文章
姚 鑫 · 四月 16, 2021 阅读大约需 12 分钟

第二章 定义和构建索引（四）

第二章 定义和构建索引（四）

位片索引

当数字数据字段用于某些数值运算时，位片索引用于该字段。位片索引将每个数值数据值表示为二进制位串。位片索引不是使用布尔标志来索引数值数据值(如在位图索引中那样)，而是以二进制值表示每个值，并为二进制值中的每个数字创建一个位图，以记录哪些行的该二进制数字具有1。这是一种高度专门化的索引类型，可以显著提高以下操作的性能：

- SUM、COUNT或AVG
  Aggregate计算。(位片索引不用于COUNT(*)计算。)。位片索引不用于其他聚合函数。
- 指定的字段 TOP n ... ORDER BY field
- 在范围条件运算中指定的字段，WHERE field > n 或 WHERE field BETWEEN lownum AND highnum、

SQL优化器确定是否应该使用定义的位片索引。通常，优化器仅在处理大量(数千)行时才使用位片索引。

可以为字符串数据字段创建位片索引，但位片索引将这些数据值表示为规范数字。换句话说，任何非数字字符串(如 “ abc ”)都将被索引为0。这种类型的位片索引可用于快速计数具有字符串字段值的记录，而不计算那些为空的记录。

在下面的例子中，Salary是位片索引的候选项:

```
SELECT AVG(Salary) FROM SalesPerson
```

位片索引可用于使用WHERE子句的查询中的聚合计算。如果WHERE子句包含大量记录，则这是最有效的。在下面的示例中，SQL优化器可能会使用Salary上的位片索引(如果已定义)；如果定义了位片索引，它还会使用REGION上的位图索引，使用定义的位图或为REGION生成位图临时文件：

```
SELECT AVG(Salary) FROM SalesPerson WHERE Region=2
```

但是，当索引无法满足WHERE条件时，不使用位片索引，而必须通过读取包含要聚合的字段的表来执行。以下示例将不使用Salary的位片索引：

```
SELECT AVG(Salary) FROM SalesPerson WHERE Name LIKE '%Mc%'
```

可以为任何包含数值的字段定义位片索引。InterSystems
SQL使用Scale参数将小数转换为位字符串，如ObjectScript $factor函数中所述。可以为数据类型字符串的字段定义位片索引；在这种情况下，出于位片索引的目的，非数字字符串数据值被视为0。

可以为系统分配的行ID为正整数值的表中的字段定义位片索引，也可以为使用%BID属性定义以支持位图(和位片)索引的表中的字段定义位片索引。

位片索引只能为单个字段名定义，不能为多个字段的连接定义。
不能指定WITH DATA子句。

下面的例子比较了位片索引和位图索引。
如果你为1、5和22行创建一个位图索引，它会为这些值创建一个索引:

```
^gloI("bitmap",1,1)= "100"
^gloI("bitmap",5,1)= "010"
^gloI("bitmap",22,1)="001"
```

如果为第1、2和3行的值1、5和22创建位切片索引，则会首先将这些值转换为位值：

```
1 =   00001
5 =   00101
22 = 10110
```

然后，它为这些位创建索引：

```
^gloI("bitslice",1,1)="110"
^gloI("bitslice",2,1)="001"
^gloI("bitslice",3,1)="011"
^gloI("bitslice",4,1)="000"
^gloI("bitslice",5,1)="001"
```

在本例中，位图索引中的值22需要设置1个全局节点；位片索引中的值22需要设置3个全局节点。

请注意，插入或更新需要在所有n个位片中设置一个位，而不是设置单个位串。这些附加的全局设置操作可能会影响涉及填充位片索引的插入和更新操作的性能。使用INSERT、UPDATE或DELETE操作填充和维护位片索引比填充位图索引或常规索引慢。维护多个位片索引和/或在频繁更新的字段上维护位片索引可能具有显著的性能成本。

在易失性表(执行许多插入、更新和删除操作)中，位片索引的存储效率可能会逐渐降低。%SYS.Maint.Bitmap实用程序方法同时压缩位图索引和位片索引，从而提高了还原效率。

重建索引

可以按如下方式构建/重新构建索引：
- 使用BUILD INDEX SQL命令构建指定索引，或构建为表、架构或当前命名空间定义的所有索引。
- 使用管理门户重建指定类(表)的所有索引。
- 使用%BuildIndices()(或%BuildIndicesAsync())方法，如本节所述。

当前数据库访问确定应如何重建现有索引：
- 非活动系统(在索引构建或重建期间没有其他进程访问数据)
- READONLY活动系统(能够在索引构建或重建期间查询数据的其他进程)
- 读写活动系统(能够在索引构建或重建期间修改数据和查询数据的其他进程)

构建索引的首选方法是使用%BuildIndices()方法或%BuildIndicesAsync()方法。
- %Library.Persistent.%BuildIndices()：%BuildIndices()作为后台进程执行，但调用方必须等待%BuildIndices()完成才能接收回控制。
- %Library.Persistent.%BuildIndicesAsync()：%BuildIndicesAsync()将%BuildIndices()作为后台进程启动，调用方立即收到控制权。%BuildIndicesAsync()的第一个参数是eueToken输出参数。其余参数与%BuildIndices()相同。

%BuildIndicesAsync()返回%Status值：Success表示%BuildIndices()辅助作业已成功排队；失败表示该辅助作业未成功排队。

%BuildIndicesAsync()向eueToken输出参数返回一个值，该值指示%BuildIndices()完成状态。要获取完成状态，请通过引用将eueToken值传递给%BuildIndicesAsyncResponse()方法。还可以指定等待布尔值。如果wait=1，则%BuildIndicesAsyncResponse()将等待，直到由eueToken标识的%BuildIndices() JOB 完成。如果wait=0，%BuildIndicesAsyncResponse()将尽快返回状态值。如果返回时%BuildIndicesAsyncResponse()
èueToken不为空，则%BuildIndices() job尚未完成。在这种情况下，可以使用eueToken再次调用%BuildIndicesAsyncResponse()。当%BuildIndicesAsyncResponse()èueToken最终为NULL时，返回的%BuildIndicesAsyncResponse()%Status值是%BuildIndicesAsync()调用的job的完成状态。

在非活动系统上构建索引

系统自动生成方法(由%Persistent类提供)，这些方法构建或清除为类(表)定义的每个索引。可以通过以下两种方式之一使用这些方法：
- 通过管理门户进行交互。
- 以编程方式，作为方法调用。

构建索引执行以下操作：
1. 删除索引的当前内容。
2. 扫描(读取每一行)主表，并为表中的每一行添加索引项。如果可能，使用特殊的$SortBegin和$SortEnd函数来确保高效地构建大型索引。在构建标准索引时，除了在内存中缓存数据之外，使用$SortBegin/$SortEnd还可以使用IRISTEMP数据库中的空间。因此，在构建非常大的标准索引时，InterSystems
IRIS可能需要IRISTEMP中大致等于最终索引大小的空间。

注：构建索引的方法仅为使用InterSystems IRIS默认存储结构的类(表)提供。映射到遗留存储结构的类不支持索引构建，因为它假定遗留应用程序管理索引的创建。

使用管理门户构建索引

可以通过执行以下操作来构建表的现有索引(重建索引)：

1. 从管理门户中选择系统资源管理器，然后选择SQL。使用页面顶部的切换选项选择一个命名空间；这将显示可用命名空间的列表。选择命名空间后，选择屏幕左侧的Schema下拉列表。这将显示当前名称空间中的模式列表，其中带有布尔标志，指示是否有任何表或视图与每个模式相关联。
2. 从此列表中选择一个架构；该架构将显示在架构框中。它的正上方是一个下拉列表，允许选择属于该模式的表、系统表、视图、过程或所有这些。选择“ 表 ”或“ 全部 ”，然后打开“ 表 ”文件夹以列出此架构中的表。如果没有表，则打开文件夹将显示空白页。(如果未选择“ 表 ”或“ 全部 ”，则打开“ 表 ”文件夹将列出整个命名空间的表。)
3. 选择其中一个列出的表。这将显示表的目录详细信息。

- 要重建所有索引：单击操作下拉列表，然后选择重建表的索引。
- 要重建单个索引：单击索引按钮以显示现有索引。每个列出的索引都有重建索引的选项。

注意：当其他用户正在访问表的数据时，不要重建索引。要在活动系统上重建索引，请参阅在活动系统上构建索引。

以编程方式构建索引

为非活动表构建索引的首选方法是使用随表的Persistent类提供的%BuildIndices()(或%BuildIndicesAsync())方法。

若要以编程方式生成一个或多个索引，请使用%Library.Persistent.%BuildIndices()方法。

生成所有索引：调用%BuildIndices()，不带参数生成为给定类(表)定义的所有索引(为其提供值)：

```
SET sc = ##class(MyApp.SalesPerson).%BuildIndices()
IF sc=1 {
```

```
    WRITE !,"??????"
} ELSE {
WRITE !,"??????",!
    DO $System.Status.DisplayError(sc) QUIT
}
```

生成指定索引：调用%BuildIndices()，并将$LIST索引名作为第一个参数，为给定类(表)生成指定的已定义索引(为其提供值)：

```
SET sc = ##class(MyApp.SalesPerson).%BuildIndices($ListBuild("NameIDX","SSNKey"))
IF sc=1 {
    WRITE !,"??????"
} ELSE {
WRITE !,"??????",!
    DO $System.Status.DisplayError(sc) QUIT
}
```

生成除以下项之外的所有索引：调用%BuildIndices()，并将索引名称的$LIST作为第七个参数来构建(为其提供值)给定类(表)的所有已定义索引(指定索引除外)：

```
SET sc = ##class(MyApp.SalesPerson).%BuildIndices("",,,,,,$ListBuild("NameIDX","SSNKey"))
IF sc=1 {
    WRITE !,"??????"
} ELSE {
WRITE !,"??????",!
    DO $System.Status.DisplayError(sc) QUIT
}
```

%BuildIndices()方法执行以下操作：

1. 对要重建的任何(非位图)索引调用$SortBegin函数(这将启动对这些索引的高性能排序操作)。
2. 循环遍历类(表)的主要数据，收集索引使用的值，并将这些值添加到索引(通过适当的排序转换)。
3. 调用$SortEnd函数来完成索引排序过程。

如果索引已经有值，则必须使用两个参数调用%BuildIndices()，其中第二个参数的值为1。
为此参数指定1将导致该方法在重新生成值之前清除这些值。
例如:

```
SET sc = ##class(MyApp.SalesPerson).%BuildIndices(,1)
IF sc=1 {
    WRITE !,"??????"
} ELSE {
WRITE !,"??????",!
    DO $System.Status.DisplayError(sc) QUIT
}
```

清除并重建所有的索引。
你也可以清除并重建索引的子集，例如:

```
    SET sc = ##class(MyApp.SalesPerson).%BuildIndices($ListBuild("NameIDX","SSNKey"),
1)
    IF sc=1 {
        WRITE !,"??????"
    } ELSE {
    WRITE !,"??????",!
        DO $System.Status.DisplayError(sc) QUIT
    }
```

注意:当表的数据被其他用户访问时，不要重建索引。
若要在活动系统上重建索引，请参见在活动系统上构建索引。

在活动系统上构建索引

在活动系统上构建(或重建)索引时，有两个问题：
- 除非正在构建的索引对SELECT
Query隐藏，否则活动Query可能返回不正确的结果。这是在构建索引之前使用SetMapSelecability()方法处理的。
- 索引构建期间对数据的活动更新不会反映在索引条目中。这是通过在生成索引时使生成操作锁定单个行来处理的。

注意：如果应用程序在单个事务内对数据执行大量更新，则可能会出现锁表争用问题。

在Readonly主动系统上构建索引

如果表当前仅用于查询操作(READONLY)，则可以在不中断查询操作的情况下构建新索引或重建现有索引。这是通过在重建索引时使索引对查询优化器不可用来实现的。

如果要为其构建一个或多个索引的所有类当前都是READONLY，请使用 “ 在读写活动系统上构建索引 ” 中描述的相同系列操作，但有以下区别：使用%BuildIndices()时，设置pLockFlag=3(共享区锁定)。

在读写活动系统上构建索引

如果持久化类(表)当前正在使用并且可用于读写访问(查询和数据修改)，则可以在不中断这些操作的情况下构建新索引或重建现有索引。如果要为其重建一个或多个索引的类当前可读写访问，则构建索引的首选方法是使用与表的持久类一起提供的%BuildIndices()(或%BuildIndicesAsync())方法。

注意：以下信息适用于动态SQL查询，而不适用于嵌入式SQL。嵌入式SQL在编译时(而不是在运行时)检查MapSelecability设置。因此，关闭索引的MapSelecability对已经编译的嵌入式SQL查询没有任何影响。因此，嵌入式SQL查询仍可能尝试使用禁用的索引，并将给出不正确的结果。

在并发读写访问期间，需要执行以下一系列操作来构建一个或多个索引：

1. 望构建的索引对查询不可用(读取访问权限)。这是使用SetMapSelecability()完成的。这使得查询优化器无法使用该索引。在重建现有索引和创建新索引时都应执行此操作。例如：

```
  SET status=$SYSTEM.SQL.Util.SetMapSelectability("Sample.MyStudents","StudentNameIDX
",0)
```

- 第一个参数是Schema.Table名称，它是SqlTableName，而不是持久类名称。例如，默认模式是SQLUser，而不是User。该值区分大小写。
- 第二个参数是SQL索引映射名称。这通常是索引的名称，指的是磁盘上存储索引的名称。对于新索引，这是在创建索引时将使用的名称。该值不区分大小写。
- 第三个参数是MapSelecability标志，其中0将索引映射定义为不可选择(OFF)，1将索引映射定义为可选择(ON)。指定0。

可以通过调用GetMapSelecability()方法来确定索引是否不可选。如果已将索引显式标记为不可选，则此方法返回0。在所有其他情况下，它返回1；它不执行表或索引是否存在的验证检查。请注意，Schema.Table名称是SqlTableName，并且区分大小写。

SetMapSelecability()和GetMapSelecability()仅适用于当前命名空间中的索引映射。如果该表映射到多个命名空间，并且需要在每个命名空间中构建索引，则应该在每个命名空间中调用SetMapSelecability()。

2. 在索引构建期间建立并发操作：

- 对于新索引：在类中创建索引定义(或在类的%Storage.SQL中创建新的SQL Index Map规范)。编译类。此时，索引存在于表定义中；这意味着对象保存、SQL INSERT操作和SQL UPDATE操作都记录在索引中。但是，由于在步骤1中调用了SetMapSelecability()，因此不会为任何数据检索选择此索引映射。SetMapSelecability()阻止查询使用区索引，但是数据映射将被投影到SQL以使用索引全局和数据全局。对于新索引，这是合适的，因为索引尚未填充。在对表运行查询之前，需要填充区索引。
- 对于现有索引：清除任何引用该表的缓存查询。索引构建执行的第一个操作是终止索引。因此，在重新生成索引时，不能依赖任何经过优化以使用该索引的代码。

3. 使用pLockFlag=2(行级锁定)的持久化类(表)的%BuildIndices()方法构建一个或多个索引。PLockFlag=2标志在重建过程中在单个行上建立独占写锁，以便并发数据修改操作与构建索引操作相协调。

默认情况下，%BuildIndices()构建为持久类定义的所有索引；可以使用pIgnoreIndexList从重建中排除索引。

默认情况下，%BuildIndices()为所有ID构建索引项。但是，可以使用pStartID和pEndID来定义ID范围。%BuildIndices()将仅为该范围内(含)的ID构建索引项。例如，如果使用带有%NOINDEX限制的INSERT将一系列新记录添加到表中，则可以稍后使用具有ID范围的%BuildIndices()为这些新记录构建索引项。还可以使用pStartID和pEndID在节中构建极大的索引。

%BuildIndices()返回%Status值。如果%BuildIndices()因检索数据时出现问题而失败，系统将生成一个SQLCODE错误和一条消息(%msg)，其中包含遇到错误的%ROWID。

4. 构建完索引后，启用映射以供查询优化器选择。将第三个参数MapSelecability标志设置为1，如下例所示：

```
  SET status=$SYSTEM.SQL.Util.SetMapSelectability("Sample.MyStudents","StudentNameIDX",1)
```

5. 再次清除引用该表的所有缓存查询。这将消除在此程序中创建的缓存查询，这些查询无法使用索引，因此不如使用索引的相同查询最佳。

这就完成了这个过程。索引已完全填充，查询优化器能够考虑该索引。

注意：%BuildIndices()只能用于重建ID值为正整数的表的索引。如果父表具有正整数ID值，还可以使用%BuildIndices()重建子表中的索引。对于其他表，请使用%ValidateIndices()方法，如验证索引中所述。因为%ValidateIndices()是构建索引的最慢方法，所以只有在没有其他选项的情况下才应该使用它。

#SQL #Caché

源URL: https://cn.community.intersystems.com/post/%E7%AC%AC%E4%BA%8C%E7%AB%A0-%E5%AE%9A%E4%B9%89%E5%92%8C%E6%9E%84%E5%BB%BA%E7%B4%A2%E5%BC%95%EF%BC%88%E5%9B%9B%EF%BC%89