

文章

姚鑫 · 三月 7, 2021 阅读大约需 17 分钟

第五章 SQL定义表（二）

第五章 SQL定义表（二）

主键

InterSystems IRIS提供了两种方法来唯一标识表中的行：RowID和主键。

可选的主键是一个有意义的值，应用程序可以使用该值唯一地标识表中的行（例如，联接中的行）。主键可以是用户指定的数据字段，也可以是多个数据字段的组合。主键值必须是唯一的，但不必是整数值。

RowID是一个内部用于标识表中行的整数值。通常，主键是由应用程序生成的值，而RowID是由InterSystems IRIS生成的唯一整数值。

系统会自动创建一个主map，以使用RowID字段访问数据行。如果定义主键字段，系统将自动创建并维护主键索引。

显然，具有两个不同的字段和索引来标识行的双重性不一定是一件好事。可以通过以下两种方式之一解析为单个行标识符和索引：

- 使用应用程序生成的主键值作为IDKEY。

可以通过使用关键字PrimaryKey和IdKey在类定义中标识主键索引来实现这一点(如果为此目的设置了PKey is IdKey标志，也可以在DDL中实现这一点)。

这使得主键索引成为表的主映射。

因此，主键将被用作行的主要内部地址。

如果主键包含多个字段，或者主键值不是整数，那么这种方法的效率会较低。

- 不要使用应用程序生成的主键值，而应在应用程序中使用系统生成的RowID整数作为应用程序使用的主键（例如，在joins中）。这样做的好处是，整数RowID有助于进行更有效的处理，包括使用位图索引。

根据应用程序的性质，可能希望解析为单个行标识符和索引，或者为应用程序生成的主键和系统生成的RowID具有单独的索引。

RowVersion，AutoIncrement和串行计数器字段

InterSystems

SQL支持三种专用数据类型，用于自动增加计数器值。这三种数据类型都是扩展%Library.BigInt数据类型类的子类。

- %Library.RowVersion：计算在命名空间范围内所有RowVersion表的插入和更新。只有在包含ROWVERSION字段的表中进行插入和更新时，此计数器才会递增。ROWVERSION值是唯一的且不可修改。此名称空间范围的计数器永远不会重置。
- %Library.Counter（也称为SERIAL计数器字段）：对表中的插入进行计数。默认情况下，此字段接收一个自动递增的整数。但是，用户可以为该字段指定一个非零的整数值。用户可以指定重复值。如果用户提供的值大于系统提供的最高值，则将自动递增计数器设置为从用户指定的值开始递增。
- %Library.AutoIncrement：计数插入到表中的次数。默认情况下，此字段接收一个自动递增的整数。但是，用户可以为该字段指定一个非零的整数值。用户可以指定重复值。指定用户值对自动增量计数器无效。

这三个字段以及IDENTITY字段均返回AUTO_INCREMENT = YES，如以下示例所示：

```
SELECT COLUMN_NAME,AUTO_INCREMENT FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =
```

'MyTable'

RowVersion Field

RowVersion字段是一个可选的用户定义字段，它提供行级版本控制，使可以确定对每个命名空间范围内的行中的数据进行更改的顺序。InterSystems IRIS维护一个整个命名空间范围的计数器，并在每次修改行数据（插入，更新或%Save）时向该字段分配一个唯一的增量正整数。因为此计数器是整个名称空间范围的，所以对具有ROWVERSION N字段的一个表进行的操作将设置ROWVERSION计数器的增量点，该值将用于同一名称空间中具有ROWVERSION字段的所有其他表。

通过指定数据类型为ROWVERSION（%Library.RowVersion）的字段来创建RowVersion字段。每个表只能指定一个ROWVERSION数据类型字段。尝试创建具有多个ROWVERSION字段的表会导致5320编译错误。

该字段可以具有任何名称，并且可以出现在任何列位置。

ROWVERSION（%Library.RowVersion）数据类型映射到BIGINT（%Library.BigInt）。

此字段从自动递增计数器接收一个从1开始的正整数。只要通过插入，更新或%Save操作修改了任何启用ROWVERSION的表中的数据，此计数器就会递增。递增的值记录在已插入或更新的行的ROWVERSION字段中。

名称空间可以包含具有RowVersion字段的表和不具有该字段的表。仅对具有RowVersion字段的表的数据更改会增加整个命名空间范围的计数器。

当用数据填充表时，InterSystems IRIS会为每个插入的行将此字段分配连续的整数。如果使用ALTER TABLE将ROWVERSION字段添加到已经包含数据的表中，则该字段将被创建为NULL以用于预先存在的字段。对该表的任何后续插入或更新都会为该行的RowVersion字段分配一个顺序整数。该字段是只读的；尝试修改RowVersion值会生成SQL CODE -138错误：无法为只读字段插入/更新值。因此，RowVersion字段被定义为唯一且不可修改，但不是必需字段或非null。

RowVersion值始终递增。它们不被重用。因此，插入和更新按时间顺序分配唯一的RowVersion值。删除操作从该序列中删除数字。因此，RowVersion值可能在数字上不连续。

此计数器永远不会重置。删除所有表数据不会重置RowVersion计数器。即使删除名称空间中包含ROWVERSION字段的所有表，也不会重置此计数器。

RowVersion字段不应包含在唯一键或主键中。RowVersion字段不能是IDKey索引的一部分。

分片表不能包含RowVersion字段。

RowVersion字段未隐藏（通过SELECT *显示）。

在同一名称空间中的三个表的以下示例中显示了这一点。

1. 创建表1和表3，每个都有一个ROWVERSION字段，并创建表2没有一个ROWVERSION字段。
2. 在Table1中插入十行。这些行的ROWVERSION值是接下来的十个计数器增量。由于以前未使用过计数器，因此它们是1到10。
3. 在Table2中插入十行。由于Table2没有ROWVERSION字段，因此计数器不会增加。
4. 更新表1的行。该行的ROWVERSION值将更改为下一个计数器增量（在这种情况下为11）。
5. 在Table3中插入十行。这些行的ROWVERSION值是接下来的十个计数器增量（12到21）。
6. 更新表1的行。该行的ROWVERSION值更改为下一个计数器增量（在这种情况下为22）。
7. 删除表1的行。ROWVERSION计数器不变。
8. 更新Table3的一行。该行的ROWVERSION值将更改为下一个计数器增量（在这种情况下为23）。

Serial Counter Field

可以使用SERIAL数据类型（在持久性类表定义中为%Library.Counter）来指定一个或多个可选的整数计数器字段，以记录在表中插入记录的顺序。每个串行计数器字段都维护自己的独立计数器。

每当将一行插入表中时，串行计数器字段都会从其自动增量计数器接收一个正整数，该行没有提供任何值（NULL）或值为0。但是，用户可以指定非零整数值插入期间针对此字段的值，将覆盖表计数器的默认值。

- 如果INSERT没有为计数器字段指定非零整数值，则计数器字段将自动接收正整数计数器值。计数从1开始。每个连续值都是从为此字段分配的最高计数器值开始的1增量。
- 如果INSERT为counter字段指定了一个非零的整数值，则该字段将接收该值。它可以是正整数或负整数，可以低于或高于当前计数器值，并且可以是已经分配给该字段的整数。如果该值大于任何分配的计数器值，它将自动增量计数器的增量起始点设置为该值。

尝试更新计数器字段值会导致SQLCODE -105错误。

TRUNCATE

TABLE命令将该计数器重置为1。即使使用DELETE命令删除表中的所有行，也不会通过DELETE命令将其重置。

分片表不能包含串行计数器字段。

AutoIncrement Field

可以使用%Library.AutoIncrement数据类型（或BIGINT AUTOINCREMENT）来指定一个整数计数器字段，以记录在表中插入记录的顺序。每个表只能指定一个%AutoIncrement数据类型字段。每当将一行插入表中时，此字段都会从自动增量计数器接收一个正整数，该行没有提供任何值（NULL）或值为0。但是，用户可以为此指定非零整数值插入过程中的字段，将覆盖表计数器的默认值。

- 如果INSERT没有为计数器字段指定非零整数值，则计数器字段将自动接收正整数计数器值。计数从1开始。每个连续值都是从为此字段分配的最高计数器值开始的1增量。
- 如果INSERT为counter字段指定了一个非零的整数值，则该字段将接收该值。它可以是正整数或负整数，可以低于或高于当前计数器值，并且可以是已经分配给该字段的整数。用户分配的值对自动增量计数器无效。

尝试更新计数器字段值会导致SQLCODE -105错误。

TRUNCATE

TABLE命令将该计数器重置为1。即使使用DELETE命令删除表中的所有行，也不会通过DELETE命令将其重置。

分片表可以包含一个AutoIncrement字段。

通过创建持久性类来定义表

在InterSystems IRIS中定义表的主要方法是使用Studio创建持久性类定义。当这些类在InterSystems IRIS数据库中保存并编译时，它们会自动投影到与类定义相对应的关系表中：每个类代表一个表；每个类代表一个表。每个属性代表一列，依此类推。可为一个类（表）定义的属性（列）的**最大数量**为1000。

例如，以下定义了持久类MyApp.Person：

```
Class MyApp.Person Extends %Persistent
{
Property Name As %String(MAXLEN=50) [Required];
Property SSN As %String(MAXLEN=15) [InitialExpression = "Unknown"];
Property DateOfBirth As %Date;
Property Sex As %String(MAXLEN=1);
}
```

编译后，这将在MyApp模式中创建MyApp.Person持久类和相应的SQL表Person。

在此示例中，指定了程序包名称MyApp。定义持久类时，未指定的程序包名称默认为User。这对应于默认的SQL模式名称SQLUser。例如，将名为“Students”的表定义为持久类将创建类User.Students，以及相应的SQL schema.table名称SQLUser.Students。

在此示例中，持久类名称Person是默认的SQL表名称。可以使用SqlTableName类关键字来提供其他SQL表名称。

可以使用DDL CREATE TABLE语句（指定SQL schema.table名称）定义相同的MyApp.Person表。成功执行此SQL语句会生成一个相应的持久性类，其包名称为MyApp，类名称为Person：

```
CREATE TABLE MyApp.Person (  
    Name VARCHAR(50) NOT NULL,  
    SSN VARCHAR(15) DEFAULT 'Unknown',  
    DateOfBirth DATE,  
    Sex VARCHAR(1)  
)
```

CREATE TABLE在相应的类定义中未指定显式的StorageStrategy。相反，它将采用已定义的默认存储策略。

默认情况下，CREATE TABLE在相应的类定义中指定Final class关键字，指示它不能具有子类。

请注意，诸如上图所示的持久性类定义在编译时会创建相应的表，但是无法使用SQL DDL命令（或通过使用Management Portal Drop操作）来修改或删除此表定义，这会向显示消息“未为类'schema.name'启用DDL ...”。必须在表类定义中指定[DdlAllowed]才能进行以下操作：

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

可以在类定义中指定%Populate以启用使用测试数据自动填充表。

```
Class MyApp.Person Extends (%Persistent,%Populate) [DdlAllowed]
```

这为该类提供了Populate（）方法。运行此方法将在表中填充十行测试数据。

定义数据值参数

每个属性（字段）定义都必须指定一个数据类型类，该类指定该属性所基于的类。指定的数据类型将字段的允许数据值限制为该数据类型。定义投影到表的持久类时，必须使用%Library包中的类指定此数据类型。可以将此类指定为%Library.Datatype或%Datatype。

许多数据类型类提供的参数使可以进一步定义允许的数据值。这些参数特定于单个数据类型。以下是一些较常见的数据定义参数：

- 数据值物理限制
- 允许的数据值：枚举或模式匹配
- 通过定义唯一索引来唯一数据值
- 通过定义SqlComputeCode计算数据值

数据值限制

对于数字数据类型，可以指定MAXVAL和MINVAL参数以限制允许值的范围。根据定义，数字数据类型具有最大支持值（正数和负数）。可以使用MAXVAL和MINVAL进一步限制允许的范围。

对于字符串数据类型，可以指定MAXLEN和MINLEN参数以限制允许的长度（以字符为单位）。根据定义，字符串数据类型具有最大支持的长度。可以使用MAXLEN和MINLEN进一步限制允许的范围。默认情况下，超过MAXLEN的数据值会生成字段验证错误：INSERT的SQLCODE -104或UPDATE的SQLCODE -105。可以指定TRUNCATE = 1以允许超过MAXLEN的字符串数据值。指定的字符串将被截断为MAXLEN长度。

允许的数据值

可以通过两种方式限制实际数据值：

- 允许值的列表（带有VALUELIST和DISPLAYLIST的枚举值）。
- 允许值的匹配模式（PATTERN）。

枚举值

通过将表定义为持久类，可以定义仅包含某些指定值的属性（字段）。这是通过指定VALUELIST参数来完成的。VALUELIST（指定逻辑存储值的列表）通常与DISPLAYLIST（指定相应的显示值的列表）一起使用。这两个列表都以列表定界符开头。几种数据类型可以指定VALUELIST和DISPLAYLIST。下面的示例定义两个带有枚举值的属性：

```
Class Sample.Students Extends %Persistent
{
Property Name As %String(MAXLEN=50) [Required];
Property DateOfBirth As %Date;
Property ChoiceStr As %String(VALUELIST="0,1,2", DISPLAYLIST="NO, YES, MAYBE");
Property ChoiceODBCStr As %EnumString(VALUELIST="0,1,2", DISPLAYLIST="NO, YES, MAYBE");
;
}
```

如果指定了VALUELIST，则INSERT或UPDATE只能指定VALUELIST中列出的值之一，或者不提供值（NULL）。VALUELIST有效值区分大小写。指定与VALUELIST值不匹配的数据值会导致字段值验证失败：INSERT的SQLCODE -104或UPDATE的SQLCODE -105。

在ODBC模式下显示时，%String和%EnumString数据类型的行为不同。使用上面的示例，当以逻辑模式显示时，ChoiceStr和ChoiceODBCStr都显示其VALUELIST值。在“显示”模式下显示时，ChoiceStr和ChoiceODBCStr均显示其DISPLAYLIST值。当以ODBC模式显示时，ChoiceStr显示VALUELIST值；否则显示VALUELIST值。ChoiceODBCStr显示DISPLAYLIST值。

值的模式匹配

几种数据类型可以指定PATTERN参数。PATTERN将允许的数据值限制为与指定的ObjectScript模式匹配的数据值，指定为带引号的字符串，省略前导问号。以下示例使用模式定义属性：

```
Class Sample.Students Extends %Persistent
{
Property Name As %String(MAXLEN=50) [Required];
Property DateOfBirth As %Date;
Property Telephone As %String(PATTERN = "3N1"-""3N1"-""4N");
}
```

由于将模式指定为带引号的字符串，因此模式中指定的文字必须将其双引号引起来。请注意，模式匹配是在MAXLEN和TRUNCATE之前应用的。因此，如果为可能超过MAXLEN并被截断的字符串指定了一个模式，则可能希望以“

.E ”（任何类型的尾随字符数不限）结束该模式。

与PATTERN不匹配的数据值会生成字段验证错误：INSERT的SQLCODE -104或UPDATE的SQLCODE -105。

唯一值

CREATE TABLE允许将字段定义为UNIQUE。这意味着每个字段值都是唯一（非重复）值。

将表定义为持久类不支持相应的uniqueness属性关键字。相反，必须同时定义属性和该属性的唯一索引。下面的示例为每个记录提供唯一的Num值：

```
Class Sample.CaveDwellers Extends %Persistent [ DdlAllowed ]
{
  Property Num As %Integer;
  Property Troglodyte As %String(MAXLEN=50);
  Index UniqueNumIdx On Num [ Type=index,Unique ];
}
```

索引名称遵循属性的命名约定。可选的Type关键字指定索引类型。 Unique关键字将属性（字段）定义为唯一。

使用INSERT或UPDATE语句时，必须具有唯一的值字段。

计算值

下面的类定义示例定义一个表，该表包含一个字段（生日），该字段在最初设置DateOfBirth字段值时使用SqlComputed来计算其值，而在更新DateOfBirth字段值时使用SqlComputeOnChange来重新计算其值。

Birthday字段值包括当前时间戳，以记录该字段值的计算/重新计算时间：

```
Class Sample.MyStudents Extends %Persistent [DdlAllowed]
{
  Property Name As %String(MAXLEN=50) [Required];
  Property DateOfBirth As %Date;
  Property Birthday As %String
    [ SqlComputeCode = {SET {Birthday}=$PIECE($ZDATE({DateOfBirth},9),",")_
      " changed: "_$ZTIMESTAMP},
      SqlComputed, SqlComputeOnChange = DateOfBirth ];
}
```java
```

```
?????`DateOfBirth`?`UPDATE`?????`DateOfBirth`???????`Birthday`?????
```

```
????????SerialObject?
```

```
??`MyData.Person`??
????`?SerialObject`????????`MyData.Person`????????????????`MyData.Person`????????????????`Hom
e`????????????????????????????
```

```
```java
```

```
Class MyData.Person Extends (%Persistent) [ DdlAllowed ]
{
  Property Name As %String(MAXLEN=50);
  Property Home As MyData.Address;
  Property Age As %Integer;
}
```

```
Class MyData.Address Extends (%SerialObject)
{
  Property Street As %String;
  Property City As %String;
  Property State As %String;
  Property PostalCode As %String;
}
```

不能直接访问串行对象属性中的数据，必须通过引用它的持久类/表访问它们：

- 要从持久性表中引用单个串行对象属性，请使用下划线。例如，SELECT名称HomeState FROM MyData.Person返回状态串行对象属性值作为字符串。串行对象属性值以查询中指定的顺序返回。
- 要引用持久性表中的所有串行对象属性，请指定引用字段。例如，SELECT Home FROM MyData.Person以%List结构形式返回所有MyData.Address属性的值。串行对象属性值以串行对象中指定的顺序返回：HomeStreet, HomeCity, HomeState, HomePostalCode。在Management Portal SQL界面“目录详细信息”中，此引用字段称为“容器”字段。这是一个Hidden字段，因此SELECT *语法不返回。
- 持久类的SELECT *单独返回所有串行对象属性，包括嵌套的串行对象。例如，SELECT * FROM MyData.Person返回Age, Name, HomeCity, HomePostalCode, HomeState和HomeStreet值（按此顺序）；它不返回Home %List结构值。串行对象属性值以排序顺序返回。SELECT
- *首先按排序顺序（通常按字母顺序）列出持久性类中的所有字段，然后按排序顺序列出嵌套的串行对象属性。

请注意，嵌入式串行对象不必与引用它的持久性表位于同一程序包中。

定义嵌入式对象可以简化持久性表定义：

- 持久表可以包含多个属性，这些属性引用同一嵌入式对象中的不同记录。例如，MyData.Person表可以包含Home和Office属性，这两个属性均引用MyData.Address串行对象类。
- 多个持久表可以引用同一嵌入式对象的实例。例如，MyData.Person表的Home属性和MyData.Employee Workplace属性都可以引用MyData.Address串行对象类。
- 一个嵌入式对象可以引用另一个嵌入式对象。例如，MyData.Address嵌入式对象包含引用MyData.Telephone嵌入式对象的Phone属性，其中包含CountryCode, AreaCode和PhoneNum属性。在持久类中，使用多个下划线来引用嵌套的串行对象属性，例如HomePhoneAreaCode。

编译串行对象类会在存储定义中生成数据规范。编译器通过在串行对象类名称后附加单词“State”来为该规范分配数据名称。因此，为MyData.Address分配了<Data name = “AddressState”>。如果此名称（在此示例中为AddressState）已经用作属性名称，则编译器将附加一个整数以创建唯一的数据名称：<Data name = “AddressState1”>。

类方法

可以将类方法指定为表定义的一部分，如下示例所示：

```
Class MyApp.Person Extends %Persistent
{
  Property Name As %String(MAXLEN=50) [Required];
  Property SSN As %String(MAXLEN=15) [InitialExpression = "Unknown"];
  Property DateOfBirth As %Date;
  Property Sex As %String(MAXLEN=1);
  ClassMethod Numbers() As %Integer [ SqlName = Numbers, SqlProc ]
  {
    QUIT 123
  }
}
```

在SELECT查询中，可以按以下方式调用此方法：

```
SELECT Name, SSN, Sample.Numbers() FROM Sample.Person
```

	Name	SSN	Expression_3
	VARCHAR (50)	VARCHAR (50)	INTEGER
1	yaoxin	111-11-1117	123
2	xiaoli	111-11-1111	123
3	姚鑫	111-11-1112	123
4	姚鑫	111-11-1113	123
5	姚鑫	111-11-1114	123
6	姚鑫	111-11-1115	123
7	姚鑫	111-11-1116	123
8	Isaacs Roberta Z	590-93-6378	123

通过创建持久性类来定义分片表

必须先建立分片环境，然后才能定义作为分片表投影的持久性类。

要将持久性类定义为分片，请指定类关键字 `Sharded = 1`。（类关键字 `Sharded = 2` 保留供生成的类内部使用。）

注意：请勿尝试设置或更改现有类定义的与分片相关的类属性。仅应为不包含数据的新表指定这些属性。这包括设置 `Sharded` 类关键字和与分片相关的索引关键字。尝试编辑现有类的任何与分片相关的属性都可能导致数据无法访问。

下例显示了 `Sharded = 1` 持久类的类定义：

```
Class Sample.MyShardT Extends %Persistent [ ClassType = persistent, DdlAllowed, Final
, Sharded = 1]
{
...
}
```

如果将一个类定义为分片，则它必须是持久性的 `ClassType`。如果未将分片类定义为 `ClassType` 持久类，则在类编译期间将返回错误，例如：ERROR # 5599：分片类 'Sample.Address' 必须为 `ClassType` 'persistent'，而不是 `ClassType` 'serial'。分片类使用的存储类必须为 `%Storage.Persistent` 或其子类 `%Storage.Shard`。如果分片类的存储类不是 `%Storage.Persistent`，则在类编译期间将返回以下错误：错误 # 5598：分片类 'Sample.Vendor' 必须使用存储类型 `%Storage.Persistent`，而不是存储类型 `%Storage.SQL`。

定义分片类时，应定义参数 `DEFAULTCONCURRENCY = 0`。

然后，可以定义 `ShardKey` 索引。

创建分片表时，将自动生成抽象的分片键索引。分片键索引的目的是用作确定行所在的分片的键。

分片类方法

分片类（`Sharded = 1`）支持 `%Library.Persistent` 方法 `%Open()`，`%OpenId()`，`%Save()`，`%Delete()` 和 `%DeleteId()` 具有以下限制：并发 `concurrency` 参数被忽略；删除将始终使用并发 `concurrency = 0`，而不管用户提供的并发值如何。完全支持回调方法 `%OnDelete()`，`%OnAfterDelete()`，`%OnOpen()`，`%OnBeforeSave()` 和 `%OnAfterSave()`。这些回调方法在分片主机上执行，而不是在分片服务器上执行。分片本地类（`Sharded = 2`）不支持这些方法。

分片类（`Sharded = 1`）不支持 `%Library.Persistent` 方法 `%LockExtent()` 和 `%UnlockExtent()`。定义并发参数的对象方法中的所有并发参数都要求值 `concurrency = 0`；否则，值为 0。可以通过设置 `DEFAULTCONCURRENCY = 0` 来建立默认值。

分片类限制

- 分片类不支持的类参数：CONNECTION，DEFAULTGLOBAL，DSINTERVAL，DSTIME，IDENTIFIEDBY，OBJJOURNAL。
- 分片类不支持的类关键字：language，ViewQuery。
- 分片类不支持的超级类：%Library.IndexBuilder，%DocDB.Document。
- 分片类不支持的属性数据类型：%Library.Text。
- 分片类不支持关系属性。
- 分片类不支持投影。
- 分片类不支持功能索引（无法定义索引TypeClass）。
- 分片类不支持使用除“对象”以外的语言的任何方法。
- 分片类不支持任何非%SQLQuery类型的类查询。

尝试使用任何这些功能来编译分片类都会导致编译时错误。

[#SQL](#) [#Caché](#) [#InterSystems IRIS](#) [#InterSystems IRIS for Health](#)

源

URL:

<https://cn.community.intersystems.com/post/%E7%AC%AC%E4%BA%94%E7%AB%A0-sql%E5%AE%9A%E4%B9%89%E8%A1%A8%EF%BC%88%E4%BA%8C%EF%BC%89>