#### 文章

姚 鑫 · 四月 6, 2021 阅读大约需 13 分钟

## 第十八章 定义和使用存储过程

## 第十八章 定义和使用存储过程

本章介绍如何在IntersystemsIRIS®数据平台上定义和使用Intersystems SQL中的存储过程。它讨论了以下内容:

- 存储过程类型的概述
- 如何定义存储过程
- 如何使用存储过程如
- 何列出存储过程及其参数。

### 概述

SQL例程是可执行的代码单元,可以由SQL查询处理器调用。 SQL例程有两种类型:功能和存储过程。从支持FunctionName()语法的任何SQL语句中调用函数。存储过程只能由CALL语句调用。函数接受某些输入定向参数并返回单个结果值。存储过程接受某些输入,输入输出和输出参数。存储过程可以是用户定义的函数,返回单个值。CALL语句也可以调用函数。

与大多数关系数据库系统一样,Intersystems Iris允许创建SQL存储过程。存储过程(SP)提供存储在数据库中的可调用可调用的程序,并且可以在SQL上下文中调用(例如,通过使用呼叫语句或通过ODBC或JDBC)。

与关系数据库不同,Intersystems Iris使可以将存储过程定义为类的方法。实际上,存储过程只不过是SQL可用的类方法。在存储过程中,可以使用基于对象的全系列Intersystems的功能。

- 可以通过查询数据库将存储过程定义为返回单个结果集数据集的查询。
- 可以将存储过程定义为可以用作用户定义函数的函数过程,返回单个值。
- 可以将存储过程定义为可以修改数据库数据并返回单个值或一个或多个结果集的方法。

可以确定使用 \$SYSTEM.SQL.Schema.ProcedureExists()方法是否已存在该过程。此方法还返回过程类型:"函数function"或"查询query"。

## 定义存储过程

与Intersystems

SQL的大多数方面一样,有两种方法可以定义存储过程:使用DDL和使用类。这些在以下部分中描述。

# 使用DDL定义存储过程

Intersystems SQL支持以下命令来创建查询:

- CREATE PROCEDURE可以创建始终作为存储过程投影的查询。 查询可以返回单个结果集。
- CREATE QUERY创建一个查询,该查询可以选择性地投影为存储过程。 查询可以返回单个结果集。

InterSystems SQL支持以下命令来创建方法或函数:

#### 第十八章 定义和使用存储过程

Published on InterSystems Developer Community (https://community.intersystems.com)

- CREATE PROCEDURE可以创建始终作为存储过程投影的方法。 方法可以返回单个值,也可以返回一个或多个结果集。
- CREATE METHOD可以创建一个方法,该方法可以选择投影为存储过程。 方法可以返回单个值,也可以返回一个或多个结果集。
- CREATE FUNCTION可以创建一个函数过程,该函数过程可以选择投影为存储过程。 函数可以返回单个值。

这些命令中指定的可执行代码块可以用InterSystems SQL或ObjectScript编写。可以在ObjectScript代码块中包含嵌入式SQL。

### SQL到类名转换

使用DDL创建存储过程时,指定的名称将转换为类名。 如果类不存在,系统将创建它。

- 如果名称是不限定的,并且没有提供FOR子句:使用系统范围的默认模式名作为包名,后跟一个点,后跟一个生成的类名,由字符串'func','meth','proc',or'query'组成,后跟去掉标点字符的SQL名。
   例如,未限定的过程名StoreName会产生如下类名User.procStoreName:
   这个过程类包含方法StoreName()。
- 如果名称是限定的,并且没有提供FOR子句:模式名被转换为包名,后跟一个点,后跟字符串'func', 'meth','proc', or 'query',后跟去掉标点字符的SQL名。 如果需要,将指定的包名转换为有效的包名。

如果名称是限定的,并且提供了FOR子句:在FOR子句中指定的限定类名将覆盖在函数、方法、过程或查询名称中指定的模式名。

SQL存储过程名称遵循标识符命名约定。
 InterSystems IRIS从SQL名称中去除标点字符,从而为过程类及其类方法生成唯一的类实体名称。

下面的规则管理模式名到有效包名的转换:

- 如果架构名称包含下划线,则此字符将转换为点,表示子包。例如,合格的名称myprocs.myname创建包myprocs。限定名称myprocs.myname创建了包含子包procs的包。

以下示例显示了标点符号在类名和SQL调用中的不同之处。它定义了一个包含包含两个点的类名的方法。从SQL中调用时,示例将第一个点替换为下划线字符:

```
Class Sample.ProcTest Extends %RegisteredObject
{
    ClassMethod myfunc(dummy As %String) As %String [ SqlProc ]
    {
        /* method code */
        Quit "abc"
    }
}

SELECT Sample.ProcTest_myfunc(Name)
FROM Sample.Person
```

# 使用类定义方法存储过程

类方法可以公开为存储过程。

这些是不返回数据的操作的理想选择,例如计算值并将其存储在数据库中的存储过程。 几乎所有类都可以将方法公开为存储过程; Published on InterSystems Developer Community (https://community.intersystems.com)

例外是生成器类,比如数据类型类([ClassType = datatype])。 生成器类没有运行时上下文。 只有在其他实体(如属性)的运行时中使用数据类型上下文才有效。

#### 要定义方法存储过程,只需定义一个类方法并设置其SqlProc关键字:

编译这个类之后,FindTotal()方法将作为存储过程MyApp.Person<u>F</u>indTotal()投影到SQL中。可以使用方法的SqlName关键字更改SQL对过程使用的名称。

该方法使用过程上下文处理程序在过程及其调用者(例如,ODBC服务器)之间来回传递过程上下文。 这个过程上下文处理程序是由InterSystems IRIS(作为%qHandle:%SQLProcContext)使用%sqlcontext对象自动生成的。

%sqlcontext由SQLCODE错误状态、SQL行数、错误消息等属性组成,使用相应的SQL变量设置,如下所示:

```
SET %sqlcontext.%SQLCode=SQLCODE
SET %sqlcontext.%ROWCOUNT=%ROWCOUNT
SET %sqlcontext.%Message=%msg
```

不需要对这些值做任何事情,但是它们的值将由客户机解释。 在每次执行之前都会重置%sqlcontext对象。

该方法不应该返回任何值。

一个类的用户定义方法的最大数目是2000个。

例如,假设有一个CalcAvgScore()方法:

```
ClassMethod CalcAvgScore(firstname As %String,lastname As %String) [sqlproc]
{
   New SQLCODE,%ROWID
   &sql(UPDATE students SET avgscore =
      (SELECT AVG(sc.score)
      FROM scores sc, students st
      WHERE sc.student_id=st.student_id
      AND st.lastname=:lastname
      AND st.firstname=:firstname)
   WHERE students.lastname=:lastname
```

```
AND students.firstname=:firstname)

IF ($GET(%sqlcontext)'= "") {
    SET %sqlcontext.%SQLCODE = SQLCODE
    SET %sqlcontext.%ROWCOUNT = %ROWCOUNT
  }
  QUIT
}
```

### 使用类定义查询存储过程

许多从数据库返回数据的存储过程可以通过标准查询接口实现。 只要可以用嵌入式SQL编写过程,这种方法就可以很好地工作。 注意,在以下示例中,使用了嵌入式SQL host变量为WHERE子句提供一个值:

```
Class MyApp.Person Extends %Persistent [DdlAllowed]
{
    /// This procedure result set is the persons in a specified Home_State, ordered b
y Name
    Query ListPersons(state As %String = "") As %SQLQuery [ SqlProc ]
    {
        SELECT ID,Name,Home_State
        FROM Sample.Person
        WHERE Home_State = :state
        ORDER BY Name
    }
}
```

要将查询公开为存储过程,可以将Studio

Inspector条目中的SQLProc字段的值更改为True,或者在查询定义中添加以下"[SQLProc]"字符串:

```
Query QueryName() As %SQLQuery( ... query definition ... )
    [ SqlProc ]
```

编译这个类之后,ListPersons查询将作为存储过程MyApp.PersonListPersons投影到SQL中。可以使用查询的SqlName关键字更改SQL用于该过程的名称。

当MyApp。

从SQL调用PersonListPersons,它将自动返回由查询的SQL语句定义的结果集。

下面是一个使用结果集的存储过程的示例:

```
Class apc.OpiLLS.SpCollectResults1 [ Abstract ]
{
/// This SP returns a number of rows (pNumRecs) from WebService.LLSResults, and updat
es a property for each record
Query MyQuery(pNumRecs As %Integer) As %Query(ROWSPEC = "Name: %String, DOB: %Date") [ S
qlProc ]
{
}
```

```
/// You put initial code here in the Execute method
ClassMethod MyQueryExecute(ByRef qHandle As %Binary, pNumRecs As %Integer) As %Status
    SET mysql="SELECT TOP ? Name, DOB FROM Sample.Person"
    SET rset=##class(%SQL.Statement).%ExecDirect(,mysql,pNumRecs)
            IF rset.%SQLCODE'=0 {QUIT rset.%SQLCODE}
    SET qHandle=rset
    QUIT $$$OK
}
/// This code is called by the SQL framework for each row, until no more rows are ret
ClassMethod MyQueryFetch(ByRef qHandle As %Binary, ByRef Row As %List,
                         ByRef AtEnd As %Integer = 0) As %Status [ PlaceAfter = NewQu
ery1Execute ]
{
     SET rset=qHandle
     SET tSC=$$$OK
     FOR {
        ///Get next row, quit if end of result set
        IF 'rset.%Next() {
                SET Row = "", AtEnd = 1
                SET tSC=$$$OK
                QUIT
        SET name=rset.Name
        SET dob=rset.DOB
        SET Row = $LISTBUILD(name,dob)
        QUIT
        QUIT tSC
}
ClassMethod MyQueryClose(ByRef qHandle As %Binary) As %Status [ PlaceAfter = NewQuery
1Execute 1
{
        KILL qHandle
                       //probably not necesary as killed by the SQL Call framework
        QUIT $$$OK
}
}
```

如果可以将查询编写为一个简单的SQL语句并通过查询向导创建它,那么就不需要了解实现查询的底层方法。

在后台,对于每个查询,类编译器都会根据存储过程的名称生成方法,包括:

- stored-procedure-nameExecute()
- stored-procedure-nameFetch()
- stored-procedure-nameFetchRows()
- stored-procedure-nameGetInfo()
- stored-procedure-nameClose()

如果查询类型为%SQLQuery,则类编译器会自动将一些嵌入式SQL插入到生成的方法中。 Execute()为SQL声明并打开存储的游标。 Fetch()被反复调用,直到它返回一个空行(SET row ="")。 还可以选择让Fetch()返回一个AtEnd=1布尔标志,以表明当前获取构成最后一行,下一个获取预期返回空行。

#### 第十八章 定义和使用存储过程

Published on InterSystems Developer Community (https://community.intersystems.com)

然而,应该总是使用空行(row ="")作为测试,以确定结果集何时结束; 当设置AtEnd=1时,应该始终设置Row=""。

FetchRows()在逻辑上等同于反复调用Fetch()。 调用GetInfo()返回存储过程签名的详细信息。 Close()关闭游标。

当从客户机调用存储过程时,会自动调用所有这些方法,但理论上可以从运行在服务器上的ObjectScript直接调用这些方法。

要将对象从Execute()传递给Fetch(),或从Fetch()传递给下一次调用Fetch(),可以将查询处理程序设置为希望传递的对象引用(oref)。

要传递多个对象,可以将qHandle设置为一个数组:

```
SET qHandle(1)=oref1,qHandle(2)=oref2
```

可以基于自定义编写的代码(而不是SQL语句)创建结果集存储过程。

对一个类的用户定义查询Query的最大数目是200。

### 自定义Query

对于复杂的查询或不适合查询模型的存储过程,通常需要通过替换查询的部分或全部方法来自定义查询。 你可以使用 %Library.Query。

如果选择类型%query (%Library.Query)而不是%SQLQuery (%Library.SQLQuery),则通常更容易实现查询。这生成了相同的5个方法,但是现在FetchRows()只是重复调用Fetch()(%SQLQuery进行了一些优化,导致了其他行为)。 GetInfo()只是从签名中获取信息,因此代码不太可能需要更改。 这将问题简化为为其他三个类中的每一个创建类方法。 请注意,在编译类时,编译器会检测到这些方法的存在,而不会覆盖它们。

这些方法需要特定的签名:它们都接受类型为%Binary的Qhandle(查询处理程序)。 这是一个指向保存查询的性质和状态的结构的指针。 它通过引用传递给Execute()和Fetch(),通过值传递给Close():

```
ClassMethod SP1Close(qHandle As %Binary) As %Status
{
    // ...
}

ClassMethod SP1Execute(ByRef qHandle As %Binary,
    p1 As %String) As %Status
{
    // ...
}

ClassMethod SP1Fetch(ByRef qHandle As %Binary,
    ByRef Row As %List, ByRef AtEnd As %Integer=0) As %Status
{
    // ...
}

Query SP1(p1 As %String)
    As %Query(CONTAINID=0,ROWSPEC="lastname:%String") [sqlproc]
```

{ }

代码通常包括SQL游标的声明和使用。

从类型为%SQLQuery的查询中生成的游标自动具有诸如Q14这样的名称。 必须确保查询具有不同的名称。

在尝试使用游标之前,类编译器必须找到游标声明。

因此,DECLARE语句(通常在Execute中)必须与Close和Fetch语句在同一个MAC例程中,并且必须出现在它们中的任何一个之前。

直接编辑源代码,在Close和Fetch定义中都使用方法关键字PLACEAFTER,以确保实现这一点。

错误消息引用内部游标名,它通常有一个额外的数字。

因此,游标Q140的错误消息可能指向Q14

# 使用存储过程

使用存储过程有两种不同的方式:

- 可以使用SQL CALL语句调用存储过程;
- 可以像使用SQL查询中的内置函数一样使用存储函数(即返回单个值的基于方法的存储过程)。

•

注意:当执行一个以SQL函数为参数的存储过程时,请使用CALL调用存储过程,示例如下:

```
CALL sp.MyProc(CURRENT_DATE)
```

SELECT查询不支持执行带有SQL函数参数的存储过程。 SELECT支持执行带有SQL函数参数的存储函数。

xDBC不支持使用SELECT或CALL来执行带有SQL函数参数的存储过程。

# 存储方法

存储函数是返回单个值的基于方法的存储过程。 例如,下面的类定义了一个存储函数Square,它返回给定值的平方:

```
Class MyApp.Utils Extends %Persistent [DdlAllowed]
{
    ClassMethod Square(val As %Integer) As %Integer [SqlProc]
     {
         Quit val * val
      }
}
```

存储的函数只是指定了SqlProc关键字的类方法。

注意:对于存储的函数,ReturnResultsets关键字必须不指定(默认)或以关键字not作为开头。

可以在SQL查询中使用存储函数,就像使用内置SQL函数一样。

Published on InterSystems Developer Community (https://community.intersystems.com)

函数的名称是存储函数(在本例中为"Square")的SQL名称,该名称由定义该函数的模式(包)名称限定(在本例中为"MyApp")。

下面的查询使用了Square函数:

```
SELECT Cost, MyApp.Utils_Square(Cost) As SquareCost FROM Products
```

如果在同一个包(模式)中定义了多个存储函数,则必须确保它们具有惟一的SQL名称。

下面的示例定义了一个名为Sample的表。

具有两个定义的数据字段(属性)和两个定义的存储函数TimePlus和DTime的工资:

```
Class Sample.Wages Extends %Persistent [ DdlAllowed ]
{
   Property Name As %String(MAXLEN = 50) [ Required ];
   Property Salary As %Integer;
   ClassMethod TimePlus(val As %Integer) As %Integer [ SqlProc ]
   {
     QUIT val * 1.5
   }
   ClassMethod DTime(val As %Integer) As %Integer [ SqlProc ]
   {
     QUIT val * 2
   }
}
```

下面的查询使用这些存储过程返回同一个表Sample.Wages中每个员工的Salary、time- half和double time工资率:

```
SELECT Name, Salary,
Sample.Wages_TimePlus(Salary) AS Overtime,
Sample.Wages DTime(Salary) AS DoubleTime FROM Sample.Wages
```

下面的查询使用这些存储过程返回不同表Sample.Employee中每个员工的Salary、time- half和double time工资率:

```
SELECT Name, Salary,
Sample.Wages_TimePlus(Salary) AS Overtime,
Sample.Wages_DTime(Salary) AS DoubleTime FROM Sample.Employee
```

### 权限

要执行一个过程,用户必须具有该过程的execute权限。 使用GRANT命令或\$SYSTEM.SQL.Security.GrantPrivilege()方法将指定过程的执行权限分配给指定用户。

通过调用\$SYSTEM.SQL.Security.CheckPrivilege()方法,可以确定指定的用户是否具有指定过程的执行权限。

要列出用户具有EXECUTE权限的所有过程,请转到管理门户。 从系统管理中选择Security,然后选择Users或Roles。 为所需的用户或角色选择Edit,然后选择SQL Procedures选项卡。 从下拉列表中选择所需的名称空间。 Published on InterSystems Developer Community (https://community.intersystems.com)

## List 存储过程

INFORMATION.SCHEMA.ROUTINES persistent类显示关于当前命名空间中所有例程和过程的信息。

当在嵌入式SQL中指定时,INFORMATION.SCHEMA。 例程需要#include %occlnclude宏预处理指令。 动态SQL不需要这个指令。

下面的例子返回例程名称、方法或查询名称、例程类型(过程或函数)、例程主体(SQL=class query with SQL, EXTERNAL=not a class query with

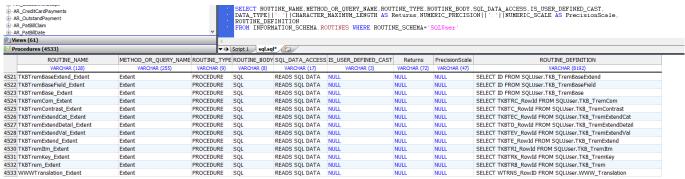
SQL)、返回数据类型,以及当前命名空间中模式 "Sample"中所有例程的例程定义:

SELECT ROUTINE\_NAME, METHOD\_OR\_QUERY\_NAME, ROUTINE\_TYPE, ROUTINE\_BODY, SQL\_DATA\_ACCESS, IS \_USER\_DEFINED\_CAST,

DATA\_TYPE||' '||CHARACTER\_MAXIMUM\_LENGTH AS Returns, NUMERIC\_PRECISION||':'||NUMERIC\_S CALE AS PrecisionScale,

ROUTINE\_DEFINITION

FROM INFORMATION\_SCHEMA.ROUTINES WHERE ROUTINE\_SCHEMA='SQLUser'



TNEORMATION:SOHEMA:PARAMETERS ETYPE.ROUTINE\_BODY, SQL\_DATA\_ACCESS,IS\_USER\_DEFINED\_CAST, DATA\_TYPE||"||CHARACTER\_MAXIMUM\_LENGTH AS Returns, NUMERIC\_PRECISION||"||NUMERIC\_SCALE AS Pressionscale, F persistent类显示关于当前命名空间中所有例程和过程的输入和输出参数的信息。

下面的示例返回例程名称、参数名称(不管是输入参数还是输出参数)以及当前命名空间中模式 "Sample"中的所有例程的参数数据类型信息:

SELECT SPECIFIC\_NAME, PARAMETER\_NAME, PARAMETER\_MODE, ORDINAL\_POSITION,

DATA\_TYPE, CHARACTER\_MAXIMUM\_LENGTH AS MaxLen, NUMERIC\_PRECISION | | ':' | | NUMERIC\_SCALE AS

PrecisionScale

FROM INFORMATION SCHEMA.PARAMETERS WHERE SPECIFIC SCHEMA='SQLUser'

使用管理门户SQL界面中的Catalog Details选项卡,可以为单个过程显示大部分相同的信息。 过程的目录详细信息包括过程类型(查询或函数)、类名称、方法或查询名称、描述以及输入和输出参数的数量。 目录详细信息存储过程信息显示还提供了运行存储过程的选项。

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

#### 源

URL:

https://cn.community.intersystems.com/post/%E7%AC%AC%E5%8D%81%E5%85%AB%E7%AB%A0-%E5%AE% 9A%E4%B9%89%E5%92%8C%E4%BD%BF%E7%94%A8%E5%AD%98%E5%82%A8%E8%BF%87%E7%A8% 8B