技术概要：使用 InterSystems 产品优化 SQL 性能

版本 2021.1
2021-07-21

InterSystems 公司 马萨诸塞州剑桥纪念大道 1 号 邮编 02142 www.intersystems.com

技术概要：使用 InterSystems 产品优化 SQL 性能

InterSystems IRIS 数据平台 版本 2021.1 2021-07-21

有关 InterSystems 产品的技术支持问题，请联系：

InterSystems 全球响应中心（WRC）
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

目录

技术概要：使用 InterSystems 产品优化 SQL 性能

本技术概要（First Look）指南向您介绍了 InterSystems SQL 查询优化，包括查询分析工具的使用，几种索引方法以及随着时间的变化查看运行时统计数据的能力。

要浏览所有的技术概要（First Look），包括其他可以在免费的云实例或 web 实例上执行的技术概要（First Look），请参见 InterSystems First Looks（《InterSystems 技术概要》）。

1 使用 InterSystems SQL 优化查询

InterSystems IRIS®数据平台为 SQL 查询性能调整提供全套工具：

- 查询计划分析的图形化显示
- 如位图和位片索引（bitslice index）等索引策略结构紧凑，可由矢量化 CPU 指令有效处理。每种索引类型都为某些查询类型（如逻辑条件、计数和聚合函数）提供了好处。通过索引，您可以在一个核心上实现每秒多达数十亿行的查询性能结果。
- 随时间变化的 SQL 查询性能指标

重要提示： 下面演示中显示的查询性能数字代表了在一台 Windows 10 笔记本电脑上进行的多次演示试验。根据您的环境，您可能会看到不同的查询性能数字。

想查看 InterSystems IRIS SQL 功能快速演示吗？请查看 SQL QuickStart（SQL 快速入门）！

2 演示：显示并解释优化前的查询计划

2.1 用前须知

阅读并完成 *First Look:InterSystems SQL（《技术概要：InterSystems SQL》）*后，您将获得最佳体验。在这里，您将再次使用 InterSystems IRIS SQL Shell；您将使用的数据来自您在技术概要（First Look）的演示时创建的百万记录股票交易数据表。

您还将运行 `TuneTableutility`，它检查表中的数据并创建 *InterSystems SQL 查询优化器*（决定如何最好地运行任何查询的引擎）使用的统计数据。这些统计数据包括表的大小（区段大小）和每列唯一值的数量（选择性）。优化器在决定连接顺序等情况下使用表大小，其中最好从较小的表开始。在表有多个索引的情况下，选择性有助于优化器选择最佳索引。在产品实例中，您通常只运行一次 `TuneTable`：在数据被加载到表中之后和运行之前。

First Look:InterSystems SQL （《技术概要：InterSystems SQL》）解释了如何执行在技术概要（First Look）和此处运行演示所需的以下步骤：

- 选择一个 InterSystems IRIS 实例。您的选择包括多种类型的已授权的和免费的评估实例；关于如何部署每种类型的信息，请参见 *InterSystems IRIS Basics: Connecting an IDE*（《*InterSystems IRIS 基础：连接一个 IDE*》）中的 Deploying InterSystems IRIS（部署 InterSystems IRIS）。
- 打开 InterSystems 终端（Terminal）（简称终端（Terminal））来运行 SQL Shell。
- 从 GitHub repo https://github.com/intersystems/FirstLook-SQLBasics 获取本指南的实用程序文件，包括
  - stock_table_demo_two.csv，其中包含一百万行股票表数据
  - Loader.xml，是一个包含实用程序方法的类文件，用于将 Stock_table_demo_two.csv 中的数据加载到 InterSystems IRIS 表中。

2.2 运行 TuneTable 实用程序

如果您的 InterSystems IRIS 实例不再包含 **StockTableDemoTwotable**，请按照 Demo：Using Bitmap Indexing To Maximize Query Performance（《演示：使用位图索引最大化查询性能》）（在执行 SELECT DISTINCT 查询之前停止）的前四个步骤重新创建并加载它。

在 SQL Shell 中，在 **FirstLook.StockTableDemoTwo** 上运行 TuneTable 实用程序，如下所示：

```
OBJ DO $SYSTEM.SQL.TuneTable("FirstLook.StockTableDemoTwo")
```

该命令在 SQL Shell 中不会生成可见的输出。

2.3 使用 EXPLAIN 关键字显示查询计划

这个演示假设您想获得所有"SELL"交易的平均价格。鉴于该表包含一百万条记录，所需的查询有可能非常慢。

虽然您可能已经想在 Price 和 TransactionType 字段上创建索引，但在开始优化工作之前，查看查询计划将具有指导意义。在 SQL Shell 中，您可以通过在查询前添加 EXPLAIN 关键字来显示查询计划。查询计划显示 SQL 查询优化器将如何使用索引（如果有索引的话），或者是否将直接读取表数据来执行语句。

要使用 EXPLAIN 关键字来显示查询计划，在 SQL Shell 中执行以下语句：

```
EXPLAIN SELECT AVG(Price) As AveragePrice FROM FirstLook.StockTableDemoTwo
     WHERE TransactionType = 'SELL'
```

这将返回格式为 XML 的查询计划：

```
Plan
"<plans>
 <plan>
 <sql>
  SELECT AVG ( Price ) AS AveragePrice FROM FirstLook . StockTableDemoTwo WHERE TransactionType = ?
  /*#OPTIONS {""DynamicSQLTypeList"":""1""} */
 </sql>
 <cost value=""1827000""/>
 Call module B.
 Output the row.
 <module name=""B"" top=""1"">
 Process query in parallel, partitioning master map FirstLook.StockTableDemoTwo.IDKEY into
     subranges of T1.ID values, piping results to temp-file A:
     SELECT count(T1.Price),sum(T1.Price) FROM %NOPARALLEL FirstLook.StockTableDemoTwo T1
     where ((%SQLUPPER(T1.TransactionType) = %SQLUPPER(?)))
 Read temp-file A, looping on a counter.
 For each row:
     Accumulate the count([value]).
     Accumulate the sum([value]).
 </module>
```

```
 </plan>
 <plan>
 <sql>
  SELECT COUNT ( T1 . Price ) , SUM ( T1 . Price ) FROM %NOPARALLEL FirstLook . StockTableDemoTwo T1
  WHERE ( ( %SQLUPPER ( T1 . TransactionType ) = %SQLUPPER ( ? ) ) ) %PARTITION BY T1 . ID > ? AND T1
 . ID <= ?
 </sql>
 <cost value=""1827000""/>
 Call module B.
 Output the row.
 <module name=""B"" top=""1"">
 Read master map FirstLook.StockTableDemoTwo.IDKEY, looping on ID (with a range condition).
 For each row:
     Accumulate the count(Price).
     Accumulate the sum(Price).
 </module>
 </plan>
</plans>"
```

您将看到为执行 SQL 查询而生成的查询计划可以分为多个*模块*，每个模块都执行计划的不同部分，例如评估子查询。

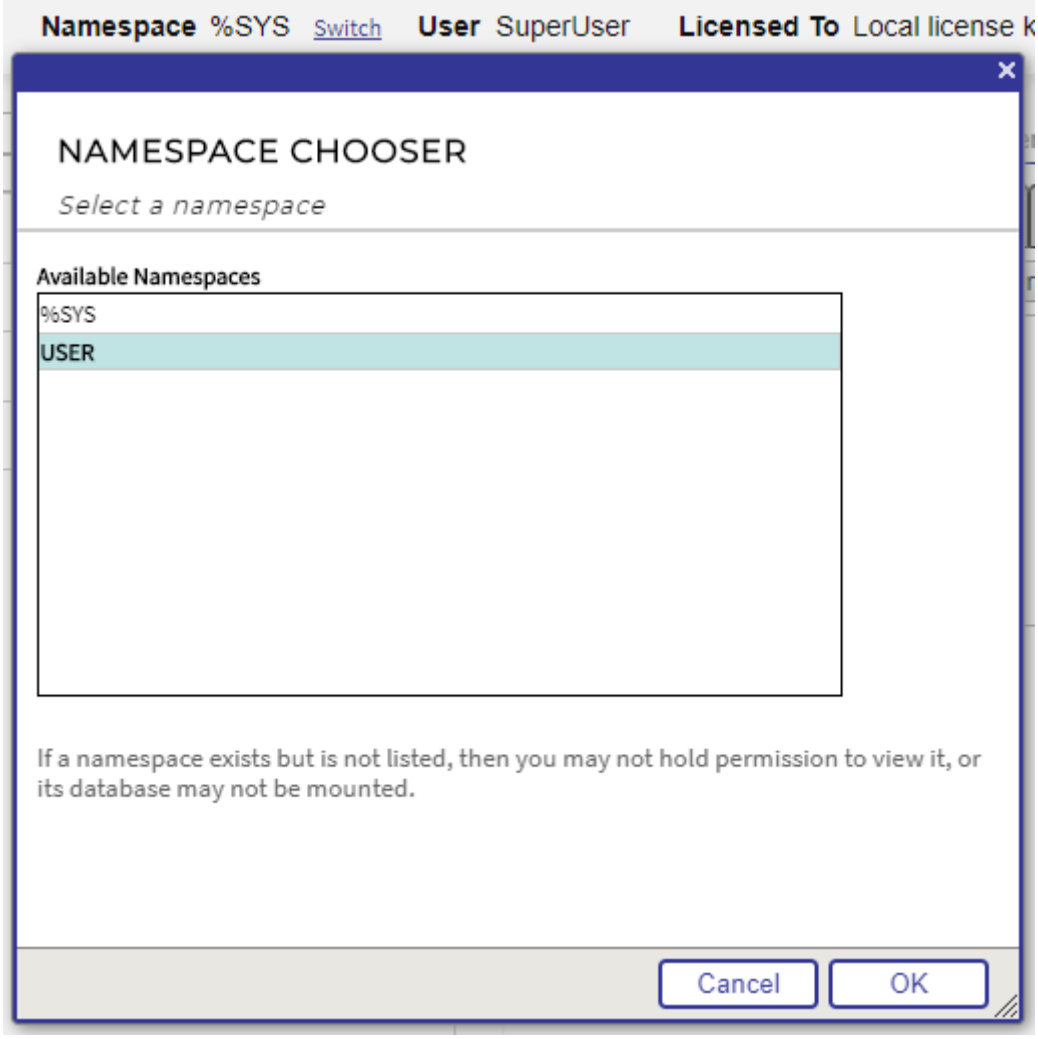
实际上，这个查询计划被分为两个独立的计划。顶部计划用于初始查询。它调用一个模块 B，在这个模块中，其中"主映射"被分区，并在每个分区上并行执行子查询。子查询的计划遵循初始查询的计划。

在 "Spotting Potential Performance Issues in Query Plan Results（在查询计划结果中发现潜在的性能问题）"中，您将学习如何识别这个查询的问题。

2.4 在管理门户（Management Portal）中使用 SQL 查询接口显示查询计划

InterSystems IRIS 在管理门户（Management Portal）中提供了一个基于 web 的接口，用于 SQL 查询执行和计划分析。要使用管理门户（Management Portal）中的SQL 查询接口显示查询计划：

1. 使用 *InterSystems IRIS Basics:Connecting an IDE（《InterSystems IRIS 基础：连接一个IDE》）*中URL described for your instance（为您的实例描述的URL），在浏览器中打开您的实例的管理门户（Management Portal）。
2. 请确保您是在 **USER** 命名空间。如果您还没有找到，可以：
   - 在屏幕的顶部面板中，点击当前命名空间名称右侧的 **SWITCH（交换）**。
   - 在弹出的窗口中，选择**USER（用户）**并点击**OK（确定）**。



3. 导航到 SQL 页面(**System Explorer（系统资源管理器）> SQL**)。
4. 省略 EXPLAIN 关键字，将"Using the EXPLAIN Keyword to Show a Query Plan（使用 EXPLAIN 关键字显示查询计划）中的查询粘贴到**Execute Query（执行查询）**标签的文本字段中。
5. 点击 **Show Plan（显示计划）**来显示查询计划。 结果将如下所示：

Statement Text

```
SELECT AVG ( Price ) AS AveragePrice FROM FirstLook . StockTableDemoTwo WHERE TransactionType = ?
/*#OPTIONS {"DynamicSQLTypeList":"1"} */
```

Query Plan

Relative Cost = 1827000

- Call module B.
- Output the row.

Module: B

- Process query in parallel, partitioning master map FirstLook.StockTableDemoTwo.IDKEY into subranges of T1.ID values, piping results to temp-file A:
  - SELECT count(T1.Price),sum(T1.Price) FROM %NOPARALLEL FirstLook.StockTableDemoTwo T1 where ((%SQLUPPER(T1.TransactionType) = %SQLUPPER(?)))
- Read temp-file A, looping on a counter.
- For each row:
  - Accumulate the count([value]).
  - Accumulate the sum([value]).

Statement Text

```
SELECT COUNT ( T1 . Price ) , SUM ( T1 . Price ) FROM %NOPARALLEL FirstLook . StockTableDemoTwo T1
WHERE ( ( %SQLUPPER ( T1 . TransactionType ) = %SQLUPPER ( ? ) ) ) %PARTITION BY T1 . ID > ? AND T1 .
ID <= ?
```

Query Plan

Relative Cost = 1827000

- Call module B.
- Output the row.

Module: B

- Read master map FirstLook.StockTableDemoTwo.IDKEY, looping on ID (with a range condition).
- For each row:
  - Accumulate the count(Price).
  - Accumulate the sum(Price).

解释这些结果是下一节的主题。

2.5 发现查询计划结果中潜在的性能问题

查看查询计划结果，您可以看到这个查询存在一些严重的潜在性能问题。如果您查看子查询的计划，也就是实际工作完成的地方，您可以看到第一个任务是"读取主映射"。这意味着InterSystems SQL 查询优化器不会使用任何索引来运行查询；

相反，查询将遍历表中的所有 ID。特别是在大表的情况下，这表明查询将不能很好地执行。

当您优化查询时，您会看到它的执行时间减少，查询计划也会有明显的变化。

注意： 相对成本可以很好地预测性能，但只是相对于某个特定的查询而言。如果您在表中添加了一个索引，并且看到相对成本下降了，那么很可能这个查询现在会运行得更快。然而，相对成本并不是为了比较两个不同查询的性能。

2.6 测试查询执行

要获取有关未优化查询将如何执行的一些实际数据，请在 SQL Shell 中运行它：

```
SELECT AVG(Price) As AveragePrice FROM FirstLook.StockTableDemoTwo
   WHERE TransactionType = 'SELL'
   GO
```

输出将如下所示:

```
AveragePrice
266.1595139195757844

1 Rows(s) Affected
statement prepare time(s)/globals/cmds/disk: 0.0009s/6/1246/0ms
          execute time(s)/globals/cmds/disk: 0.2599s/1000075/8502571/0ms
                          cached query class: %sqlcq.USER.cls5
```

语句准备和执行指标分别列出。特别注意以下两项：

- 执行时间为 0.2599 秒。虽然这个时间看起来并不算长，但可以通过使用索引大大改善。
- 在执行步骤中读取的 globals 数量为 1,000,075。(Globals 是 InterSystems IRIS 用来存储数据的多维稀疏数组（multidimensional sparse arrays）；更多信息，请参见 *Introduction to InterSystems IRIS Programming*（《*InterSystems IRIS 编程简介*》）中的 Introduction to Globals（Globals 简介）一章。为了提高查询性能，应该减少这个数字。您会在下一节看到这种情况。

重要提示： 准备工作只需一次：第一次重新规划查询。如果修改了相关表或添加或删除了索引，查询将自动重新规划。大多数应用程序只准备一次查询，但会多次执行。因此，在这个演示中，我们的重点将是 *调整执行性能*。

3 演示：测试查询优化

3.1 向价格字段添加位片索引（Bitslice Index）

如果您的查询将包含一个或多个字段上的聚合函数，那么为这些字段中的一个或多个字段添加位片索引（bitslice index）可能会提高性能。

位片索引（bitslice index）将字段中的每个数字数据值表示为二进制位字符串，二进制值中的每个数字都有一个位图，以记录哪些行的二进制数字为 1。

因为我们想要获得所有"SELL"交易的平均价格，所以在 `Price` 字段中添加一个位片索引（bitslice index）是有意义的。要在 `Price` 字段上创建位片索引（bitslice index） `PriceIdx`，请在 SQL Shell 中执行以下语句：

```
CREATE BITSLICE INDEX PriceIdx ON TABLE FirstLook.StockTableDemoTwo (Price)

9.      CREATE BITSLICE INDEX PriceIdx ON TABLE FirstLook.StockTableDemoTwo (Price)

0 Rows Affected
statement prepare time(s)/globals/cmds/disk: 0.0091s/2000/13151/0ms
          execute time(s)/globals/cmds/disk: 1.4268s/2087789/55765062/1ms
                          cached query class: %sqlcq.USER.cls7
```

但是，正如您将在下面看到的，仅仅因为您创建了索引并不一定意味着 InterSystems SQL 查询优化器会使用它 。

3.2 测试位片索引（Bitslice Index）的效果

要查看新的位片索引（bitslice index）是否会对查询的执行方式或运行速度产生任何影响，请使用上述任何一种方法（SQL Shell 或管理门户（Management Portal））来显示查询计划。

正如您将看到的，查询计划仍然与以前相同。InterSystems SQL 查询优化器将不会使用新的索引。

Statement Text

```
SELECT AVG ( Price ) AS AveragePrice FROM FirstLook . StockTableDemoTwo WHERE TransactionType = ?
/*#OPTIONS {"DynamicSQLTypeList":"1"} */
```

Query Plan

Relative Cost = 1827000

- Call module B.
- Output the row.

Module: B

- Process query in parallel, partitioning master map FirstLook.StockTableDemoTwo.IDKEY into subranges of T1.ID values, piping results to temp-file A:
  - SELECT count(T1.Price),sum(T1.Price) FROM %NOPARALLEL FirstLook.StockTableDemoTwo T1 where ((%SQLUPPER(T1.TransactionType) = %SQLUPPER(?)))
- Read temp-file A, looping on a counter.
- For each row:
  - Accumulate the count([value]).
  - Accumulate the sum([value]).

Statement Text

```
SELECT COUNT ( T1 . Price ) , SUM ( T1 . Price ) FROM %NOPARALLEL FirstLook . StockTableDemoTwo T1
WHERE ( ( %SQLUPPER ( T1 . TransactionType ) = %SQLUPPER ( ? ) ) ) %PARTITION BY T1 . ID > ? AND T1 .
ID <= ?
```

Query Plan

Relative Cost = 1827000

- Call module B.
- Output the row.

Module: B

- Read master map FirstLook.StockTableDemoTwo.IDKEY, looping on ID (with a range condition).
- For each row:
  - Accumulate the count(Price).
  - Accumulate the sum(Price).

运行查询产生的性能统计数据与您创建位片索引（bitslice index）之前几乎相同（0.2559 秒的执行时间与 0.2599 秒相比）。InterSystems IRIS 智能地缓存查询计划和数据，因此同一查询的后续运行可能会提高性能，鉴于查询性能时间略有不同，这里可能就是这种情况。 在机器上运行的其他应用程序也会影响性能。

```
        SELECT AVG(Price) As AveragePrice FROM FirstLook.StockTableDemoTwo
        WHERE TransactionType = 'SELL'
        GO

10.     SELECT AVG(Price) As AveragePrice FROM FirstLook.StockTableDemoTwo
        WHERE TransactionType = 'SELL'

AveragePrice
266.1595139195757844

1 Rows(s) Affected

statement prepare time(s)/globals/cmds/disk: 0.0569s/35431/227191/0ms
          execute time(s)/globals/cmds/disk: 0.2559s/1000075/8502571/0ms
                          cached query class: %sqlcq.USER.cls8
```

如果从查询中删除WHERE 子句，当您显示查询计划时，将会看到完全不同的结果：

Statement Text

```
SELECT AVG ( Price ) AS AveragePrice FROM FirstLook . StockTableDemoTwo
```

Query Plan

Relative Cost = 16362

- Call module D once.
- Call module G once.
- Output the row.

Module: D

- Read bitslice index FirstLook.StockTableDemoTwo.PriceIdx, looping on bitslice power.
- For each bitslice power:
  Call module E.
  - Accumulate the sum(Price).

Module: G

- Read extent bitmap FirstLook.StockTableDemoTwo.%%DDLBEIndex, looping on bitmap chunks.
- For each bitmap chunk:
  - Read a chunk from bitslice index FirstLook.StockTableDemoTwo.PriceIdx and perform a bitwise AND NOT operation.
  - Accumulate the count(bits).

Module: E

- Read bitslice index FirstLook.StockTableDemoTwo.PriceIdx, using the given bitslice power, and looping on bitslice chunks.
- For each bitslice chunk:
  - Accumulate the aggregate for the power.

如您所见，读取位片索引（bitslice index）是查询计划的第一步。在这个计划中没有读取到"总映射"。

SQL 查询优化器也使用第二个索引 FirstLook.StockTableDemoTwo.$StockTableDemoTwo。这是*位图范围索引*， 在执行 CREATE TABLESQL 语句时自动创建。它是表中所有行的位图索引（bitmap index），而不仅仅是一个字段，每个位的值反映了该行是否实际存在。

然而，我们真正想要运行的查询包含一个WHERE 子句。因此，我们必须找到一种方法，让SQL 查询优化器在WHERE 子句存在时使用索引。

3.3 向交易类型字段（TransactionType Field）添加位图索引（Bitmap Index）

如果您阅读了InterSystems SQL Optimization Guide（《InterSystems SQL 优化指南》），您会发现InterSystems SQL 查询优化器在与WHERE 子句字段上的位图索引（bitmap index）结合使用时，通常会使用位片索引（bitslice index）。

这是因为没有WHERE子句的聚合查询可以简单地聚合索引中的所有数据。然而，为了只聚合满足WHERE条件的行，查询必须*掩码*那些不满足条件的行的位片索引（bitslice index）的位。在WHERE子句中的字段上有一个位图索引（bitmap index），允许有效地构建这个掩码。

幸运的是，查询中的另一个字段TransactionType是位图索引（bitmap index）的一个很好的候选者，因为它的可能值计数是两个（"SELL"和"BUY"）。

要向TransactionType 字段添加位图索引（bitmap index），请在SQL Shell中执行以下语句：

```
CREATE BITMAP INDEX TransactionTypeIdx ON TABLE FirstLook.StockTableDemoTwo
        (TransactionType)

11.     CREATE BITMAP INDEX TransactionTypeIdx ON TABLE FirstLook.StockTableDemoTwo
        (TransactionType)

0 Rows Affected
statement prepare time(s)/globals/cmds/disk: 0.0069s/2001/13291/0ms
          execute time(s)/globals/cmds/disk: 1.1046s/2088960/19771584/0ms
                        cached query class: %sqlcq.USER.cls7
```

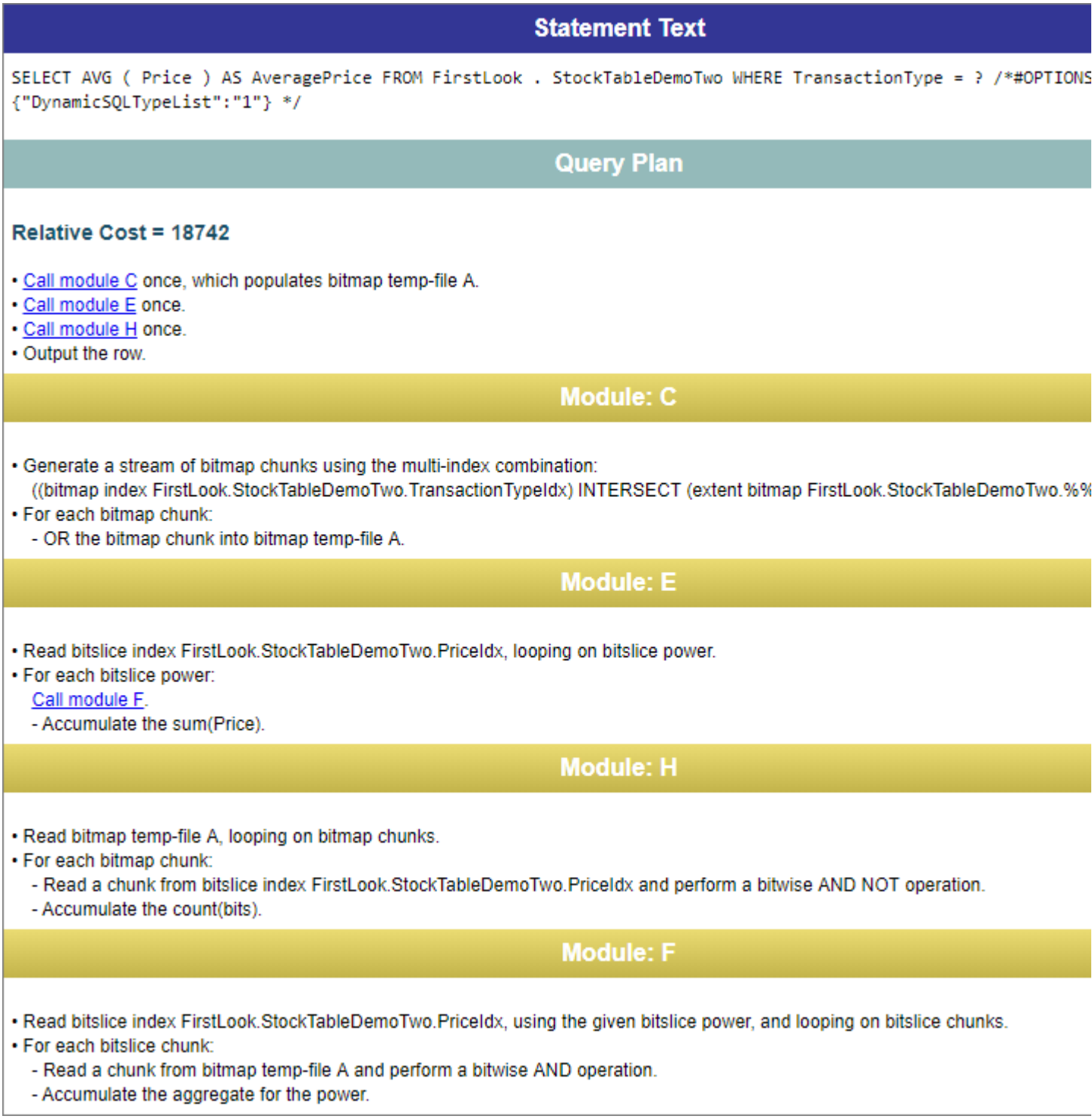
3.4 重新测试查询性能

现在，您已经添加了位片索引（bitslice index）和位图索引（bitmap index）：如果您显示的查询计划

```
SELECT AVG(Price) as AveragePrice FROM FirstLook.StockTableDemoTwo
     WHERE TransactionType = 'SELL'
```

在SQL Shell或管理门户（Management Portal）中，您会看到查询优化器使用您创建的两个索引来获得最佳性能。

还请注意，18742的相对成本只是未优化查询的一小部分，其成本为1827000。



Statement Text

```
SELECT AVG ( Price ) AS AveragePrice FROM FirstLook . StockTableDemoTwo WHERE TransactionType = ? /*#OPTIONS
{"DynamicSQLTypeList":"1"} */
```

Query Plan

Relative Cost = 18742

- Call module C once, which populates bitmap temp-file A.
- Call module E once.
- Call module H once.
- Output the row.

Module: C

- Generate a stream of bitmap chunks using the multi-index combination:
  ((bitmap index FirstLook.StockTableDemoTwo.TransactionTypeIdx) INTERSECT (extent bitmap FirstLook.StockTableDemoTwo.%%
- For each bitmap chunk:
  - OR the bitmap chunk into bitmap temp-file A.

Module: E

- Read bitslice index FirstLook.StockTableDemoTwo.PriceIdx, looping on bitslice power.
- For each bitslice power:
  Call module F.
  - Accumulate the sum(Price).

Module: H

- Read bitmap temp-file A, looping on bitmap chunks.
- For each bitmap chunk:
  - Read a chunk from bitslice index FirstLook.StockTableDemoTwo.PriceIdx and perform a bitwise AND NOT operation.
  - Accumulate the count(bits).

Module: F

- Read bitslice index FirstLook.StockTableDemoTwo.PriceIdx, using the given bitslice power, and looping on bitslice chunks.
- For each bitslice chunk:
  - Read a chunk from bitmap temp-file A and perform a bitwise AND operation.
  - Accumulate the aggregate for the power.

最后，如果您在 SQL Shell 中运行这个查询，您会看到对 globals 的更有效使用（594 而不是 1000075）。

最关键的是，有索引的查询运行速度比无索引的查询快了近 85 倍（执行时间为 0.0031 秒，而非 0.2599）。

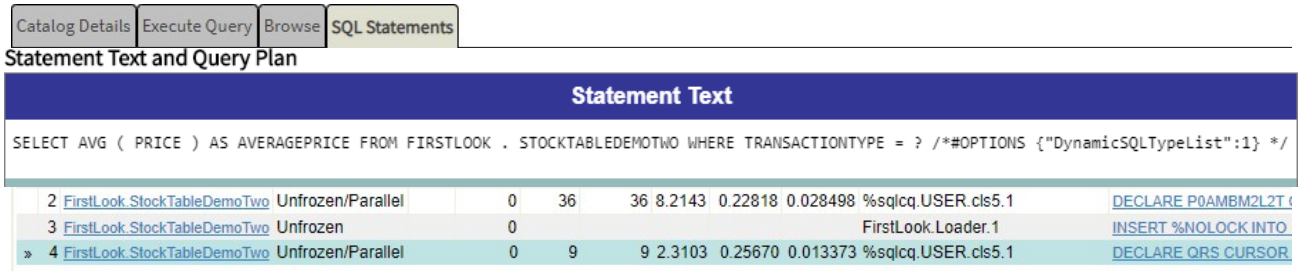
```
        SELECT AVG(Price) As AveragePrice FROM FirstLook.StockTableDemoTwo
        WHERE TransactionType = 'SELL'
        GO

12.     SELECT AVG(Price) As AveragePrice FROM FirstLook.StockTableDemoTwo WHERE TransactionType =
'SELL'

AveragePrice
266.1595139195757844

1 Rows(s) Affected
statement prepare time(s)/globals/cmds/disk: 0.0554s/34877/186130/0ms
          execute time(s)/globals/cmds/disk: 0.0031s/594/2878/0ms
                        cached query class: %sqlcq.USER.cls8
```

为了随时间跟踪查询的性能，InterSystems IRIS 提供了查询统计数据，您将在下一节了解如何查看这些数据。

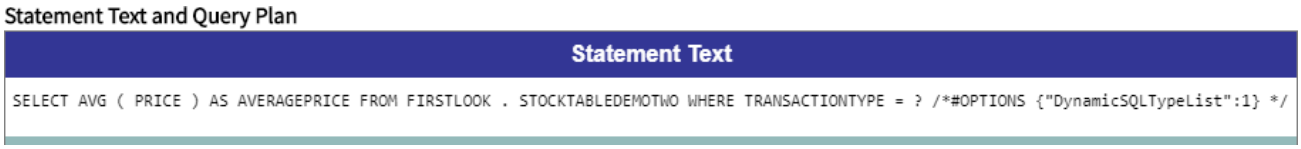
4 查看随时间变化的查询性能

要跟踪运行缓慢的查询或查看新查询在产品中的运行情况，您可以使用管理门户（Management Portal）中的 **SQL Statements（SQL 语句）**视图。要导航到这个视图，在管理门户（Management Portal）中打开 SQL 查询接口，并点击 **SQL Statements（SQL语句）**。

例如，如果您上面调整的查询在其原始（未优化的）计划下运行了 9 次，您可能会看到类似这样的结果：

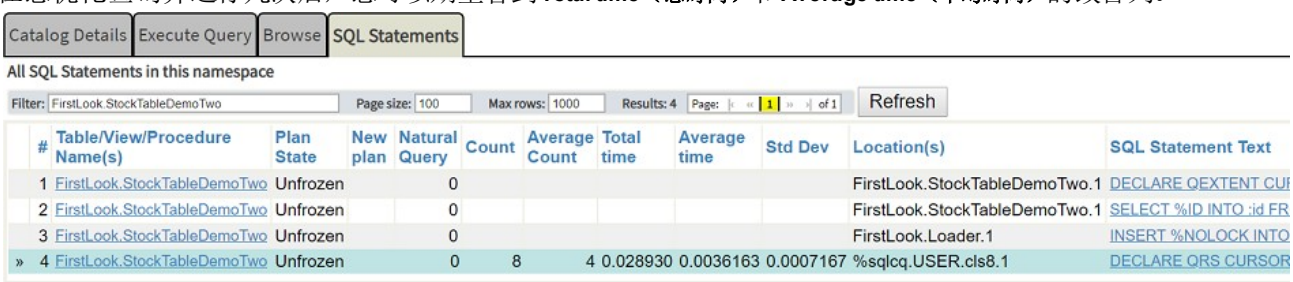


点击 **SQL Statement Text column（SQL 语句文本列）**中的语句链接， 允许您以SQL 形式查看查询：



您还可以使用缓存查询类的名称将 SQL 语句执行与 **SQL Statements（SQL 语句）**视图联系起来，该名称是SQL Shell 中输出的最后一行，并列在 **SQL Statements（SQL 语句）** 的 **Location(s)（位置）**列中。

在您优化查询并运行几次后，您可以期望看到**Total time（总时间）**和**Average time（平均时间）**的改善列。



请注意，**Count（计数）**的值已经下降。这是因为位图和位片索引（bitslice index）的添加引起了查询计划的改变，这反过来又触发了对相关类的缓存查询的删除。该查询在新查询计划下总共运行了 8 次，平均每天 4 次。

5 了解有关InterSystems SQL 的更多信息

要了解更多有关 SQL 和 InterSystems IRIS 的信息，请参见：

5.1 介绍材料

- First Look: InterSystems SQL（《技术概要：InterSystems SQL》）
- Using InterSystems SQL（《使用 InterSystems SQL》）
- InterSystems SQL Reference（《InterSystems SQL 参考书目》）
- InterSystems SQL Overview（《InterSystems SQL 概述》）

5.2 SQL 开发

- SQL – Things You Should Know（《SQL – 您应该知道的事情》）
- Developing with InterSystems Objects and SQL（《使用 InterSystems Objects 和 SQL 开发》）

5.3 查询优化

- InterSystems SQL Optimization Guide（《InterSystems SQL 优化指南》）
- Optimizing SQL Queries（《优化 SQL 查询》）

5.4 分片和可扩展性

- First Look: Scaling for Data Volume with Sharding（《技术概要：带分片的数据卷扩展》）
- Scalability Guide（《可扩展性指南》）

5.5 SQL 搜索

- First Look: SQL Search with InterSystems IRIS（《技术概要：使用 InterSystems IRIS 进行 SQL 搜索》）
- Using InterSystems SQL Search（《使用 InterSystems SQL 搜索》）
- Creating iFind Indices for Searching Text Fields（《创建用于搜索文本字段的 iFind 索引》）

5.6 JDBC

- First Look: JDBC and InterSystems IRIS (《技术概要：JDBC 和 InterSystems IRIS》)
- Using Java JDBC with InterSystems IRIS（《在 InterSystems IRIS 中使用 Java JDBC》）<文档>
- Java Overview (《Java 概述》)

- Using JDBC with InterSystems IRIS（《在 InterSystems IRIS 中使用 JDBC》）<在线学习>