文章
姚 鑫 · 六月 8, 2022 阅读大约需 4 分钟

第三章 数据类型（二）

第三章 数据类型（二）

SQL 系统数据类型映射

上表中为 DDL 和 IRIS 数据类型表达式显示的语法是为 SQL.SystemDataTypes
配置的默认映射。对于提供的系统数据类型和用户数据类型，有单独的映射表可用。

要查看和修改当前数据类型映射，请转到管理门户，选择系统管理、配置、SQL 和对象设置、系统 DDL 映射。

了解 DDL 数据类型映射

将数据类型从 DDL 映射到 IRIS 时，常规参数和函数参数遵循以下规则：

- 常规参数 - 这些在 DDL 数据类型和 IRIS 数据类型中以 %# 格式标识。例如：

  ```
  VARCHAR(%1)
  ```

映射到：

```
%String(MAXLEN=%1)
```

因此，DDL 数据类型为：

```
VARCHAR(10)
```

映射到：

```
%String(MAXLEN=10)
```

- 函数参数 — 当 DDL 数据类型中的参数必须经过一些转换才能放入 IRIS
  数据类型中时，使用这些参数。这方面的一个例子是将 DDL 数据类型的数值精度和比例参数转换为 IRIS
  数据类型的 MAXVAL、MINVAL 和 SCALE 参数。例如：

  ```
  DECIMAL(%1,%2)
  ```

映射到：

```
%Numeric(MAXVAL=<|'$$maxval^%apiSQL(%1,%2)'|>,
         MINVAL=<|'$$minval^%apiSQL(%1,%2)'|>,
         SCALE=%2)
```

DDL 数据类型 DECIMAL 采用参数 Precision (%1) 和 Scale (%2)，但IRIS 数据类型 %Numeric 没有精度参数。因此，要将 DECIMAL 转换为 %Numeric，必须将 Precision 参数转换为适当的 %Numeric 参数，在这种情况下，通过将IRIS 函数格式、maxval 和 minval 应用于 DECIMAL 提供的参数。特殊的 <|'xxx'|>语法（如上所示）指示 DDL 处理器进行参数替换，然后使用提供的值调用函数。 <|'xxx'|> 表达式随后被函数调用返回的值替换。

考虑这个具有实际值的示例，可能存在精度为 4 位、小数位数为 2 的 DECIMAL 数据类型：

```
DECIMAL(4,2)
```

映射到：

```
%Numeric(MAXVAL=<|'$$maxval^%apiSQL(4,2)'|>,
         MINVAL=<|'$$minval^%apiSQL(4,2)'|>,
         SCALE=2)
```

计算为：

```
%Numeric(MAXVAL=99.99,MINVAL=-99.99,SCALE=2)
```

- 附加参数——数据类型类可以定义不能使用 DDL 数据类型定义的附加数据定义参数。这些包括数据验证操作，例如允许的数据值的枚举列表、允许的数据值的模式匹配以及超过 MAXLEN 最大长度的数据值的自动截断。

数据类型优先级

当一个操作可以返回多个不同的值，并且这些值可能具有不同的数据类型时，IRIS 将返回值分配给具有最高优先级的数据类型。例如，NUMERIC 数据类型可以包含所有可能的 INTEGER 数据类型值，但 INTEGER 数据类型不能包含所有可能的 NUMERIC 数据类型值。因此 NUMERIC 具有更高的优先级（更具包容性）。

例如，如果 CASE 语句有一个数据类型为 INTEGER 的可能结果值，以及一个数据类型为 NUMERIC 的可能结果值，则无论采用这两种情况中的哪一种，实际结果始终为 NUMERIC 类型。

数据类型的优先级如下，从最高（包括最多）到最低：

```
LONGVARBINARY
LONGVARCHAR
VARBINARY
VARCHAR
GUID
TIMESTAMP
DOUBLE
NUMERIC
BIGINT
INTEGER
```

```
DATE
TIME
SMALLINT
TINYINT
BIT
```

规格化和验证

%Library.DataType超类包含特定数据类型的类。这些数据类型类提供了 Normalize()
方法来将输入值规范化为数据类型格式，并提供 IsValid()
方法来确定输入值是否对该数据类型有效，以及各种模式转换方法，例如 LogicalToDisplay( ) 和
DisplayToLogical()。

以下示例显示了%TimeStamp 数据类型的 Normalize() 方法：

```
/// d ##class(PHA.TEST.SQLFunction).NormalizeValidate()
ClassMethod NormalizeValidate()
{
    s indate = 64701
    s tsdate = ##class(%Library.TimeStamp).Normalize(indate)
    w "%TimeStamp date: ",tsdate
}
```

```
DHC-APP>d ##class(PHA.TEST.SQLFunction).NormalizeValidate()
%TimeStamp date: 2018-02-22 00:00:00
```

```
/// d ##class(PHA.TEST.SQLFunction).NormalizeValidate2()
ClassMethod NormalizeValidate2()
{
    s indate = "2018-2-22"
    s tsdate = ##class(%Library.TimeStamp).Normalize(indate)
    w "%TimeStamp date: ",tsdate
}
```

```
DHC-APP> d ##class(PHA.TEST.SQLFunction).NormalizeValidate2()
%TimeStamp date: 2018-02-22 00:00:00
```

以下示例显示了 %TimeStamp 数据类型的 IsValid() 方法：

```
/// d ##class(PHA.TEST.SQLFunction).NormalizeValidate3()
ClassMethod NormalizeValidate3()
{
    s datestr = "July 4, 2018"
    s stat = ##class(%Library.TimeStamp).IsValid(datestr)
    if stat = 1 {
        w datestr," is a valid %TimeStamp",!
    } else {
        w datestr," is not a valid %TimeStamp",!
    }
}
```

```
DHC-APP> d ##class(PHA.TEST.SQLFunction).NormalizeValidate3()
July 4, 2018 is not a valid %TimeStamp


/// d ##class(PHA.TEST.SQLFunction).NormalizeValidate4()
ClassMethod NormalizeValidate4()
{
    s leapdate = "2016-02-29 00:00:00"
    s noleap = "2018-02-29 00:00:00"
    s stat = ##class(%Library.TimeStamp).IsValid(leapdate)
    if stat = 1 {
        w leapdate," is a valid %TimeStamp",!
    } else {
        w leapdate," is not a valid %TimeStamp",!
    }
    s stat = ##class(%Library.TimeStamp).IsValid(noleap)
    if stat = 1 {
        w noleap," is a valid %TimeStamp",!
    } else {
        w noleap," is not a valid %TimeStamp",!
    }
}


DHC-APP> d ##class(PHA.TEST.SQLFunction).NormalizeValidate4()
2016-02-29 00:00:00 is a valid %TimeStamp
2018-02-29 00:00:00 is not a valid %TimeStamp
```

#SQL #Caché

源
URL:
https://cn.community.intersystems.com/post/%E7%AC%AC%E4%B8%89%E7%AB%A0-%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B%EF%BC%88%E4%BA%8C%EF%BC%89