

文章

姚鑫 · 六月 9, 2022 阅读大约需 6 分钟

第四章 数据类型（三）

第四章 数据类型（三）

日期、时间、PosixTime 和时间戳数据类型

可以定义日期、时间和时间戳数据类型，并通过标准 SQL 日期和时间函数相互转换日期和时间戳。例如，可以使用 `CURRENTDATE` 或 `CURRENTTIMESTAMP` 作为使用该数据类型定义的字段的输入，或者使用 `DATEADD`、`DATEDIFF`、`DATENAME` 或 `DATEPART` 来操作使用该数据类型存储的日期值。

数据类型类 `%Library.Date`、`%Library.Time`、`%Library.PosixTime`、`%Library.TimeStamp` 和 `%MV.Date` 对于 `SqlCategory` 的处理方式如下：

1. `%Library.Date` 类以及逻辑值为 `+$HOROLOG ($HOROLOG` 的日期部分) 的任何用户定义数据类型类都应使用 `DATE` 作为 `SqlCategory`。默认情况下，`DATE` 和对应的 `%Library.Date` 数据类型只接受正整数，0 代表 1840-12-31。要支持早于 1840-12-31 的日期，必须在表中定义数据类型为 `%Library.Date(MINVAL=-nnn)` 的日期字段，其中 `MINVAL` 是从 1840-12-31 倒数的负天数最大为 -672045 (0001-01-01)。`%Library.Date` 可以将日期值存储为 -672045 到 2980013 范围内的无符号或负整数。日期值可以按如下方式输入：
 - 逻辑模式接受 `+$HOROLOG` 整数值，例如 65619 (2020 年 8 月 28 日)。
 - 显示模式使用 `DisplayToLogical()` 转换方法。它接受当前语言环境的显示格式的日期，例如 “8/28/2020”。它还接受逻辑日期值 (`+$HOROLOG` 整数值)。
 - ODBC 模式使用 `ODBCToLogical()` 转换方法。它接受 ODBC 标准格式的日期，例如 “2020-08-28”。它还接受逻辑日期值 (`+$HOROLOG` 整数值)。
2. `%Library.Time` 类和任何逻辑值为 `$PIECE($HOROLOG, ",", 2)` (`$HOROLOG` 的时间部分) 的用户定义数据类型类都应使用 `TIME` 作为 `SqlCategory`。`%Library.Time` 将时间值存储为 0 到 86399 范围内的无符号整数 (自午夜以来的秒数)。时间值可以按如下方式输入：
 - 逻辑模式接受 `$PIECE($HOROLOG, ",", 2)` 整数值，例如 84444 (23:27:24)。
 - 显示模式使用 `DisplayToLogical()` 转换方法。它接受当前语言环境的显示格式的时间，例如 “23:27:24”。
 - ODBC 模式使用 `ODBCToLogical()` 转换方法。它接受 ODBC 标准格式的时间，例如 “23:27:24”。它还接受逻辑时间值 (0 到 86399 范围内的整数)。

`TIME` 支持小数秒，因此此数据类型也可用于 `HH:MM:SS.FF` 到用户指定的精度 (F) 小数位数，最多为 9。要支持小数秒，请设置 `PRECISION` 范围。例如，`TIME(0) (%Time(PRECISION=0))` 舍入到最接近的秒数；`TIME(2) (%Time(PRECISION=2))` 将 (或零填充) 四舍五入到精度的两位小数。

如果提供的数据还指定了精度 (例如，`CURRENTTIME(3)`)，则存储的小数位数如下：

- 如果 `TIME` 未指定精度，而数据指定了精度，则使用数据的精度。
- 如果 `TIME` 未指定精度且数据未指定精度，则使用系统范围配置的时间精度。
- 如果 `TIME` 指定精度而数据未指定精度，则使用系统范围配置的时间精度作为数据精度。
- 如果 `TIME` 指定了精度并且数据精度小于 `TIME` 精度，则使用数据精度。
- 如果 `TIME` 指定了精度并且数据精度大于 `TIME` 精度，则使用 `TIME` 精度。

SQL 元数据将时间精度的小数位报告为 “scale”；它使用 “precision 精度” 一词来表示数据的总长度。使用 `TIME` 数据类型的字段报告精度和比例元数据如下：`TIME(0) (%Time(PRECISION=0))` 的元数据精度为 8

(nn:nn:nn), 比例为 0。TIME(2) (%Time(PRECISION=2)) 的元数据精度为 11 (nn:nn:nn:ff), 比例为 2。TIME (%Time 或 %Time(PPRECISION=*)) 采用其小数秒精度来自提供的数据, 因此元数据精度为 18 和未定义的比例。

3. %Library.PosixTime 类和任何具有编码的有符号 64 位整数逻辑值的用户定义数据类型类都应使用 POSIXTIME 作为 SqlCategory。 %PosixTime 是从 1970-01-01 00:00:00 以来的秒数 (和 小数秒) 计算的编码时间戳。该日期之后的时间戳由正 %PosixTime 值表示, 该日期之前的时间戳由负 %PosixTime 值表示。 %PosixTime 支持最多 6 位精度的小数秒。 %PosixTime 支持的最早日期为 0001-01-01 00:00:00, 其逻辑值为 -6979664624441081856。支持的最后日期为 9999-12-31 23:59:59.999999, 其逻辑值为 1406323805406846975。

因为 %PosixTime 值始终由编码的 64 位整数表示, 所以它始终可以明确地区分于 %Date 或 %TimeStamp 值。例如, 1970-01-01 00:00:00 的 %PosixTime 值为 1152921504606846976, 2017-01-01 00:00:00 的 %PosixTime 值为 1154404733406846976, 1969-12-01 的 %PosixTime 值 00:00:00 是 -6917531706041081856。

%PosixTime 比 %TimeStamp 更可取, 因为它比 %TimeStamp 数据类型占用更少的磁盘空间和内存, 并且提供比 %TimeStamp 更好的性能。

可以使用 ODBC 显示模式集成 %PosixTime 和 %TimeStamp 值:

- %PosixTime 和 %TimeStamp 数据类型的逻辑模式值完全不同: %PosixTime 是有符号整数, %TimeStamp 是包含 ODBC 格式时间戳的字符串。
- 显示方式: %PosixTime 显示使用当前 locale 时间和日期格式参数 (例如 02/22/2018 08:14:11); %TimeStamp 显示为 ODBC 格式的时间戳。
- ODBC 模式: %PosixTime 和 %TimeStamp 都显示为 ODBC 格式的时间戳。精度的小数位数可能不同。

可以使用 TO_POSIXTIME 函数或 TO_POSIXTIME() 方法将 %TimeStamp 值转换为 %PosixTime。可以使用 IsValid() 方法来确定数值是否为有效的 %PosixTime 值。

4. %Library.TimeStamp 类和任何具有 YYYY-MM-DD HH:MM:SS.FF 逻辑值的用户定义数据类型类都应使用 TIMESTAMP 作为 SqlCategory。请注意, %Library.TimeStamp 的最大精度来自系统平台的精度, 最多为 9 位小数秒, 而 %Library.PosixTime 的最大精度为 6 位。因此, 在某些平台上, %Library.TimeStamp 可能比 %Library.PosixTime 更精确。 %Library.TimeStamp 规范化会自动将精度超过 9 位的输入值截断为 9 位小数秒。
5. %Library.DateTime 是 %Library.TimeStamp 的子类。它定义了一个名为 DATEFORMAT 的类型参数, 它覆盖了 DisplayToLogical() 和 OdbcToLogical() 方法来处理 TSQL 应用程序习惯的不精确的日期时间输入。
6. %MV.Date 类, 或任何具有 \$HOROLOGY-46385 逻辑日期值的用户定义数据类型类, 应使用 MVDATE 作为 SqlCategory。
7. 不适合上述任何逻辑值的用户定义日期数据类型应将数据类型的 SqlCategory 定义为 DATE, 并在数据类型类中提供 LogicalToDate() 方法以将用户定义的逻辑日期值转换为 %Library.Date 逻辑值和 DateToLogical() 方法, 用于将 %Library.Date 逻辑值转换为用户定义的逻辑日期值。
8. 不适合上述任何逻辑值的用户定义时间数据类型应将数据类型的 SqlCategory 定义为 TIME, 并在数据类型类中提供 LogicalToTime() 方法以将用户定义的逻辑时间值转换为 %Library.Time 逻辑值和 TimeToLogical() 方法, 用于将 %Library.Time 逻辑值转换为用户定义的逻辑时间值。
9. 不适合上述任何逻辑值的用户定义时间戳数据类型应将数据类型的 SqlCategory 定义为 TIMESTAMP, 并在数据类型类中提供 LogicalToTimeStamp() 方法以将用户定义的逻辑时间戳值转换为 %Library.TimeStamp 逻辑值和 TimeStampToLogical() 方法, 用于将 %Library.TimeStamp 逻辑值转换为用户定义的逻辑时间戳值。

可以使用 =, <, >, < 运算符将 POSIXTIME 与 DATE 或 TIMESTAMP 值进行比较。

在将 FMTIMESTAMP 类别值与 DATE 类别值进行比较时, IRIS 在将其与 DATE 进行比较之前不会从 FMTIMESTAMP 值中去除时间。这与比较 TIMESTAMP 与 DATE 值以及比较 TIMESTAMP 与 MVDATE 值的行为相同。它还与其他 SQL 供应商比较时间戳和日期的方式兼容。这意味着当使用 SQL 相等 (=)

运算符进行比较时，FMTIMESTAMP 320110202.12 和 DATE 62124 的比较相等。应用程序必须将 FMTIMESTAMP 值转换为 DATE 或 FMDATE 值以仅比较值的日期部分。

1840 年 12 月 31 日之前的日期

日期通常由 DATE 数据类型或 TIMESTAMP 数据类型表示。

DATE 数据类型以 \$HOROLOG 格式存储日期，作为从 1840 年 12 月 31 日的任意开始日期算起的正整数天数。默认情况下，日期只能由正整数 (MINVAL=0) 表示，它对应于到 1840 年 12 月 31 日。但是，可以更改 %Library.Date MINVAL 类型参数以启用存储 1840 年 12 月 31 日之前的日期。通过将 MINVAL 设置为负数，可以存储 12 月 31 日之前的日期，1840 为负整数。最早允许的 MINVAL 值为 -672045。这对应于第 1 年 (CE) 的 1 月 1 日。DATE 数据类型不能表示 BCE（也称为 BC）日期。

TIMESTAMP 数据类型默认为 1840-12-31 00:00:00 作为最早允许的时间戳。但是，可以更改 MINVAL 参数以定义可以存储 1840 年 12 月 31 日之前的日期的字段或属性。例如，MyTS %Library.TimeStamp(MINVAL='1492-01-01 00:00:00')。最早允许的 MINVAL 值是 0001-01-01 00:00:00。这对应于第 1 年 (CE) 的 1 月 1 日。%TimeStamp 数据类型不能表示 BCE（也称为 BC）日期。

注意：请注意，这些日期计算并未考虑公历改革（1582 年 10 月 15 日颁布，但直到 1752 年才在英国及其殖民地采用）引起的日期变化。

可以重新定义语言环境的最短日期，如下所示：

```
s oldMinDate = ##class(%SYS.NLS.Format).GetFormatItem("DATEMINIMUM")
if oldMinDate = 0 {
    d ##class(%SYS.NLS.Format).SetFormatItem("DATEMINIMUM", -672045)
    s newMinDate = ##class(%SYS.NLS.Format).GetFormatItem("DATEMINIMUM")
    w "Changed earliest date to ",newMinDate
} else {
    w "Earliest date was already reset to ",oldMinDate
}
```

上面的示例将语言环境的 MINVAL 设置为允许的最早日期 (1/1/01)。

注意：IRIS 不支持使用带有负逻辑 DATE 值的儒略日期（%Library.Date 值，MINVAL<0）。因此，这些 MINVAL<0 值与 TOCHAR 函数返回的儒略日期格式不兼容。

[#SQL #Caché](#)

源

URL:

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