SinoDB 时间序列用户手册

星瑞格数据库管理系统 (SinoDB)

TimeSeries 用户手册

福建星瑞格软件有限公司

www.sinoregal.cn

■版权声明

版本记录

版本号	版本日期	状态	修改人	审核人	修改记录
V1.0	20240701	C	蔡志扬	张道山	创建文档

状态：C–创建文档，A–增加内容，M–修改内容，D–删除内容

目录

关于本手册

本章内容

- 虚拟表接口概述
- 创建虚拟表
- 使用虚拟表进行查询
- 虚拟表的限制
- 跟踪

第六章 日历模式例程

- AndOp
- OrOp
- NotOp
- CalPattStartDate
- Collapse
- Expand

第七章 日历例程

- AndOp
- OrOp
- CalIndex
- CalRange
- CalStamp
- CalStartDate

第八章 时间序列SQL例程

介绍

本章内容

- 关于本手册
- 本手册的组织结构
- 用户类型
- 软件依赖关系
- 在线文档

关于本手册

本手册是 SinoDB TimeSeries DataBlade 模块的用户指南。TimeSeries DataBlade 模块是一种扩展，可以在 SinoDB 数据库中处理时间序列数据。时间序列数据是按时间顺序排列的数据集，如股票市场数据、气象数据等。

本手册的组织结构

本手册按以下方式组织：

1. **入门配置**：介绍如何安装和设置 TimeSeries DataBlade 模块
2. **时间序列概述**：解释时间序列的基本概念和结构
3. **数据类型和系统表**：描述 TimeSeries DataBlade 模块中的数据类型和系统表
4. **创建和加载时间序列**：指导如何创建和加载时间序列数据
5. **使用虚拟表接口**：介绍如何使用虚拟表接口来访问时间序列数据
6. **日历模式例程**：创建日历模式的联合和交集
7. **日历例程**：创建日历的联合和交集
8. **时间序列SQL例程**：创建和加载时间序列，并进行统计计算、算术计算、管理容器、在时间戳和偏移量之间转换、提取时间序列的时间段等

用户类型

本手册适用于以下用户：

- **数据库管理员**：负责安装和管理 TimeSeries DataBlade 模块。
- **应用程序开发人员**：使用 TimeSeries DataBlade 模块进行应用开发。
- **最终用户**：使用 TimeSeries DataBlade 模块查询和分析时间序列数据。

软件依赖关系

在安装和使用 TimeSeries DataBlade 模块之前，确保系统满足以下要求：

- 安装了SinoDB。
- 安装了 BladeManager，用于安装和管理 DataBlade 模块。

在线文档

在线文档可以从以下位置获取：

- Sinoregal社区：https://forum.sinoregal.cn/

第一章 配置时间序列

本章内容

- 配置 TimeSeries DataBlade 模块
- 验证安装

配置 TimeSeries DataBlade 模块

数据库安装完成后，需要对 TimeSeries DataBlade 模块进行一些配置，以确保其正确工作。

1. **创建数据库**：

- 使用以下 SQL 命令创建一个新的数据库：

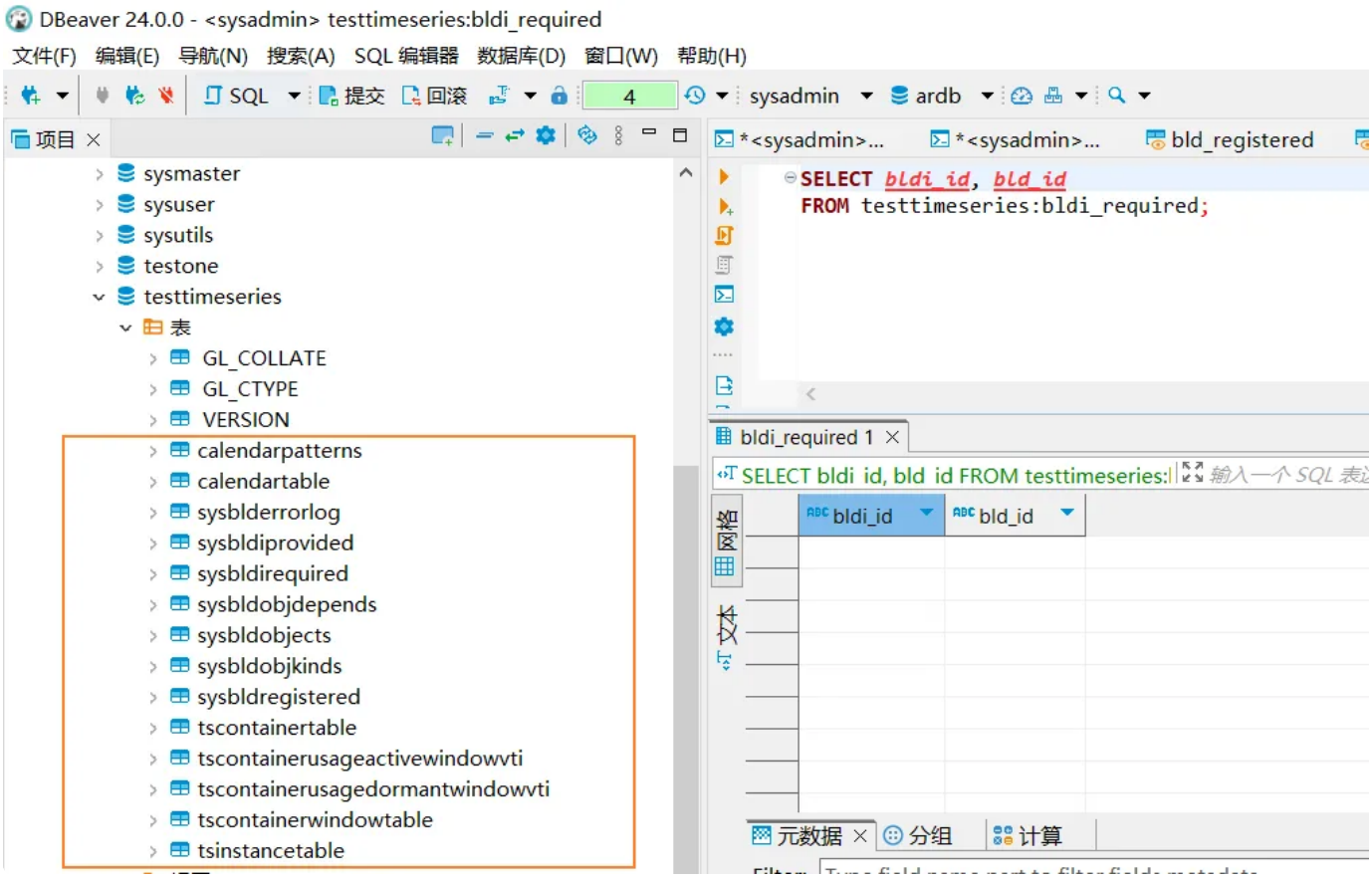
```
1   CREATE DATABASE TestTimeSeries  WITH LOG ;
```

2. **注册 DataBlade**：

- 运行以下 命令将 DataBlade 注册到数据库中。
- TimeSeries.6.00.FC7 在数据库安装目录$SINODBMSDIR/extend下

```
[sinodbms@SFQ112 examples]$ blademgr
sinodb168>register TimeSeries.6.00.FC7 TestTimeSeries
Register module TimeSeries.6.00.FC7 into database TestTimeSeries? [Y/n]Y
Registering DataBlade module... (may take a while).
DataBlade TimeSeries.6.00.FC7 was successfully registered in database TestTimeSeries.
```

注册成功后，会自动创建TimeSeries相关的表、函数等。



验证安装

为了验证安装和配置是否成功，可以执行一些基本的操作，如插入和查询数据。

1. **插入数据**：

- 使用以下 SQL 命令建立Calendar：

```
insert into Calendartable (c_name, c_calendar) values ('daycal','startdate (2024-01-01 00:00:00),pattern({1 on},day)');
```

2. **查询数据**：

- 使用以下 SQL 命令查询插入的数据：

```
1   SELECT * FROM Calendartable where c_name='daycal';
```

- 确认查询结果与插入的数据一致。

第二章 时间序列概述

本章内容

- 什么是时间序列?
- 时间序列的数据结构
- 时间序列的种类
- 时间序列组织
- 时间序列的优点

什么是时间序列?

时间序列是带有时间戳的一系列数据条目，这类数据被许多不同行业的应用程序存储和分析，包括金额、制造业、新闻业、科学和工程、IOT等。例如，一家生物技术公司可以测量其培养细胞的容器条件。时间序列每分钟记录温度、压力、酸度和其他参数。该公司将使用这些数据来监测培养物的健康状况，并分析数据以确定最佳的生长条件。

尽管关系数据库管理系统可以通过每个带时间戳的数据条目存储一行来存储标准类型的时间序列，但性能很差，存储效率很低。另外，非关系时间序列实现也存在一些限制，比如缺乏通用性和可扩展性、对数据的类型和结构有预定义的限制，以及无法将时间序列数据与其他信息组合在一起。

相反，SinoDB TimeSeries DataBlade模块提供：

高性能存储和访问架构，由容器组成，用于保存数据库表外的时间序列数据。

一套丰富的时间序列分析例程。

完全支持时间序列数据作为对象数据类型，使用TimeSeries数据类型。

有了SinoDB TimeSeries DataBlade模块，数据库管理系统得到了扩展，可以将时间序列和时间数据作为数据库中的第一类类型来“理解”。时间序列条目是数据库可以操作的对象，而不是不透明的大对象。特定时间序列的所有条目位于数据库表的同一行中。例如，有关IBM股票表现的所有信息可能位于一行中，其中包含两列:一列标识股票名称，另一列包含所有时间序列数据。因此，条目已经由股票名称标

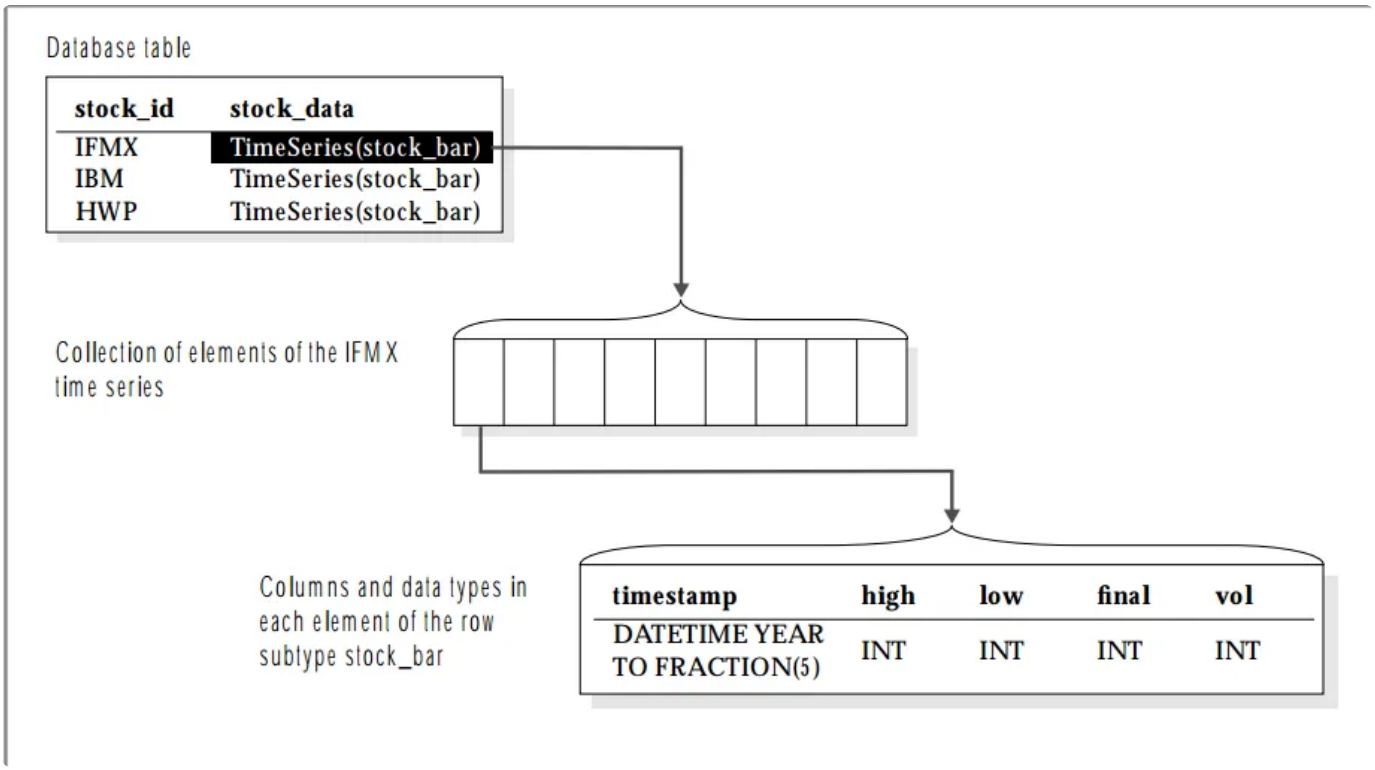
识，避免了传统关系数据库必须执行的排序步骤之一。在时间序列列中，条目还按时间戳进行索引。这使得检索在时间上接近的值特别有效。

使用可重用的业务日历有效地存储时间序列，这些日历是用SinoDB TimeSeries DataBlade模块提供的Calendar和CalendarPattern数据类型创建的。日历确定数据的有效性，并根据用户指定的模式组织时间序列数据。此外，SinoDB TimeSeries DataBlade模块提供系统表来存储有关时间序列、日历、日历模式和容器的信息。

时间序列的数据结构

时间序列数据通常由时间戳和对应的值组成。一个典型的时间序列数据结构如下：

- **时间戳**：表示数据点的时间，例如日期和时间。
- **值**：表示在该时间点上记录的数值。



本例中的数据库表有两列:一个stock_id列包含股票名称，一个stock_data列包含时间序列。表中的每一行都包含不同的时间序列。在本例中，表中的所有三行都有一个子类型为stock_bar的时间序列。时间序列行子类型定义了元素的结构。

时间序列由元素组成，每个元素表示特定时间戳的单行数据。元素按时间戳排序。

每个元素都是包含列的行数据类型，从时间戳列开始。时间戳列必须为DATETIME YEAR TO FRACTION(5)类型，它指定时间戳具有存储从一年到10微秒的时间范围的精度。时间戳必须是唯一的;多

个表项不能具有相同的时间戳。

SinoDB TimeSeries DataBlade模块支持具有通用数据选项的SinoDB Dynamic Server中允许的行数据类型的大多数数据类型，但定义了Assign或Destroy函数的数据类型除外。

允许其他行数据类型或集合作为行子类型中的列。元素中的列数不受限制。前面的示例包含用于股票的最高、最低和最终值以及股票交易量的列。

在本例中，类型为stock_bar的每个时间序列必须具有相同的列;但是，每个时间序列可以有自己的日历和时间序列开始日期。

时间序列种类

有两种时间序列:规则和不规则。相对于日历，规则时间序列存储有规则间隔时间点的数据，而不规则时间序列存储任意时间点的数据。常规时间序列适用于在可预测的时间点记录条目的应用程序，例如每个工作日记录的股票汇总数据。

当数据不可预测地到达时，例如当应用程序记录每笔股票交易时，不规则时间序列是合适的。

时间序列中的每个元素表示与时间间隔相关联的数据。

与时间间隔相关联的时间戳标志着时间间隔的开始。时间戳的条目通常总结该时间戳间隔内的事件(例如，一分钟内股票的高、低和成交量)。在常规时间序列中，每个元素的长度相同。

常规元素仅在与时间序列关联的日历定义的间隔长度内存在，缺少的元素为空。在不规则时间序列中，每个元素可以是不同的长度。不规则元素会持续到下一个元素;没有空元素。

两种时间序列的主要区别是不规则时间序列缺乏偏移量的概念;它们在与元素相关联的时间点与其相对于时间序列开始的位置之间没有映射。正因为如此，规则时间序列比不规则时间序列的存储效率更高。时间戳不存储在常规时间序列中，而是根据元素的偏移量计算。

在大多数情况下，规则时间序列和不规则时间序列具有相同的特征。因此，大多数时间序列例程既适用于规则时间序列，也适用于不规则时间序列。时间序列是规则的还是不规则的，在创建时间序列时指定。

时间序列存储

时间序列数据可能会变得太大，无法放入表的一行中，目前的限制是大约2048字节。发生这种情况时，SinoDB TimeSeries DataBlade模块将时间序列的数据部分移动到容器中。容器是SinoDB TimeSeries DataBlade模块创建和维护的一种结构，用于保存时间序列数据。不同的时间序列可以使用同一个容器，只要它们都是规则时间序列或都是不规则时间序列。容器存在于dbspace中，dbspace是物理内存的命名

部分。您可以使用ON-Monitor或onspaces实用程序创建dbspace，也可以使用默认的dbspace rootdbs。

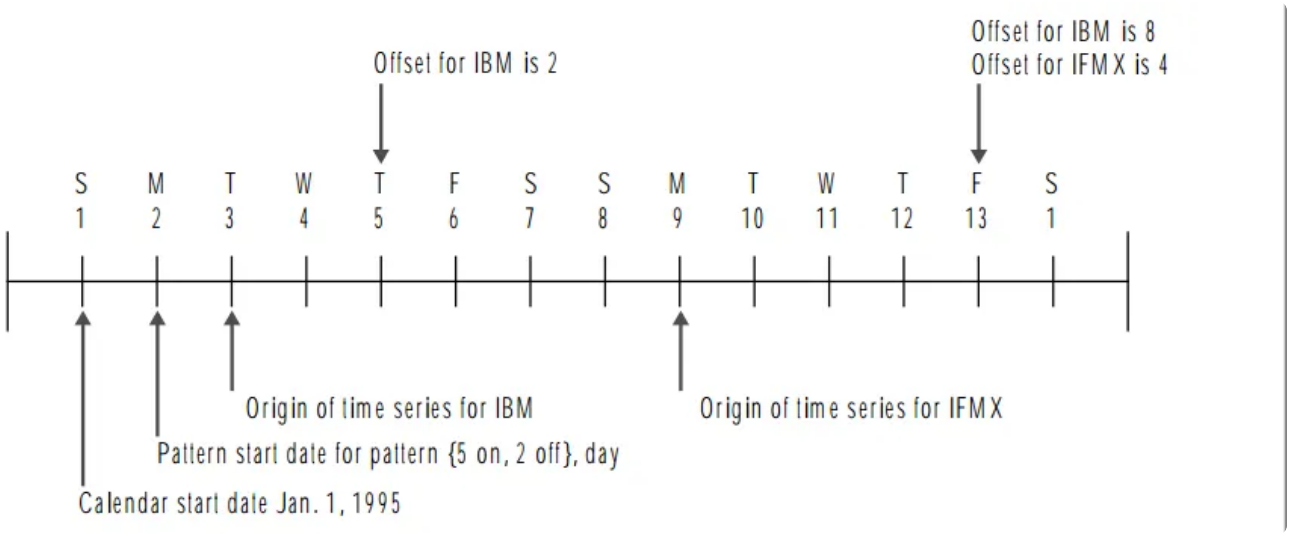
您可以通过运行SQL TSContainerCreate过程调用来创建容器，并通过运行TSContainerDestroy过程来删除它。您可以在创建时间序列时指定要用于时间序列的容器。当时间序列在容器中有数据时，原始表中只剩下一个小表头。这个头由SinoDB TimeSeries DataBlade模块在内部维护，它包含标识SinoDB TimeSeries DataBlade模块在容器中放置数据的位置的信息。

您可以使用threshold参数控制时间序列何时从行内表示提升到容器内表示，您可以在创建时间序列时设置该参数(参见后续的“TSCreate”)。一旦数据被移动到容器中，它就不能移动回一行，即使元素的数量下降到阈值以下。除非时间序列的数据变得太大，无法在表的一行中容纳，否则不需要容器。

时间序列组织

时间序列由使用Calendar和CalendarPattern数据类型的日历控制。

- Calendars
- Calendar patterns
- Time series origin
- Offset



这张插图是1995年头两个星期的情况。它显示:

- 两个常规时间序列(IBM和IFMX)，它们的起源和偏移量
- 具有其开始日期、日历模式和日历模式开始日期的日历。

Calendars

每个时间序列都与一个日历相关联。日历定义了一组有效的时间，时间序列可以在这些时间记录数据;它决定何时以及多久接受一次条目。因此，日历控制数据库中的数据。对于常规时间序列，日历通过将时间戳转换为偏移量来创建向量结构。尽管不规则时间序列没有偏移量，但它们仍然使用日历来定义有效的输入时间。

日历由Calendar数据类型表示，它由以下值组成:

start date	日历开始的日期。与日历及其时间序列关联的所有其他日期必须与此日期相同或晚于此日期。
pattern	一组有规律地出现的有效和无效的时间间隔。
interval	日历模式的校准
pattern start date	日历模式开始的日期。此日期必须晚于或等于日历开始日期，并且早于时间序列原点(开始日期)或与之相同。此日期必须小于日历开始日期之后的一个日历模式长度。

例如，上图所示的日历具有以下特点:

■ 日历从1995年1月1日开始。

■ 日历模式是五天上班，两天休息。

■ 间隔为天。

■ 日历模式从1995年1月2日星期一开始。

规则时间序列和不规则时间序列都可以使用相同的日历。

来自Calendar数据类型的日历信息存储在CalendarTable系统表中

Calendar Patterns

每个日历都有一个有效或无效时间间隔的日历模式，日历模式的开始由日历模式开始日期指定。在有效的时间间隔内记录数据，在无效的时间间隔内不记录数据。在日历模式语法中，on表示有效的时间段，off表示无效的时间间隔。日历模式还指示测量间隔的时间单位，例如分钟、小时、天或月。

日历模式由CalendarPattern数据类型表示，指定为:

{pattern specification}, interval

例如，上图所示的日历模式是:{5 on, 2 off}， day

由于日历模式的开始日期是1995年1月2日星期一，因此日历模式指定从星期一开始，5天有效(工作日)，2天无效(周末)。有效条目的间隔为一天;每周有5个条目。

日历必须包含日历模式开始时间和日历开始时间，因为日历和日历模式开始时间可能不一致。例如，大多数年份不是从一周的第一天开始的。

日历开始时间和日历模式开始时间必须在同一个日历模式长度内。日历模式长度是模式重复之前的间隔数。例如，如果日历模式长度为一周(间隔为一天)，则日历开始时间最多可以比日历模式开始时间早一周。

偶尔，如果您有一个规则的时间序列，您将有没有数据的元素。例如，如果您有每日日历，则可能无法获得假日的数据。日历中的这些异常被标记为空元素。但是，您可以隐藏异常，使它们不包含在计算或分析中。

您可以使用带有WithinR和WithinC函数的时间序列日历模式来搜索指定时间点周围的数据。WithinR执行相对搜索。相对搜索从起始时间点向前或向后搜索，沿着给定的时间间隔移动到未来或过去。WithinC执行校准搜索。校准后的搜索将向前和向后进行，直至给定起始时间点周围的“自然”间隔边界。

Time Series Origin

每个时间序列都有自己的起始时间点，它必须位于日历和日历模式的起始时间戳之前或之后。这是用户感兴趣的第一个时间点。在常规时间序列中，它被视为偏移量0。

例如，上图中的日历从1995年1月1日开始，日历模式从1995年1月2日开始，IBM的时间序列从1995年1月3日开始。

如果不声明一个更早的时间点，就不能引用日历中原点之前的时间点。

Offsets

一个规则的时间序列可以被认为是一个点的向量。点与原点之间的间隔数称为偏移量。每个偏移量都与一个时间戳相关联。

日历确定向量上所有可能的点。时间序列原点后的每个条目都有一个分配给它的偏移数。使用偏移量可以加快计算速度，并节省数据库中的空间，因为可以计算偏移量而不是存储时间戳。

例如，从上图中，IBM的时间序列向量为:

Day	T	W	T	F	M	T	W	T	F
Date	3	4	5	6	9	10	11	12	13
Offset	0	1	2	3	4	5	6	7	8

在日历模式中标记为无效的日期(星期六和星期日)没有偏移量。

IFMX的时间序列向量为:

Day	M	T	W	T	F
Date	9	10	11	12	13
Offset	0	1	2	3	4

常规时间序列具有可空的时间戳。当您将没有时间戳的元素插入到常规时间序列中时，它将被放入向量上的下一个可用偏移量中。在vector中最后一个偏移量之后输入一个时间戳超过一个偏移量的元素而跳过的任何偏移量都被标记为null。跳过的偏移量可以稍后更新。

偏移量和时间戳都可以用作元素的索引。与常规时间序列关联的日历会自动在时间戳和偏移量之间进行转换。

时间序列的优点

使用时间序列数据有许多优点，包括：

- **趋势分析**：可以分析数据随时间的变化趋势。
- **预测**：可以根据历史数据进行预测。
- **异常检测**：可以检测数据中的异常点。

第三章 数据类型和系统表

本章内容

- 时间序列数据类型
- 系统表概述

时间序列数据类型

TimeSeries DataBlade 模块提供了一些特殊的数据类型，用于处理时间序列数据。

■ CalendarPattern

■ Calendar

■ TimeSeries

1.CalendarPattern数据类型是一个不透明的数据类型，它是日历模式的结构。日历模式指定间隔持续时间以及有效(开启)和无效(关闭)间隔的模式。间隔可以是以下时间单位:

■ Second

■ Minute

■ Hour

■ Day

■ Week

■ Month

■ Year

CalendarPattern数据类型具有以下格式:{n on|off[, n on|off, ...]}, interval

花括号内的信息是模式规范。模式规范有一个或多个元素，这些元素由n(间隔单元的数量)组成，并且可以打开或关闭，以表示有效或无效的间隔。

元素之间用逗号分隔。日历模式长度是日历模式重新开始之前的间隔时间;一旦耗尽模式规范中的所有时间点，就会重复该模式。因此，具有每日间隔的周日历模式必须包含正好7个间隔，具有小时间隔的日日历模式必须包含正好24个间隔，依此类推。日历模式开始的时间由日历模式开始日期指定。

insert into CalendarPatterns values('workweek_day','{1 off, 5 on, 1 off}, day');

2.Calendar数据类型是一个不透明的数据类型。

下面的示例将一个名为yearcal24的日历插入到CalendarTable表中:

insert into CalendarTable(c_name, c_calendar) values ('yearcal24','startdate(2024-01-01 00:00:00.00000),pattstart(2024-01-07 00:00:00.00000),pattname(workweek_day)');

这个日历从2024年1月1日开始;它的模式始于2024年1月7日;它使用workweek_day模式。
下面的示例创建一个指定模式的小时日历:

insert into CalendarTable(c_name, c_calendar) values('my_cal','startdate(2024-01-01 00:00:00.00000), pattstart(2024-01-02 00:00:00.00000),pattern({24 off, 120 on, 24 off},hour)');

3.要创建TimeSeries列，首先使用create ROW TYPE语句创建TimeSeries子类型。

所有TimeSeries子类型必须以DATETIME YEAR TO FRACTION(5)数据类型作为第一列。行子类型的其余列可以是具有通用数据选项的SinoDB Dynamic Server支持的行数据类型的任何数据类型，但不具有Assign或Destroy函数的数据类型除外。

创建TimeSeries子类型之后，使用create table语句创建包含TimeSeries列的表【下一章节详细介绍】。 TimeSeries列可以包含规则或不规则的时间序列;单个时间序列元素的最大允许大小是1920字节。不能在TimeSeries类型的列上放置索引。

当前具有通用数据选项限制的SinoDB Dynamic Server可以将哪些数据类型包含在行类型中，这些限制适用于TimeSeries子类型。行类型不能包含:

■SERIAL和SERIAL8类型。

■没有Assign或Destroy函数赋值的类型，包括大型对象类型和一些用户定义的类型。

4.TimeSeries返回类型

返回的时间序列保留日历信息，并在可能的情况下保留阈值和容器信息。

有些返回时间序列子类型的函数要求将返回值强制转换为特定的时间序列类型，有些则不要求。对于Clip、WithinC和WithinR这样的函数，返回类型总是与参数时间序列的类型相同，不需要强制转换。

但是，对于其他函数，如AggregateBy、Apply和Union，结果时间序列的类型不一定与时间序列参数相同。这些函数要求将它们的返回类型强制转换为特定的时间序列类型。由于这些函数返回的时间序列可能无法使用原始时间序列的容器，因此在运行SetContainerName函数指定要使用的容器之前，生成的时间序列不会与容器关联。

系统表概述

系统表是数据库中用于存储元数据的表。例如，系统表可以存储关于表结构、索引和用户的信息。

SinoDB TimeSeries DataBlade模块中包含的系统表有:

■ CalendarPatterns.

■ CalendarTable.

■ TSInstanceTable.

■ TSContainerTable.

当将日历插入到CalendarTable表中时，它会从CalendarPatterns表中提取信息。SinoDB TimeSeries DataBlade模块仅引用日历和日历模式信息的CalendarTable;除非更新或重新创建calendartables，否则对CalendarPatterns表的更改不会产生任何影响。

TSInstanceTable包含所有时间序列的相关信息。

TSInstanceTable表、TSContainerTable表由SinoDB TimeSeries DataBlade模块管理，用户不直接修改它，通常也不需要查看它。当创建或销毁大型时间序列时，将自动插入或删除该表中的行。

使用TSContainerCreate和TSContainerDestroy过程创建和销毁容器，它们在TSContainerTable表中插入和删除特殊行。要确定数据库中存在哪些容器，请执行以下查询:

select name from TSContainerTable;

第四章 创建和加载时间序列

本章内容

- 创建时间序列列和相应表
- 创建时间序列容器
- 删除容器
- 创建和填充时间序列

创建时间序列列和相应表

要创建TimeSeries类型的列，必须首先创建行子类型来表示时间序列的每个元素中保存的数据。此类子类型的一个示例是每日股票栏，其中包含时间戳以及一天的高点、低点、成交量和收盘价。

SinoDB TimeSeries DataBlade模块对行子类型中的数据类型施加的唯一限制是，第一列必须是DATETIME YEAR TO FRACTION(5)类型的时间戳。此数据类型指定时间戳包含从一年到10微秒的所有时间间隔的条目，例如:1995-01-01 12:00:05.00000。子类型中的列数不受限制。

要创建行子类型，请使用SQL create row TYPE语句。

示例:常规时间序列的stock_bar子类型

```
1   create row type stock_bar(timestamp  datetime year to fraction(5),
2                             high decimal(18,2),low decimal(18,2),final decima
    l(18,2),vol decimal(18,2));
```

示例:不规则时间序列的stock_trade子类型

```
create row type stock_trade(
timestamp datetime year to fraction(5),
price double precision,
vol double precision,
trade int,
broker int,
buyer int,
seller int
);
```

创建了子类型之后，使用CREATE TABLE语句创建一个包含该子类型的列的表。

创建带有TimeSeries子类型列的表的语法是:

例如:常规时间序列的daily_stocks表。daily_stocks表中的每一行都可以保存特定股票的stock_bar时间序列。此表用于保存规则时间序列。

```
create table daily_stocks (
stock_id int,
stock_name lvarchar,
stock_data TimeSeries(stock_bar)
);
```

例如:不规则时间序列的activity_stocks表。activity_stocks表中的每一行都可以保存特定股票的股票交易时间序列。该表用于保存不规则的时间序列。

```
create table activity_stocks(
stock_id int,
activity_data TimeSeries(stock_trade)
);
```

创建时间序列容器

您可以使用TSContainerCreate过程创建一个具有指定名称的新容器。

```
TSContainerCreate(container_name varchar(18,1),dbspace_name varchar(18,1),
ts_type varchar(18,1),container_size integer,container_grow integer);
```

container_name新容器名。容器名称必须唯一。

dbspace_name存放容器的dbspace的名称。

ts_type将放置在容器中的时间序列的类型。

此参数必须是以时间戳开头的现有行类型的名称。

container_size容器的初始大小，单位为千字节。如果此参数为0或更小，则使用默认大小为16 KB。如果该参数为正数，则至少为4页。

container_grow容器增长的增量，单位为千字节。如果此参数为0或更小，则使用默认大小为16 KB。如果该参数为正数，则至少为4页。

TSContainerCreate过程接受以下参数:容器名称、dbspace的名称、时间序列类型的名称、容器的大小和增长增量的大小。您想要使用的dbspace必须已经存在，或者您可以使用默认的dbspace rootdbs。

作为TSContainerCreate的结果，当第一个时间序列插入到容器中时，Informix TimeSeries DataBlade模块将创建一个容器。当常规和不规则时间序列超过指定的大小(在创建时间序列时指定)时，它们都存储在容器中。

只有对TSContainerTable表具有更新权限的用户才能执行此过程。

下面的示例为时间序列类型stock_bar在rootdbs空间中创建一个名为new_cont的新容器:

```
1   execute procedure TSContainerCreate('new_cont','rootdbs','stock_bar',0,0);
```

删除容器TSContainerDestroy

TSContainerDestroy过程从TSContainerTable表中删除容器行，并删除容器及其相应的系统编目行。

TSContainerDestroy(container_name varchar(18,1));

只有当容器中不存在时间序列时，才允许销毁容器;即使是空的时间序列也能防止容器被破坏。

只有对TSContainerTable表具有更新权限的用户才能执行此过程。

例：execute procedure TSContainerDestroy('new_cont');

创建和填充时间序列

有几种方法可以创建时间序列的实例，这取决于是否有要加载的现有数据，以及如果有，该数据的格式是什么。下表列出了用于创建和填充时间序列的选项。

创建一个空的时间序列	TSCreate(规则时间序列)或TSCreateIrr(不规则时间序列)
创建具有元数据的空时间序列	使用元数据参数TSCreate(常规时间序列)，或者使用元数据参数TSCreateIrr(不规则时间序列)
创建并填充时间序列	带有set_ts参数的TSCreate(常规时间序列)，或者带有set_ts参数的TSCreateIrr(不规则时间序列)

用元数据创建和填充时间序列	TSCreate带有set_ts和metadata参数(常规时间序列)，或者TSCreateIrr带有set_ts和metadata参数(不规则时间序列)
填充现有的时间序列	BulkLoad Other functions, such as PutElem

1.使用TSCreate或TSCreateIrr创建空时间序列：

TSCreate和TSCreateIrr函数基于日历名称、起始时间戳、阈值、标志、元素数量和容器名称创建空时间序列.

```
insert into daily_stocks values(901,'IBM',TSCreate('daycal','1995-01-03 00:00:00.00000',20,0,0,NULL));
```

2.创建和填充时间序列

可以同时使用TSCreate或TSCreateIrr函数创建时间序列并将数据插入到包含时间序列列的表中，该函数带有一个附加参数，即一组行类型。如果要加载的数据位于单独的表中，则此方法非常有用。

例如，如果有一个名为activity_load_tab的表，其中列set_data类型为SET(stock_trade)，可以使用以下查询创建一个时间序列并将其插入activity_stocks表:

```
insert into activity_stocks
 select 1234,
 TSCreateIrr('daycal','2024-01-03 00:00:00.00000'::datetime year to fraction(5),
 20,0,NULL,set_data)::timeseries(stock_trade)
```

3.创建包含元数据的时间序列

要将元数据添加到时间序列，必须使用以下SQL语句创建基于TimeSeriesMeta数据类型的不同数据类型:创建不同类型MyMetaData as TimeSeriesMeta当创建不同数据类型时，它将继承源数据类型的表示和例程。在这种情况下，TimeSeriesMeta数据类型是可变长度的不透明数据类型，最大长度为512字节。TimeSeriesMeta数据类型是几个例程中的参数，这些例程将自动接受不同的MyMetaData数据类型。

```
create distinct type MyMetaData as TimeSeriesMeta
```

不透明的数据类型需要支持函数，比如输入、输出、发送、接收等等。当使用不透明数据类型作为参数调用时间序列函数时，将调用这些函数。但是，由于不提供timesiesmeta数据类型的支持函数，SinoDB TimeSeries DataBlade模块允许您定制MyMetaData的支持函数，以符合您的元数据需求。

一旦使用元数据创建了时间序列，就可以添加、更改、删除和检索元数据。您还可以检索元数据类型的名称。

4.用输入函数创建时间序列

可以使用时间序列输入函数用INSERT语句创建时间序列。

insert into table_name values(

'col1_value',

'col2_value',

...,

'parameter_input_string'

);

parameter_input_string值包含时间序列信息。所有数据类型都有一个关联的输入函数，当ASCII数据插入到列中时自动调用该函数。在TimeSeries数据类型的情况下，输入在文本中嵌入了几段数据。该信息用于传递日历的名称、起始时间戳、阈值、容器和初始时间序列数据。参数输入字符串的格式为:

paramname(value), paramname(value), ..., [data_element, ...]

参数paramname可以是以下参数中的任意一个，顺序任意:

- calendar
- origin
- threshold
- container
- irregular
- datafile

这些值是特定于参数的，并且每个值都有不同的格式。下表显示了与每个参数相关的值。

calendar	必需	要使用的日历的名称。没有默认名称。
origin	No	时间序列起源的时间戳。默认的起始日期是日历开始日期
threshold	No	上面的数据放在容器中而不是放在行中的元素数。 默认值是20。行内时间序列不应大于1500字节。
container	No	要使用的容器的名称。默认是没有容器;时间序列必须适合数据库行，否则永远不会分配给表。如果时间序列超过阈值大小，则必须设置容器。

irregular	视情况	没有值，只有字符串“不规则”。非规则时间序列必须包含该参数，规则时间序列不能包含该参数。
datafile	No	要使用的输入文件名。格式与BulkLoad函数相同。如果数据文件存在，则不允许使用“括号”中的数据。默认为NULL。
metadata	No	要添加到时间序列中的元数据。 可以是NULL。如果提供了元数据，则还必须提供元数据类型。
metatype	No	元数据数据类型

如果输入字符串中不存在参数，则使用其默认值。

如果没有指定数据文件，则可以在参数后面提供要放置在时间序列中的数据(数据元素)，用方括号括起来:

[(value, value, value, ...)@timestamp, (...), ...]

元素由数据值组成，每个数据值之间用逗号分隔。每个元素中的数据值对应于TimeSeries子类型中的列，但不包括初始时间戳列。每个元素都用圆括号括起来，后面跟着@符号和时间戳。时间戳对于规则时间序列是可选的，但是对于不规则时间序列是必须的。

元素之间用逗号分隔。Null数据值或元素用单词Null表示。如果没有数据元素存在，该函数将创建一个空的时间序列。

示例:创建并插入常规时间序列

下面是在表daily_stocks中创建常规时间序列的INSERT语句示例:

```
insert into daily_stocks values(1234,'sinoregal','origin(2024-01-03 00:00:00.00000),calendar(daycal),[(350,310,340,1999),(362,320,350,2500)]');
```

这个INSERT语句创建一个从2024年1月3日开始的时间序列，该时间序列是在名为daycal的日历指定的日期开始的。时间序列中的前两个元素使用带括号的数据填充。由于未指定阈值参数，因此使用其默认值。因此，如果在时间序列中放置了超过20个元素，SinoDB TimeSeries DataBlade模块将尝试将数据移动到容器中，但由于没有指定容器，因此会引发错误。

示例:创建并插入不规则时间序列

下面是在表activity_stocks中创建不规则时间序列的INSERT语句示例:

```
insert into activity_stocks values (600,'irregular,container(ctnr_stock),origin(2024-06-06 00:00:00.00000),calendar(daycal),[(6.25,1000,1,7,2,1)@2024-
```

INSERT语句创建一个时间序列，该序列从2024年06月06日开始，在名为daycal的日历指定的一天的时间开始。用指定的时间戳插入两行数据。

5.将数据加载到现有时间序列中

创建时间序列之后，有几种方法可以将数据加载到该序列中。可以使用的函数有BulkLoad, PutElem, PutSet, InsElem和InsSet。

可以使用BulkLoad函数将数据加载到现有的时间序列中。

这个函数接受一个现有的时间序列和一个文件名作为参数。文件名用于客户机上的文件，该文件包含要加载到时间序列中的行类型数据。

```
update table_name
set TimeSeries_col=BulkLoad(TimeSeries_col, 'filename')
where col1='value';
```

TimeSeries_col参数是包含行类型的列的名称。filename参数是数据文件的名称。WHERE子句指定要更新表中的哪一行。

BulkLoad加载的文件支持数据格式：使用类型构造函数

```
insert into daily_stocks values(999,'HP',TSCreate('daycal','2024-01-03 00:00:00.00000',20,0,0,NULL));

update daily_stocks
set stock_data=BulkLoad(stock_data,'/home/sinodb168/sam.dat')
where stock_name='HP';
```

类型构造函数格式遵循行类型约定，即由括号包围并以row类型构造函数开头的逗号分隔的列。典型文件的前两行是这样的:

row(2024-01-03 00:00:00.00000, 1.1, 2.2)

row(2024-01-04 00:00:00.00000, 10.1, 20.2)

```
[sinodbms@SFQ112 sinodb168]$ dbaccess testtimeseries -

Database selected.

> update daily_stocks set stock_data=BulkLoad(stock_data,'/home/sinodb168/sam.dat') where stock_name='HP';

1 row(s) updated.

> select * from daily_stocks;



stock_id    901
stock_name  IBM
stock_data  origin(1995-01-03 00:00:00.00000), calendar(daycal), container(), t
            hreshold(20), regular, []

stock_id    1234
stock_name  sinoregal
stock_data  origin(2024-01-03 00:00:00.00000), calendar(daycal), container(), t
            hreshold(20), regular, [(350.0000000000,310.0000000000,340.00000000
            00,1999.000000000), (362.0000000000,320.0000000000,350.0000000000,2
            500.000000000)]

stock_id    999
stock_name  HP
stock_data  origin(2024-01-03 00:00:00.00000), calendar(daycal), container(), t
            hreshold(20), regular, [(1.100000020000,2.200000050000,NULL,NULL),
            (10.10000040000,20.20000080000,NULL,NULL)]

3 row(s) retrieved.

>
```

当您使用onload实用程序时，也会生成这种文件格式。

如果在行数据类型中的列中包含集合，请使用类型构造函数(SET、MULTISET或LIST)和花括号括住集合值。包含行集的行具有以下格式:

row(timestamp, set{row(value, value), row(value, value)}, value)

您还可以使用以下任何函数将数据加载到时间序列中:

■PutElem使用单个元素更新时间序列。

■PutSet用一组元素更新时间序列。

■InsElem将元素插入到时间序列中。

■InsSet将给定集合中的每个元素插入到时间序列中。

这些函数向时间序列添加或更新一个或一组元素。它们必须在带有SET子句的SQL UPDATE语句中使用:

```
update table_name
set TimeSeries_col=FunctionName(TimeSeries_type, data)
where col1='value';
```

TimeSeries_col参数是时间序列所在列的名称。FunctionName参数是函数的名称。data参数采用行类型数据元素格式。WHERE子句指定要更新表中的哪一行.

```
update daily_stocks
set stock_data = PutElem(stock_data,
row(NULL::datetime year to fraction(5),
2.3, 3.4, 5.6, 67)::stock_bar)
where stock_name = 'IBM';
```

注意：

将数据加载到TimeSeries列后，运行以下命令:

update statistics high for table daily_stocks;

update statistics high for table daily_stocks (stock_id);

这提高了任何后续加载、插入和删除操作的性能。

第五章 使用虚拟表接口

本章内容

- 虚拟表接口概述
- 创建虚拟表
- 使用虚拟表进行查询
- 虚拟表的限制
- 跟踪

虚拟表接口概述

虚拟表接口允许您使用 SQL 访问和操作时间序列数据。通过虚拟表接口，您可以将时间序列数据视为传统的关系表，从而利用 SQL 的强大功能进行数据操作和查询。

例如，表daily_stocks包含一个名为stock_data的TimeSeries列，其中包含具有最高价格、最低价格、收盘价格和成交量列的行类型。

stock_id	stock_name	stock_data
900	IFMX	(*t1*, 7.25, 6.75, 7, 1000000), (*t2*, 7.5, 6.875, 7.125, 1500000), ...
901	IBM	(*t1*, 97, 94.25, 95, 2000000), (*t2*, 97, 95.5, 96, 3000000), ...
905	FNM	(*t1*, 49.25, 47.75, 48, 2500000), (*t2*, 48.75, 48, 48.25, 3000000), ...

虚拟表接口生成一个虚拟关系表，该表接受stock_data列行元素，并使它们成为单独的列。

stock_id	stock_name	timestamp*	high	low	final	vol
900	IFMX	*t1*	7.25	6.75	7	1000000
900	IFMX	*t2*	7.5	6.875	7.125	1500000
...	...	...			...	...
901	IBM	*t1*	97	94.25	95	2000000
901	IBM	*t2*	97	95.5	96	3000000
...	...	...			...	...
905	FNM	*t1*	49.25	47.75	48	2500000
905	FNM	*t2*	48.75	48	48.25	3000000
...	...	...			...	

* In this column, *t1* and *t2* are DATETIME values.

对虚拟表的SQL SELECT语句返回普通数据类型格式的数据，而不是TimeSeries数据类型格式的数据。

虚拟表不是存储在数据库中的真实表，因此没有重复的数据存储。在任何给定时刻，虚表中可见的数据在基表中也是可见的。

要查询stock_data列，必须使用TimeSeries DataBlade模块提供的函数。

select stock_id,Apply('$final', stock_data)::TimeSeries(one_real) from daily_stocks

具体函数的使用在后续介绍章节里。

在这个查询中，one_real是一个用来保存查询结果的行类型，它是用下面的语句创建的:

create row type one_real(timestamp datetime year to fraction(5),result real);

创建虚拟表

要创建一个虚拟表，可以使用TSCreateVirtualTab存储过程。

创建虚拟表时，必须指定虚拟表如何处理插入其中的数据，以便更新底层基表。TSCreateVirtualTab的两个参数指定了这种行为:

■NewTimeSeries

■TSVTMode

语法:

TSCreateVirtualTab(

VirtualTableName lvarchar, --新虚拟表的名称

BaseTableName lvarchar, --基表的名称

NewTimeSeries lvarchar，--(可选)如果允许对尚不存在的时间序列插入，则要创建新的空时间序列。

TSVTMode integer default 0 --(可选)设置虚拟表模式，取值为0和1

TSColName lvarchar default NULL -(可选)对于具有多个TimeSeries列的基表，使用此参数指定要用于创建虚拟表的TimeSeries列的名称。TSColName的默认值是NULL，在这种情况下，基表必须只有一个TimeSeries列。

)

“TSVTMode”的默认值为0。默认情况下，TimeSeries DataBlade模块使用PutElemNoDups向底层时间序列添加元素。如果一个元素在同一时间点已经存在，那么新元素将替换现有的元素。这使您能够执行基础时间序列的批量更新。TSVTMode = 1。TimeSeries DataBlade模块使用PutElem向底层时间序列添加元素。

定义虚拟表：

- 使用以下 SQL 命令定义一个虚拟表：

```
execute procedure TSCreateVirtualTab('daily_stocks_virt',
'daily_stocks',
'calendar(daycal), origin(2024-01-03 00:00:00.00000)'
);
```

上面的示例基于表daily_stocks创建一个名为daily_stocks_virt的虚拟表。由于此示例为NewTimeSeries参数指定了一个值，因此如果底层基表中不存在某个元素的时间序列，则虚拟表daily_stocks_virt允许插入。如果执行这样的插入，TimeSeries DataBlade模块将创建一个新的空时间序列，该序列使用日历daycal，起始日期为2024年1月3日。

```
execute procedure
TSCreateVirtualTab('daily_stocks_no_ts', 'daily_stocks');
```

因为该语句没有指定NewTimeSeries参数，所以daily_stocks_no_ts不允许插入在daily_stocks中没有相应时间序列的元素。此外，该语句省略了TSVTMode参数，其默认值为0。因此，如果将数据插入到daily_stocks_no_ts中，TimeSeries DataBlade模块将使用putelemnodup向daily_stocks中的底层时间序列添加一个元素。

虚拟表daily_stocks_no_ts如下所示:

stock_id	stock_name	timestamp*	high	low	final	vol
900	IFMX	*t1*	7.25	6.75	7	1000000
900	IFMX	*t2*	7.5	6.875	7.125	1500000
...	...	...			...	...
901	IBM	*t1*	97	94.25	95	2000000

使用虚拟表进行查询

通过虚拟表，可以像查询普通表一样查询时间序列数据。例如：

获取特定时间段内的价格和成交量的查询如下所示:

```
select * from daily_stocks_no_ts
where timestamp between t1 and t5;
```

时间序列函数难以完成的一些复杂任务(例如使用ORDER BY子句)现在变得简单了。例如:

```
select * from daily_stocks_no_ts
where timestamp between t1 and t5
order by vol;
```

向虚拟表中插入数据也很简单。要向IBM股票添加新元素，请使用以下查询:

```
insert into daily_stocks_virt
values(901,'IBM', t6, 55, 53, 54, 2000000);
```

执行后，会将元素(t6, 55,53,54,2000000)添加到基表daily_stocks中

使用虚拟表的限制

1、列名冲突

由于TimeSeries行类型中的列名用作结果虚拟表中的列名，因此必须确保这些列名不与基表中其他列的名称冲突。

2、在虚拟表上创建索引

不能在时间序列虚拟表上创建索引。

3、虚拟表的大小

虚拟表(非时间序列和TimeSeries列的组合)中的行总长度受到与任何其他数据库表相同的限制。服务器目前最多允许32 KB。

4、修改虚表中的数据

可以对时间序列虚拟表使用SELECT和INSERT语句。

不能使用UPDATE或DELETE语句，但是可以通过在虚拟表中插入具有相同时间戳的新元素来更新基表中的时间序列元素。

5、具有多个TimeSeries列的基表

只能基于单个TimeSeries列创建虚拟表。如果你的基表有多个TimeSeries列，你必须使用TSCreateVirtualTab过程的TSColName参数来指定使用哪个TimeSeries列。

跟踪TRC

跟踪函数可用于帮助您调试使用虚拟表的工作。除非您与SinoDB技术支持或工程专业人员一起工作，否则不应该使用这些跟踪函数。

TSSetTraceFile允许你指定一个附加跟踪信息的文件。

TSSetTraceLevel设置要执行的跟踪级别:实际上，打开或关闭跟踪。

TSSetTraceFile语法和使用：

TSSetTraceFile(traceFileName lvarchar) --附加跟踪信息的文件的完整路径和名称

returns integer; --成功时返回0，失败时返回-1。

使用TSSetTraceFile指定的文件将覆盖任何当前跟踪文件。

该文件驻留在服务器计算机上。默认的跟踪文件为/tmp/session_number.trc。

TSSetTraceFile调用mi_set_trace_file()数据库DataBlade API 函数。

execute function TSSetTraceFile('/tmp/test1.trc');

TSSetTraceLevel语法和使用

TSSetTraceLevel(traceLevelSpec lvarchar) --指定特定跟踪类的跟踪级别的字符串。格式为TS_VTI_DEBUG

returns integer;

TSSetTraceLevel设置跟踪类的跟踪级别。跟踪级别决定为给定的跟踪类记录哪些信息。虚拟表信息的跟踪类是TS_VTI_DEBUG。对TS_VTI_DEBUG跟踪类启用跟踪的级别是1001。必须将跟踪级别设置为1001或更高才能启用跟踪。缺省情况下，跟踪级别低于1001。

TSSetTraceLevel调用mi_set_trace_level() DataBlade API函数。

execute function TSSetTraceLevel('TS_VTI_DEBUG 1001');

第六章 日历模式例程

描述了SinoDB TimeSeries DataBlade模块提供的供用户执行的例程。

AndOp

AndOp函数返回两个日历模式的交集。

AndOp (cal_patt1 CalendarPattern,cal_patt2 CalendarPattern) returns CalendarPattern;

这个函数返回一个日历模式，其中每个间隔在两个日历模式中都是开启的，其余的都是关闭的。如果给定的模式没有相同的间隔单位，则扩展具有较大间隔单位的模式以匹配另一个模式。

第一个AndOp语句返回两个日日历模式的交集，第二个AndOp语句返回一个小时和一个日日历模式的交集:

```
select * from CalendarPatterns where cp_name = 'workweek_day';
cp_name workweek_day
cp_pattern {1 off,5 on,1 off},day

select * from CalendarPatterns  where cp_name = 'fourday_day';
cp_name fourday_day
cp_pattern {1 off,4 on,2 off},day

select * from CalendarPatterns  where cp_name = 'workweek_hour';
cp_name workweek_hour
cp_pattern {32 off,9 on,15 off,9 on,15 off,9 on,15 off, 9
on,15 off,9 on,31 off},hour

select AndOp(p1.cp_pattern, p2.cp_pattern) from CalendarPatterns p1, Calen
darPatterns p2
 where p1.cp_name = 'workweek_day' and p2.cp_name = 'fourday_day';
(expression)
{1 off,4 on,2 off},day

select AndOp(p1.cp_pattern, p2.cp_pattern) from CalendarPatterns p1, Calen
darPatterns p2
 where p1.cp_name = 'workweek_hour' and p2.cp_name = 'fourday_day';
(expression)
{32 off,9 on,15 off,9 on,15 off,9 on,15 off,9
```

OrOp

OrOp函数返回两个日历模式的并集

OrOp（ cal_patt1 CalendarPattern,cal_patt2 CalendarPattern)

returns CalendarPattern;

这个函数返回一个日历模式，其中每个间隔都是打开的，其余的都是关闭的。如果两个图案具有不同大小的间隔单元，则生成的图案具有两个间隔中较小的那个。

下面的第一个OrOp语句返回两个日日历模式的并集。第二个OrOp语句返回一个小时和一个日日历模式的并集:

```
select * from CalendarPatterns where cp_name = 'workweek_day';
cp_name workweek_day
cp_pattern {1 off,5 on,1 off},day

select * from CalendarPatterns where cp_name = 'fourday_day';
cp_name fourday_day
cp_pattern {1 off,4 on,2 off},day

select * from CalendarPatterns where cp_name = 'workweek_hour';
cp_name workweek_hour
cp_pattern {32 off,9 on,15 off,9 on,15 off,9 on,15 off, 9
on,15 off,9 on,31 off},hour

select OrOp(p1.cp_pattern, p2.cp_pattern) from CalendarPatterns p1, Calend
arPatterns p2
 where p1.cp_name = 'workweek_day' and p2.cp_name = 'fourday_day';
(expression)
{1 off,5 on,1 off},day
(expression)
{24 off,96 on,8 off,9 on,31 off},hour
```

NotOp

NotOp函数在给定的日历模式中关闭所有打开间隔，并打开所有关闭间隔。

NotOp (cal_patt CalendarPattern)

returns CalendarPattern;

下面的语句转换workweek_day日历

```
select * from CalendarPatterns where cp_name = 'workweek_day';
cp_name workweek_day
cp_pattern {1 off,5 on,1 off},day

select NotOp(cp_pattern) from CalendarPatterns  where cp_name = 'workweek_d
ay';
(expression)
{1 on,5 off,1 on}, day
```

CalPattStartDate

CalPattStartDate函数接受一个日历名称，并返回一个DATETIME，其中包含该日历模式的开始日期。

CalPattStartDate(calname lvarchar）returns datetime year to fraction(5);

等效的API函数是ts_cal_pattstartdate()。

下面的示例返回CalendarTable表中每个日历的日历模式的开始日期:

```
select c_name, CalPattStartDate(c_name) from CalendarTable;
```

Collapse

Collapse函数将给定的日历模式折叠为目标单元，目标单元必须具有比给定日历模式更大的间隔单元。

Collapse (cal_patt CalendarPattern,interval lvarchar)

returns CalendarPattern;

---The destination time interval: minute, hour, day, week,month, or year.

下面的语句将小时日历模式转换为日日历模式:

```
select * from CalendarPatterns where cp_name = 'workweek_hour';
cp_name workweek_hour
cp_pattern {32 off,9 on,15 off,9 on,15 off,9 on,15 off,9
on,15 off,9 on,31 off},hour

select Collapse(cp_pattern, 'day') from CalendarPatterns where cp_name = 'w
orkweek_hour';
(expression)
{1 off,5 on,1 off},day
```

Expand

Expand函数将给定的日历模式扩展为目标单元，目标单元必须具有比给定日历模式更小的间隔单元。

Expand (cal_patt CalendarPattern,interval lvarchar)

returns CalendarPattern;

-----The destination time interval: second, minute, hour, day,week, or month.

下面的语句将日日历模式转换为小时日历模式:

```
select * from CalendarPatterns where cp_name = 'workweek_day';
cp_name workweek_day
cp_pattern {1 off,5 on,1 off},day

select Expand(cp_pattern, 'hour') from CalendarPatterns  where cp_name = 'w
orkweek_day';
(expression)
{24 off,120 on,24 off},hour
```

第七章 日历例程

描述了SinoDB TimeSeries DataBlade模块提供的供用户执行的例程。

AndOp

AndOp函数返回两个日历的交集。

AndOp（ cal1 Calendar,cal2 Calendar）returns Calendar;

这个函数返回一个日历，其中包含两个日历中都打开的所有间隔，其余的都关闭。生成的日历采用两个开始日期中较晚的日期和两个模式开始日期中较晚的日期。如果两个日历具有不同大小的间隔单位，则生成的日历具有两个间隔中较小的那个。

下面的AndOp语句返回小时日历和具有不同开始日期的日历的交集:

```
select c_calendar from CalendarTable where c_name = 'hourcal';
c_calendar startdate(1994-01-01 00:00:00), pattstart(1994-
 01-02 00:00:00), pattern({32 off,9 on,15 off,9
 on,15 off,9 on,15 off,9 on,15 off, 9 on,31
 off},hour)

select c_calendar from CalendarTable where c_name = 'daycal';
c_calendar startdate(1994-04-01 00:00:00), pattstart(1994-
04-03 00:00:00), pattern({1 off,5 on,1 off},day)

select AndOp(c1.c_calendar, c2.c_calendar) from CalendarTable c1, Calendar
Table c2
 where c1.c_name = 'daycal' and c2.c_name ='hourcal';

(expression)
startdate(1994-04-01 00:00:00), pattstart(19
94-04-03 00:00:00), pattern({32 off,9 on,15
off,9 on,15 off,9 on,15 off,9 on,15 off, 9 on
,31 off},hour)
```

OrOp

OrOp函数返回两个日历的并集

OrOp（ cal1 Calendar,cal2 Calendar)

returns Calendar;

这个函数返回一个日历，其中每个间隔都是打开的，其余的都是关闭的。生成的日历采用两个开始日期和两个模式开始日期中较早的一个。如果两个日历具有不同大小的间隔单位，则生成的日历具有两个间隔中较小的那个。

下面的OrOp函数返回小时日历和具有不同开始日期的日日历的并集:

```
select c_calendar from CalendarTable where c_name = 'hourcal';
c_calendar startdate(1994-01-01 00:00:00), pattstart(1994-
 01-02 00:00:00), pattern({32 off,9 on,15 off,9
 on,15 off,9 on,15 off,9 on,15 off, 9 on,31
 off},hour)

select c_calendar from CalendarTable where c_name = 'daycal';
c_calendar startdate(1994-04-01 00:00:00), pattstart(1994-
04-03 00:00:00), pattern({1 off,5 on,1 off},day)

select OrOp(c1.c_calendar, c2.c_calendar) from CalendarTable c1, CalendarT
able c2
 where c1.c_name = 'daycal' and c2.c_name ='hourcal';
(expression)
startdate(1994-01-01 00:00:00), pattstart(19
94-01-02 00:00:00), pattern({24 off,120 on,24
off} hour)
```

CalIndex

CalIndex函数返回日历中两个给定时间戳之间的有效间隔数。

CalIndex(cal_name lvarchar,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5))

returns integer;

等效的API函数是ts_cal_index()。

下面的查询返回日历日中2024-01-03和2024-01-05之间的间隔的个数:

```
select CalIndex('daycal','2024-01-03 00:00:00.00000',
'2024-01-05 00:00:00.00000')
 from systables  where tabid = 1;
```

CalRange

CalRange函数返回一个范围内的一组有效时间戳。

CalRange(cal_name lvarchar,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5))

returns list(datetime year to fraction(5));

CalRange(cal_name lvarchar,

begin_stamp datetime year to fraction(5),

num_stamps integer)

returns list(datetime year to fraction(5));

第一种语法指定两个给定时间戳之间的范围。第二种语法指定在给定时间戳之后返回的有效时间戳的数量.

等效的API函数是ts_cal_range()。

下面的查询返回日历daycal中2024-01-03和2024-01-15之间的所有时间戳的列表:

```
execute function CalRange('daycal',
'2024-01-03 00:00:00.00000',
'2024-01-15 00:00:00.00000'::datetime year to fraction(5));
```

下面的查询返回日历daycal中2024-01-03后面的两个时间戳的列表:

```
execute function CalRange('daycal','2024-01-03 00:00:00.00000', 2);
```

CalStamp

CalStamp函数在给定时间戳之后返回给定日历间隔数的时间戳。

CalStamp(cal_name lvarchar, tstamp datetime year to fraction(5), num_stamps integer)

returns datetime year to fraction(5);

等效的API函数是ts_cal_stamp()。

下面的例子返回的时间戳是在2024-01-03之后的两个间隔:

```
execute function CalStamp('daycal','2024-01-03 00:00:00.00000', 2);
```

CalStartDate

CalStartDate函数接受一个日历名称，并返回一个包含该日历开始日期的DATETIME。

CalStartDate(cal_name lvarchar)

returns datetime year to fraction(5);

等效的API函数是ts_cal_startdate()。

下面的示例返回CalendarTable表中所有日历的开始日期:

```
1   select c_name, CalStartDate(c_name) from CalendarTable;
```

第八章 时间序列SQL例程

时间序列SQL例程创建特定时间序列类型的实例，并向其中添加数据或更改数据。还提供了SQL例程来检查、分析、操作和聚合时间序列中的数据。

类型	描述	函数名
从时间序列中获取信息	得到原点	GetOrigin
	获取间隔	GetInterval
	获取日历	GetCalendar
	获取日历名称	GetCalendarName
	获取容器名称	GetContainerName
	获取用户定义的元数据	GetMetaData
	获取元数据类型	GetMetaTypeName
	确定时间序列是否规则	IsRegular
	如果时间序列存储在容器中，则获取实例ID	InstanceId

类型	描述	函数名
在时间戳和偏移量之间进行转换	返回给定时间戳的偏移量	GetIndex(仅普通)
	返回给定偏移量的时间戳	GetStamp(仅普通)

类型	描述	函数名

计算元素的数量	返回元素的个数	GetNelems
	获取两个时间戳之间的元素数量	ClipGetCount

类型	描述	函数名
选择单个元素	获取与给定时间戳相关联的元素	GetElem
	获取时间戳处或之前的元素	GetLastValid
	获取时间戳后面的元素	GetNextValid
	获取时间戳前的元素	GetPreviousValid
	获取指定位置的元素	GetNthElem(仅限常规)
	得到第一个元素	GetFirstElem
	获取最后一个元素	GetLastElem

类型	描述	函数名
修改元素或一组元素	添加或更新单个元素	PutElem
	添加或更新单个元素	PutElemNoDups
	在给定的偏移量处添加或更新单个元素	PutNthElem(仅限常规)
	添加或更新整个集合	PutSet
	在给定的时间点删除元素	DelElem
	删除指定时间段内的所有元素	DelClip
	插入元素	InsElem
	插入一组	InsSet
	更新元素	UpdElem
	更新一个集合	UpdSet
	把一个时间序列的每个元素放到另一个时间序列中	PutTimeSeries
修改元数据	更新用户自定义元数据	UpdMetaData

类型	描述	函数名
使元素对扫描可见或不可见	使元素不可见	HideElem
	使元素可见	RevealElem

类型	描述	函数名
提取并使用时间序列的一部分	提取两个时间戳之间的周期或对应于一组值，并对每个条目运行表达式或函数	Apply
	提取两个时间点之间的数据	Clip
	剪辑一定数量的元素	ClipCount
	提取一个包含给定时间的时间段	WithinC
	提取在给定时间开始或结束的时间段	WithinR
将新日历应用于时间序列	应用日历	ApplyCalendar

类型	描述	函数名
创建并加载时间序列	从客户端文件加载数据	BulkLoad
	创建常规空时间序列，或常规填充时间序列，或具有元数据的常规时间序列	TSCreate
	创建不规则的空时间序列，或不规则的填充时间序列，或具有元数据的不规则时间序列	TSCreateIrr

类型	描述	函数名
找出时间序列的交集或并集	构建多个时间序列的交集，并可选择剪辑结果	Intersect
	构建多个时间序列的并集，并可选择剪辑结果	Union
对时间序列进行转置	将时间序列数据转换为表格形式	Transpose

类型	描述	函数名

对时间序列执行统计计算	对时间序列类型执行求和	Sum
	对SMALLFLOAT或双精度值求和	TSAddPrevious
	计算衰减函数	TSDecay
	计算SMALLFLOAT或DOUBLE PRECISION值的运行平均值	TSRunningAvg
	计算SMALLFLOAT或DOUBLE PRECISION值的总和	TSRunningSum
	比较SMALLFLOAT或DOUBLE PRECISION值	TSCmp
	返回先前保存的值	TSPrevious

类型	描述	函数名
对一个或两个时间序列执行算术运算	将两个时间序列相加	Plus
	用一个时间序列减去另一个时间序列	Minus
	用一个时间序列乘以另一个	Times
	用一个时间序列除以另一个	Divide
	把第一个论点提高到第二个论点的倍数	Pow
	得到绝对值	Abs
	对时间序列取幂	Exp
	得到时间序列的自然对数	Logn

类型	描述	函数名
对一个或两个时间序列执行算术运算(续)	求一个时间序列除以另一个时间序列的模数或余数	Mod
	否定一个时间序列	Negate
	返回参数;被绑定到一元+操作符正	Positive
	将时间序列四舍五入到最接近的整数	Round

	得到时间序列的平方根	Sqrt
聚合时间序列中的值	AggregateBy函数使用一组值聚合时间序列中的值。此函数通常用于将具有小间隔的时间序列转换为具有较大间隔的时间序列，例如，将每日时间序列转换为每周时间序列。	AggregateBy
创建一个滞后的时间序列	创建一个滞后于源时间序列给定偏移量的时间序列	Lag
重置原点	重置原点	SetOrigin
管理容器	创建一个容器	TSContainerCreate
	销毁容器	TSContainerDestroy
	设置容器名称	SetContainerName

AggregateBy

AggregateBy(agg_express lvarchar,　--这些SQL聚合操作符的逗号分隔列表:MIN、MAX、SUM、AVG、FIRST、LAST或n

cal_name lvarchar,ts TimeSeries)　--定义聚合周期的日历的名称。

returns TimeSeries;

下面的查询将daily_stock时间序列聚合为一个weekly时间序列:

insert into weekly_stocks(stock_id, stock_name, stock_data)

select stock_id, stock_name,

AggregateBy('max($high), min($low),

last($final),sum($vol)',

'weeklycal', stock_data)::TimeSeries(daybar)

from daily_stocks;

下面的查询子句选择每周的第二个价格:

AggregateBy('Nth($price, 2)', 'weekly', ts)

这个查询子句选择每周的第二个到最后一个价格:

AggregateBy('Nth($price, -2)', 'weekly', ts)

Apply

Apply函数查询一个或多个时间序列，并对选定的时间序列元素应用用户指定的SQL表达式或函数。

Apply(sql_express lvarchar,

ts TimeSeries, ...)

returns TimeSeries;

Apply(sql_express lvarchar,

set_ts set(TimeSeries))

returns TimeSeries;

Apply(sql_express lvarchar,

filter lvarchar,

ts TimeSeries, ...)

returns TimeSeries;

Apply(sql_express lvarchar,

filter lvarchar,

set_ts set(TimeSeries))

returns TimeSeries;

Apply(sql_express lvarchar,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5),

ts TimeSeries, ...)

returns TimeSeries with (handlesnulls);

Apply(sql_express lvarchar,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5),

set_ts set(TimeSeries))

returns TimeSeries with (handlesnulls);

Apply(sql_express lvarchar,

filter lvarchar,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5),

ts TimeSeries, ...)

returns TimeSeries with (handlesnulls);

Apply(sql_express lvarchar,

filter lvarchar,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5),

set_ts set(TimeSeries))

returns TimeSeries with (handlesnulls);

该函数在给定的时间序列上运行用户指定的SQL表达式，并生成一个新的时间序列，其中包含输入时间序列的每个限定元素处的表达式结果。

可以通过指定要剪辑的时间段和使用筛选器表达式来限定输入时间序列中的元素。

sql_express参数是为每个选定元素运行的以逗号分隔的表达式列表。您可以运行的表达式数量没有限制。表达式的结果必须匹配结果时间序列的相应列减去第一个时间戳列。不要将第一个时间戳指定为第一个表达式;为每个表达式结果生成第一个时间戳。

表达式的参数可以是输入元素或输入时间序列的任何列。应该使用$，后跟给定时间序列在输入时间序列列表中的位置，以表示其数据元素，再加上一个点，然后是列的编号。位置号和列号都是从零开始的。

例如，$0表示第一个输入时间序列的元素，$0.0表示它的时间戳列，$0.1是时间戳列后面的一列。引用列的另一种方法是直接使用列名，而不是使用列号。

```
select Apply('$high-$low',
datetime(2024-01-01) year to day,
datetime(2024-01-06) year to day,
stock_data)::TimeSeries(one_real)
from daily_stocks
where stock_name = 'IBM';
```

```
select Apply(
'($0.high+$1.high)/2, ($0.low+$1.low)/2,
($0.final+$1.final)/2, ($0.vol+$1.vol)/2',
datetime(2024-01-04) year to day,
datetime(2024-01-05) year to day,
t1.stock_data, t2.stock_data)
::TimeSeries(stock_bar)
from daily_stocks t1, daily_stocks t2
where t1.stock_name = 'IBM' and t2.stock_name = 'HWP';
```

下面的示例显示了带有过滤器但不带剪辑范围的应用。

由此产生的时间序列包含交易范围超过最低价10%的交易日的收盘价:

```
 create function ts_sum(a stock_bar)
  returns one_real;
  return row(null::datetime year to fraction(5),
 (a.high + a.low + a.final + a.vol))::one_real;
 end function;

 select Apply('ts_sum',
 '2024-01-03 00:00:00.00000'::datetime year
 to fraction(5),
 '2024-01-03 00:00:00.00000'::datetime year
 to fraction(5),stock_data)::TimeSeries(one_real)
 from daily_stocks where stock_id = 901;
```

下面的查询生成每日平均股票价格的时间序列(实际上是每日最高价和最低价的平均值)。为了方便起见，本例首先将高点和低点投影到不同的时间序列中:

```
create row type price( timestamp datetime year to fraction(5),val real);
create row type simple_series( stock_id int, data TimeSeries(price));
create table daily_high of type simple_series;
insert into daily_high
 select stock_id,Apply('$high','2024-01-03 00:00:00.00000' ::datetime year
 to fraction(5),'2024-01-10 00:00:00.00000'::datetime year to fraction(5),
stock_data)::TimeSeries(price)
from daily_stocks;

create table daily_low of type simple_series;
insert into daily_low
 select stock_id,
 Apply('$low',
 '2024-01-03 00:00:00.00000'
 ::datetime year to fraction(5),
 '2024-01-10 00:00:00.00000'
 ::datetime year to fraction(5),
 stock_data)::TimeSeries(price)
from daily_stocks;
create table daily_avg of type simple_series;
insert into daily_avg
 select l.stock_id, (l.data + h.data)/2
 from daily_low l, daily_high h
 where l.stock_id = h.stock_id;
```

ApplyCalendar

ApplyCalendar函数将新日历应用于时间序列。

ApplyCalendar (ts TimeSeries,cal_name lvarchar)

returns TimeSeries;

如果参数指定的日历的间隔小于附加到原始时间序列的日历，并且原始时间序列是规则的，则生成的时间序列具有更高的频率，因此可以比原始时间序列具有更多的元素。例如，将每天有八个有效时间点的小时日历应用于每日时间序列，将新时间序列中的每个每日条目转换为八个小时条目。

假设fourdaycal是一个包含四天工作周的日历，下面的查询将返回给定股票在四个工作日中的每个工作日的数据时间序列:

select ApplyCalendar(stock_data,'fourdaycal')

from daily_stocks

where stock_name = 'IBM';

ApplyUnaryTsOp

ApplyUnaryTsOp函数将一元算术函数应用于时间序列。

ApplyUnaryTsOp(func_name lvarchar,

ts TimeSeries)

returns TimeSeries;

```
create row type simple_series( stock_id int, data
TimeSeries(one_real));
create table daily_high of type simple_series;
insert into daily_high
select stock_id,
Apply( '$0.high',
NULL::datetime year to fraction(5),
NULL::datetime year to fraction(5),
stock_data)::TimeSeries(one_real)
from daily_stocks;
create table daily_low of type simple_series;
insert into daily_low
select stock_id,
Apply( '$0.low',
NULL::datetime year to fraction(5),
NULL::datetime year to fraction(5),
stock_data)::TimeSeries(one_real)
from daily_stocks;
create table daily_avg of type simple_series;
insert into daily_avg
select l.stock_id, ApplyBinaryTSOp("plus", l.data,
h.data)/2
from daily_low l, daily_high h
where l.stock_id = h.stock_id;
create table log_high of type simple_series;
insert into log_high
select stock_id, ApplyUnaryTsOp( "logn",
data) from daily_avg;
```

Clip

Clip函数提取时间序列中两个时间点之间的数据，然后创建并返回包含该数据的新时间序列。这允许您从大型时间序列中提取感兴趣的时间段，并将它们与大型序列分开存储或操作。

Clip(ts TimeSeries,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5)

[, flag integer])

returns TimeSeries;

Clip(ts TimeSeries,

begin_stamp datetime year to fraction(5),

end_offset integer

[, flag integer])

returns TimeSeries;

Clip(ts TimeSeries,

begin_offset integer,

end_stamp datetime year to fraction(5)

[, flag integer])

returns TimeSeries;

Clip(ts TimeSeries,

begin_offset integer,

end_offset integer

[, flag integer])

returns TimeSeries;

Clip函数都接受一个时间序列、一个起始点和一个范围的结束点。

对于常规时间序列，起始点和结束点可以是整数或时间戳。如果起始点是整数，则它是时间序列中某个条目的绝对偏移量。如果它是一个时间戳，Clip函数使用时间序列的日历来查找与时间戳对应的偏移量。如果时间序列中没有与请求的时间戳完全一致的条目，Clip将使用紧跟着给定时间戳的日历时间戳作为范围的起始点。结束点的使用方式与开始点相同，只是它指定了范围的结束，而不是开始。

对于不规则时间序列，起始点和结束点只允许使用时间戳。

开始和结束偏移量的数据包含在结果时间序列中。

如果未设置flag参数(默认值为0)，则结果时间序列的原点是起始点参数和输入时间序列的原点的后一个。如果设置了标志(值为1)，则结果时间序列的原点被设置为起始点参数和输入时间序列的原点中较早的一个。在这种情况下，时间序列起源之前的时间点被设置为NULL。

下面的查询从时间序列中提取数据，并创建一个包含给定股票一周数据的表:

```
create table week_1_analysis (stock_id int, stock_data TimeSeries(stock_bar
));
insert into week_1_analysis
 select stock_id,
Clip(stock_data,
'2024-01-03 00:00:00.00000'
 ::datetime year to fraction(5),
'2024-01-07 00:00:00.00000'
 ::datetime year to fraction(5))
from daily_stocks  where stock_name = 'IBM';
```

下面的查询显示时间序列中给定股票的前六个条目:

select Clip(stock_data, 0, 5) from daily_stocks where stock_name ='IBM';

ClipCount

ClipCount函数是Clip的一个变体，其中第一个整数参数被解释为一个计数。如果计数为正数，则ClipCount从时间戳处或之后的第一个元素开始，并剪辑下一个计数项。如果计数为负，则ClipCount从时间戳处或之前的第一个元素开始，并剪辑前一个计数项。

ClipCount(ts TimeSeries,

begin_stamp datetime year to fraction(5),

num_stamps integer

[, flag integer])

returns TimeSeries;

如果日历中没有与请求的时间戳完全一致的条目，ClipCount将使用日历的第一个有效时间戳，该时间戳紧跟着给定的时间戳作为范围的起点。如果起始点为空，则使用时间序列的原点。

下面的示例截取ID为600的股票在2024年3月14日上午9:30时或之后的前30个元素，并返回整个结果时间序列:

select ClipCount(activity_data,'2024-01-01 09:30:00.00000', 30)

from activity_stocks where stock_id = 600;

以下示例截取ID为600的股票在2024年8月22日或之前的午夜12:00的前60个元素:

select ClipCount(activity_data,'1994-08-22 00:00:00.00000', -60)

from activity_stocks where stock_id = 600;

ClipGetCount

ClipGetCount函数返回当前时间序列中出现在由时间戳分隔的时间段内的元素数。

ClipGetCount(ts TimeSeries,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5))

returns integer;

对于不规则的时间序列，删除的元素不计算在内。对于常规时间序列，只计算非空的条目，因此ClipGetCount可能返回与getelems不同的值。

如果起始点为空，则使用时间序列原点。如果结束点为空，则使用时间序列的结束。

下面的语句返回10:30 A.M.之间的元素数

2024年3月14日和2024年3月19日午夜，包括在内:

```
select ClipGetCount(activity_data,
'2024-03-14 10:30:00.00000','2024-03-19 00:00:00.00000')
 from activity_stocks  where stock_id = 600;
```

DelClip

DelClip函数删除指定时间范围内的所有元素，包括分隔时间点。

DelClip(ts TimeSeries,

begin_stamp datetime year to fraction(5),

end_stamp datetime year to fraction(5))

returns TimeSeries;

下面的示例删除给定日期内6小时范围内的所有元素:

```
update activity_stocks
set activity_data = DelClip(activity_data,
 '2024-01-05 00:00:00.00000'
 ::datetime year to fraction(5),
 '2024-01-06 00:00:00.00000'
 ::datetime year to fraction(5))
where stock_id = 600;
```

DelElem

DelElem函数在给定的时间点删除元素。

DelElem(ts TimeSeries,

tstamp datetime year to fraction(5))

returns TimeSeries;

如果在指定的时间点上没有元素，则不会删除任何元素，也不会引发错误。

```
update activity_stocks
set activity_data = DelElem(activity_data,
'2024-01-05 12:58:09.23456'
::datetime year to fraction(5))
where stock_id = 600;
```

GetCalendar

GetCalendar函数返回与给定时间序列关联的日历。

GetCalendarName函数返回给定时间序列使用的日历的名称。

GetContainerName函数返回给定时间序列的容器名称

```
select GetCalendar(stock_data)
from daily_stocks
where stock_name ='IBM';
(expression)
startdate(2024-01-01 00:00:00),pattstart(1994-
 01-02 00:00:00),pattern({1 off,5 on,1 off},day)


select GetCalendarName(stock_data)
from daily_stocks
where stock_name ='IBM';
(expression) daycal

select GetContainerName(activity_data)
from activity_stocks
where stock_id = 600;
```

GetElem

GetElem函数为给定的时间戳提取元素

GetFirstElem函数返回时间序列中的第一个元素。

GetLastElem函数返回存储在时间序列中的最后一个条目。

select GetElem(stock_data,'2024-01-04 00:00:00.00000')from daily_stocks

select GetFirstElem(activity_data) from activity_stocks where stock_id = 600;

select GetLastElem(stock_data) from daily_stocks

GetIndex

GetIndex函数返回与所提供的时间戳相关联的常规时间序列条目的索引(偏移量)

select stock_name, GetIndex(stock_data,'2024-01-05 00:00:00.00000') from daily_stocks;

GetInterval函数返回时间序列使用的间隔

select stock_name from daily_stocks

where GetInterval(stock_data) <> 'day';

GetOrigin函数返回时间序列的原点

select GetOrigin(stock_data) from daily_stocks

GetThreshold函数返回与指定时间序列关联的阈值。

select GetThreshold(stock_data) from daily_stocks;

HideElem函数将给定时间戳中的一个元素或一组元素标记为不可见。

select HideElem(stock_data,'2024-01-03 00:00:00.00000') from daily_stocks

InsElem

InsElem函数将元素插入到时间序列中。

update activity_stocks

set activity_data =InsElem(activity_data,

row('2023-10-06 08:06:56.00000', 6.50, 2000,1, 007, 3, 1)::stock_trade)

where stock_id = 600;

InsSet

InsSet函数将给定集合的每个元素插入到时间序列中。

update activity_stocks

set activity_data = (select InsSet(activity_data, set_data)

from activity_load_tab where stock_id = 600) where stock_id = 600;

InstanceId

InstanceId函数确定时间序列是否存储在容器中，如果是，则返回该时间序列的实例ID。

select stock_id, InstanceId(activity_data) from activity_stocks;

PutElem

PutElem函数在提供的行类型所指示的时间点向时间序列添加一个元素。

update daily_stocks set stock_data = PutElem(stock_data,

row(NULL::datetime year to fraction(5),2.3, 3.4, 5.6, 67)::stock_bar)

where stock_name ='IBM';

PutElemNoDups函数将单个元素插入到时间序列中。如果在指定的时间点已经有一个元素，它将被新元素替换。

update activity_stocks

set activity_data = PutElemNoDups(activity_data,row('2023-08-25 09:06:00.00000', 6.25,

1000, 1, 007, 2, 1)::stock_trade)

where stock_id = 600;

PutTimeSeries函数将第一个时间序列的每个元素(隐藏元素除外)放入第二个时间序列。

下面的示例将常规时间序列转换为不规则时间序列。daily_stocks表保存有规律的时间序列数据，activity_stocks表保存有不规则的时间序列数据。此外，daily_stocks时间序列中的元素从stock_bar转换为stock_trade:

update activity_stocks

 set activity_data = PutTimeSeries(activity_data,'calendar(daycal), irregular'
::TimeSeries(stock_trade))

 where stock_id = 600;

SetContainerName

当函数(例如Apply)返回的时间序列的元素与源时间序列不同时，需要此函数。发生这种情况时，返回的时间序列不能使用源时间序列的容器。因此，如果希望将返回的时间序列插入到表中，应该使用

SetContainerName为其分配一个容器。

下面的示例创建容器tsirr并为其设置时间序列:

```
execute procedure TSContainerCreate('tsirr','rootdbs','stock_bar_union',0,0);

select SetContainerName(Union(s1.stock_data,s2.stock_data)::TimeSeries(stock_bar_union),‘tsirr’)

from daily_stocks s1, daily_stocks s2

where s1.stock_name = 'IBM' and s2.stock_name = 'HWP';
```

结论

本用户指南介绍了 TimeSeries DataBlade 模块的基本和高级功能，包括数据创建、修改、查询、分析和管理等方面。通过学习和使用这些功能，用户可以高效地管理和分析时间序列数据。
