiedades de recopilación:

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

Bitslice – Representación bitmap de la representación de la cadena de bits de datos numéricos

```
Index SalaryIDX On Salary [ Type = bitslice ]; //In Sample.Employee
```

A diferencia de los índices bitmap, que contienen marcas (flags) que representan qué filas contienen un valor específico, los índices bitslice primero convierten valores numéricos de decimal a binario y luego crean un bitmap en cada dígito del valor binario.

Tomemos el ejemplo anterior y, para ser realistas, simplifiquemos Salary (salario) como unidades de $1000. Así, si el salario de un empleado se guarda como 65, se entiende que representa $65.000.

Digamos por ejemplo que tenemos Employee con ID 1 y salario 15, ID2 con salario 40, ID 3 con salario 64 e ID 4 con salario 130. Los valores de bits correspondientes son:

	0	0	0	0	1	1	1	
	0	0	1	0	1	0	0	
	0	1	0	0	0	0	0	
	1	0	0	0	0	0	1	

Nuestra cadena de bits abarca 8 dígitos. La representación bitmap correspondiente (los valores de índices bitslice) se almacena básicamente así:

```
^Sample.PersonI("SalaryIDX",1,1) = "1000" ; Row 1 has value in 1’s
place
```

```
     ^Sample.PersonI("SalaryIDX",2,1) = "1001" ; Rows 1 and 4 have values in
2’s place

     ^Sample.PersonI("SalaryIDX",3,1) = "1000" ; Row 1 has value in 4’s
place

     ^Sample.PersonI("SalaryIDX",4,1) = "1100" ; Rows 1 and 2 have values in
8’s place

     ^Sample.PersonI("SalaryIDX",5,1) = "0000" ; etc…

     ^Sample.PersonI("SalaryIDX",6,1) = "0100"

     ^Sample.PersonI("SalaryIDX",7,1) = "0010"

     ^Sample.PersonI("SalaryIDX",8,1) = "0001"
```

Ten en cuenta que las operaciones que modifican Sample.Employee o los salarios de sus filas (es decir, INSERT, UPDATES y DELETE) ahora requieren actualizar cada uno de estos nodos globales, o bitslices. Añadir un índice bitslice a múltiples propiedades de una tabla o una propiedad que se modifica con frecuencia puede conllevar riesgos de rendimiento. En general, mantener un índice bitslice es más costoso que mantener índices de tipo estándar o bitmap.

Los índices bitslice son altamente especializados y por eso tienen casos de uso específicos: consultas que deben realizar cálculos agregados (p.ej. SUM, COUNT o AVG).

Además, solo pueden usarse de forma efectiva sobre valores numéricos (las cadenas de caracteres se convierten a un 0 binario).

Si es necesario leer la tabla de datos, en lugar de los índices, para verificar la condición de una consulta, los índices bitslice no se elegirán para ejecutar la consulta. Digamos que Sample.Person no tiene un índice en Name. Si quisiéramos calcular el salario medio de los empleados cuyo apellido es Smith (“SELECT AVG(Salary) FROM Sample.Employee WHERE Name %STARTSWITH 'Smith,' “) necesitaríamos leer filas de datos para aplicar la condición WHERE, por lo que en la práctica no se usaría el índice bitslice.

Hay varios problemas de almacenamiento similares para índices bitslice y bitmap en tablas en las que se crean o eliminan filas frecuentemente.

Datos - Índices con datos almacenados en sus nodos globales.

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

En varios de los ejemplos anteriores, puedes haber observado la cadena “~Sample.Employee~” almacenada como el valor del propio nodo. Recuerda que Sample.Employee hereda índices de Sample.Person. Cuando hacemos una consulta sobre los empleados (Employees) en particular, leemos el valor en los nodos de índices que coinciden con la condición de nuestra propiedad para verificar que dicha persona (Person) también sea un empleado.

También podemos definir explícitamente qué valores almacenar. Definir los datos en tus nodos globales de índices puede ahorrarte lecturas de todos los datos globales. Esto puede ser útil para consultas ordenadas o selectivas frecuentes.

Tomamos como ejemplo el índice anterior. Si quisiéramos extraer información identificatoria sobre una persona dado su nombre completo o parcial (p. ej. para buscar información del cliente en una aplicación de atención al público), podríamos tener una consulta como "SELECT SSN, Name, DOB FROM Sample.Person WHERE Name %STARTSWITH 'Smith,J' ORDER BY Name". Como las condiciones de nuestra consulta sobre Name y los valores que estamos recuperando se encuentran todos dentro de los nodos globales QuickSearchIDX, solo necesitamos leer nuestro global I para ejecutar esta consulta.

Ten en cuenta que los valores de datos no pueden almacenarse con índices bitmap o bitslice.

```
     ^Sample.PersonI("QuickSearchIDX"," LARSON,KIRSTEN
A.",100115)=$lb("~Sample.Employee~","555-55-5555",51274,"Larson,Kirsten A.")
```

Índices iFind

¿Alguna vez ha oído hablar de ellos? Yo tampoco. Los índices iFind se usan en propiedades de flujo, pero para usarlos se deben especificar sus nombres con palabras clave en la consulta.

Podría explicarlo más en detalle, pero Kyle Baxter tiene un artículo muy útil sobre esto:

https://community.intersystems.com/post/free-text-search-way-search-your-text-fields-sql-developers-are-hiding-you

Continúa leyendo la Parte 2, sobre la gestión de índices definidos.

#Indexación #Mejores prácticas #Rendimiento #SQL #Caché #InterSystems IRIS

URL de fuente: https://es.community.intersystems.com/post/conoce-tus-%C3%ADndices