Snappy Data数据类型说明

## 类型预览

|  |  |
| --- | --- |
| SnappyData是数据类型 | Java数据类型 |
| [BigInt](#_BIGINT) | java.lang.Long |
| [Blob](#_BLOB) | java.lang.Blob |
| [Char](#_CHAR) | java.lang.String |
| [Char for big data](#_CHAR_FOR_BIT) | 根据需要使用byte []，java.io.InputStream或java.lang.OutputStream |
| [Clob](#_CLOB) | java.sql.Clob |
| [Date](#_DATE) | java.sql.Date |
| [Decimal](#_DECIMAL) | java.math.BigDecima |
| [Double](#_DOUBLE) | java.lang.Double |
| [Double precision](#_DOUBLE_PRECISION) | java.lang.Double或java.lang.Float取决于精度 |
| [Float](#_FLOAT) | java.lang.Double或java.lang.Float取决于精度 |
| [Integer](#_INTEGER) | java.lang.Integer |
| [Json](#_INTEGER) | java.sql.Clob |
| [Long varchar](#_LONG_VARCHAR) | java.lang.String |
| [Long varchar for big data](#_LONG_VARCHAR_FOR) | 根据需要使用byte []或java.io.InputStream / java.lang.OutputStream |
| [Numeric](#_NUMERIC) | java.sql.Types.NUMERIC |
| [Real](#_REAL) | java.lang.Float |
| [Time](#_TIME) | java.sql.Time |
| [Timestamp](#_TIMESTAMP) | java.sql.Timestamp |
| [User-Defined Types](#_User-Defined_Types) | java.lang.String |
| [Varchar](#_VARCHAR) |  |
| [Varchar for big data](#_VARCHAR_FOR_BIT) | 根据需要使用byte []或java.io.InputStream / java.lang.OutputStream |
| [Xml](#_XML) | java.lang.String |

## BIGINT

为长整数值提供8字节存储。如果该值溢出较小类型允许的最大值，则尝试将BIGINT值放入具有较小尺寸/精度的另一精确数字类型（例如INT）中。

对于表达式中其他类型的行为，请参阅表达式中的数字类型升级，将另一数字数据类型的列中的一个数字数据类型的值存储。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.Long |
| 最小值 | java.lang.Long.MIN\_VALUE（-9223372036854775808） |
| 最大值 | java.lang.Long.MAX\_VALUE（9223372036854775807） |
| JDBC元数据类型 | java.sql.Types.BIGINT |
| JDBC方法 | ResultSet.getLong，PreparedStatement.setLong |

## BLOB

二进制大对象表示长度不同的原始字节数组。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.Blob |
| 最大长度（也是默认长度） | 2 GB - 1（或2,147,483,647） |
| JDBC元数据类型 | java.sql.Types.BLOB |
| JDBC方法 | ResultSet.getBlob，PreparedStatement.setBlob |

{ BLOB | BINARY LARGE OBJECT } [ ( length [{ K | M | G }] ) ]

默认情况下，BLOB的长度以字节数表示。后缀K，M和G代表千字节，兆字节和千兆字节，分别使用1024，1024 \* 1024或1024 \* 1024 \* 1024的倍数。

CREATE TABLE blob\_data(id INT primary key, data BLOB(10M));

–- search for a blob

select length(data) from blob\_data where id = 100;

## CHAR

提供固定长度的字符串。如果字符串值小于预期长度，则插入空格以将字符串填充到预期长度。如果字符串值长于预期长度，则会修剪任何尾部空白以使长度与预期长度相同，如果需要截断空格以外的字符，则会引发异常。对于比较操作，较短的CHAR字符串用空格填充到较长的值。类似地，当在表达式中混合CHAR和VARCHAR时，较短的值用空格填充到更长的字符串长度。

要在字符串中表示单引号，请使用两个引号：

VALUES 'going to Chandra''s place'

CHAR的长度是无符号整数常数。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.String |
| 最大长度 | java.lang.Integer.MAX\_VALUE（2147483647） |
| 默认长度 | 1 |
| JDBC元数据类型 | java.sql.Types.CHAR |
| JDBC方法 | ResultSet.getString，PreparedStatement.setString |

CHAR[ACTER] [(length)]

## CHAR FOR BIT DATA

提供固定长度的字节串。由于非结构化数据或其他原因，CHAR类型不合适时使用。如果值小于预期长度，则填充0x20字节值。对于CHAR FOR BIT DATA和VARCHAR FOR BIT DATA类型之间的比较，根据SQL-92标准（这与某些DBMS的行为不同），不执行填充。涉及VITAR FOR BIT DATA和CHAR FOR BIT DATA的操作产生VARCHAR FOR BIT DATA类型的值。

长度是以字节为单位的值的长度的无符号整数常量。

|  |  |
| --- | --- |
| 等效的Java类型 | 根据需要使用byte []，java.io.InputStream或java.lang.OutputStream |
| 最大长度 | 254字节 |
| 默认长度 | 1 |
| JDBC元数据类型 | java.sql.Types.BINARY |
| JDBC方法 | ResultSet.getBytes，PreparedStatement.setBytes |

{ CHAR | CHARACTER }[(length)] FOR BIT DATA

## CLOB

字符大对象表示长度不同的字符数组。它用于存储大量基于字符的数据，如文档。

长度以数字字符表示，除非您指定分别使用1024，1024 \* 1024或1024 \* 1024 \* 1024的倍数的后缀K，M或G。

|  |  |
| --- | --- |
| 等效的Java类型 | java.sql.Clob |
| 最大长度（也是默认长度） | 2 GB - 1（或2,147,483,647） |
| JDBC元数据类型 | java.sql.Types.CLOB |
| JDBC方法 | ResultSet.getClob，PreparedStatement.setClob |

{ CLOB | CHARACTER LARGE OBJECT } [ ( length [{ K | M | G }] ) ]

CREATE TABLE clob\_data(id INT primary key, text CLOB(10M));

–- search for a clob

select text from clob\_data where id = 100;

## DATE

提供将日期存储为年月日。支持的格式有：

yyyy-mm-dd

mm/dd/yyyy

dd.mm.yyyy

年（yyyy）必须始终以四位数表示，而月（mm）和日（dd）可能有一位或两位数字。DATE，TIME和TIMESTAMP不得在表达式中彼此混合，除了具有明确的CAST。

|  |  |
| --- | --- |
| 等效的Java类型 | java.sql.Date |
| JDBC元数据类型 | java.sql.Types.DATE |
| JDBC方法 | ResultSet.getDate，PreparedStatement.setDate |

VALUES '2010-05-04'

VALUES DATE('2001-10-12')

后一个示例使用内置函数和过程中描述的DATE（）函数。

## DECIMAL

提供具有指定精度和比例的精确十进制值。精度是小数点左侧和右侧的总位数，而缩放比例是小数点右侧小数位数。

只要非分数精度不会丢失，数值（例如INT，BIGINT，SMALLINT）可以放入DECIMAL，否则抛出范围异常（SQLState：“22003”）。当截断来自DECIMAL的尾数时，该值被舍入。

对于表达式中其他类型的行为，请参阅表达式中的数字类型升级，十进制算术的缩放和另一数字数据类型的列中的一个数值数据类型的值。

|  |  |
| --- | --- |
| 等效的Java类型 | java.math.BigDecimal |
| 精度最小/最大 | 1到31 |
| 缩小最小/最大 | 小于或等于精度 |
| 默认精度 | 五 |
| 默认比例 | 0 |
| JDBC元数据类型 | java.sql.Types.DECIMAL |
| JDBC方法 | ResultSet.getBigDecimal，PreparedStatement.setBigDecimal |

{ DECIMAL | DEC } [(precision [, scale ])]

-- this cast loses fractional precision

values cast (23.8372 AS decimal(4,1));

-–- results in:

23.8

-- this cast is outside the range

values cast (97824 AS decimal(4,1));

–-- throws exception:

ERROR 22003: The resulting value is outside the range for the data type DECIMAL/NUMERIC(4,1).

## DOUBLE

使用IEEE浮点表示法为数字提供8字节存储。

算术运算不会使其结果值为零。如果值太小，您将收到异常。数字浮点常数限制为30个字符的长度。

对于表达式中其他类型的行为，请参阅表达式中的数字类型升级，并在另一数字数据类型的列中存储一个数字数据类型的值。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.Double  注意：如下所述，最大/最小限制与java.lang.Double的不同。 |
| 最小值 | -1.79769E + 30 |
| 最大值 | 1.79769E + 308 |
| 最小正值 | 2.225E-307 |
| 最大的负值 | -2.225E-307 |
| 默认精度 | 5 |
| 默认比例 | 0 |
| JDBC元数据类型 | java.sql.Types.DOUBLE |
| JDBC方法 | ResultSet.getDouble，PreparedStatement.setDouble |

–- examples of valid values

values 233.31E3;

values 8928E+06;

-- this example will throw a range exception (SQLState: "42820")

values 123456789012345678901234567890123456789e0;

## DOUBLE PRECISION

Synonyn为DOUBLE。

## FLOAT

具有REAL或DOUBLE数据类型的别名，具体取决于指定的精度。默认精度为53，相当于DOUBLE。23或更小的精度使FLOAT等效于REAL，而大于23则相当于DOUBLE。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.Double或java.lang.Float取决于精度 |
| 最小/最大限度 | 如果精度小于23，则与FLOAT相同。否则，与DOUBLE相同的最小/最大极限相同。 |
| 默认精度 | 53 |
| JDBC元数据类型 | java.sql.Types.FLOAT |
| JDBC方法 | ResultSet.getFloat / getDouble，PreparedStatement.setFloat / setDouble |

FLOAT [(precision)]

## INTEGER

为整数值提供4字节存储。INT可以用作CREATE TABLE中的INTEGER的同义词。

对于表达式中其他类型的行为，请参阅表达式中的数字类型升级，并在另一数字数据类型的列中存储一个数字数据类型的值。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.Integer |
| 最小值 | java.lang.Integer.MIN\_VALUE（-2147483648） |
| 最大值 | java.lang.Integer.MAX\_VALUE（21474836487） |
| JDBC元数据类型 | java.sql.Types.INTEGER |
| JDBC方法 | ResultSet.getInt，PreparedStatement.setInt |

## JSON

JSON格式（JavaScript对象符号）字符串。JSON类型的列可以与JSON\_EVALPATH函数一起使用，以提取JSON属性或使用JSON路径过滤查询结果。

RowStore尝试在插入期间使用public com.gemstone.gemfire.pdx.JSONFormatter API 解析插入的数据为JSON 。如果RowStore无法将数据解析为JSON，则抛出SQLException并且插入失败。

注意： RowStore不保留您插入的JSON数据中存在的空格或其他格式。此外，RowStore在查询JSON列值时，会添加空格和换行符来格式化字符串输出。RowStore不对JSON数据类型的列执行索引。

|  |  |
| --- | --- |
| 等效的Java类型 | java.sql.Clob |
| 最大长度（也是默认长度） | 2 GB - 1（或2,147,483,647字节） |
| JDBC元数据类型 | java.sql.Types.CLOB |
| JDBC方法 | ResultSet.getClob，PreparedStatement.setClob |

CREATE TABLE json\_store (id INT primary key, storedata JSON);

–- insert JSON data as a string value delimited by single quotes

INSERT INTO json\_data VALUES ( 1,

'{ "store": {

"book": [

{ "category": "reference",

"author": "Nigel Rees",

"title": "Sayings of the Century",

"price": 8.95

},

{ "category": "fiction",

"author": "Evelyn Waugh",

"title": "Sword of Honour",

"price": 12.99

},

{ "category": "fiction",

"author": "Herman Melville",

"title": "Moby Dick",

"isbn": "0-553-21311-3",

"price": 8.99

},

{ "category": "fiction",

"author": "J. R. R. Tolkien",

"title": "The Lord of the Rings",

"isbn": "0-395-19395-8",

"price": 22.99

}

],

"bicycle": {

"color": "red",

"price": 19.95

}

}

}' );

另请参见JSON\_EVALPATH中的示例。

## LONG VARCHAR

与VARCHAR相同，但不必指定最大长度为32700。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.String |
| 最大长度 | 32700 |
| JDBC元数据类型 | java.sql.Types.LONGVARCHAR |
| JDBC方法 | ResultSet.getString，PreparedStatement.setString |

## LONG VARCHAR FOR BIT DATA

与VARCHAR FOR BIT DATA相同，除了不需要指定最大长度为32700。

|  |  |
| --- | --- |
| 等效的Java类型 | 根据需要使用byte []或java.io.InputStream / java.lang.OutputStream |
| 最大长度 | 32700字节 |
| JDBC元数据类型 | java.sql.Types.LONGVARBINARY |
| JDBC方法 | ResultSet.getBytes，PreparedStatement.setBytes |

## NUMERIC

DECIMAL数据类型的同义词。

下面列出了DECIMAL的元数据差异。否则，NUMERIC的行为与DECIMAL相同。

|  |  |
| --- | --- |
| JDBC元数据类型 | java.sql.Types.NUMERIC |

NUMERIC [(precision [, scale ])]

## REAL

使用IEEE浮点表示法为数字提供4字节存储。

算术运算不会使其结果值为零。如果值太小，您将收到异常。常量总是映射到DOUBLE - 使用显式CAST将常量转换为REAL。

对于表达式中其他类型的行为，请参阅表达式中的数字类型升级，将另一数字数据类型的列中的一个数字数据类型的值存储。

最大/最小限制与以下所述的java.lang.Float不同。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.Float |
| 最小值 | -3.402E + 38 |
| 最大值 | 3.402E + 38 |
| 最小正值 | 1.175E-37 |
| 最大的负值 | -1.175E-37 |
| JDBC元数据类型 | java.sql.Types.REAL |
| JDBC方法 | ResultSet.getFloat，PreparedStatement.setFloat |

–- examples of valid values

values cast (233.31E3 as REAL);

values cast (8928E+06 as REAL);

-- this example will throw a range exception (SQLState: "42820")

values cast (123456789012345678901234567890123456789e0 as REAL);

## SMALLINT

为短整型值提供2字节存储。

对于表达式中其他类型的行为，请参阅表达式中的数字类型升级，并在另一数字数据类型的列中存储一个数字数据类型的值。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.Short中 |
| 最小值 | java.lang.Short.MIN\_VALUE（-32768） |
| 最大值 | java.lang.Short.MAX\_VALUE（32767） |
| JDBC元数据类型 | java.sql.Types.SMALLINT |
| JDBC方法 | ResultSet.getShort，PreparedStatement.setShort |

## TIME

提供以小时 - 分钟秒的时间存储。支持的格式有：

hh:mm[:ss]

hh.mm[.ss]

hh[:mm] {AM | PM}

小时（hh）可能有一个或两个数字，而分钟（mm）和秒（ss）（如果存在）必须有两位数字。DATE，TIME和TIMESTAMP不得在表达式中彼此混合，除了具有明确的CAST。

|  |  |
| --- | --- |
| 等效的Java类型 | java.sql.Time |
| JDBC元数据类型 | java.sql.Types.TIME |
| JDBC方法 | ResultSet.getTime，PreparedStatement.setTime |

VALUES '13:23:12'

VALUES TIME('13:23:12')

后一个例子使用内置函数和过程中描述的TIME（）函数。

## TIMESTAMP

提供DATE和TIME两者的组合值。此外，它允许具有最多六位数的小数秒。支持的格式有：

yyyy-MM-dd hh:mm:ss[.nnnnnn]

yyyy-MM-dd-hh.mm.ss[.nnnnnn]

年（yyyy）必须始终以四位数表示。月（MM），天（dd）和小时（hh）可能有一位或两位数字，而分钟（mm）和秒（ss）必须有两位数字。微秒（如果存在）可能有一到六位数。DATE，TIME和TIMESTAMP不得在表达式中彼此混合，除了具有明确的CAST。

|  |  |
| --- | --- |
| 等效的Java类型 | java.sql.Timestamp |
| JDBC元数据类型 | java.sql.Types.TIMESTAMP |
| JDBC方法 | ResultSet.getTimestamp，PreparedStatement.setTimestamp |

VALUES '2000-02-03 12:23:04'

VALUES TIMESTAMP(' 2000-02-03 12:23:04.827')

VALUES TIMESTAMP('2000-02-03 12:23:04')

后一个示例使用内置函数和过程中描述的TIMESTAMP（）函数。

## User-Defined Types

RowStore允许您创建用户定义的类型。用户定义的类型是一个可序列化的Java类，其实例存储在列中。该类必须实现java.io.Serializable接口。为了有效的数据传输和存储，该类也应该实现java.io.Externalizable接口。如果该类实现了Apache Geode com.gemstone.gemfire.DataSerializable接口，那么它将被用于最佳的存储和对等传输。

有关创建和删除类型的信息，请参阅CREATE TYPE和DROP TYPE。

有关编写实现用户定义类型的Java类的信息，请参阅编程用户定义的类型。

VARCHAR

提供长度最大限制的可变长度字符串。如果字符串值长于最大长度，则会修剪任何尾部空白以使其长度与最大长度相同，如果需要截断空格以外的字符，则会引发异常。在表达式中混合CHAR和VARCHAR时，较短的值用空格填充到更长的字符串长度。

字符串常量的类型是CHAR，而不是VARCHAR。要在字符串中表示单引号，请使用两个引号：

VALUES 'going to Chandra''s place'

VARCHAR的长度是无符号整数常数。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.String |
| 最大长度 | 32672 |
| JDBC元数据类型 | java.sql.Types.VARCHAR |
| JDBC方法 | ResultSet.getString，PreparedStatement.setString |

{ VARCHAR | CHAR VARYING | CHARACTER VARYING }(length)

## VARCHAR

提供长度最大限制的可变长度字符串。如果字符串值长于最大长度，则会修剪任何尾部空白以使其长度与最大长度相同，如果需要截断空格以外的字符，则会引发异常。在表达式中混合CHAR和VARCHAR时，较短的值用空格填充到更长的字符串长度。

字符串常量的类型是CHAR，而不是VARCHAR。要在字符串中表示单引号，请使用两个引号：

VALUES 'going to Chandra''s place'

VARCHAR的长度是无符号整数常数。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.String中 |
| 最大长度 | 32672 |
| JDBC元数据类型 | java.sql.Types.VARCHAR |
| JDBC方法 | ResultSet.getString，PreparedStatement.setString |

{ VARCHAR | CHAR VARYING | CHARACTER VARYING }(length)

## VARCHAR FOR BIT DATA

提供长度最大限制的可变长度字节串。当由于非结构化数据或其他原因而导致字符类型不合适时使用。对于CHAR FOR BIT DATA和VARCHAR FOR BIT DATA类型之间的比较，根据SQL-92标准（这与某些DBMS的行为不同），不执行填充。涉及VITAR FOR BIT DATA和CHAR FOR BIT DATA的操作产生VARCHAR FOR BIT DATA类型的值。

字节字面值的类型始终是位数据的VARCHAR，而不是字节数据的CHAR。

长度是以字节为单位的值的长度的无符号整数常量。

|  |  |
| --- | --- |
| 等效的Java类型 | 根据需要使用byte []或java.io.InputStream / java.lang.OutputStream |
| 最大长度 | 32672字节 |
| JDBC元数据类型 | java.sql.Types.VARBINARY |
| JDBC方法 | ResultSet.getBytes，PreparedStatement.setBytes |

{ VARCHAR | CHAR VARYING | CHARACTER VARYING }[(length)] FOR BIT DATA

## XML

此类型可用于可扩展标记语言（XML）文档：

存储符合格式良好的XML（DOCUMENT（ANY））值的SQL / XML定义的XML文档

瞬态地为XML（SEQUENCE）值，可能不是格式正确的XML（DOCUMENT（ANY））值

注意： RowStore不支持SQL标准中的SQLXML类型，或者是SQLXML类型，而是将值作为普通字符串存储，并且用户必须根据需要明确地使用XMLSERIALIZE，XMLPARSE函数进行插入和查询。

对于执行XML相关功能的应用程序，应用程序必须使用内置JAXP / Xalan支持的JDK（例如Sun JDK5），或者必须为JAXP解析器和服务器类路径中的Xalan提供类（serializer.jar，xalan.jar ）。

要使用JDBC或ADO.NET驱动程序从RowStore数据库获取XML值，请在SQL查询中使用XMLSERIALIZE运算符。例如：

SELECT XMLSERIALIZE(doc as CLOB) FROM t1

然后通过使用与目标序列化类型相对应的ResultSet.getXXX方法检索XML值（例如，使用CLOB数据类型的上述示例中的getClob）。要使用JDBC将XML值存储到RowStore数据库中，请在SQL语句中使用XMLPARSE运算符。例如：

INSERT INTO t1(doc) VALUES XMLPARSE( DOCUMENT CAST (? AS CLOB) PRESERVE WHITESPACE)

然后使用与String类型兼容的任何PreparedStatement.setXXX方法（例如，在上述示例中使用PreparedStatement.setString或PreparedStatement.setCharacterStream）。

|  |  |
| --- | --- |
| 等效的Java类型 | java.lang.String; 使用显式XML函数进行插入/查询，如上所述 |
| 最大长度 | java.lang.Integer.MAX\_VALUE（2147483647） |
| JDBC元数据类型 | 没有 |
| JDBC方法 | 根据目标序列化类型 |