文章
姚 鑫 · 九月 21, 2021 阅读大约需 9 分钟

# 第二十二章 SQL命令 CREATE TRIGGER（二）

# 第二十二章 SQL命令 CREATE TRIGGER（二）

## SQL触发器代码

如果LANGUAGE SQL(默认)，被触发的语句是一个SQL过程块，包括一个SQL过程语句后跟一个分号，或者关键字BEGIN后跟一个或多个SQL过程语句，每个SQL过程语句后跟一个分号，最后以END关键字结束。

被触发的操作是原子的，它要么完全应用，要么根本不应用，并且不能包含COMMIT或ROLLBACK语句。
关键字BEGIN ATOMIC与关键字BEGIN是同义词。

如果语言是SQL, CREATE TRIGGER语句可以选择包含引用子句、WHEN子句和/或UPDATE OF子句。
UPDATE OF子句指定，只有在对为该触发器指定的一个或多个列执行UPDATE时，才应该执行该触发器。
带有LANGUAGE OBJECTSCRIPT的CREATE TRIGGER语句不能包含这些子句。

SQL触发器代码作为嵌入式SQL执行。
这意味着IRIS将SQL触发器代码转换为ObjectScript;
因此，如果查看与SQL触发器代码对应的类定义，将在触发器定义中看到Language=objectscript。

在执行SQL触发器代码时，系统会自动重置(NEWs)触发器代码中使用的所有变量。
在执行每条SQL语句之后 IRIS会检查SQLCODE。
如果发生错误，IRIS将%ok变量设置为0，终止并回滚触发器代码操作和相关的INSERT、UPDATE或DELETE。

## ObjectScript触发代码

如果LANGUAGE OBJECTSCRIPT，则CREATE TRIGGER语句不能包含引用子句、WHEN子句或UPDATE OF子句。
使用LANGUAGE OBJECTSCRIPT指定这些仅sql子句将分别导致编译时SQLCODE错误-49、-57或-50。

如果LANGUAGE
OBJECTSCRIPT，则触发语句是一个由一个或多个OBJECTSCRIPT语句组成的块，用花括号括起来。

因为触发器的代码不是作为过程生成的，所以触发器中的所有局部变量都是公共变量。
这意味着触发器中的所有变量都应该用NEW语句显式声明;
这可以防止它们在调用触发器的代码中与变量发生冲突。

如果触发器代码包含宏预处理器语句(#命令、##函数或$$$宏引用)，这些语句将在CREATE trigger DDL代码本身之前编译。

ObjectScript触发器代码可以包含嵌入式SQL。

通过将%ok变量设置为0，可以在触发器代码中发出错误。
这将创建一个运行时错误，该错误将中止并回滚触发器的执行。
它生成适当的SQLCODE错误(例如，SQLCODE -131 " After insert trigger failed ")，并返回用户指定的%msg变量的值作为字符串，以描述触发代码错误的原因。
请注意，将%ok设置为非数字值将设置%ok=0。

即使是多事件触发器，系统也只生成一次触发器代码。

## 字段引用和伪字段引用

在ObjectScript中编写的触发器代码可以包含字段引用，指定为{fieldname}，其中fieldname指定当前表中已有的字段。
花括号内不允许有空格。

你可以在字段名后面加上*N (new)， *O (old)，或*C (compare)来指定如何处理插入、更新或删除的字段数据值，如下所示:

- {fieldname*N}
  - 对于UPDATE，在进行指定更改后返回新的字段值。
  - 对于INSERT，返回插入的值。
  - 对于DELETE，返回删除前的字段值。
- {fieldname*O}
  - 对于UPDATE，返回进行指定更改之前的旧字段值。
  - 对于INSERT，返回NULL。
  - 对于DELETE，返回删除前的字段值。
- {fieldname*C}
  - 对于UPDATE，如果新值与旧值不同，则返回1(TRUE)，否则返回0(FALSE)。
  - 对于INSERT，如果插入的值非NULL，则返回1(TRUE)，否则返回0(FALSE)。
  - 对于DELETE，如果要删除的值非NULL，则返回1(TRUE)，否则返回0(FALSE)。

对于UPDATE、INSERT或DELETE，{fieldname}返回与{fieldname*N}相同的值。

例如，以下触发器返回插入到Sample.Employee中的新行的Name字段值。(可以从SQL Shell执行插入以查看此结果)：

```
CREATE TRIGGER InsertNameTrig AFTER INSERT ON Sample.Employee
   LANGUAGE OBJECTSCRIPT
   {WRITE "The employee ",{Name*N}," was ",{%%OPERATION},"ed on ",{%%TABLENAME},!}
```

在设置字段值的语句中不允许回车。

可以使用GetAllColumns()方法列出为表定义的字段名称。

用ObjectScript编写的触发器代码还可以包含伪字段引用变量{%%CLASSNAME}、{%%CLASSNAMEQ}、{%%OPERATION}、{%%TABLENAME}和{%%ID}。伪字段在类编译时被转换为特定值。所有这些伪字段关键字都不区分大小写。

- {%%CLASSNAME}和{%%CLASSNAMEQ}都转换为投影SQL表定义的类的名称。{%%CLASSNAME}返回不带引号的字符串，{%%CLASSNAMEQ}返回带引号的字符串。
- 根据调用触发器的操作，{%%operation}转换为字符串文字，可以是INSERT、UPDATE或DELETE。
- {%%TABLENAME}转换为表的完全限定名称。
- {%%ID}转换为RowID名称。当不知道RowID字段的名称时，此引用非常有用。

## 引用流属性

在触发器定义(如{StreamField}、{StreamField*O}或{StreamField*N})中引用流字段/属性时，{StreamField}引用的值是流的OID(对象ID)值。

对于BEFORE INSERT或BEFORE UPDATE触发器，如果INSERT/UPDATE/ObjectSave指定了新值，则{StreamField*N}值将是临时流对象的OID或新的文字流值。对于BEFORE UPDATE触发器，如果没有为流字段/属性指定新值，

则{StreamField*O}和{StreamField*N}都将是当前字段/属性流对象的OID。

## 引用SQLComputed属性

当触发器定义中引用瞬态SqlComputed字段/属性(“ calculate ”或显式地“ transient ”)时，触发器不会识别Get()/Set()方法覆盖。
使用SQLCOMPUTED/SQLCOMPUTONCHANGE，而不是覆盖属性的Get()或Set()方法。

使用Get()/Set()方法覆盖可能会导致以下错误结果:{property*O}值是用SQL确定的，没有使用覆盖的Get()/Set()方法。
因为属性没有存储在磁盘上，{property*O}使用SqlComputeCode“ 重新创建 ”旧值。
然而，{property*N}使用覆盖的Get()/Set()方法来访问属性的值。
因此，即使属性实际上没有改变，也有可能{property*O}和{property*N}是不同的(因此{property*C}=1)。

## 标签

触发器代码可能包含行标签(标签)。
若要在触发器代码中指定标签，请在标签行前面加上冒号，以指示该行应从第一列开始。
IRIS去掉冒号并将其余行作为标签处理。
但是，因为触发器代码是在任何过程块的作用域之外生成的，所以在整个类定义中每个标签必须是唯一的。
编译到类例程中的任何其他代码都不能定义相同的标签，包括在其他触发器、非过程块方法、SqlComputeCode和其他代码中。

注意:对标签使用冒号前缀要优先于对主机变量引用使用冒号前缀。
为了避免这种冲突，建议嵌入式SQL触发器代码行永远不要以主机变量引用开始。
如果必须以主机变量引用开始触发器代码行，可以通过加倍冒号前缀将其指定为主机变量(而不是标签)。

## 方法调用

可以从触发器代码中调用类方法，因为类方法不依赖于开放对象。
必须使用##class(classname).Method()语法来调用方法。
不能使用..Method()语法，因为该语法需要当前打开的对象。

可以将当前行字段的值作为类方法的参数传递，但类方法本身不能使用字段语法。

# 列出现有触发器

可以使用INFORMATION.SCHEMA.TRIGGERS类列出当前定义的触发器。
这个类列出每个触发器的名称、关联的模式和表名称以及触发器创建时间戳。
对于每个触发器，它列出EVENTMANIPULATION属性(INSERT, UPDATE, DELETE, INSERT/UPDATE, INSERT/UPDATE/DELETE)和ACTIONTIMING属性(BEFORE, AFTER)。
它还列出了ACTIONSTATEMENT，这是生成的SQL触发器代码。

# 引发运行时错误

触发器及其调用事件作为单个行上的原子操作执行。
那就是:
- 回滚触发器失败之前，不执行关联的INSERT、UPDATE或DELETE操作，并释放该行上的所有锁。
- 回滚失败的AFTER触发器，回滚关联的INSERT、UPDATE或DELETE操作，并释放该行上的所有锁。
- 回滚失败的INSERT、UPDATE或DELETE操作，回滚关联的BEFORE触发器，释放该行上的所有锁。
- 回滚失败的INSERT、UPDATE或DELETE操作，不执行关联的AFTER触发器，释放该行上的所有锁。

请注意，仅为当前行操作维护完整性。应用程序必须使用事务处理语句处理涉及多行操作的数据完整性问题。

因为触发器是原子操作，所以不能在触发器代码中编写事务语句(如COMMIT和ROLLBACKS)。

如果INSERT、UPDATE或DELETE操作导致执行多个触发器，则一个触发器失败会导致所有其余触发器保持未执行状态。
- SQLCODE-415：如果触发器代码中存在错误(例如，对不存在的表或未定义的变量的引用)，则触发器代码的执行在运行时会失败， IRIS会发出SQLCODE-415错误 “ FATAL ERROR OVERT INGRED INTERT SQL FILER ”。
- SQLCODE-130到-135：当触发器操作失败时， IRIS在运行时发出SQLCODE错误代码-130到-135之一，指示失败的触发器类型。可以通过在触发器代码中将%ok变量设置为0来强制触发器失败。这将发出相应的SQLCODE错误(例如，SQLCODE-131 “ AFTER INSERT TRIGGER
FAILED ”)，并以字符串形式返回用户指定的%msg变量值，以描述触发器代码错误的原因。

# 示例

下面的示例演示使用ObjectScript DELETE触发器创建触发器。它假设有一个包含记录的数据表(TestDummy)。它使用嵌入式SQL创建一个日志表(TestDummyLog)和一个删除触发器，该触发器在对数据表执行删除操作时写入日志表。触发器插入数据表的名称、已删除行的RowId、当前日期和执行的操作类型(%oper特殊变量)，在本例中为 “ DELETE ”：

```
ClassMethod CreateTrigger()
{
    &sql(
        CREATE TABLE TestDummyLog
        (
            TableName VARCHAR(40),
            IDVal INTEGER,
            LogDate DATE,
            Operation VARCHAR(40)
        )
    )
    w !,"SQL??????: ",SQLCODE

    &sql(
        CREATE TRIGGER TrigTestDummy AFTER DELETE ON TestDummy
            LANGUAGE OBJECTSCRIPT
            {
                NEW id
                SET id = {ID}
                &sql(
                    INSERT INTO TestDummyLog
                    (
                        TableName, IDVal, LogDate, Operation
                    )
                    VALUES
                    (
                        'TestDummy', :id, +$HOROLOG, :%oper)
                    )
            }
    )
    w !,"SQL??????: ",SQLCODE
}
```

以下示例演示了使用SQL INSERT触发器的CREATE TRIGGER。第一个嵌入式SQL程序创建表、该表的插入触发器和日志表以供触发器使用。第二个嵌入式SQL程序针对该表发出INSERT命令，该命令调用触发器，该触发器在日志表中记录一个条目。显示日志条目后，程序将删除这两个表，以便可以重复运行此程序：

```
ClassMethod CreateTrigger1()
{
    d $SYSTEM.Security.Login("_SYSTEM","SYS")
    &sql(
        CREATE TABLE TestDummy
        (
            testnum     INT NOT NULL,
            firstword   CHAR (30) NOT NULL,
            lastword    CHAR (30) NOT NULL,
            CONSTRAINT TestDummyPK PRIMARY KEY (testnum)
        )
    )
    w !,"SQL????: ",SQLCODE
    &sql(
        CREATE TABLE TestDummyLog
        (
            entry CHAR (60) NOT NULL
        )
    )
    w !,"SQL??????: ",SQLCODE
    &sql(
        CREATE TRIGGER TrigTestDummy AFTER INSERT ON TestDummy
        LANGUAGE SQL
        BEGIN
            INSERT INTO TestDummyLog
            (
                entry
            )
            VALUES
            (
                CURRENT_TIMESTAMP||' INSERT to TestDummy'
            );
        END
    )
    w !,"SQL??????: ",SQLCODE
}


ClassMethod CreateTrigger2()
{
    n SQLCODE, %ROWCOUNT, %ROWID
    &sql(
        INSERT INTO sqluser.TestDummy
        (
            testnum, firstword, lastword
        )
        VALUES
        (
            46639, 'hello', 'goodbye'
        )
    )
    if SQLCODE = 0 {
        w !,"Insert succeeded"
        w !,"Row count=",%ROWCOUNT
        w !,"Row ID=",%ROWID
    } else {
        w !,"Insert failed, SQLCODE=",SQLCODE
```

```
    }
    &sql(
        SELECT entry INTO :logitem FROM sqluser.TestDummyLog
    )
    w !,"Log entry: ",logitem
    &sql(DROP TABLE TestDummy)
    &sql(DROP TABLE TestDummyLog)
    w !,"finished!"
}
```

下面的示例包括一个WHEN子句，该子句指定只有在满足括号中的谓词条件时才应执行操作：

```
CREATE TRIGGER Trigger_2 AFTER INSERT ON Table_1
  WHEN (f1 %STARTSWITH 'A')
  BEGIN
    INSERT INTO Log_Table VALUES (new_row.Category);
  END
```

以下示例定义在Sample.Employee中插入、更新或删除行后返回旧名称字段值和新名称字段值的触发器。(可以在SQL Shell中执行触发事件操作来查看此结果)：

```
CREATE TRIGGER EmployNameTrig AFTER INSERT,UPDATE,DELETE ON Sample.Employee
   LANGUAGE OBJECTSCRIPT
   {WRITE "Employee old name:",{Name*O}," new name:",{Name*N}," ",{%%OPERATION}," on
",{%%TABLENAME},!}
```

#SQL #Caché

---

**源URL:**
https://cn.community.intersystems.com/post/%E7%AC%AC%E4%BA%8C%E5%8D%81%E4%BA%8C%E7%AB%A0-sql%E5%91%BD%E4%BB%A4-create-trigger%EF%BC%88%E4%BA%8C%EF%BC%89