文章
Jingwei Wang · 七月 29, 2022 阅读大约需 33 分钟

InterSystems SQL 的优化 - 第一部分 - 定义和构建索引

什么时候使用索引

索引提供了一种机制，通过维护常用数据的分类子集来优化查询。确定哪些字段应该被编入索引需要一些思考：太少或错误的索引，关键查询会运行得太慢；太多的索引会减慢INSERT和UPDATE的性能（因为索引值必须被设置或更新）。

索引什么

为了确定添加索引是否能提高查询性能，从管理门户的SQL界面运行查询，并在Performance中注意global引用的数量。添加索引，然后重新运行查询，注意 global引用的数量。一个有用的索引应该减少global引用的数量。你可以通过使用%NOINDEX关键字作为WHERE子句或ON子句条件的前言来阻止索引的使用。

你应该对JOIN中指定的字段（属性）进行索引。例如，LEFT OUTER JOIN从左表开始，然后查看右表，因此，你应该对右表的字段进行索引。在下面的例子中，你应该为T2.f2编制索引。一个INNER JOIN应该在两个ON子句字段上都有索引。

```
FROM Table1 AS T1 LEFT OUTER JOIN Table2 AS T2 ON T1.f1 = T2.f2
```

如果查询计划中的第一个项目是 "read master map"，或者查询计划调用的模块的第一个项目是 "read master map"，那么查询的第一个map就是master map而不是索引map。因为主图读取的是数据本身，而不是数据的索引，这几乎总是表明一个低效的查询计划。除非该表相对较小，否则你应该创建一个索引，以便当你重新运行这个查询时，查询计划的第一张map显示 "read index map"。

你应该为WHERE子句中指定的相等条件的字段建立索引。

你可能希望为WHERE子句范围条件中指定的字段，以及GROUP BY和ORDER BY子句中指定的字段建立索引。

在某些情况下，基于范围条件的索引会使查询变慢。如果绝大多数的记录都满足指定的范围条件，就会出现这种情况。例如，如果查询子句WHERE Date < CURRENTDATE被用于一个数据库，其中大部分记录来自以前的日期，在Date上建立索引实际上会降低查询速度。这是因为查询优化器假定范围条件将返回相对较少的行，并针对这种情况进行优化。你可以通过在范围条件前加上%NOINDEX来确定是否发生了这种情况，然后再次运行查询。

如果你使用一个索引字段进行比较，那么在比较中指定的字段应该具有和它在相应索引中相同的collation类型。例如，在SELECT的WHERE子句或JOIN的ON子句中的Name字段应该具有与为Name字段定义的索引相同的排序方式。如果字段的排序和索引的排序不匹配，那么索引的有效性就会降低，或者根本就不能使用。

在map中存储的表

一个SQL表被存储为一组map。每个表都有一个主图，包含了表中的所有数据；表还可能有其他图，如索引图和位图。每个map可以被想象成一个多维global，一些字段的数据在一个或多个子标中，其余字段存储在节点值中。这些子标控制着哪些数据被访问。

对于主映射，RowID或IDKEY字段通常被用作映射子标。

对于索引map，通常使用其他字段作为前导下标，RowID/IDKEY字段作为额外的低级下标。

对于一个位图，位图层可以被认为是一个额外的RowID下标层。然而，位图只能用于正整数的RowIDs。

Master Map

系统为每个表自动定义了一个主图（Data/Master）。master map不是一个索引，它是一个使用其map下标字段直接访问数据本身的map。默认情况下，主图的下标字段是系统定义的RowID字段。默认情况下，这种使用RowID字段的直接数据访问是用SQL map名称（SQL索引名称）IDKEY表示的。

默认情况下，一个用户定义的主键不是IDKEY。这是因为使用RowID整数的Master Map查询几乎总是比通过主键值查询更有效率。然而，如果你指定主键是IDKEY，主键索引就会被定义为表的主映射，SQL Map Name是主键SQL索引的名称。

对于一个单字段的主键/IDKEY，主键索引是Master Map，但是Master Map的数据访问列仍然是RowID。这是因为记录的唯一主键字段值和它的RowID值之间存在一对一的匹配，而且RowID被认为是更有效的查询。对于多字段主键/IDKEY，主映射被赋予主键索引名称，主映射的数据访问列是主键字段。

选择一个索引类型

下面是在位图索引和标准索引之间进行选择的一般指导原则：

- 一般来说，可以对这些类型使用标准索引：主键、外键、唯一键、关系、简单的对象引用。
- 否则，位图索引通常是比较好的（假设该表使用系统分配的数字ID号）。

其他因素：每个属性上的独立位图索引通常比多个属性上的位图索引有更好的性能。这是因为SQL引擎可以使用AND和OR操作有效地组合独立的位图索引。

如果一个属性（或一组你真正需要一起索引的属性）有超过10,000-20,000个不同的值（或值组合），请考虑标准索引。然而，如果这些值的分布非常不均匀，以至于少量的值占了相当一部分行，那么位图索引可能会好得多。一般来说，我们的目标是减少索引所需的总体大小。

定义和建立索引

以下是建立索引的原则：

- 你可以在表中的字段值上定义一个索引，或者在类中的相应属性上定义一个索引。
- 你也可以在几个字段/属性的组合值上定义一个索引。无论你是用SQL字段和表的语法，还是用类的属性语法来定义，都会创建相同的索引。
- 当某些类型的字段（属性）被定义时，InterSystems IRIS会自动定义索引，例如主键和唯一值属性。
- 你可以为同一个字段（属性）定义一个以上的索引，为不同的目的提供不同类型的索引。

每当对数据库进行数据插入、更新或删除操作时，InterSystems IRIS都会填充和维护索引（默认情况下），无论是使用SQL字段和表语法，还是使用类属性语法。你可以覆盖这个默认值（通过使用%NOINDEX关键字）来快速对数据进行修改，然后作为一个单独的操作建立或重建相应的索引。你可以在用数据填充一个表之前定义索引，你也可以为一个已经填充了数据的表定义索引，然后作为一个单独的操作填充（建立）索引。

InterSystems IRIS在准备和执行SQL查询时，会使用可用的索引。默认情况下，它选择使用哪些索引来优化查询性能。你可以覆盖这个默认值，以防止在特定的查询或所有的查询中使用一个或多个索引，视情况而定。

索引属性

每个索引都有一个唯一的名字。这个名称用于数据库管理目的（报告、建立索引、删除索引等等）。像其他SQL实体

一样，一个索引也有一个SQL索引名和一个相应的索引属性名；这些名称在允许的字符、大小写敏感度和最大长度上有所不同。

如果使用SQL CREATE INDEX命令定义，系统会生成一个相应的索引属性名。如果使用持久化类定义，SqlName关键字允许用户指定一个不同的SQL索引名称（SQL映射名称）。管理门户SQL界面的目录详情显示了每个索引的SQL索引名称（SQL映射名称）和相应的索引属性名称（索引名称）。

索引类型是由两个索引类关键字Type和Extent定义的。InterSystems IRIS可用的索引类型包括：

- 标准索引（类型=索引）- 一个持久的数组，将索引值与包含该值的行的RowID联系起来。任何没有明确定义为位图索引、位片索引或范围索引的索引都是一个标准索引。
- 位图索引（类型=位图）--一种特殊的索引，它使用一系列的bitstring来表示对应于给定索引值的RowID值的集合；InterSystems IRIS包括一些针对位图索引的性能优化。
- 位片索引（类型=位片）--一种特殊的索引，能够非常快速地评估某些表达式，如sum范围条件。某些SQL查询会自动使用位片索引。
- Extent Indices（位图范围索） - 一个范围内所有对象的索引。

一个表（类）的**最大索引数是400**。

系统自动定义的索引

当你定义一个表时，系统会自动定义某些索引。以下索引在你定义表时自动生成，并在你添加或修改表数据时被填充。如果你定义了：

- 一个不是IDKEY的主键，系统会生成一个Unique类型的相应索引。主键索引的名称可以是用户指定的，也可以是从表的名称中衍生出来的。例如，如果你定义了一个未命名的主键，相应的索引将被命名为tablenamePKEY#，其中#是每个唯一键和主键约束的一个连续的整数。
- 一个UNIQUE字段，InterSystems IRIS为每个UNIQUE字段生成一个索引，名称为tablameUNIQUE#，其中#是每个唯一键和主键约束的连续整数。
- 一个UNIQUE约束，系统为每个具有指定名称的UNIQUE约束生成一个索引，为共同定义一个唯一值的字段生成索引。
- 一个分片Shard Key，系统为分片键字段生成一个索引，名为ShardKey。

你可以通过管理门户的SQL目录细节标签查看这些索引。CREATE INDEX命令可以用来添加一个UNIQUE字段约束；DROP INDEX命令可以用来删除一个UNIQUE字段约束。

默认情况下，系统会在RowID字段上生成IDKEY索引。定义一个IDENTITY字段并不产生索引。然而，如果你定义了一个IDENTITY字段并使该字段成为主键，InterSystems IRIS会在IDENTITY字段上定义IDKEY索引并使其成为主键索引。这在下面的例子中显示。

```
CREATE TABLE Sample.MyStudents (
          FirstName VARCHAR(12),
          LastName VARCHAR(12),
          StudentID IDENTITY,
          CONSTRAINT StudentPK PRIMARY KEY (StudentID) )
```

同样，如果你定义了一个IDENTITY字段，并给该字段一个UNIQUE约束，InterSystems IRIS会明确地在IDENTITY字段上定义一个IdKey/Unique索引。这在下面的例子中显示。

```
CREATE TABLE Sample.MyStudents (
          FirstName VARCHAR(12),
          LastName VARCHAR(12),
```

```
          StudentID IDENTITY,
          CONSTRAINT StudentU UNIQUE (StudentID) )
```

这些IDENTITY索引操作只在没有明确定义的IdKey索引和表不包含数据的情况下发生。

手动定义索引

有两种方法来定义索引。

- 使用类定义来定义索引，其中包括：
  - 可以被索引的属性
  - 多个属性的索引
  - 索引的整理
  - 在索引中使用Unique、PrimaryKey和IdKey关键字
  - 定义SQL搜索索引
  - 用索引存储数据
  - 对NULL进行索引
  - 索引集合
  - 阵列集合的索引
  - 用(ELEMENTS)和(KEYS)对数据类型属性进行索引
  - 为嵌入式对象（%SerialObject）属性建立索引
  - 关于在类中定义的索引的说明
- 使用DDL定义索引

使用类定义来定义索引

```
Class MyApp.Student Extends %Persistent [DdlAllowed]
{
 Property Prop1 As %String;
 Property Prop2 As %String;

 //????????
 Index Prop1IDX On Prop1;

 //????????
 Index MIXIDX On (Prop1, Prop2)

 //???????collation????
 Index Prop2IDX On Prop2 As Exact;

 //??????collation????
 Index MIX2IDX On ?Prop1 As SQLUPPER,Prop2 As Exact);

 //??Unique???????
 Index Prop1IDX On Prop1 [Unique]

 //??PrimaryKey???????
 Index Prop1IDX On Prop1 [PrimaryKey]

 //??IdKey???????
 Index Prop1IDX On Prop1 [IdKey]

 //?????? -- ???????
 Index Prop1IDX On (Prop1) As %iFind.Index.Basic
```

```
 //List??Array??????
 Index EmpIndex On Employees(KEYS)
 Index EmpIndex On Employees(ELEMENTS)
 Index EmpIndex On (Employees(KEYS), Employees(ELEMENTS));

 //????????
 Index StateInx On Home.State;

 //??????
 Index ReginIDX On Region [Type = bitmap]

 //??????
 Index AgeIDX On Age [Type = bitslice]
}
```

在类定义中处理索引时，有几点需要注意：

- 索引定义只从主（第一）超类中继承。
- 如果你使用Studio为一个有数据存储的类添加（或删除）一个索引定义，你必须通过构建索引来手动填充索引。

使用类定义定义%BID位图索引

如果表的ID不是正整数，你可以创建一个%BID属性，用来创建位图索引定义。你可以对具有任何数据类型的ID字段的表，以及由多个字段组成的IDKEY（其中包括子表）使用这个选项。可以为任一数据存储类型创建%BID位图：默认结构表或%Storage.SQL表。这个功能被称为 "任何表的位图Bitmaps for Any Table"，或BAT。

要在这样的表中启用位图索引，你必须做以下工作。

0. 为该类定义一个%BID属性/字段。这可以是该类的一个现有属性，也可以是一个新的属性。它可以有任何名字。如果这是一个新的属性，你必须为表中的所有现有行填充这个属性/字段。这个%BID字段必须用一个数据类型来定义，限制字段数据值为唯一的正整数。例如：

   ```
   Property MyBID As %Counter;
   ```

1. 定义一个新的类参数来定义哪个属性是%BID字段。这个参数被命名为BIDField。这个参数被设置为%BID属性的SQLFieldName。例如，参数BIDField = "MyBID"。

   ```
   Parameter BIDField = "MyBID";
   ```

2. 为%BID定义一个索引。例如，

   ```
   Index BIDIdx On MyBID [ Type = key, Unique ] ?
   ```

3. 定义%BID定位器索引。这将%BID索引与表的ID关键字段联系起来。下面的例子是关于一个有两个字段组成的复合IDKey的表。

   ```
   Index IDIdx On (IDfield1, IDfield2) [ IdKey, Unique ] ?
   Index BIDLocIdx On (IDfield1, IDfield2, MyBID) [ Data = IdKey, Unique ] ?
   ```

这个表现在支持位图索引。你可以使用标准语法根据需要定义位图索引。例如:

```
Index RegionIDX On Region [Type = bitmap]
```

该表现在也支持位片索引。你可以使用标准语法定义位片索引。

注意:
要建立或重建一个%BID位图索引，你必须使用**%BuildIndices()**
。%ConstructIndicesParallel()方法不支持%BID位图索引。

使用DDL定义索引

DDL索引命令做了以下工作。

- 它们更新相应的类和表的定义，在这些定义上添加或删除索引。修改后的类定义被重新编译。
- 它们根据需要在数据库中添加或删除索引数据。CREATE INDEX命令使用当前存储在数据库中的数据来填充索引。同样，DROP INDEX命令从数据库中删除了索引数据（也就是实际的索引）。

CREATE INDEX

```
//????
CREATE INDEX StateIdx ON TABLE Sample.Person (Home_State)
?
//??????
CREATE BITMAPEXTENT INDEX Patient ON TABLE Sample.Patient
?
//????
CREATE BITMAP INDEX RegionIDX ON TABLE MyApp.SalesPerson (Region)
?
//????
CREATE BITSLICE INDEX AgeIDX ON TABLE MyApp.SalesPerson (Age)
?
```

DROP INDEX

```
DROP INDEX PeopleIndex ON TABLE Employee
```

为嵌入式对象(%SerialObject)属性创建索引

要为嵌入式对象中的一个属性建立索引，你要在引用该嵌入式对象的持久化类中创建一个索引。属性名必须指定表（%Persistent类）中的引用字段的名称和嵌入对象（%SerialObject）中的属性，如下面的例子所示。

```
Class Sample.Person Extends (%Persistent) [ DdlAllowed ]
{  Property Name As %String(MAXLEN=50);
  Property Home As Sample.Address;
  Index StateInx On Home.State;
}
```

这里Home是Sample.Person中的一个属性，它引用嵌入式对象Sample.Address，其中包含State属性，如下例所示
。

```
Class Sample.Address Extends (%SerialObject)
{  Property Street As %String;
```

```
  Property City As %String;
  Property State As %String;
  Property PostalCode As %String;
}
```

只有与持久化类属性引用相关的嵌入式对象实例中的数据值被索引。你不能直接索引%SerialObject属性。%Library.SerialObject（以及所有没有明确定义SqlCategory的%SerialObject的子类）的SqlCategory是STRING。

你也可以使用SQL CREATE INDEX语句在嵌入式对象属性上定义一个索引，如下面的例子所示。

```
CREATE INDEX StateIdx ON TABLE Sample.Person (Home_State)
```

位图索引

位图索引是一种特殊类型的索引，它使用一系列的bit字符串来表示与给定的索引数据值相对应的ID值集合。

位图索引有以下重要特点：

- 位图是高度压缩的：位图索引可以比标准索引小得多。这大大减少了磁盘和缓存的使用。
- 位图操作为事务处理进行了优化：你可以在表内使用位图索引，与使用标准索引相比，没有性能损失。
- 位图上的逻辑操作（计数、和、以及OR）被优化为高性能。
- SQL引擎包括一些特殊的优化，可以利用位图索引的优势。

位图索引的创建取决于表的唯一标识字段的性质。

- 如果表的ID字段被定义为一个具有正整数值的单一字段，你可以使用这个ID字段为一个字段定义位图索引。这种类型的表要么使用系统分配的唯一正整数ID，要么使用IdKey定义自定义ID值，其中IdKey基于类型为%Integer且MINVAL>0的单一属性，或类型为%Numeric且SCALE=0且MINVAL>0的单一属性。
- 如果表的ID字段没有定义为具有正整数值的单一字段（例如，一个子表），你可以定义一个%BID（位图ID）字段，它采取正整数，作为一个代理ID字段；这允许你为这个表中的字段创建位图索引。

本章讨论了与位图索引有关的下列主题 “

- 位图索引操作
- 通过使用类定义来定义位图索引
- 使用DDL定义位图索引
- 生成一个位图范围索引
- 选择一个索引类型
- 对位图索引的限制
- 维护位图索引
- 位图分块的SQL操作

位图索引操作原理

位图索引的工作方式如下。假设你有一个包含若干列的Person表，这个表中的每一行都有一个系统分配的RowID号码（一组递增的整数值）。一个位图索引使用一组bitstring（一个包含1和0值的字符串）。在一个bitstring内，一个bit的序号位置对应于被索引表的RowID。对于一个给定的值，例如State是 "NY"，有一个bitstring，对应于包含 "NY "的行，每个位置都是1，其他不包含'NY'的行，位置都是0。

例如，一个关于State的位图索引可能看起来像这样:

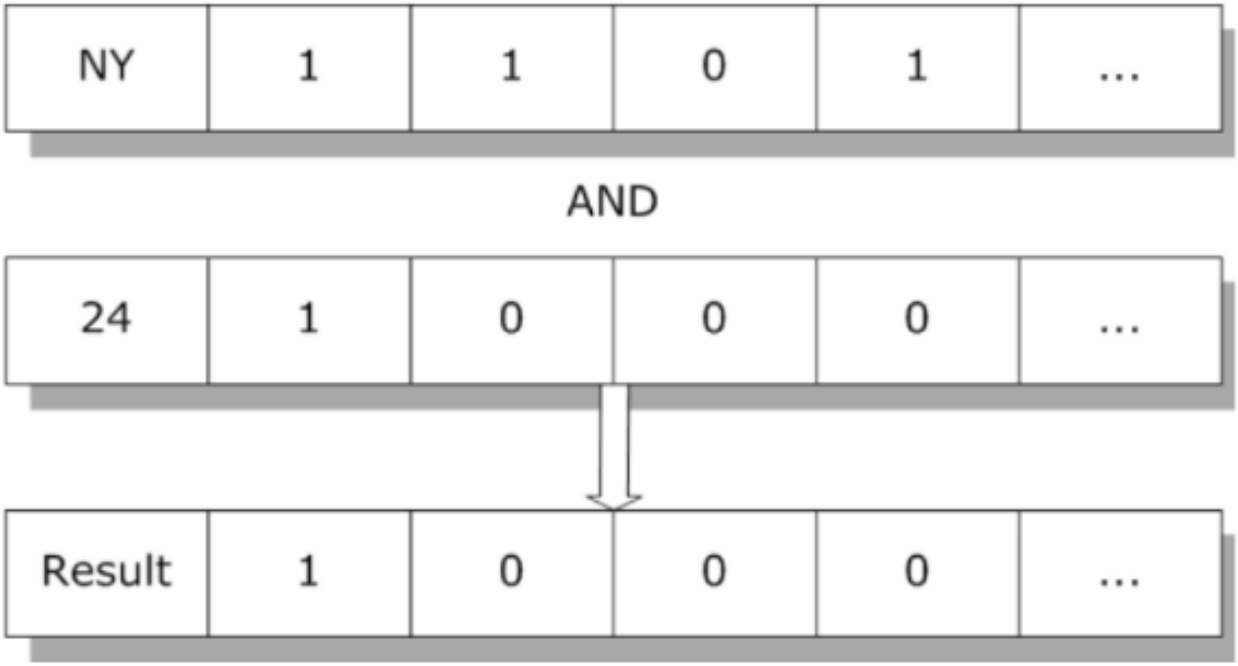
StateIndex					
	Row 1	Row 2	Row 3	Row 4	...
CA	0	0	1	0	...
NY	1	1	0	1	...
WY	0	0	0	0	...
...	...	...	...	...	...

而关于年龄的索引可能看起来像这样。

AgeIndex					
	Row 1	Row 2	Row 3	Row 4	...
24	1	0	0	0	...
35	0	1	0	0	...
48	0	0	1	0	...
...	...	...	...	...	...

注意: 这里显示的年龄字段可以是一个普通的数据字段，也可以是一个可以靠其他字段得出的字段（计算和SQLComputed)。

除了在标准操作中使用位图索引外，SQL引擎还可以使用位图索引，使用多个索引的组合有效地执行基于集合的特殊操作。例如，为了找到所有24岁并居住在纽约的Person实例，SQL引擎可以简单地执行Age和State索引的 AND 逻辑。结果如下：



SQL引擎可以使用位图索引进行以下操作:

- 在一个给定的表上对多个条件进行ANDing。
- 在一个给定的表上的多个条件的ORing。
- 在一个给定的表上的RANGE条件。
- 在一个给定的表上进行COUNT操作。

位图索引的限制

所有的位图索引都有以下限制：

- 不能在一个UNIQUE列上定义一个位图索引。
- 不能在位图索引中存储数据值。
- 不能在一个字段上定义位图索引，除非该字段的SqlCategory是INTEGER, DATE, POSIXTIME, 或NUMERIC (with scale=0) 。
- 对于一个包含超过100万条记录的表来说，当唯一值的数量超过10,000时，位图索引的效率就不如标准索引。因此，对于一个大表，建议你避免对任何包含（或可能包含）超过10,000个唯一值的字段使用位图索引；对于任何规模的表，避免对任何可能包含超过20,000个唯一值的字段使用位图索引。这些都是一般的近似值，不是精确的数字。

你必须创建一个%BID属性来支持表上的位图索引，该表

- 使用一个非整数字段作为唯一的ID键。
- 使用一个多字段的ID键。
- 是父-子关系中的一个子表。

你可以使用$SYSTEM.SQL.Util.SetOption()方法SET status=$SYSTEM.SQL.Util.SetOption("BitmapFriendlyCheck",1,.oldval)来设置一个系统范围的配置参数，在编译时检查这个限制，确定%Storage.SQL类中是否允许定义位图索引。这个检查只适用于使用%Storage.SQL的类。你可以使用$SYSTEM.SQL.Util.GetOption("BitmapFriendlyCheck")来确定这个选项的当前配置。

应用逻辑的限制:

- 一个位图结构可以用一个bitstring数组来表示，其中数组的每个元素代表一个具有固定bit数的"块"。因为未定义等同于一个全部为0位的块，所以数组可以是稀疏的。一个代表所有0 bit的块的数组元素根本就不需要存在。由于这个原因，应用逻辑应该避免依赖0值位的$BITCOUNT(str,0)计数。
- 因为一个bitstring包含内部格式化，应用逻辑不应该依赖于一个bitstring的物理长度，也不应该依赖于对两个具有相同位值的bitstring进行等价。在回滚操作之后，一个bitstring被恢复到事务之前的位值。然而，由于内部格式化，回滚的bitstring可能不等同于事务之前的bitstring，也不具有相同的物理长度。

维护位图索引

在一个不稳定的表（一个经历了许多INSERT和DELETE操作的表），位图索引的存储可能会逐渐变得不那么有效。

为了维护位图索引，你可以运行%SYS.Maint.Bitmap工具方法来压缩位图索引，将其恢复到最佳效率。你可以使用OneClass()方法来压缩单个类的位图索引。或者你可以使用Namespace()方法来压缩整个命名空间的位图索引。这些维护方法可以在一个生产系统上运行。

```
d ##class(%SYS.Maint.Bitmap).Namespace("Samples",1,1,"2014-01-17 09:00:00")
d ##class(%SYS.Maint.Bitmap).OneClass("BitMap.Test",1,1)
```

运行%SYS.Maint.Bitmap实用方法的结果被写到调用该方法的进程中。这些结果也被写入%SYS.Maint.BitmapResults类中。

位片索引

当一个数字数据字段被用于某些数字操作时，位片索引被用于该字段。位片索引将每个数字数据值表示为一个二进制位串。比起使用布尔标志对数字数据值进行索引（如在位图索引中），位片索引用二进制表示每个值，并为二进制值中的每个数字创建一个位图来记录哪些行的二进制数字为1。这是一种高度专业化的索引类型，可以大幅度提高以下操作的性能。位片索引适用于以下类型：

- SUM, COUNT, 或AVG聚合计算。(位片索引不用于COUNT(*)计算) 位片索引不用于其他聚合函数。
- 在 TOP n ... ORDER BY field 的操作。
- 在一个范围条件操作中指定的字段，例如 WHERE field > n or WHERE field BETWEEN lownum AND highnum

SQL优化器决定是否应该使用定义的位片索引。通常情况下，优化器只在处理大量的行（成千上万）时才使用位片索引。

在一个不稳定的表（一个经历了许多INSERT、UPDATE和DELETE操作的表）中，位片索引的存储效率会逐渐降低。%SYS.Maint.Bitmap工具方法同时压缩了位图索引和位片索引，恢复了效率。更多细节，请看 "维护位图索引"。

位图范围索引

位图范围索引是针对表的行的位图索引，而不是针对表的任何指定字段。在位图范围索引中，每个位代表一个连续的RowID整数值，每个位的值指定相应的行是否存在。InterSystems SQL使用这种索引来提高COUNT(*)*的性能，COUNT()*返回表中的记录数（行数）。
一个表最多可以有一个位图范围索引。试图创建一个以上的位图范围索引，会导致SQLCODE

-400错误，并出现%msg ERROR #5445: 定义了多个范围索引。

一个位图索引需要一个位图范围索引。只有在定义了一个或多个位图索引的情况下，定义一个持久化类才会生成一个位图范围索引。因此，当编译一个包含位图索引的持久化类时，如果没有为该类定义位图范围索引，类编译器会生成一个位图范围索引。

如果你从持久化类定义中删除了所有位图索引，那么位图范围索引会自动删除。但是，如果你重命名位图范围索引（例如，使用CREATE BITMAPEXTENT INDEX命令），删除位图索引并不会删除位图范围索引。

当为一个类建立索引时，如果你明确地建立了位图范围索引，或者你建立了位图索引而位图范围索引是空的，那么位图范围索引就会被建立。

一个类从主超类继承它的位图范围索引，如果它存在的话，可以定义或生成。一个位图范围索引被定义为：type = bitmap, extent = true。这意味着从主超类继承的位图范围索引被认为是一个位图索引，如果在子类中没有明确定义位图范围索引，则会触发在子类中生成一个位图范围索引。

InterSystems IRIS不会根据未来的可能性在超类中生成一个位图范围索引。这意味着InterSystems IRIS永远不会在一个持久性类中生成一个位图范围索引，除非有一个类型=位图的索引存在。假设某个未来的子类可能引入一个类型=位图的索引是不够的。

注意：在生产系统上为一个类添加位图索引的过程中需要特别注意（当用户使用一个特定的类，编译该类，并随后为它建立位图索引结构）。在这样的系统上，位图范围索引可能会在编译完成和索引建立过程的中间阶段被填充。这可能导致索引构建程序没有隐含地构建位图范围索引，这导致了部分位图范围索引不能被构建。

在一个经历了许多DELETE操作的表中，位图范围索引的存储会逐渐变得不那么有效。你可以通过使用BUILD INDEX命令重建位图范围索引，或者从管理门户选择表的Catalog Details标签，Maps/Indices选项并选择Rebuild Index。

构建索引

构建一个索引会做以下事情。

- 移除索引的当前内容。
- 扫描（读取每一行）主表，为表中的每一行添加索引条目。如果可能的话，使用特殊的$SortBegin和$SortEnd函数来确保建立大型索引的效率。当建立标准索引时，除了在内存中缓存数据外，这种对$SortBegin/$SortEnd的使用会占用IRISTEMP数据库的空间。因此，当建立一个非常大的标准索引时，InterSystems IRIS可能需要IRISTEMP中的空间，大致相当于最终索引的大小。

注意：建立索引的方法只提供给使用InterSystems IRIS默认存储结构的类（表）。

当前的数据库访问决定了你应该如何重建一个现有的索引：

- 非活跃系统（在索引建立或重建期间没有其他进程访问数据）
- 只读活跃系统（其他进程在索引建立或重建期间能够查询数据）
- 读和写活跃系统（其他进程能够修改数据，并在索引建立或重建期间查询数据）。

你可以按以下方式建立/重新建立索引。

- 使用BUILD INDEX SQL命令来构建指定的索引，或构建为一个表、一个Schema或当前命名空间定义的所有索引：

  ```
  //????????
  BUILD INDEX FOR TABLE table-name
  ```

```
//???Schema?????????
BUILD INDEX FOR SCHEMA schema-name
//??????????
BUILD INDEX FOR ALL
```

- 使用管理门户为一个指定的类（表）重建所有的索引。 系统资源管理器 - SQL- 操作 - 重建表索引
- 使用%BuildIndices()（或%BuildIndicesAsync()）方法，如本节所述。

建立索引的首选方法是使用%BuildIndices()方法或%BuildIndicesAsync()方法。

- %Library.Persistent.%BuildIndices()。%BuildIndices()作为一个后台进程执行，但调用者必须等待%BuildIndices()完成后才能接收控制权。
- %Library.Persistent.%BuildIndicesAsync()。BuildIndicesAsync()将%BuildIndices()作为一个后台进程启动，调用者立即收到控制权。%BuildIndicesAsync()的第一个参数是queueToken输出参数。其余的参数与%BuildIndices()相同。
    - %BuildIndicesAsync()返回一个%Status值：成功表示%BuildIndices()工作任务成功排队；失败表示工作任务没有成功排队。
    - %BuildIndicesAsync()向queueToken输出参数返回一个值，表示%BuildIndices()完成状态。为了获得完成状态，你通过引用queueToken值传递给%BuildIndicesAsyncResponse()方法。你还指定了wait布尔值。如果wait=1，%BuildIndicesAsyncResponse()将等待，直到由queueToken识别的%BuildIndices()作业完成。如果wait=0，%BuildIndicesAsyncResponse()将尽可能快地返回一个状态值。如果%BuildIndicesAsyncResponse()的queueToken在返回时不是NULL，那么%BuildIndices()作业还未完成。在这种情况下，queueToken可以用来再次调用%BuildIndicesAsyncResponse()。当%BuildIndicesAsyncResponse()的queueToken最终为空时，那么%BuildIndicesAsyncResponse()返回的%Status值就是由%BuildIndicesAsync()调用的作业的完成状态。

在一个不活跃的系统上构建索引：

系统自动生成方法（由%Persistent类提供），用于构建（即为其提供数值）或清除（即为其移除数值）为类（表）定义的每个索引。你可以通过以下两种方式使用这些方法。

- 通过管理门户： 系统资源管理器 - SQL- 操作 - 重建表索引 注意：当其他用户正在访问该表的数据时，不要重建索引。要在一个活跃系统上重建索引，请看在活跃系统上建立索引。
- 以编程方式调用。
    - 构建所有索引：调用%BuildIndices()，没有参数，为给定的类（表）构建（提供）所有索引。

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

    - 构建指定的索引：调用%BuildIndices()，以$List的索引名称作为第一个参数，为一个给定的类（表）建立（提供）指定的索引。

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

    - 构建所有索引，除了某些指定索引：调用%BuildIndices()，将$List的索引名称作为第七个参数，为一个给定的类（表）构建（提供值）所有定义的索引，除了指定的索引。

      ```
      SET sc = ##class(MyApp.SalesPerson).%BuildIndices("",,,,,,$ListBuild("Nam
      eIDX", "SSNKey"))
      IF sc=1 {WRITE !, "Successful index build" }
      ELSE {WRITE !"Index build failed",!
      ```

```
                    DO $System.Status.DisplayError(sc) QUIT}
```

%BuildIndices()方法做了以下工作:

0. 在任何要重建的（非位图）索引上调用$SortBegin函数（这将为这些索引启动一个高性能的排序操作）。
1. 循环处理该类的主要数据（表），收集索引使用的值，并将这些值添加到索引中（有适当的整理转换）。
2. 调用$SortEnd函数来完成对索引的排序过程。

如果索引已经有了值，你必须用两个参数调用%BuildIndices()，其中第二个参数的值为1，为这个参数指定1会使该方法在重建索引之前清除这些值。比如说:

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

这就清除并重建了所有的索引。你也可以清除和重建索引的一个子集，如：

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

注意：当其他用户正在访问该表的数据时，不要重建索引。要在一个活跃系统上重建索引，请看在活跃系统上建立索引。

在活跃系统上建立索引

当在一个活跃系统上建立（或重建）索引时，有两个问题：

- 活跃查询可能会返回不正确的结果，除非正在建立的索引被隐藏在SELECT查询之外。在建立索引之前，可以使用SetMapSelectability()方法来处理这个问题。
- 在建立索引期间对数据的主动更新可能不会反映在索引条目中。这可以通过在构建索引时让构建操作锁定个别行来处理。

注意：如果一个应用程序在一个事务中对数据进行大量的更新，可能会出现锁表争夺的问题。

在一个只读活跃系统上构建索引

如果一个表目前只用于查询操作（只读），你可以在不中断查询操作的情况下建立新的索引或重建现有索引。这是通过在重建这些索引时使索引对查询优化器不可用来实现的。

如果你想建立一个或多个索引的所有类目前都是只读的，使用 "在读和写活跃系统上建立索引"中描述的相同系列操作，但有以下区别：当你使用%BuildIndices()时，设置pLockFlag=3（共享范围锁）。

在读和写活跃系统上建立索引

如果一个持久化的类（表）目前正在使用，并且可以进行读和写访问（查询和数据修改），你可以建立新的索引或者重建现有的索引而不中断这些操作。如果你想重建一个或多个索引的类目前可以被读和写访问，建立索引的首选方法是使用表的持久化类提供的%BuildIndices()（或%BuildIndicesAsync()）方法。

在并发的读和写访问中，建立一个或多个索引需要进行以下一系列操作：

0. 使你希望建立的索引对查询不可用（READ访问）。这是用SetMapSelectability()完成的。这使得该索引不能被查询优化器使用。这个操作应该在重建一个现有的索引和创建一个新的索引时执行。比如说

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

   第一个参数是Schema.Table名称，即SqlTableName，而不是持久化类的名称。例如，默认的Schema是SQLUser，而不是User。这个值是区分大小写的。 第二个参数是SQL索引图名称。这通常是索引的名称，指的是索引存储在磁盘上的名称。对于一个新的索引，这是你在创建索引时要使用的名称。这个值是不区分大小写的。 第三个参数是MapSelectability标志，其中0定义索引图为不可选择（关闭），1定义索引图为可选择（打开）。指定为0。 你可以通过调用GetMapSelectability()方法来确定一个索引是否是不可选择的。如果你已经明确地将一个索引标记为不可选择，这个方法返回0。在所有其他情况下，它返回1；它不对表或索引的存在进行验证检查。注意，Schema.Table名称是SqlTableName，并且区分大小写。 SetMapSelectability()和GetMapSelectability()仅适用于当前命名空间的索引映射。如果该表被映射到多个命名空间，并且需要在每个命名空间建立索引，则应在每个命名空间调用SetMapSelectability()。
1. 在索引建立的过程中建立并发操作。
   - 对于一个新的索引 在类中创建索引定义（或者在类的%Storage.SQL中创建新的SQL索引映射规范），编译该类。对于一个新的索引，这是合适的，因为索引还没有被填充。在可以对表进行查询之前，需要对范围索引进行填充。
   - 对于一个现有的索引。清除任何引用该表的缓存查询。索引构建所执行的第一个操作是杀死索引。因此，当索引被重建时，你不能依靠任何被优化的代码来使用该索引。

     ```
     //???????????????
     PURGE CACHED QUERIES
     ?
     //????????n?????????
     PURGE CACHED QUERIES BY AGE n
     Do $SYSTEM.SQL.Purge(n)
     ?
     //??????????
     PURGE CACHED QUERIES BY TABLE table-name
     ??
     Do $SYSTEM.SQL.PurgeForTable("MedLab.Patient")
     ?
     //?????????
     PURGE [CACHED] QUERIES BY NAME class-name
     ```

2. 使用你的持久化类（表）的%BuildIndices()方法，使用**pLockFlag=2（行级锁）**来建立索引。pLockFlag=2标志在重建过程中对个别行建立了一个排他性的写锁，这样并发的数据修改操作就可以与构建索引操作相协调。 默认情况下，%BuildIndices()会构建所有为持久化类定义的索引；你可以使用pIgnoreIndexList来排除重建索引。 默认情况下，%BuildIndices()为所有ID建立索引条目。然而，你可以你可以使用pStartID和pEndID来定义一个ID的范围。%BuildIndices()将只为该范围内的ID建立索引条目。例如，如果你使用带有%NOINDEX限制的INSERT将一系列新记录添加到表中，你可以随后使用带有ID范围的%BuildIndices()来为这些新记录建立索引条目。 %BuildIndices()返回一个%Status值。如果%BuildIndices()由于检索数据的问题而失败，系统会产生一个SQLCODE错误和一个消息（%msg），其中包括遇到错误的%ROWID。
3. 一旦你完成了建立索引，请启用查询优化器的MapSelectability。设置第三个参数，MapSelectability标志为1，如下面的例子所示。

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

4. 再一次，清除任何引用该表的缓存查询。这将消除在此过程中创建的无法使用索引的缓存查询，因此，与使用索引的相同查询相比，缓存查询的效果较差。

这样就完成了这个过程。索引被完全填充，并且查询优化器能够考虑该索引。

注意：%BuildIndices()只能用于为有正整数ID值的表重建索引。如果父表有正的整数ID值，你也可以使用%BuildIndices()来重建子表的索引。对于其他表，使用%ValidateIndices()方法。因为%ValidateIndices()是建立索引的最慢的方法，所以只有在没有其他选择的情况下才应该使用它。

验证索引

你可以使用以下任一方法来验证索引：

- $SYSTEM.OBJ.ValidateIndices()验证一个表的索引，同时也验证该表的集合子表的任何索引。
- %Library.Storage.%ValidateIndices()验证一个表的索引。集合子表的索引必须用单独的%ValidateIndices()调用进行验证。

这两种方法都检查指定表的一个或多个索引的数据完整性，并可选择纠正发现的任何索引完整性问题。它们分两步进行索引验证。

0. 确认为表（类）中的每一行（对象）正确定义了一个索引实体。
1. 遍历每个索引，对于每个被索引的条目，确保在表（类）中有一个值和匹配的条目。

如果任何一种方法发现不一致的地方，它可以选择性地纠正索引结构和内容。它可以验证并选择性地纠正标准索引、位图索引、位图范围索引和位片索引。默认情况下，这两个方法都验证索引，但不纠正索引。

只有在遵循以下条件的情况下，%ValidateIndices()才能用于纠正(建立)读和写活跃系统中的索引：

- SetMapSelectability()被使用，如上所述；%ValidateIndices()参数必须包括autoCorrect=1和lockOption>0。因为%ValidateIndices()的速度明显较慢，%BuildIndices()是在活跃系统上建立索引的首选方法。

%ValidateIndices()通常从终端运行。它显示输出到当前设备。这个方法可以应用于指定的%List索引名称，或者为指定表（类）定义的所有索引。它只对那些起源于指定类的索引进行操作；如果一个索引起源于一个超类，该索引可以通过在超类上调用%ValidateIndices()进行验证。

只读类不支持%ValidateIndices()。

%ValidateIndices()被支持用于分片类和分片主类表（Sharded=1）。你可以调用%ValidateIndices，要么直接作为一个类方法，要么从$SYSTEM.OBJ.ValidateIndices上调用分片主类。然后在每个分片上的分片本地类上执行索引验证，并将结果返回给分片主类上的调用者。当在分片类上使用%ValidateIndices()时，verbose标志被强制为0，没有输出到当前设备。任何发现/纠正的问题都会在byreference errors()数组中返回。

下面的例子使用%ValidateIndices()来验证和纠正Sample.Person表的所有索引：

```
SET status=##class(Sample.Person).%ValidateIndices("",1,2,1)
IF status=1 {WRITE !, "Successful index validation/correction" }
ELSE {WRITE !, "Index validation/correction failed",!
      DO $System.Status.DisplayError(status) QUIT}?
```

第一个参数（""）指定要验证所有的索引；第二个参数（1）指定要修正索引差异；
第三个参数（2）指定要对整个表进行独占锁定；
第四个参数（1）指定使用多个进程（如果有的话）来执行验证。该方法返回一个%Status值。

通过名称验证索引:

```
SET IndList=$LISTBUILD("NameIDX", "SSNKey")
SET status=##class(Sample.Person).%ValidateIndices(IndList,1,2,1)
IF status=1 {WRITE !, "Successful index validation/correction" }
ELSE {WRITE !, "Index validation/correction failed",!
      DO $System.Status.DisplayError(status) QUIT}?
```

%ValidateIndices()的第一个参数或$SYSTEM.OBJ.ValidateIndices()的第二个参数指定哪些索引将作为%List结构被验证。不管第一个参数的值如何，IdKey索引总是被验证的。你可以通过指定一个空字符串值（""）来验证表的所有索引。你可以通过指定一个列表结构来验证该表的单个索引。下面的例子验证了IdKey索引和两个指定的索引。NameIDX和SSNKey。

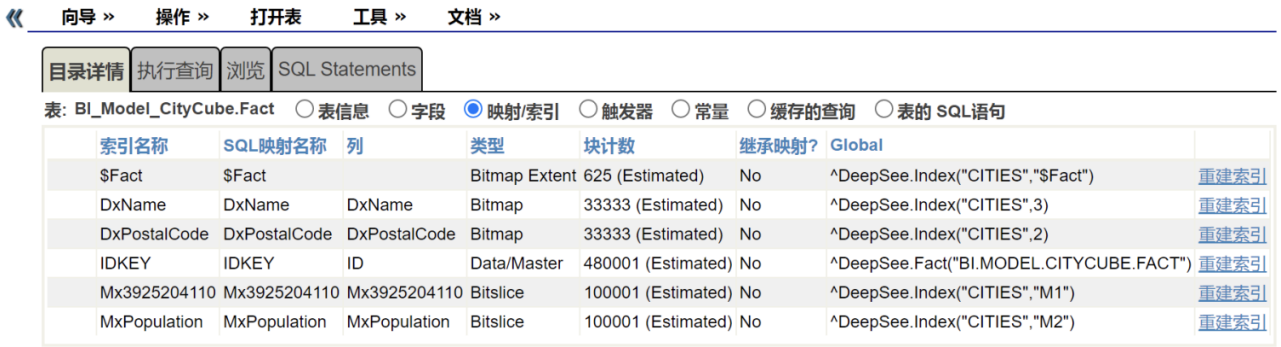
对于这两种方法，如果索引列表包含一个不存在的索引名称，该方法不执行索引验证，并返回%Status错误。如果索引列表中包含一个重复的有效索引名称，该方法将验证指定的索引，忽略重复的索引，不发出错误。

索引信息查询

INFORMATION.SCHEMA.INDEXES
持久化类显示当前命名空间中所有列索引的信息。它为每个被索引的列返回一条记录。它提供了一些索引的属性，包括索引的名称，表的名称，以及索引所对应的列的名称。每条列记录还提供了该列在索引图中的序号位置；除非索引映射到多个列，否则这个值是1。它还提供了布尔属性PRIMARYKEY和NONUNIQUE（0=索引值必须是唯一的）。

```
SELECT Index_Name,Table_Schema,Table_Name,Column_Name,Ordinal_Position,
Primary_Key,Non_Unique
FROM INFORMATION_SCHEMA.INDEXES WHERE NOT Table_Schema %STARTSWITH '%'
```

你可以使用管理界面的目录详情的映射/索引选项列出所选表的索引。这将为每个索引显示一行，并显INFORMATION.SCHEMA.INDEXES没有提供的索引信息。



索引名称	SQL映射名称	列	类型	块计数	继承映射?	Global	
$Fact	$Fact		Bitmap Extent	625 (Estimated)	No	^DeepSee.Index("CITIES","$Fact")	重建索引
DxName	DxName	DxName	Bitmap	33333 (Estimated)	No	^DeepSee.Index("CITIES",3)	重建索引
DxPostalCode	DxPostalCode	DxPostalCode	Bitmap	33333 (Estimated)	No	^DeepSee.Index("CITIES",2)	重建索引
IDKEY	IDKEY	ID	Data/Master	480001 (Estimated)	No	^DeepSee.Fact("BI.MODEL.CITYCUBE.FACT")	重建索引
Mx3925204110	Mx3925204110	Mx3925204110	Bitslice	100001 (Estimated)	No	^DeepSee.Index("CITIES","M1")	重建索引
MxPopulation	MxPopulation	MxPopulation	Bitslice	100001 (Estimated)	No	^DeepSee.Index("CITIES","M2")	重建索引

使用索引对一个对象进行open、exist和delete的方法

InterSystems IRIS的索引支持以下操作：

- 通过索引键打开一个实例
- 检查一个实例是否存在
- 删除一个实例

通过索引key打开一个实例

对于ID键、主键或唯一索引，indexnameOpen()方法（其中indexname是索引的名称）允许你打开其索引属性值与所

提供的值一致的对象。

例如，假设一个类包括下面的索引定义。

```
Index SSNKey On SSN [ Unique ] ?
```

那么，如果被引用的对象已经被存储到磁盘，并且有一个唯一的ID值，你可以按以下方式调用该方法。

```
 SET person = ##class(Sample.Person).SSNKeyOpen("111-22-3333",2,.sc)
```

第一个参数对应于索引中的属性。

第二个参数指定要打开对象的并发值（这里是2 - 共享锁）。

第三个参数可以接受%Status代码，以防该方法无法打开一个实例，如果找到一个匹配的实例，该方法会返回一个OREF。如果该方法没有找到一个与所提供的值相匹配的对象，那么一个错误信息将被写入状态参数sc中。

这个方法被实现为%Compiler.Type.Index.Open()方法；这个方法类似于%Persistent.Open()和%Persistent.OpenId()方法，除了它使用索引定义中的属性而不是OID或ID参数。

检查一个对象是否存在

indexnameExists()方法（其中indexname是索引的名称）检查是否存在一个具有该方法的参数所指定的索引属性值的实例。该方法有一个参数对应于索引中的每个属性；它的最后一个可选参数可以接收对象的ID，如果有一个与提供的值相匹配的话。该方法返回一个布尔值，表示成功（1）或失败（0）。这个方法被实现为%Compiler.Type.Index.Exists()方法。

例如，假设一个类包括下面的索引定义。

```
Index SSNKey On SSN [ Unique ];
```

那么，如果被引用的对象已经被存储到磁盘，并且有一个唯一的ID值，你可以按以下方式调用该方法。

```
 SET success = ##class(Sample.Person).SSNKeyExists("111-22-3333",.id)
```

成功完成后，success等于1，id包含与找到的对象匹配的ID。

该方法返回所有索引的值，除了。

- 位图索引，或位图范围索引。
- 当索引包括（ELEMENTS）或（KEYS）表达式时。

删除一个对象

indexnameDelete()
方法（其中indexname是索引的名称）是为了与Unique、PrimaryKey和/或IdKey索引一起使用；它删除其键值与提供的键属性/列值相匹配的实例。有一个可选的参数，你可以用它来指定该操作的并发设置。该方法返回一个%Status代码。它被实现为%Compiler.Type.Index.Delete()方法。

#SQL #InterSystems IRIS

源

URL:https://cn.community.intersystems.com/post/intersystems-sql-%E7%9A%84%E4%BC%98%E5%8C%96-%E7%AC%AC%E4%B8%80%E9%83%A8%E5%88%86-%E5%AE%9A%E4%B9%89%E5%92%8C%E6%9E%84%E5%BB%BA%E7%B4%A2%E5%BC%95