文章
姚 鑫 · 三月 14, 2021 阅读大约需 7 分钟

第十章 SQL排序（一）

第十章 SQL排序

排序规则指定值的排序和比较方式，并且是InterSystems SQL和InterSystemsIRIS®数据平台对象的一部分。有两种基本排序规则：数字和字符串。

- 数值排序规则按以下顺序基于完整数字对数字进行排序：null，然后是负数，从最大到最小，零，然后是正数，从最小到最大。这将创建如下序列：–210，–185，–54，–34，-.02、0、1、2、10、17、100、120。
- 字符串归类通过对每个顺序字符进行归类来对字符串进行排序。这将创建以下顺序：null，A，AA，AA，AAA，AAB，AB，B。对于数字，这将创建以下顺序：–.02，–185，–210，–34，–54，0、1、10、100、120、17、2。

默认的字符串排序规则是SQLUPPER；为每个名称空间设置此默认值。 SQLUPPER排序规则将所有字母都转换为大写（出于排序的目的），并在字符串的开头附加一个空格字符。此转换仅用于整理目的；在InterSystems中，无论所应用的排序规则如何，SQL字符串通常以大写和小写字母显示，并且字符串的长度不包括附加的空格字符。

时间戳记是一个字符串，因此遵循当前的字符串排序规则。但是，由于时间戳是ODBC格式，因此如果指定了前导零，则字符串排序规则与时间顺序相同。

- 字符串表达式（例如使用标量字符串函数LEFT或SUBSTR的表达式）使其结果归类为EXACT。
- 两个文字的任何比较都使用EXACT归类。

可以使用“ ObjectScript排序后 ”运算符来确定两个值的相对排序顺序。

可以按以下方式指定排序规则：
- 命名空间默认值
- 表字段/属性定义
- 索引定义查询
- SELECT项
- 查询DISTINCT和GROUP BY子句

排序类型

排序规则可以在字段/属性的定义或索引的定义中指定为关键字。

可以通过对查询子句中的字段名应用排序规则函数来指定排序规则。
在指定排序函数时必须使用%前缀。

排序规则采用升序的ASCII/Unicode序列，具有以下转换:
- **EXACT - 强制字符串数据区分大小写。**
如果字符串数据包含规范数字格式的值(例如123或-.57)，则不建议使用。
- **SQLSTRING - 去除末尾的空格(空格、制表符等)，并在字符串的开头添加一个前导空格。**
它将任何只包含空格(空格、制表符等)的值作为SQL空字符串进行排序。
SQLSTRING支持可选的maxlen整数值。
- **SQLUPPER -**
将所有字母字符转换为大写，去除末尾的空格(空格、制表符等)，然后在字符串的开头添加一个前导空格字符。

附加这个空格字符的原因是为了强制将数值作为字符串进行整理(因为空格字符不是有效的数字字符)。
这种转换还导致SQL将SQL空字符串('')值和任何只包含空格(空格、制表符等)的值作为单个空格字符进行整理。
SQLUPPER支持可选的maxlen整数值。
注意，SQLUPPER转换与SQL函数UPPER的结果不同。
- **TRUNCATE —增强字符串数据的区分大小写，并且（与EXACT不同）允许指定截断该值的长度。当索引比下标支持的数据长的精确数据时，此功能很有用。它采用%TRUNCATE（string，n）形式的正整数参数将字符串截断为前n个字符，从而改善了对长字符串的索引和排序。如果未为TRUNCATE指定长度，则其行为与EXACT相同；同时支持此行为。如果仅在定义了长度的情况下使用TRUNCATE而在没有定义长度的情况下使用EXACT，则定义和代码可能更易于维护。**
- **PLUS —使值成为数字。非数字字符串值将返回0。**
- **MINUS — 使数值成为数字并更改其符号。非数字字符串值将返回0。**

注意：还有多种传统排序规则类型，不建议使用。

在SQL查询中，可以指定不带括号%SQLUPPER Name或带括号%SQLUPPER（Name）的排序规则函数。如果排序规则函数指定了截断，则必须使用括号%SQLUPPER（Name，10）。

三种排序规则类型：SQLSTRING，SQLUPPER和TRUNCATE支持可选的maxlen整数值。如果指定，maxlen会将字符串的分析截断为前n个字符。在对长字符串进行索引和排序时，可以使用它来提高性能。可以在查询中使用maxlen进行排序，分组或返回截断的字符串值。

还可以使用 %SYSTEM.Util.Collation()方法执行排序规则类型转换。

命名空间范围的默认排序规则

每个名称空间都有一个当前的字符串排序规则设置。此字符串排序规则是为%Library.String中的数据类型定义的。默认值为SQLUPPER。此默认值可以更改。

可以基于每个命名空间定义排序规则默认值。默认情况下，名称空间没有分配的排序规则，这意味着它们使用SQLUPPER排序规则。可以为命名空间分配其他默认排序规则。此名称空间默认排序规则适用于所有进程，并且在InterSystems上保持不变，IRIS会重新启动，直到明确重置为止。

```
/// d ##class(PHA.TEST.SQL).Collation()
ClassMethod Collation()
{
    SET stat=$$GetEnvironment^%apiOBJ("collation","%Library.String",.collval)
    WRITE "???? ",$NAMESPACE,!
    ZWRITE collval
SetNamespaceCollation
    DO SetEnvironment^%apiOBJ("collation","%Library.String","SQLstring")
    SET stat=$$GetEnvironment^%apiOBJ("collation","%Library.String",.collnew)
    WRITE "user-assigned??? ",$NAMESPACE,!
    ZWRITE collnew
ResetCollationDefault
    DO SetEnvironment^%apiOBJ("collation","%Library.String",.collval)
    SET stat=$$GetEnvironment^%apiOBJ("collation","%Library.String",.collreset)
    WRITE "?????????? ",$NAMESPACE,!
    ZWRITE collreset
}
```

```
DHC-APP>d ##class(PHA.TEST.SQL).Collation()
???? DHC-APP
user-assigned??? DHC-APP
collnew="SQLstring"
```

```
?????????? DHC-APP
```

注意，如果从未设置名称空间排序的默认值，那么$$GetEnvironment将返回一个未定义的排序变量，例如本例中的.collval。
这个未定义的排序规则默认为SQLUPPER。

注意:如果数据包含德语文本，大写排序规则可能不是理想的默认设置。
这是因为德语eszett字符($CHAR(223))只有小写形式。
相当于大写的是两个字母“ SS ”。
转换为大写的SQL排序规则不会转换eszett, eszett保持为单个小写字母不变。

表字段/属性定义排序

在SQL中，排序规则可以分配为字段/属性定义的一部分。字段使用的数据类型确定其默认排序规则。字符串数据类型的默认排序规则为SQLUPPER。非字符串数据类型不支持排序规则分配。

可以在CREATE TABLE和ALTER TABLE中为字段指定排序规则：

```
CREATE TABLE Sample.MyNames (
    LastName CHAR(30),
    FirstName CHAR(30) COLLATE SQLstring)
```

注意：使用CREATE TABLE和ALTER TABLE为字段指定排序规则时，%前缀是可选的：COLLATE SQLstring或COLLATE %SQLstring。

在使用持久类定义定义表时，可以为属性指定排序规则：

```
Class Sample.MyNames Extends %Persistent [DdlAllowed]
{
Property LastName As %String;
Property FirstName As %String(COLLATION = "SQLstring");
}
```

注意：在为类定义和类方法指定排序规则时，请勿将%前缀用于排序规则类型名称。

在这些示例中，LastName字段采用默认排序规则（SQLUPPER，不区分大小写），FirstName字段使用区分大小写的SQLSTRING排序规则进行定义。

如果更改类属性的排序规则，并且已经存储了该类的数据，则该属性上的所有索引都将变为无效。必须基于此属性重建所有索引。

索引定义排序

CREATE INDEX命令无法指定索引排序规则类型。索引使用与要索引的字段相同的排序规则。

定义为类定义一部分的索引可以指定排序规则类型。默认情况下，给定一个或多个给定属性的索引使用属性数据的排序规则类型。例如，假设已定义类型为%String的属性Name：

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{
```

```
Property Name As %String;
Index NameIDX On Name;
}
```

名称的排序规则为SQLUPPER（%String的默认值）。假设“ Person ”表包含以下数据：

ID	Name
1	Jones
2	JOHNSON
3	Smith
4	jones
5	SMITH

然后，Name上的索引将包含以下条目：

Name	ID(s)
JOHNSON	2
JONES	1, 4
SMITH	3, 5

SQL引擎可以将此索引直接用于ORDER BY或使用“ Name ”字段进行比较操作。

可以通过在索引定义中添加一个As子句来覆盖用于索引的默认排序规则：

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{
Property Name As %String;
Index NameIDX On Name As SQLstring;
}
```

在这种情况下，NameIDX索引现在将以SQLSTRING（区分大小写）的形式存储值。使用上面示例中的数据：

Name	ID(s)
JOHNSON	2
Jones	1
jones	4
SMITH	5
Smith	3

在这种情况下，对于需要区分大小写排序规则的任何查询，SQL Engine都可以利用此索引。

通常，不必更改索引的排序规则。如果要使用其他排序规则，最好在属性级别定义它，然后让属性上的所有索引都采用正确的排序规则。

如果使用索引属性执行属性比较，则在比较中指定的属性应与相应索引具有相同的排序规则类型。例如，SELECT的WHERE子句或JOIN的ON子句中的Name属性应与为Name属性定义的索引具有相同的排序规则。如果属性归类和索引归类之间不匹配，则索引可能无效或根本不使用。

如果将索引定义为使用多个属性，则可以分别指定每个索引的排序规则：

```
Index MyIDX On (Name As SQLstring, Code As Exact);
```

#SQL #Caché #InterSystems IRIS #InterSystems IRIS for Health

源URL:

https://cn.community.intersystems.com/post/%E7%AC%AC%E5%8D%81%E7%AB%A0-sql%E6%8E%92%E5%BA%8F%EF%BC%88%E4%B8%80%EF%BC%89