Oracle8*i*™ Time Series

User's Guide

Release 8.1.5

February 1999

A67294-01

Enabling the Information Age™

Oracle8*i* Time Series User’s Guide

A67294-01

Release 8.1.5

Contents

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle8*i* Time Series User's Guide, Release 8.1.5

A67294-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this guide?

If you find any errors or have any other suggestions for improvement, please indicate the book title and (if possible) the chapter, section, and page number. You can send comments to us in the following ways:

- Electronic mail: nedc_doc@us.oracle.com
- FAX - 603-897-3316. Attn: Time Series writer
- Postal service:
  Oracle Corporation
  Time Series Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you would like a reply, please include your name and contact information.

If you have problems with the software, please contact your local Oracle Worldwide Support Center.

Preface

This guide describes how to use Oracle8*i* Time Series. (In previous releases, this product was called the Oracle8 Time Series Cartridge.)

For changes to this guide for the current release, see "Changes to This Guide" at the end of this Preface.

Intended Audience

This guide is intended for anyone who is interested in storing, retrieving, and manipulating time series data in an Oracle database, including developers wishing to extend Oracle8*i* Time Series.

Structure

This guide contains the following chapters, appendixes, and glossary:

Chapter 1	Introduces object types and the contents of the Oracle8*i* Time Series product.
Chapter 2	Explains time series concepts and operations.
Chapter 3	Explains important procedures for using Oracle8*i* Time Series.
Chapter 4	Provides reference information on calendar functions.
Chapter 5	Provides reference information on time series functions.
Chapter 6	Provides reference information on time scaling functions.
Chapter 7	Provides reference information on administrative tools procedures for fast and easy creation of time series schema objects.
Appendix A	Lists potential errors, their causes, and user actions to correct them.

Appendix B	Describes the views that Oracle8*i* Time Series uses to store information about time series schema objects.
Appendix C	Lists deprecated features. These are features that still work for this release, but are not documented as available for use. You are encouraged not to use these features because they might not work in future releases.
Glossary	Contains definitions of important terms related to Oracle8*i* Time Series.

Related Documents

For information added after the production of this guide, see the README file in the following directory:

- $ORACLE_HOME/ord/ts/admin (Solaris systems)
- $ORACLE_HOME\ord80\ts\admin (Windows NT systems)

The location of the README file is operating system-dependent.

For more information, see the following manuals in the Oracle8*i* documentation set:

- *PL/SQL User's Guide and Reference*
- *Oracle Call Interface Programmer's Guide*
- *Oracle8i Application Developer's Guide - Fundamentals*

Conventions

The following conventions are used in this guide:

Convention	Meaning
. . .	A vertical ellipsis in an example means that lines not directly related to the example have been omitted.
. . .	A horizontal ellipsis in an example means that part of the statement or command not directly related to the example has been omitted
boldface text	Boldface text indicates a term defined in the text.
italicized text	Italicized text indicates emphasis or a user-defined variable, schema name, or object data type.
< >	Angle brackets enclose user-supplied names.

Convention	Meaning
[ ]	Brackets enclose optional clauses from which you can choose one or none.

Changes to This Guide

The following substantive changes have been made to this guide since its previous (and initial) version for release 8.0.4.

Other minor corrections and clarifications have also been included.

See also the description of deprecated features in Appendix C.

Calendar Enhancements

Calendar enhancements include:

- New frequencies: *week*, *10_day*, *semi_monthly*, *quarter*, and *semi_annual*
- Greater flexibility in pattern definition, including pattern bit numbers other than 0 and 1 and anchor dates other than the first interval of the period
- New simpler method of deriving calendar exceptions

These calendar enhancements are included in Section 2.2.

New Calendar Functions

The following calendar functions have been added:

- Day
- GenDateRangeTab
- GetIntervalStart
- GetIntervalEnd
- Hour
- Minute
- Month
- Quarter
- Second
- Semi_annual

- Semi_monthly
- Ten_day
- Week
- Year

These functions are documented in Chapter 4.

Irregular Time Series

Oracle8*i* Time Series now supports irregular time series, which are time series without associated calendars. Using an irregular time series lets you handle unpredictable data, and it also lets you conveniently process predictable data (although some Oracle8*i* Time Series features are unavailable with this approach). Irregular time series are explained in Section 2.1.1.

Object Storage Model

Time series can now be stored in an object table (nested index-organized table). This storage model is described in Section 2.6.2.

Administrative Tools Procedures

Administrative tools procedures are provided to simplify the creation and maintenance of time series schema objects. These procedures are introduced in Section 2.12. Reference information on these procedures is in Chapter 7.

The quick-start demo (described in Section 1.6.1) uses the administrative tools procedures.

Time Scaling Functions in Separate Package

All time scaling functions have been placed in a new TimeScale package. These functions are now documented in a separate chapter (Chapter 6).

The use of the TimeSeries package for scaleup functions that were available in release 8.0.4 is a deprecated feature (see Section C.4).

New Scaleup Functions

The following scaleup functions are been added:

- ScaleupAvgX
- ScaleupGMean
- ScaleupSumAnnual

These functions are documented in Chapter 6.

Scaledown Functions

Scaledown functions are now provided:

- ScaledownInterpolate
- ScaledownRepeat
- ScaledownSplit

These functions are documented in Chapter 6.

New tsname Parameter

Functions that return a time series accept an optional *tsname* parameter specifying a name for the resulting time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

Lookback Window (k) Parameter for Mavg and Msum

The lookback window (*k*) parameter for the Mavg and Msum functions now comes before any optional start-end date range. (The old format is a deprecated feature. See Appendix C for information about deprecated features.)

Tables and Views for Time Series Data

The following clarification is added to the ORDTNumSeriesIOTRef data type description in Section 2.7.2: "*table_name* can be a view, but the view must be updatable and must map to an IOT. If the view includes any functions, they must include the PRAGMA RESTRICT_REFERENCES compiler directive with the keywords WNPS, RNPS, and WNDS."

Glossary Added

A glossary of Oracle8*i* Time Series terms has been added.

1 Introduction

Oracle8*i* Time Series (in previous releases called the Oracle8 Time Series Cartridge) is an extension to Oracle8*i* that provides storage and retrieval of timestamped data through object types. Oracle8*i* Time Series is a building block for applications rather than being an end-user application in itself. It consists of data types along with related functions for managing and processing time series data.

For example, applications can use this product to process historical data derived from financial market transactions, such as trades of stocks, bonds, and mutual fund shares. In such applications, the functions included with Oracle8*i* Time Series let you conveniently perform operations ranging from the simple to the complex, such as:

- Finding the opening, closing, low, and high prices for a stock on a specific date
- Calculating monthly volumes for a stock for a specific year
- Deriving the 30-day moving average for a stock over a year

Time series applications have certain distinct requirements and some degree of commonality. The time series data types accommodate the commonality and support extensions that address application-specific requirements. With Oracle8*i* Time Series, time series data can be managed more conveniently and efficiently than is possible using only traditional data types and user-defined functions.

You can use or adapt existing tables for time series applications, or you can create new tables. You can also extend the capabilities of Oracle8*i* Time Series to add or modify functions and to create customized calendars.

1.1 Oracle8*i* Time Series and Object-Relational Technology

The Oracle8*i* architecture allows clients, application-specific servers, and database servers to be extended easily and reliably. Oracle8*i* Time Series provides support for time series domain-specific types, functions, and interfaces. The product focuses on a set of time series data representation and access mechanisms sufficient to support many applications and the development of more specialized time series functions.

The objects option makes Oracle8*i* an **object-relational** database management system, which means that users can define additional kinds of data -- specifying both the structure of the data and the ways of operating on it -- and use these types within the relational model. This approach adds value to the data stored in a database.

Oracle8*i* with the objects option stores structured business data in its natural form and allows applications to retrieve it that way. For that reason, it works efficiently with applications developed using object-oriented programming techniques.

1.2 Storing and Accessing Data

Oracle8*i* Time Series can store time series data in the database under transactional control.

Once stored in the database, this data can be queried and retrieved by finding a row in a table that contains the primary key (which includes the timestamp) using the various alphanumeric columns (attributes) of the table. Typical queries might include the following:

- Select the closing price from a stock market data table where the ticker (stock symbol) is XYZ and the date is 30-May-1997.
- Select the 30-day moving average of stock XYZ for the month of May 1997.

Applications access and manipulate time series data using SQL or PL/SQL^{TM}. See the *Oracle8i SQL Reference* manual for information on SQL syntax.

1.3 Time Series Usage Models

Most Oracle8*i* Time Series users fit into one of a few usage models, depending on their needs. The two basic usage models are as follows:

- **No need for calendars:** You do not need to use a calendar if the timestamps have no pattern, if the timestamps have a pattern but it does not need to be checked, or if the pattern is important but the timestamps have extraneous elements (for example, hourly timestamps created using SYSDATE, which includes the minutes and seconds).

  Many time series applications, including some for financial markets and other environments with regular data, do not need to use calendars.

- **Need for calendars:** You may need to use calendars for any of several reasons, such as to use the Lead and Lag functions (which require calendars) or to ensure the validity of insert, update, and delete operations on the timestamped data. However, depending on your needs, you may or may not need to specify certain elements in the calendar definition, such as:

  – Lower and upper date boundaries for the calendar

  – Exception timestamps (for example, to identify holidays)

  For example, you may be interested only in the pattern of timestamps, but not in defining date boundaries for the calendar or specifying exceptions for holidays. In this case, if no data exists for a valid timestamp (for example, no price for a stock on Friday, 04-Jul-1997 because U.S. financial markets were closed that day), you can simply insert a null (that is, treat it as a valid timestamp but with a null associated data value) because you are confident that your data is accurate.

This rest of this section describes these usage models. It does not explain in detail any of the concepts mentioned; these are explained in Chapter 2. You may want to find the model that best fits your needs, follow the instructions in that section, and refer to the other sections in this document as necessary.

1.3.1 No Need for Calendars

Many Oracle8*i* Time Series users do not need to use calendars with their timestamped data. Situations where calendars are not needed include the following:

- The timestamps have no pattern. Examples include timestamps for the opening or closing of a valve, fluctuations in electrical power demand, transactions at an automatic teller machine (ATM), and trades during the day on a financial market.
- The timestamps have a pattern, but you do not need to use it. (You must also assume that all timestamps for the data are correct.) For example, you can derive a 30-day moving average for stock XYZ for 1998 without using a calendar, as long as you know that the closing price data is valid (that is, there are closing prices for all trading days and no closing prices for nontrading days).

  One variation of having a pattern but no need to use it occurs when the timestamps contain extraneous elements. For example, an electric utility may want to collect hourly data on power demand use for different regions, but it is unimportant whether the timestamp is exactly on the hour or contains minutes and seconds. For example, using SYSDATE to create timestamps might result in data for 4 p.m. (16:00) being stored with a timestamp of 16:00:03, 15:59:37, or 16:01:30.

You can use all time series and time scaling functions except Lead and Lag without a calendar. You can use all time scaling functions except ScaledownRepeat and ScaledownSplit without a calendar for the input data (for example, daily trading volume for stock XYZ); however, you must have a calendar to which to scale the data (for example, a monthly calendar for deriving monthly trading volume for stock XYZ).

Note: A time series used without an associated calendar is called an **irregular time series**, regardless of whether or not the timestamps are predictable. For more information about irregular time series, see Section 2.1.1.

If you do not need to use a calendar with input timestamped data, you can follow these steps to use Oracle8*i* Time Series:

1. Use the administrative tools procedures to create the time series schema objects. See the description of the quick-start demo in Section 1.6.1, and use that demo file as a model for creating your own definitions. Note that this demo also includes a calendar definition to be used for daily to monthly scaling.
2. Load the data. The quick-start demo provides an example using the SQL*Loader utility.
3. Use standard time series and time scaling functions for queries. See the quick-start demo for some examples.

1.3.2 Need for Calendars

Many Oracle8*i* Time Series users need to use calendars to take advantage of the full range of functions, including Lead and Lag. They also want to identify a pattern for the timestamps and to perform at least some validation of those timestamps. The extent of calendar maintenance required depends on whether they specify any of the following for each calendar:

- Lower and upper date boundaries for the calendar (to let you use time series functions to ensure that all timestamps are within a defined date range)
- Exception timestamps (for example, to identify holidays)

These users can also use shared calendars (described in Section 2.2) to associate multiple time series with a single calendar.

The rest of this section describes some calendar usage models involving different levels of specification and maintenance.

1.3.2.1 Minimal Calendar Maintenance

Many Oracle8*i* Time Series users need to use calendars with their timestamped data, but do not want or need to do substantial maintenance of calendars. They need to use calendars to use the full range of functions, including Lead and Lag, but they do not need to define beginning and ending boundary dates for calendars or to specify every holiday within the date range (including adding or changing holidays as needed). They are confident that the timestamps are correct and valid.

In this usage model, each time series has a calendar with a pattern. For example, for daily stock market data, a calendar is defined with a frequency of *day* and a pattern of '0,1,1,1,1,1,0' to reflect a Monday-to-Friday normal business week. However, no beginning or ending date for the calendar is specified, and no exceptions are defined for any Monday-to-Friday dates on which the markets are closed. If the data does not include a timestamp for a particular Monday-to-Friday date (for example, Friday, 04-Jul-1997), you must insert a null value for the data associated with that timestamp.

This approach allows for some validation of input data. For example, trading data with a timestamp of Saturday, 08-Aug-1998 would be invalid. However, this approach does not catch many possible kinds of input timestamp errors. For example, the following errors would *not* be detected:

- A timestamp of 19-Aug-1997 (a Tuesday) when 19-Aug-1998 (a Wednesday) was intended, if the year was incorrectly typed and the time series is supposed to contain only 1998 data, but the calendar does not specify a starting date
- Price data entered for 25-Dec-1998 (a Friday), a holiday for U.S. financial markets

If you need to use a calendar but do not need to maintain calendars to enforce input timestamp validation, you can follow these steps to use Oracle8*i* Time Series:

1. Use the administrative tools procedures to create the time series schema objects. See the description of the quick-start demo in Section 1.6.1, and use that demo file as a model for creating your own definitions. Note that this demo creates a table to hold calendars and a calendar definition for use with daily to monthly scaling.

2. Create one or more calendar definitions in which only the essential elements are defined (frequency and pattern). The following example creates a Monday-to-Friday calendar with no date boundaries and no exceptions (see Section 3.2 for more detailed information about calendar definition):

```
INSERT INTO tsquick_cal VALUES(
     ORDSYS.ORDTCalendar(
       0,                  -- Calendar type (0 = standard)
       'BUSINESSDAYS',     -- Name of this calendar
       4,                  -- 4 = frequency code for day
       ORDSYS.ORDTPattern(  -- Pattern definition (required)
          ORDSYS.ORDTPatternBits(0.1,1,1,1,1,0),
            TO_DATE('05-JAN-1998','DD-MON-YYYY')),
       NULL,               -- No lower date boundary (minDate)
       NULL,               -- No upper date boundary (maxDate)
       NULL, NULL)         -- No off- or on-exceptions
);
```

3. Load the data. The quick-start demo provides an example using the SQL*Loader utility.

4. Use standard time series and time scaling functions for queries. See the quick-start demo and the usage demo (see Section 1.6.2) for some examples.

1.3.2.2 Complete Calendar Definition and Maintenance

Some Oracle8*i* Time Series users need to create and maintain calendars, specifying the beginning and ending boundary dates for calendars and exceptions to the normal pattern, such as all holidays and any normally "off" days that become work days. These users may need to check the data to ensure that all timestamps are valid.

In this usage model, each time series has a calendar with a pattern, starting and ending date boundaries, and full specification of all exceptions (such as holidays). Users adopting this usage model will be able to use Oracle8*i* Time Series functions to determine if any timestamps in the input data are invalid. For example, the following errors *would be* detected:

- A timestamp of 19-Aug-1997 (a Tuesday) when 19-Aug-1998 (a Wednesday) was intended, if the year was incorrectly typed and the time series is restricted to 1998 data
- Price data entered for 25-Dec-1998 (a Friday), a holiday for U.S. financial markets

If you need to perform complete calendar definition and maintenance, read the information about calendars in Section 2.2 and follow the guidelines in Chapter 3.

1.4 Installing the Kit

Oracle8*i* Time Series installation consists of the following basic steps:

1. Installing the software on your computer

   Use the Oracle Universal Installer to install the software.

2. Loading the necessary objects into the database

   You can use the Oracle Database Configuration Assistant (ODCA) to automate the creation of the necessary objects. If you are not familiar with Oracle8*i* database creation, you are especially encouraged to use the ODCA. If you plan to create the database without using the ODCA, instructions are provided in Section 1.4.3.

Oracle8*i* Time Series is installed under the ORDSYS schema.

1.4.1 Required Software for Using Oracle8*i* Time Series

To use Oracle8*i* Time Series, at least the following software components must be installed: Oracle8*i* (RDBMS), PL/SQL (on systems on which it is a separate installation option), and Oracle8*i* Time Series. These components can be installed all at once, or Oracle8*i* Time Series can be added to an existing Oracle8*i* installation that includes PL/SQL.

1.4.2 After Installing Oracle8*i* Time Series

After installing Oracle8*i* Time Series, read the README.txt file for your platform, which can be found either in $ORACLE_HOME/ord/ts/admin (UNIX systems) or $ORACLE_HOME\ord80\ts\admin (Windows NT systems). Follow any instructions appropriate for your environment (for example, adjusting certain quota values, if necessary).

You may also want to do either or both of the following:

- Create public synonyms for the Oracle8*i* Time Series packages (see Section 1.5), to eliminate the need to type the schema name with the package name when calling functions or procedures.
- Run the quick-start demo or the usage demo, or both (see Section 1.6), to familiarize yourself with the Oracle8*i* Time Series product.

1.4.3 Creating Database Objects Without Using ODCA

The following instructions are for database administrators planning to create the database without using the Oracle Database Configuration Assistant (ODCA).

1. Create and start the database.

   The ORDSYS schema shares the SYSTEM tablespace. You should allow approximately 25 megabytes for the SYSTEM tablespace, so that the Oracle8*i* Time Series components and metadata can be accommodated.

   For detailed information about database creation and startup, see the *Oracle8i Installation and Configuration Guide* for your operating system, the *Oracle8i Administrator's Guide*, and the *Oracle8i Concepts* manual.

2. Install shared components.

   Connect as user SYS, and run the following SQL procedure to install ORDSYS and certain shared components.

   On Solaris systems (example showing the default SYS password):

   ```
   SVRMGR> connect sys/change_on_install as sysdba
   SVRMGR> @<ORACLE_HOME>/ord/admin/ordinst.sql
   ```

   Replace <ORACLE_HOME> with your $ORACLE_HOME directory.

   On NT systems (example showing the default SYS password):

   ```
   SVRMGR> connect sys/change_on_install as sysdba
   SVRMGR> @c:\orant\ord\admin\ordinst.sql
   ```

   c:\orant is the usual $ORACLE_HOME directory.

3. Install the Oracle8*i* Time Series components (data types, packages, and metadata tables).

   On Solaris systems:

   ```
   SVRMGR> @<ORACLE_HOME>/ord/ts/admin/tsinst.sql
   ```

   Replace <ORACLE_HOME> with your $ORACLE_HOME directory.

   On NT systems:

   ```
   SVRMGR> @c:\orant\ord\ts\admin\tsinst.sql
   ```

   c:\orant is the usual $ORACLE_HOME directory.

The user group PUBLIC is granted execute privilege on all Oracle8*i* Time Series data types and packages.

1.5 Creating Public Synonyms for Oracle8*i* Time Series Packages

All Oracle8*i* Time Series packages and data types are installed under the ORDSYS schema, and all users must include the ORDSYS schema name when referring to these packages and data types. However, to simplify references to packages, you can define public synonyms for packages that contain the functions and procedures documented in this guide.

To create public synonyms, run the ordtsyn.sql file supplied with Oracle8*i* Time Series in the admin directory. The ordtsyn.sql file creates the following public synonyms:

```
CREATE PUBLIC SYNONYM TimeSeries FOR ORDSYS.TimeSeries;
CREATE PUBLIC SYNONYM Calendar   FOR ORDSYS.Calendar;
CREATE PUBLIC SYNONYM TSTools    FOR ORDSYS.TSTools;
CREATE PUBLIC SYNONYM TimeScale  FOR ORDSYS.TimeScale;
```

1.6 Oracle8*i* Time Series Demos (Demonstrations)

Table 1–1 shows the demos (files that demonstrate capabilities) included with Oracle8*i* Time Series. This table includes a description of each demo and the default directory in which its files are installed. (The exact location and directory syntax are system-dependent.)

Table 1–1 Oracle8i Time Series Demos

Description	Directory
Quick-start demo: quick and easy start using Oracle8*i* Time Series (See Section 1.6.1.)	demo/tsquick
Usage demo for end users and product developers who want to use existing Oracle8*i* Time Series features (See Section 1.6.2.)	demo/usage
Electric utility application demonstrating how to compute peak and off-peak summaries of 15-minute data	demo/usageutl
Java-based retrieval of time series data, using the prototype Oracle8*i* Time Series Java API and designed to run in a Web browser (See Section 1.7.)	demo/applet
Simple Java code segments that perform time series operations and print the results (See Section 1.7.)	demo/java

Table 1–1 Oracle8i Time Series Demos (Cont.)

Description	Directory
Demo showing the use of administrative tools procedures to "retrofit" existing time series detail tables; also, how to support time series queries for multiple qualifier columns in the time series detail table	demo/retrofit
Advanced-developer demo for those who want to extend Oracle8*i* Time Series features	demo/extend
OCI demo showing how to call Oracle8*i* Time Series functions using the Oracle Call Interface	demo/oci
PRO*C/C++ demo showing how to call Oracle8*i* Time Series functions in applications created using the Oracle Pro*C/C++ Precompiler	demo/proc
Oracle Developer demo showing how to call Oracle8*i* Time Series functions in an Oracle Forms^{TM} application	demo/dev2k

The README.txt file in the demo directory introduces the demos and describes each briefly. Also, the directory for each demo contains a README.txt file with a more detailed description of that demo.

1.6.1 Quick-Start Demo

The quick-start demo provides a quick and easy start using Oracle8*i* Time Series. It uses the same stock market trading data as in the usage demo (described in Section 1.6.2); however, it simplifies the process by:

- Using the Oracle8*i* Time Series administrative tools procedures to create the schema objects (accepting defaults for most object names)
- Not associating a calendar with the detail data (the daily stock market trading data)

  This approach assumes that the existing stockdemo data is valid, and it is used here solely to make the quick-start demo simpler. Note, however, that using a calendar with detail data is required if you need to use Oracle8*i* Time Series functions to validate time series data (for example, to check that trading data is not entered for an invalid date, such as for a nontrading date or a date outside a desired start-end date range). Using a calendar with detail data is also required for using the Lead and Lag functions.

The administrative tools procedures create all the schema objects needed for this demo, including:

- TSQUICK: the object view. Use this when using Oracle8*i* Time Series functions. For example:

  ```
  SELECT ticker, ORDSYS.TimeSeries.TSAvg(close) FROM TSQUICK ts;
  ```

- TSQUICK_RVW: the relational view. Use this for protected insert, update, and delete operations. Uses an INSTEAD OF trigger.
- TSQUICK_TAB: the table for detail data.
- TSQUICK_MAP: the mapping (metadata) table. A null calendar is later associated with each ticker.
- TSQUICK_CAL: the table for calendar definitions. A monthly calendar is later defined, for use with scaleup operations.

All of these schema objects, as well as concepts related to calendars, are explained in Chapter 2. The administrative tools procedures are introduced in Section 2.12.

The quick-start demo also includes queries using several Oracle8*i* Time Series functions.

1.6.1.1 Running the Quick-Start Demo

After Oracle8*i* Time Series has been installed, you can run the quick-start demo by going to the appropriate directory (see Table 1–1) and invoking the tsquick.sql procedure, as follows:

```
% svrmgrl
SVRMGR> @tsquick
```

1.6.1.2 Quick-Start Demo Files

The quick-start demo files are listed in Table 1–2.

Table 1–2 Quick-Start Demo Files

File	Description
tsquick.sql	Main procedure file: creates all schema objects, loads tables, performs queries.
tsquick.ctl	SQL*Loader control file
tsquick.dat	SQL*Loader data file
README.txt	Description and instructions for the demo

1.6.2 Usage Demo

The usage demo is a working example of using Oracle8*i* Time Series. The example models a historical database of stock pricing and provides sample queries using this data.

The usage demo is designed to guide you through Oracle8*i* Time Series in a step-by-step fashion. It includes example code for creating and populating tables and calendars, constructing relational views, constructing views to synthesize the interface to Oracle8*i* Time Series functions, and running some example queries.

1.6.2.1 Running the Usage Demo

After Oracle8*i* Time Series has been installed, you can run the usage demo by going to the appropriate directory (see Table 1–1) and invoking the demo.sql procedure, as follows:

```
% svrmgrl
SVRMGR> @demo
```

1.6.2.2 Usage Demo Files

The usage demo files include examples of bulk and incremental loading; defining tables, calendars, and views; and running example queries. These files are listed in Table 1–3.

Table 1–3 Usage Demo Files

File	Description
demo.sql	Main procedure file
stockdat.ctl	SQL*Loader control file
stockdat.dat	SQL*Loader data file for time series data
tables.sql	DDL for tables
popcal.sql	Defines calendars and populates calendar table
queries.sql	Example time series queries (SQL)
queplsql.sql	Example time series queries (PL/SQL)
calqueries.sql	Example calendar queries (PL/SQL)
incload.sql	Incremental load script
stockinc.ctl	SQL*Loader control file for incremental load
stockinc.dat	SQL*Loader data file for incremental time series data

Table 1–3 Usage Demo Files (Cont.)

File	Description
verical.sql	Verifies the correctness of calendars associated with the demo
verits.sql	Verifies the correctness of a time series (ACME) associated with the demo
cleanup.sql	Deletes database objects created by the demo
README.txt	Description and instructions for the demo

1.6.2.3 Tables and Views in the Usage Demo

The stock database consists of three tables:

- *stockdemo* stores historical time series pricing data.
- *stockdemo_calendars* stores instances of calendars.
- *stockdemo_metadata* maintains mapping between time series (here, for tickers) and calendars.

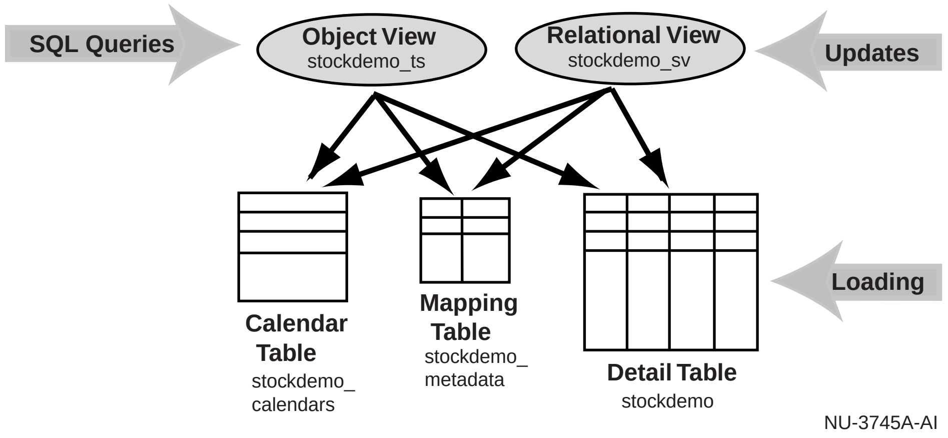
To maintain time series consistency and provide a collection-based interface for time series functions, two views are constructed using these tables.

- *stockdemo_sv* is a relational view. A relational view should be used for any insert, update, and delete operations to time series data.
- *stockdemo_ts* is an object (reference-based) view. A reference-based view provides an object model of a time series, and it can be used for efficient read-only access using Oracle8*i* Time Series functions.

The relational view ensures that insert, update, and delete operations maintain a time series that is consistent with the associated calendar. (Time series consistency is explained in Section 2.8.) The relational view and the object view access the three underlying tables. The object view synthesizes references to collections.

Figure 1–1 shows the relationships between the object and relational views and the underlying tables.

Figure 1–1 Tables and Views in the Time Series Usage Demo



1.7 Java Client-Side API (Prototype)

A prototype Java client-side application programming interface (API) is provided in the following file:

```
<ORACLE_HOME>/ord/ts/jlib/thindriver.zip
```

Documentation (generated by javadoc) for this API is in the following directory:

```
<ORACLE_HOME>/ord/ts/doc
```

The README.txt file in this directory discusses Java support.

The following directories contain demos (introduced in Table 1–1 in Section 1.6) that can help you to learn and use the API:

- The demo/java directory contains sample code segments that demonstrate the basic use of the API. Although your applications will probably be more complex, you can use the techniques in these code segments to retrieve time series data.
- The demo/applet directory contains a demonstration applet (built using JDeveloper) that runs in a Web browser or using appletviewer. This applet can connect to the database, perform queries on the sample time series data, and display the results.

2 Time Series Concepts

This chapter explains concepts related to Oracle8*i* Time Series, and it provides information on using the product. It contains the following major sections:

- Section 2.1, "Overview of Time Series Data"
- Section 2.2, "Calendars"
- Section 2.3, "Data Types"
- Section 2.4, "Conventions and Semantics" (including null values and off-exceptions)
- Section 2.5, "Oracle8i Time Series Architecture"
- Section 2.6, "Storage of Time Series Data"
- Section 2.7, "Interfaces to Time Series and Time Scaling Functions"
- Section 2.8, "Consistency of Time Series Data"
- Section 2.9, "Calendar Functions"
- Section 2.10, "Time Series Functions"
- Section 2.11, "Time Scaling Functions"
- Section 2.12, "Administrative Tools Procedures"

2.1 Overview of Time Series Data

A time series is a set of timestamped data entries. A time series allows a natural association of data collected over intervals of time. For example, summaries of stock market trading or banking transactions are typically collected daily, and are naturally modeled with time series.

2.1.1 Regular and Irregular Time Series

A time series can be regular or irregular, depending on whether or not the time series has an associated calendar.

- A **regular** time series has an associated calendar. In a regular time series, data arrives predictably at predefined intervals. For example, daily summaries of stock market data form regular time series, and such time series might include the set of trade volumes and opening, high, low, and closing prices for stock XYZ for the year 1997.
- An **irregular** time series does not have an associated calendar. Often, irregular time series are data-driven, where unpredictable bursts of data arrive at unspecified points in time or most timestamps cannot be characterized by a repeating pattern. For example, account deposits and withdrawals from a bank automated teller machine (ATM) form an irregular time series. An irregular time series may have long periods with no data or short periods with bursts of data.

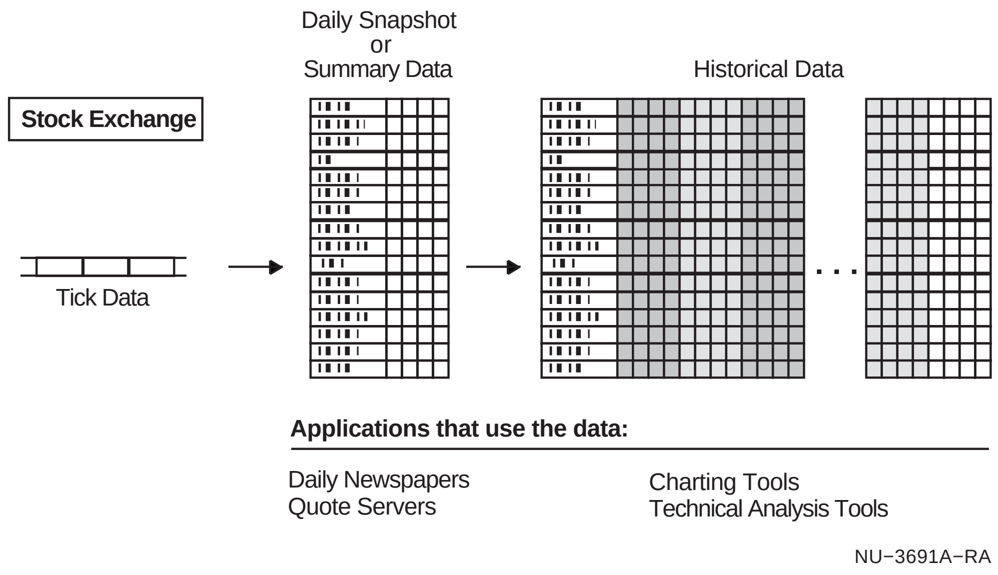
  However, an irregular time series does not have to be used only for high-volume collection of unpredictable data. An irregular time series can be used with predictable data where it is simply not necessary to deal with a calendar. This approach is used in the quick-start demo described in Section 1.6.1.

2.1.2 Data Generation for a Time Series

Data generation for a time series begins with individual transactions, such as trades on a stock exchange or purchases of products. Each transaction has a timestamp and sufficient information to identify that transaction uniquely (such as a stock ticker or a product ID), as well as other pertinent information (such as the price and information to identify the party initiating the purchase or sale).

Individual transaction data is typically *rolled up* to produce summary data for a meaningful time period, such as a daily summary indicating the trade volume and the opening, high, low, and closing prices for each stock traded that day. This summary data is collected to produce historical data, such as a table of all daily volumes and opening, high, low, and closing prices for all stocks traded for the year 1997. For example, Figure 2–1 shows how data related to securities on a stock exchange is generated.

Figure 2–1 Data Generation in Equities Markets



In Figure 2–1, each trade on the stock exchange includes several items of information, including a ticker and a price (for example, stock XYZ at 37.50). The daily summary data includes the opening, high, low, and closing prices for each ticker (for example, for XYZ: 37.75, 38.25, 37.00, 37.625). The daily data for each ticker is appended to the historical data for the ticker. The daily data is used for such purposes as quote server applications and listing in the next day’s newspapers; the historical data is used by such applications as price and volume charting and technical analysis.

The data-collection model for historical data has the following characteristics:

- At daily intervals, historical data is updated with daily summary data (main update cycle).
- At some period after the main update cycle, corrections of the daily summary data may need to be applied.
- Queries may be executed at any time, even during the update cycle.
- Queries do not observe the current day's summary information until after the main update cycle has completed.

This historical data is modeled using multiple regular time series.

Oracle8*i* Time Series and the Oracle8*i* utilities, with their bulk-loading capabilities and transactional semantics, are well suited for the requirements of time series data generation.

2.1.3 Historical Data

Oracle8*i* Time Series is especially useful in dealing with historical data. This type of data typically has relatively simple metadata but massive data storage requirements. That is, the data attributes (columns) are relatively few and easy to understand (such as ticker, volume, and opening, high, low, and closing prices); however, the number of rows is enormous (for example, data for all listed stocks for all trading days for several years). Moreover, the number of functions that users might want to perform on the data is large: for example, finding various sums, counts, maximum and minimum values, averages, number of trading days between two dates, moving average, and so on.

Figure 2–2 shows an example of historical data stored in a database.

Figure 2–2 Historical Data for Stocks

Ticker	Tstamp	Open	High	Low	Close	Volume
XYZ	01-02-1997	21.75	22.75	21.50	22.00	352,000
XYZ	01-03-1997	22.125	22.50	21.00	21.75	530,000
XYZ	01-06-1997	21.625	22.00	21.625	21.875	490,000
...	...	...	...	...	...	...
YZA	01-02-1997	44.25	44.25	43.50	43.875	125,000
YZA	01-03-1997	43.75	44.25	43.75	44.125	97,000
YZA	01-06-1997	44.25	44.50	44.125	44.125	107,000
...	...	...	...	...	...	...

Stock market historical databases have the following general characteristics:

- Multiple stocks, each identified by the ticker symbol, can be stored in the database.
- Each stock can have multiple attributes (*ticker, tstamp, open, high, low, close, volume*).
- Each stock has one or more associated time series.

- Each time series has an associated calendar (see Section 2.2).

This kind of financial historical data is used in examples in this guide and in the usage demo (see Section 1.6) provided with Oracle8*i* Time Series.

2.2 Calendars

An Oracle8*i* Time Series calendar maps human-meaningful time values to underlying machine representations of time.

A calendar can be associated with a time series. However, a calendar *does not need* to be associated with a time series unless you need to do any of the following:

- Use Oracle8*i* Time Series functions to validate time series data (for example, to check that trading data is not entered for an invalid date, such as for a nontrading date or a date outside a desired start-end date range)
- Use the Lead, Lag, or Fill functions
- Use the time scaling functions (where the created time series must have an associated calendar, for example, to provide a monthly roll-up of data)

A time series with an associated calendar is called a *regular* time series, as described in Section 2.1.1.

A calendar is, of course, necessary for using any of the calendar functions (described in Section 2.9).

A business day calendar, for example, can define the days of the week on which stocks are traded. The holidays, when trading does not occur, are also included in the calendar as exceptions.

If you have more than one time series with the same timestamps, you can associate these time series with the same calendar (that is, use a **shared calendar**). For example, a calendar of U.S. stock market trading days could be used for all stocks that trade every day. For any stocks that do not fit this pattern, you could create private calendars (for example, an *ACME_cal* calendar for stock *ACME*).

The following are key components of a calendar:

- Frequency

  A **frequency** specifies the granularity of the calendar representation. The supported frequencies are *second, minute, hour, day, week, 10-day, semi-monthly, month, quarter, semi-annual,* and *year.*

- Pattern

The **pattern** specifies the repeating pattern of frequencies and an anchor date that identifies a valid timestamp for the first element in the pattern. For example, if the frequency is set to *day*, the pattern can define which days of the week are included in the calendar. For example, a pattern of '0,1,1,1,1,1,0' over a *day* frequency defines a calendar over all weekdays. If an anchor date of 01-Jun-1997 (or any Sunday) is specified, then the 7-day pattern begins each Sunday; and Sunday and Saturday (0) are excluded from the calendar, while Monday through Friday (1) are included in the calendar.

- Exceptions

  Exceptions are timestamps that do not conform to the calendar pattern but that are significant for the calendar definition. There are two kinds of exceptions: off-exceptions and on-exceptions:

  – An **off-exception** is an exception to the nonzero bits in the pattern, and thus is a timestamp to be excluded from the calendar. For example, to ensure that Wednesday, 25-Dec-1996, is excluded from the calendar when Wednesdays normally are included, define that date as an off-exception.

  – An **on-exception** is an exception to the zero (0) bits in the pattern, and thus is a timestamp to be included in the calendar. For example, to ensure that Saturday, 28-Jun-1997, is included in the calendar when Saturdays are excluded, define that date as an on-exception.

    On-exceptions can also be used with a zero pattern. Section 2.11 includes a description of using such a calendar for scaling, with quarterly dividend payment dates as on-exceptions.

2.2.1 Frequency

Each frequency has an associated integer code that is used in function calls. Table 2–1 lists the supported frequencies and their integer codes. The frequencies are explained in Table 2–2.

Table 2–1 Frequency Codes

Frequency	Integer Code
second	1
minute	2
hour	3
day	4
week	5

Table 2–1 Frequency Codes(Cont.)

Frequency	Integer Code
month	6
quarter	7
year	8
10-day	10
semi-monthly	16
semi-annual	18

Some frequencies allow flexibility in defining pattern anchor dates, whereas other frequencies are more restrictive. Table 2–2 explains the frequencies and any requirements and options relating to the pattern anchor date.

Table 2–2 Frequencies and Their Requirements

Frequency	Explanation and Requirements
second	Every second. The anchor date can be any timestamp with a valid value for seconds.
minute	Every minute. The anchor date can be any timestamp with a valid value for minutes. The value for seconds should be zero.
hour	Every hour. The anchor date can be any timestamp with a valid value for hours. The values for minutes and seconds should be zero.
day	Every day. The anchor date can be any timestamp with a valid value for the day. The values for hours, minutes, and seconds should be zero
week	Once every 7 days. Can start on any day of the week. For example, defining a weekly calendar with an anchor date of 23-Jun-1998 means that each timestamp must be for a Tuesday.
month	Once every month. Can start on days 1-28 or 31. (Defining an anchor date of the 31st of a month means the last day of each month.) For example, defining a monthly calendar with an anchor date of 01-Jul-1998 means that each timestamp must be for the 1st of a month.

Table 2–2 Frequencies and Their Requirements (Cont.)

Frequency	Explanation and Requirements
quarter	Four times per year. Can start on days 1-28 or 31 of any month. (Defining an anchor date of the 31st of a month means the last day of each month.) For example, defining a quarterly calendar with an anchor date of 01-Jan-1998 means that each timestamp must be for the 1st of January, April, July, or October. Defining a quarterly calendar with an anchor date of 15-Feb-1998 means that each timestamp must be for the 15th of February, May, August, or November.
year	Once per year. Can start on days 1-28 or 31 of any month. (Defining an anchor date of the 31st of a month means the last day of that month.) For example, defining an annual calendar with an anchor date of 01-Jan-1998 means that each timestamp must be for the 1st of January. Defining an annual calendar with an anchor date of 15-Feb-1998 means that each timestamp must be for the 15th of February.
10-day	The 1st, 11th, and 21st days of each month. (Used for automobile sales data.) No other dates are permitted for a 10-day calendar, and any anchor date is ignored.
semi-monthly	The 1st and 16th days of each month. No other dates are permitted for a semi-monthly calendar, and any anchor date is ignored.
semi-annual	Twice per year. Can start on days 1-28 or 31 of any month. (Defining an anchor date of the 31st of a month means the last day of each month.) For example, defining a semi-annual calendar with an anchor date of 01-Jan-1998 means that each timestamp must be for the 1st of January or July. Defining a semi-annual calendar with an anchor date of 15-Feb-1998 means that each timestamp must be for the 15th of February or August.

2.2.2 Precision

Each frequency has an associated precision. Oracle8*i* Time Series functions require that input timestamps be of the precision of the frequency associated with the calendar. (The SetPrecision function is the exception: this function takes a calendar and a timestamp and returns a timestamp that conforms to the frequency of the associated calendar.)

A timestamp that is not consistent with the frequency is said to be **imprecise**. For example, a timestamp of 09-Sep-1997 is imprecise if it is input to a function that is dealing with a calendar whose frequency is 6 (*month*) or 8 (*year*) and whose pattern anchor date is not the 9th of a month. When you create a calendar, all timestamps used in the calendar definition (the anchor date for the pattern, and all off- and on-exceptions) must be precise with respect to the frequency. For example, the calendar will not be valid if you specify a frequency of *day*, an anchor date of 01-Jun-1998 13:00:00. An anchor date of just 01-Jun-1999, however, would valid in this case. (The calendar data types and their attributes are presented in Section 2.3.1.)

Table 2–3 shows the frequencies, their precision conventions, and an example timestamp of each precision using a pattern anchor date of 01-Jan-1998 00:00:00 (midnight), which was a Thursday.

Table 2–3 Precisions Using 01-Jan-1998 00:00:00 Anchor Date

Frequency	Precision Convention	Example Result
second	MM-DD-YYYY HH24:MI:SS	09-09-1997 09:09:09
minute	MM-DD-YYYY HH24:MI:00	09-09-1997 09:09:00
hour	MM-DD-YYYY HH24:00:00	09-09-1997 09:00:00
day	MM-DD-YYYY 00:00:00 (midnight)	09-09-1997 00:00:00
week	MM-DD-YYYY 00:00:00 (midnight of the preceding Thursday)	09-04-1997 00:00:00
month	MM-01-YYYY 00:00:00 (midnight of first day of month)	09-01-1997 00:00:00
quarter	MM-01-YYYY 00:00:00 (midnight of first day of quarter)	07-01-1997 00:00:00
year	01-01-YYYY 00:00:00 (midnight of first day of year)	01-01-1997 00:00:00
10-day	MM-DD-YYYY 00:00:00 (midnight of 1st, 11th, or 21st of month)	09-01-1997 00:00:00
semi-monthly	MM-DD-YYYY 00:00:00 (midnight of 1st or 15th of month	09-01-1997 00:00:00
semi-annual	MM-01-YYYY 00:00:00 (midnight of first day of half year)	07-01-1997 00:00:00

2.2.3 Pattern

A calendar pattern is specified as one or more zeroes (0) and / or positive integers.

For patterns represented by zeroes and/or ones, each '1' represents a valid timestamp of the frequency and each '0' represents an invalid timestamp. For example:

- A calendar with a *day* frequency and a single '1' pattern (ORDSYS.ORDTPatternBits(1)) has timestamps defined for each day. (The ORDTPatternBits data type is defined in Section 2.3.1.)
- A calendar with a *day* frequency, a Sunday anchor date, and a pattern of a '1' and six '0's (ORDSYS.ORDTPatternBits(1,0,0,0,0,0,0)) is in effect a weekly calendar where all Sunday timestamps are included and all other days of the week are excluded.

For patterns containing one or more integers greater than 1, each such integer represents an interval that is a multiple of the frequency. For example, a calendar with a *day* frequency, a Sunday anchor date, and a pattern of '7' (ORDSYS.ORDTPatternBits(7)) is in effect a weekly calendar where all Sunday timestamps are included and all other days of the week are excluded.

Note that while the actual timestamps that are valid for the calendar will be identical for each of the preceding weekly calendar examples (ORDSYS.ORDTPatternBits(1,0,0,0,0,0,0) and ORDSYS.ORDTPatternBits(7) with *day* frequency and Sunday anchor dates), these calendars have two different interpretations for use in the context of time scaling. For example, if the ScaleupSum function is invoked on a time series containing data defined for every day for scaling to these two calendars, the following differences in behavior occur:

- With the first target calendar (ORDSYS.ORDTPatternBits(1,0,0,0,0,0,0)), each timestamp in the resulting time series contains a sum of 1 day (Sunday) of data.
- With the second target calendar (ORDSYS.ORDTPatternBits(7)), each timestamp in the resulting time series contains a sum of 7 days of data.

2.2.4 Overview of Calendar Definition

To define a calendar, you create a table in which to store calendar definitions and then store a row for each calendar to be defined.

Example 2–1 creates a table named *stockdemo_calendars* and defines a calendar named *BusinessDays*. The *BusinessDays* calendar includes Mondays through Fridays, but excludes 28-Nov-1996 and 25-Dec-1996. Explanatory notes follow the example. (For more information and examples of calendar creation, see Section 3.2.)

Example 2–1 Overview of Calendar Definition

```
CREATE TABLE stockdemo_calendars of ORDSYS.ORDTCalendar (
    name CONSTRAINT calkey PRIMARY KEY);

INSERT INTO stockdemo_calendars VALUES(
  ORDSYS.ORDTCalendar( ❶
         0 ❷
         'BusinessDays', ❸
         4, ❹
         ORDSYS.ORDTPattern( ❺
                  ORDSYS.ORDTPatternBits(0,1,1,1,1,1,0),
                       TO_DATE('01-JAN-1995','DD-MON-YYYY')),
         TO_DATE('01-JAN-1990','DD-MON-YYYY'), ❻
         TO_DATE('01-JAN-2001','DD-MON-YYYY'),

         ORDSYS.ORDTExceptions(TO_DATE('28-NOV-1996','DD-MON-YYYY'),
               TO_DATE('25-DEC-1996','DD-MON-YYYY')), ❼
         ORDSYS.ORDTExceptions() ❽
         ));
```

Notes on Example 2–1:

❶ The *stockdemo_calendars* table has rows of data type ORDTCalendar, which is described in Section 2.3.1.

❷ 0 indicates that this is a standard calendar (the only type of calendar currently supported).

❸ *BusinessDays* is the name of this calendar.

❹ 4 is the frequency code for *day*.

❺ The calendar's pattern consists of an excluded occurrence followed by five included occurrences followed by an excluded occurrence (`0,1,1,1,1,1,0`). Because the frequency is *day* and because the anchor date (01-Jan-1995) is a Sunday, Sundays are excluded, Mondays through Fridays are included, and Saturdays are excluded.

❻ The calendar begins at the start of 01-Jan-1990 and ends at the start of 01-Jan-2001.

Note: *minDate* and *maxDate* can each be null. If *minDate* is null, the calendar has no lower boundary date; if *maxDate* is null, the calendar has no upper boundary date. Specifying a null *minDate* and *maxDate* simplifies calendar maintenance, but means you cannot have timestamps validated against the calendar's desired date range.

---

❼ 28-Nov-1996 and 25-Dec-1996 are off-exceptions (that is, excluded from the calendar).

Note: All exceptions (off- and on-) must be specified in ascending sorted order.

---

❽ *ORDSYS.ORDTExceptions( )* indicates that there are no on-exceptions (that is, no Saturday or Sunday dates to be included in the calendar).

2.2.5 Deriving Calendar Exceptions from Time Series Data

When you want to create calendars that conform to time series data, you can use the DeriveExceptions function to simplify the process. You can use one of several approaches with DeriveExceptions, depending on your needs and the requirements for each approach:

- The first approach (Approach 1) uses a DeriveExceptions call with input parameters of a time series and optionally a starting and ending date. A calendar is returned with the appropriate exception lists populated. This returned calendar is defined on the pattern and frequency of the calendar associated with the time series, and it is consistent with the timestamps of the time series.

  Approach 1 is the most convenient, and is recommended for most customers.

- A variation of the first approach (Approach 1A) uses a DeriveExceptions call with input parameters of a calendar and an ORDTDateTab instance and optionally a starting and ending date. (An ORDTDateTab instance is a collection of dates; these dates can be compared with the set of valid timestamps implied by the calendar.) A calendar is returned with the appropriate exception lists populated. This returned calendar is defined on the pattern and frequency of the input calendar, and it is consistent with the timestamps of the ORDTDateTab instance.

- The final approach (Approach 2) uses a DeriveExceptions call with two time series as input parameters and optionally a starting and ending date. The first time series is essentially an expansion of a pattern-only calendar. As in the first two approaches, a calendar is returned with the appropriate exception lists populated. The returned calendar is defined on the pattern and frequency of the calendar of the first input time series, and it is consistent with the timestamps of the second input time series.

While Approaches 1 and 1A can be performed in a single step, Approach 2 requires an additional step (before DeriveExceptions is called) in order to construct the first time series.

Although Approaches 1 and 1A are simpler in practice, Approach 2 has significant performance advantages when you need to define multiple calendars that have the same frequency and pattern but different exception lists. The first two approaches are less efficient than Approach 2 in this case, because the internal implementation of the first two approaches generates a collection of dates based on the input calendar. If you need to derive exceptions for multiple calendars defined on the same frequency and pattern, this date-generation operation is performed multiple times. You can avoid these multiple date-generation operations by using Approach 2.

Section 3.8 contains more detailed information about using each approach to deriving calendar exceptions.

2.3 Data Types

Oracle8*i* Time Series provides data types for working with calendars and time series.

All Oracle8*i* Time Series data types are installed under the ORDSYS schema, and all users must include the ORDSYS schema name when referring to these data types.

Note: The CREATE TYPE statements in this section do not include the TIMESTAMP and OID keywords that are part of the object type definitions when the product is installed. These keywords are used internally by products for version control.

2.3.1 Calendar Data Types

Oracle8*i* Time Series provides the following calendar data types. (Time series data types are described in Section 2.3.2.)

- Calendar (ORDTCalendar) (Sections 2.2.4 and 3.2 contain calendar definitions with explanatory notes.)

```
CREATE TYPE ORDSYS.ORDTCalendar AS OBJECT (
  caltype INTEGER,
  name VARCHAR2(256),
  frequency INTEGER,
  pattern ORDSYS.ORDTPattern,
  minDate DATE,
  maxDate DATE,
  offExceptions ORDSYS.ORDTExceptions,
  onExceptions ORDSYS.ORDTExceptions);
```

- Pattern (ORDTPatternBits and ORDTPattern)

```
CREATE TYPE ORDSYS.ORDTPatternBits AS VARRAY(32500) OF INTEGER;

CREATE TYPE ORDSYS.ORDTPattern AS OBJECT (
  patBits ORDSYS.ORDTPatternBits,
  patAnchor DATE);
```

- Exception (ORDTExceptions)

```
CREATE TYPE ORDSYS.ORDTExceptions AS VARRAY(32500) OF DATE;
```

2.3.2 Time Series Data Types

Oracle8*i* Time Series provides the following time series data types. (Calendar data types are described in Section 2.3.1.)

```
CREATE TYPE ORDSYS.ORDTNumCell AS OBJECT
   (tstamp DATE, value NUMBER);

CREATE TYPE ORDSYS.ORDTNumTab AS TABLE OF
   ORDSYS.ORDTNumCell;

CREATE TYPE ORDSYS.ORDTNumSeries AS OBJECT
   (
   name         VARCHAR2(256),
   cal          ORDSYS.ORDTCalendar,
   series       ORDSYS.ORDTNumTab
   );
```

```
CREATE TYPE ORDSYS.ORDTNumSeriesIOTRef AS OBJECT
   (
   name                VARCHAR2(256),
   cal                 REF ORDSYS.ORDTCalendar,
   table_name          VARCHAR2(256),
   tstamp_colname      VARCHAR2(30),
   value_colname       VARCHAR2(30),
   qualifier_colname   VARCHAR2(30),
   qualifier_value     VARCHAR2(4000)
   );

CREATE TYPE ORDSYS.ORDTVarchar2Cell AS OBJECT
   (tstamp DATE, value VARCHAR2(4000));

CREATE TYPE ORDSYS.ORDTVarchar2Tab AS TABLE OF
   ORDSYS.ORDTVarchar2Cell;

CREATE TYPE ORDSYS.ORDTVarchar2Series AS OBJECT
   (
   name         VARCHAR2(256),
   cal          ORDSYS.ORDTCalendar,
   series       ORDSYS.ORDTVarchar2Tab
   );

CREATE TYPE ORDSYS.ORDTVarchar2SeriesIOTRef AS OBJECT
   (
   name                VARCHAR2(256),
   cal                 REF ORDSYS.ORDTCalendar,
   table_name          VARCHAR2(256),
   tstamp_colname      VARCHAR2(30),
   value_colname       VARCHAR2(30),
   qualifier_colname   VARCHAR2(30),
   qualifier_value     VARCHAR2(4000)
   );

CREATE TYPE ORDSYS.ORDTDateTab AS TABLE OF DATE;
```

The preceding statements show the definition of a numeric time series and a character time series (instance-based and reference-based interfaces), each composed of a calendar instance and a collection. The collection (ORDTxxxTab) is defined as a table of ORDTxxxCell (except for ORDTDateTab, which is a table of DATE). Oracle8*i* Time Series data types, such as ORDTNumSeries and ORDTVarchar2Series, are input and output parameters of time series functions.

The following statements show the definitions for the ORDTDateRange and ORDTDateRangeTab types. The latter is returned by the GenDateRangeTab function, which is described in Chapter 4.

```
CREATE TYPE ORDSYS.ORDTDateRange AS OBJECT
   (start_date DATE, end_date DATE);

CREATE TYPE ORDSYS.ORDTDateRangeTabTab AS TABLE OF
   ORDSYS.ORDTDateRange;
```

2.4 Conventions and Semantics

For time series functions that accept two time series, both time series must be defined on calendars that have the same frequency and the same pattern. The calendars may have different exceptions lists and different starting and ending dates.

2.4.1 Semantics of Null Operands

A number of time series functions perform arithmetic, comparison, and grouping operations. When nulls are encountered in this context, the default behavior is to mirror SQL:

- *Group functions* ignore nulls. When all values encountered are null, a null is returned.

  For example, the sum of (1, NULL, NULL, 3) returns 4. The sum of (NULL, NULL, NULL, NULL) returns null.

- *Functions that operate on time series* ignore nulls, but return a null if all values encountered are null. Such functions include Mavg (Moving Average) and Msum (Moving Sum)

  For example, if there are 5 nulls in the last 30 timestamps for (and including) a specific date, the 30-day moving average on that date is computed using only 25 values (that is, adding only the values that are not null and dividing by 25). However, if all 30 dates (the date and the 29 previous dates) have nulls, the moving average for that date is null.

- Any *arithmetic expression* containing a null returns a null.

  For example, 10 + NULL returns null.

- A *comparison operator* that encounters a null returns a null.

  For example, a GT comparison of 30-Jun-1997 and null returns null.

Note that because PL/SQL does not implement UNKNOWN, these semantics are slightly different than the SQL treatment of comparisons with nulls. In SQL, a comparison operator that encounters a null returns UNKNOWN, which is like a null, except that operations on UNKNOWN return UNKNOWN.

- *Scaleup* functions return a null if all timestamps for a scaling interval contain nulls.

  For example, if you are scaling up daily data from 01-Jan-1997 through 30-Jun-1997 to monthly data, and if there are no values for the month of February, a null is returned for February and scaled data is returned for the other months. (Note that this behavior differs from the standard GROUP BY scaling in SQL, in which February would be missing in the scaled results.)

Some functions allow alternate semantics in the form of an option. For example, the scaleup functions allow you to specify IgnoreNulls or IgnoreNullsOFF, as explained in Section 2.11.2. The reference information for each function describes any alternate semantics options.

2.4.2 Semantics of Off-Exception Operands

In comparisons of two time series, it is possible that a timestamp valid for one time series is not valid for the other time series. Operations on two time series having similar calendars return a time series that is defined over a new calendar. This new calendar is derived from the two input calendars, using all of the following:

- The union of the off-exceptions
- The intersection of the on-exceptions
- Bounded by [max(minDate1, minDate2), min(maxDate1, maxDate2)]

For example, assume the following two calendars:

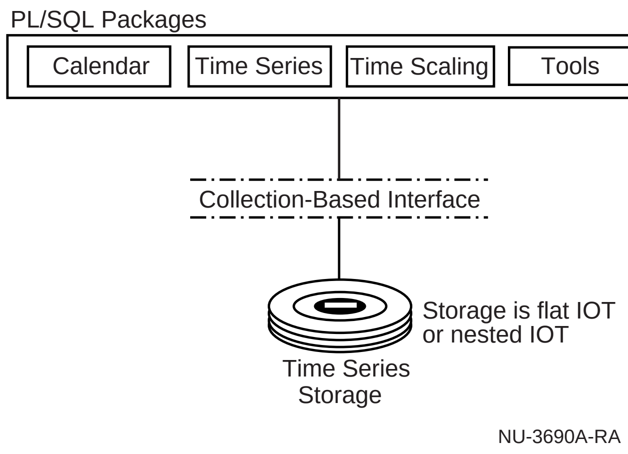
- Calendar 1: 01-Jan-1997 through 01-Dec-1997, daily pattern ’0,1,1,1,1,1,0’ (Monday through Friday), off-exception 01-May, on-exceptions 29-Mar and 29-Jun.
- Calendar 2: 01-Feb-1997 through 01-Jan-1998; daily pattern ’0,1,1,1,1,1,0’ (Monday through Friday), off-exceptions 01-May and 14-Jul, on-exceptions 29-Jun and 28-Sep.

The new (derived) calendar is: 01-Feb-1997 through 01-Dec-1997, daily pattern ’0,1,1,1,1,1,0’ (Monday through Friday), off-exceptions 01-May-1997 and 14-Jul-1997, on-exception 29-Jun-1997.

2.5 Oracle8*i* Time Series Architecture

Figure 2–3 shows the Oracle8*i* Time Series architecture. At the lowest level, a storage option is required, and this must be a flat index-organized table (IOT), a flat table, or a nested IOT. The actual product consists of PL/SQL packages for calendar, time series, and time scaling functions and for administrative tools procedures. In addition, a collection-based interface between time series storage and the packaged functions is provided.

Figure 2–3 Time Series Architecture



The rest of this chapter describes this architecture, working from bottom to top in Figure 2–3:

- Storage of time series detail data
- Interfaces (instance-based and reference-based) to time series and time scaling functions
- Calendar functions
- Time series functions
- Time scaling functions
- Administrative tools procedures

2.6 Storage of Time Series Data

Using Oracle8*i* Time Series involves storing three different kinds of information:

- Time series detail data (for example, prices and volume for each stock for each trading day)
- Calendars
- Time series metadata

In the flat table or IOT storage model, these requirements are implemented using three separate tables, and time series detail data is stored as multiple rows in the table or IOT. (Figure 1–1 in Section 1.6.2 shows the tables created by the usage demo.)

In the nested table storage model, the time series detail data is stored as an object in a nested table, that is, as rows at the second level of a nested IOT.

In the time series detail table, the data should be stored in timestamp order, because many of the analytical functions require access to the data in this order. (If timestamps are not in order, the functions perform an internal sort before processing the timestamps.)

2.6.1 Flat IOT or Flat Table Storage

A time series is stored as multiple rows in a flat index-organized table (IOT) or a flat table.[1] Each row stores a ticker, a timestamp, and composite data. This storage option is shown in Figure 2–2.

The flat IOT or table model has some benefits compared to nested IOTs:

- Only one timestamp column need be stored for multiple attributes.

  For time series data such as stock market data, where multiple attributes (such as open, high, low, volume and close) share a single timestamp, only a single timestamp column needs to be stored, thus providing efficient utilization of disk storage.

- Migration from legacy systems is easier.

  Migrating legacy data is simplest if the target schema is a flat table or flat IOT.

[1] A time series can be stored in a standard table; however, for performance reasons it is recommended that you use an IOT rather than a standard table.

However, using this model means that ensuring time series and calendar integrity cannot be encapsulated. The highest-integrity solution to time series and calendar integrity would disable insert, update, and delete operations using SQL and would implement these operations using member methods of time series and time scaling functions. This approach is not possible with a flat IOT storage model. Instead, a relational view must be defined to ensure integrity (see Section 2.8.2).

To minimize the burden of creating the relational view and other required schema objects, you can use the administrative tools procedures described in Section 2.12.

2.6.2 Nested IOT Storage (Object Model)

A time series is stored as rows in a nested IOT. At the first nesting level, the ticker symbol and any metadata associated with the time series are stored. At the second level and associated with each ticker symbol, timestamp and composite data is stored.

The nested IOT storage model has the following advantages compared to the flat IOT or table model:

- Locator-based access to nested IOTs provides optimized retrieval of time series data with standard mechanisms.

  Time series analysis functions, such as Moving Average, operate on time series data represented by collections that are passed as parameters to PL/SQL functions. This strategy is inefficient when only a portion of the time series is accessed because the server materializes the entire collection into the object cache.

  Locator-based access to nested IOTs provides a solution to this problem, by providing PL/SQL functions a handle to the nested table.

- Nested tables allow time series metadata to be stored with time series data in a single structure.

  The two levels of a nested table allow you to store metadata associated with each time series (such as textual descriptions of the instrument, or information related to stock splits). This simplifies schema management.

- A nested storage model cleanly supports the storage of derived time series.

  Derived time series are those that are computed by applying functions to existing time series. In a composite (multiple-attribute) model, the storage of derived time series is complicated by the fact that derived time series data is available for only one of many columns (at least initially).

With a nested table storage model, a single value column is stored. This cleanly enables the storage of derived time series.

- When encapsulated types are available, it will be possible to support calendar and time series integrity by requiring insert, update, delete, and append operations to be executed using methods.

However, the nested IOT storage model has the following disadvantage compared to the flat IOT model: each attribute column requires a separate timestamp column. This increases the storage overhead for multiple-attribute time series data (such as daily stock market data that includes open, high, low, close, and volume attributes). However, many forms of time series data are single-attribute (such as the monthly unemployment rate), and for these formats the nested IOT storage model is ideal.

2.7 Interfaces to Time Series and Time Scaling Functions

The interfaces to the time series and time scaling functions rely on the following aspects of the Oracle8*i* Time Series architecture:

- Time series detail data is stored as relational data (in a flat IOT or flat table), one timestamp per row.
- Calendars are stored in object tables.
- Time series and time scaling functions expect time series data and calendars to be formatted as objects. A time series object is typically the first parameter to a function.

Two basic interfaces to time series and time scaling functions are defined:

- An instance-based object interface

  In the instance-based interface, the first input parameter to a time series function is an instance of a time series (for example, ORDTNumSeries).

- A reference-based object interface

  In the reference-based interface, the first input parameter to a time series function is a reference to a time series (for example, ORDTNumSeriesIOTRef). The reference-based interface requires that you provide enough descriptive information to enable the functions to execute dynamic SQL to obtain an instance of a time series.

The data types related to the instance-based and reference-based interfaces (for example, ORDTNumSeries and ORDTNumSeriesIOTRef) are discussed in Sections 2.7.1 and 2.7.2.

Note that both types of interfaces return only instances of time series (for example, ORDTNumSeries). Also, because nesting of time series and time scaling functions is allowed (for example, SELECT (Lead(Mavg, ...) ...)), the instance-based interface is used internally for the second and subsequent levels of nesting.

When possible, you should use the reference-based interface. Although this interface may be difficult to understand initially, it offers significant performance advantages over the instance-based interface. The examples in this guide emphasize the reference-based interface.

2.7.1 Instance-Based Interface

Time series and time scaling functions operate on instances of time series objects (for example, an ORDTNumSeries). An instance of a time series object includes a name field, an instance of a calendar, and an instance of a time series. For example, as the following data type definitions for a numeric time series show, ORDTNumTab defines a collection and ORDTNumSeries combines a calendar instance with a collection:

```
CREATE TYPE ORDSYS.ORDTNumCell AS OBJECT (tstamp DATE, value NUMBER);
CREATE TYPE ORDSYS.ORDTNumTab AS TABLE OF ORDTNumCell;
CREATE TYPE ORDSYS.ORDTNumSeries AS OBJECT (
     name VARCHAR2(256),
     cal ORDTCalendar,
     series ORDTNumTab
);
```

For a numeric time series, the time series data is contained in the ORDTNumTab structure. This structure is a table (collection) of a DATE column and a NUMBER column.

Figure 2–4 shows an example of an ORDTNumTab collection type.

Figure 2–4 Example of ORDTNumTab Collection Type

Tstamp	Value
01-01-1996	22.00
01-02-1996	23.00
...	...
12-31-1996	...

Functions such as Mavg (Moving Average, described in Section 2.10.7) use the ORDTNumTab structure as the source data for performing computations, and they use the ORDTCalendar data type to enable navigation through the time series data. The calendar-based navigation is especially useful for functions such as Mavg, which has as input parameters the starting date (*startDate*) and ending date (*endDate*) for which to return moving averages and an integer (*k*) indicating the lookback window (*k* denoting the number of timestamps, including the current one, over which to compute the moving average). Calendar-based navigation is used to determine the date that is *k-1* timestamps previous to *startDate*.

Although time series and time scaling functions operate on time series instances, they are invoked from SQL using a REF to a time series. For a numeric time series, this type is an ORDTNumSeriesIOTRef. (Section 2.7.2 explains the use of REFs in the reference-based interface.) The REF contains enough information so that time series and time scaling functions can derive the instance (ORDTNumSeries) at runtime (using dynamic SQL).

The convention of defining an interface on a DATE column and a *single* NUMBER column provides a uniform interface for time series and time scaling functions. Because the underlying IOT that stores time series detail data may have multiple NUMBER columns, the view defining the REF also maps the underlying storage to conform to the two-column interface defined by the ORDTNumSeries data type.

The following are the key aspects of the instance-based interface to time series and time scaling functions:

- The input parameter of a time series function is a REF to a time series object (for example, ORDTNumSeriesIOTRef).
- Time series and time scaling functions operate on time series instances (for example, ORDTNumSeries).

> **Note:** In addition to numeric series, a character time series is also provided, with the data types ORDTVarchar2Series and ORDTVarchar2SeriesIOTRef.

- You should use a view to construct the reference descriptor.
- The REF associates the calendar with the time series.
- Instances of calendars are typically stored in a table separate from time series detail data.

- It is important to ensure and maintain consistency between time series data and the corresponding calendar. Section 2.8 discusses consistency of time series data, including ways of ensuring consistency.

2.7.2 Reference-Based Interface

Oracle8*i* Time Series provides a reference-based interface for time series and time scaling functions.

This interface provides efficient performance, especially when only a portion of the time series is accessed. The performance benefit of this interface results from the fact that at runtime, the reference-based interface materializes only those rows within the specified date range, as opposed to materializing the entire collection of rows from the time series.

Note: You should use administrative tools procedures (documented in Chapter 7) to create and maintain objects for use with the reference-based interface. Normally, you should not manually create any views used by the reference-based interface, because the specific view definitions may change in future releases and because the administrative tools procedures provide a simple interface.

The reference-based interface uses the ORDTNumSeriesIOTRef and ORDTVarchar2SeriesIOTRef data types, which include a REF to a calendar, plus several literal values. At runtime, reference-based time series and time scaling functions use these literal values to form and execute a SQL statement (using dynamic SQL) that derives an instance of a time series that contains only the timestamps needed for this instance. The function determines which timestamps are needed based on the *startDate* and *endDate* parameters.

The ORDTNumSeriesIOTRef data type is defined as follows:

```
CREATE TYPE ORDSYS.ORDTNumSeriesIOTRef AS OBJECT
   (
   name                VARCHAR2(256),
   cal                 REF ORDSYS.ORDTCalendar,
   table_name          VARCHAR2(256),
   tstamp_colname      VARCHAR2(30),
   value_colname       VARCHAR2(30),
   qualifier_colname   VARCHAR2(30),
   qualifier_value     VARCHAR2(4000)
);
```

The attributes of the ORDTNumSeriesIOTRef data type are as follows:

- *name* is the name of the time series.
- *cal* is a REF to the calendar.
- *table_name* is the fully qualified name of the flat IOT.

  table_name can be a view, but the view must be updatable and must map to an IOT. If the view includes any functions, they must include the PRAGMA RESTRICT_REFERENCES compiler directive with the keywords WNPS, RNPS, and WNDS.
- *tstamp_colname* is the name of tstamp column in the flat IOT.
- *value_colname* is the name of the value column in the flat IOT (for example, *close* for the closing price).
- *qualifier_colname* is the name of the column that identifies a time series instance (for example, *ticker*).
- *qualifier_value* is the value of the column that identifies a time series instance (for example, *ACME*, which is the ticker for Acme Corporation).

In the Oracle8*i* Time Series usage demo, the view *stockdemo_ts* uses the reference-based interface to time series and time scaling functions. The *stockdemo_ts* view determines which calendar should be associated with the time series by accessing the calendar (*stockdemo_calendars*) and metadata (*stockdemo_metadata*) tables. The pricing data is accessed through the underlying table containing historical time series pricing data (*stockdemo*). For an explanation of the relationship between the reference-based view and the underlying tables in the usage demo, see Section 1.6.2.3.

The *stockdemo_ts* view is generated by the administrative tools procedures (documented in Chapter 7) with the following definition for the current release of Oracle8*i* Time Series. (This definition may change in future releases, and therefore you are encouraged to use the administrative tools procedures rather than manually creating such a view.)

```
CREATE OR REPLACE VIEW stockdemo_ts(ticker,open,high,low,close,volume) AS
  SELECT meta.tickername,
  ORDSYS.ORDTNumSeriesIOTRef(
       substr(meta.tickername, 1, 230) || ' open NumSeries',
              Ref(cal), 'tsdev.stockdemo',
              'tstamp', 'open', 'ticker', meta.tickername),
  ORDSYS.ORDTNumSeriesIOTRef(
       substr(meta.tickername, 1, 230) || ' high NumSeries',
              Ref(cal), 'tsdev.stockdemo',
              'tstamp', 'high', 'ticker', meta.tickername),
  ORDSYS.ORDTNumSeriesIOTRef(
       substr(meta.tickername, 1, 230) || ' low NumSeries',
               Ref(cal), 'tsdev.stockdemo',
               'tstamp', 'low', 'ticker', meta.tickername),
  ORDSYS.ORDTNumSeriesIOTRef(
       substr(meta.tickername, 1, 230) || ' close NumSeries',
               Ref(cal), 'tsdev.stockdemo',
               'tstamp', 'close', 'ticker', meta.tickername),
  ORDSYS.ORDTNumSeriesIOTRef(
       substr(meta.tickername, 1, 230) || ' volume NumSeries',
               Ref(cal), 'tsdev.stockdemo',
               'tstamp', 'volume', 'ticker', meta.tickername)
  FROM stockdemo_metadata meta, stockdemo_calendars cal
  WHERE meta.calendarname = cal.name;
```

Depending on which column is selected, a different literal value is applied as an attribute of the ORDTNumSeriesIOTRef data type. For example, for the following query:

```
SELECT ORDSYS.TimeSeries.Mavg(close,
                              to_date('02-DEC-96','DD-MON-YY'),
                              to_date('31-DEC-96','DD-MON-YY'),
                              10)
FROM TSDEV.stockdemo_ts
WHERE ticker='ACME';
```

The literal value *close* is used as the *value_colname* column name. The other attributes of the ORDTNumSeriesIOTRef data type include the timestamp column name (*tstam*p), a qualifying column name (*ticker*), and the actual value of the qualifying column (*meta.tickername*).

The implementation of time series and time scaling functions uses the information stored in the ORDTNumSeriesIOTRef data type to generate the appropriate dynamic SQL statement at runtime. Using the preceding example, to instantiate a time series object (that is, to convert an ORDTNumSeriesIOTRef to an ORDTNumSeries), the Mavg function generates a query that performs the following action (with the logic shown, not the exact syntax):

```
SELECT tstamp, close
FROM tsdev.stockdemo_ts
WHERE ticker='ACME' and tstamp BETWEEN <a date range adjusted
```

```
                        to reflect the 10-day window and the
                        calendar, including any holidays>;
```

The Mavg function computes the moving average and returns the result as a time series instance (ORDTNumSeries). For more information about the Mavg function, see Section 2.10.7.

2.8 Consistency of Time Series Data

Most time series and time scaling functions rely on calendars that are consistent with time series data.[1] By assuming a time series is consistent with its calendar, time series and time scaling functions can use the calendar as a basis for navigation of time series data.

Time series consistency must be maintained; otherwise, functions might raise exceptions or return incorrect results.

2.8.1 Rules for Time Series Consistency

For a time series to be consistent, the following must be true:

- All timestamps are sorted in ascending sequence.
- There are no duplicate timestamps.
- All timestamps match the precision of the calendar.
- No timestamps are beyond the bounds of the calendar (*minDate* and *maxDate*).
- All timestamps conform to the pattern specification, except those listed in the off-exceptions list or the on-exceptions list.
- The time series data is contiguous. That is, between the smallest (earliest) and largest (latest) timestamps in the time series, the time series data contains timestamps for all valid calendar timestamps.

If some mechanism is not used to enforce these consistency rules, accidental or malicious actions could destroy the integrity of the time series data. For example, a user might delete rows from the middle of the time series, rather than being restricted to deleting rows at the beginning and the end of the date range for the time series.

[1] An exception is the Fill function, which can be used to add pairs of timestamps and values to make a time series consistent with the calendar.

2.8.2 Enforcing Time Series Consistency with Relational Views

Enforcing time series consistency can be accomplished with a relational view of time series data that uses an INSTEAD OF trigger to maintain time series consistency. (For an explanation of INSTEAD OF triggers, see the *Oracle8i Concepts* manual.) This relational view is intended to be used for limited or moderate insert, update, and delete operations; it is not intended for bulk changes to time series data.

The usage demo (see Section 1.6) includes a relational view named *stockdemo_sv*. This view:

- Enables view updates to be propagated to the underlying table
- Ensures that the underlying table can be updated only by using a view mechanism, provided that users are granted update access to the relational view and not granted update access to the underlying table
- Ensures that update, delete, and insert operations affecting time series data are constrained to conform to the calendar associated with the time series

2.8.2.1 Precision

With the relational view, if a timestamp to be inserted is imprecise, an exception is raised. If a timestamp to be deleted is imprecise and if a matching timestamp exists in the time series, the deletion is permitted.

2.8.2.2 INSTEAD OF Trigger

An INSTEAD OF trigger in a relational view enforces rules on insert, delete, and update operations. These rules maintain time series data that conforms to the associated calendar.

For *insert* operations, the following rules apply:

- For an empty time series, the new timestamp must be a valid date in the calendar.
- For a time series that is not empty, an insertion is allowed immediately after the last timestamp or immediately before the first timestamp, but nowhere else.

For *delete* operations, the following rules apply:

- For an empty time series, an exception is raised.
- For a time series that is not empty, only the following can be deleted: the first or last timestamp, or an imprecise timestamp where a matching timestamp exists in the time series.

For *update* operations, the following rules apply:

- The timestamp must exist in the time series.
- Updates are not allowed to the timestamp and qualifier columns (for example, *tstamp* and *ticker* in the usage demo relational view).

The INSTEAD OF trigger in a relational view enables you to ensure that a time series meets the consistency requirements described in Section 2.8.1.

The INSTEAD OF trigger allows for multiple timestamps to be inserted or deleted in a single query, given that the group of timestamps inserted or deleted are in the proper order. For example, a specified number of timestamps can be deleted from the beginning of a time series by using a simple range restriction on the timestamp. A specified number of timestamps can be inserted at the end of a time series by using a subquery that references another table containing time series data.

2.8.3 Bulk Loading and Consistency

The SQL*Loader utility is useful for loading large amounts of data into a table. For better performance, you should perform bulk loads on underlying tables instead of on relational views. However, after you load data into the tables, you must ensure time series consistency by using one of the following approaches:

- Adjust calendars to be consistent with the time series.

  If you are sure that all timestamps are correct, it is safe to adjust the calendar to be consistent with the time series. This strategy is normally appropriate when there is a unique calendar per time series.

  The DeriveExceptions function is useful for adjusting a calendar to be consistent with the time series.

- Validate that each time series is consistent with the calendar.

  If you expect time series data to adhere to a predefined calendar, validating each time series is the better approach. This approach is particularly useful if the same calendar is used for all time series data being loaded.

  The IsValidTimeSeries function can be used to check if the time series is consistent with the calendar.

  For better performance in the case of a shared calendar for all time series, you may want to customize time series validation using PL/SQL. This involves writing custom utility functions that call Oracle8*i* Time Series product-developer calendar functions (see Section 2.9.2) to test and maintain time series consistency.

Section 3.4 contains additional information and examples of bulk and incremental loading of time series data.

2.9 Calendar Functions

Oracle8*i* Time Series provides calendar functions for querying and modifying calendars. The calendar functions can be divided into the following categories:

- *End-user* functions allow application developers to use the main calendar-related features.
- *Product-developer* functions allow developers to modify or supplement Oracle8*i* Time Series capabilities by creating value-added enhancements.

Reference information for all calendar functions is in Chapter 4.

2.9.1 End-User Functions

End-user functions let you use the main calendar-related features of Oracle8*i* Time Series. If you do not need to modify or expand the product's capabilities, you probably can limit your use of calendar functions to those listed in Table 2–4.

Table 2–4 End-User Calendar Functions

Function	Description
	Calendar-Creation Functions[1]
Second	Creates a calendar with a frequency of *second*.
Minute	Creates a calendar with a frequency of *minute*.
Hour	Creates a calendar with a frequency of *hour*.
Day	Creates a calendar with a frequency of *day*.
Week	Creates a calendar with a frequency of *week*.
Ten_day	Creates a calendar with a frequency of *10-day*.
Semi_monthly	Creates a calendar with a frequency of *semi-monthly*.
Month	Creates a calendar with a frequency of *month*.
Quarter	Creates a calendar with a frequency of *quarter*.
Semi_annual	Creates a calendar with a frequency of *semi-annual*.
Year	Creates a calendar with a frequency of *year*.
	Calendar-Related Functions

Table 2–4 End-User Calendar Functions (Cont.)

Function	Description
EqualCals	Returns 1 if the two calendars are equivalent. If a date range is provided, tests only equivalence between the supplied dates.
GenDateRangeTab	Returns a table of date ranges that represent all of the valid intervals in the input calendar (or from *startDate* through *endDate*).
IntersectCals	Intersects two calendars.
UnionCals	Returns the union of two calendars.
IsValidCal	Returns 1 if a calendar is valid and 0 if a calendar is not valid.
ValidateCal	Validates a calendar; repairs errors where possible.
Exception-Related Functions	
InsertExceptions	Inserts a list of timestamps into the appropriate exceptions list or lists.
DeleteExceptions	Deletes a list of timestamps from the appropriate exceptions list or lists.

[1] The calendar-creation functions create a calendar of with a frequency corresponding to the function name, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, and a specified or default name and anchor date.

2.9.2 Product-Developer Functions

Product-developer functions let you modify and expand the Oracle8*i* Time Series capabilities. For example, you could use product-developer calendar functions in creating a new function that modified the information returned for the moving average or that returned a net present value for a portfolio of stocks at a specified date.

Note: It is recommended that you not modify the functions provided with Oracle8*i* Time Series. If you want a function with a behavior different from an existing function, create a new function with a different name or put the function in a different package, or do both. For example, if you work for XYZ Corporation and create a modified moving average function, you could name the function *MavgXYZ* and put it in a package named *XYZPackage*.

Table 2–5 lists the product-developer calendar functions.

Table 2–5 Product-Developer Calendar Functions

Function	Description
Calendar-Related Functions	
CombineCals	Combines two calendars. Similar to IntersectCals, except the patterns must be identical.
Exception-Related Functions	
NumOffExceptions	Returns the number of off-exceptions between two dates.
NumOnExceptions	Returns the number of on-exceptions between two dates.
Date and Index-Related Functions	
IsValidDate	Determines if a supplied date is valid.
OffsetDate	Returns a date that is *k* dates in the future (or *k* in the past if *k* is negative) of the supplied date.
GetIntervalStart	Returns the start of the interval that includes the input timestamp.
GetIntervalEnd	Returns the end of the interval that includes the input timestamp.
NumInvalidTstampsBetween	Returns the number of invalid timestamps between two dates.
NumTstampsBetween	Returns the number of valid timestamps between two dates.
TstampsBetween	Returns the valid timestamps between two dates.
InvalidTstampsBetween	Returns the invalid timestamps between two dates.
SetPrecision	Sets the precision of the input timestamp to correspond to the frequency of the input calendar.

For an example of using product-developer functions, see Section 3.9.

2.10 Time Series Functions

Time series functions operate on a time series. A time series data type is always used as the input parameter to a time series function.

Reference information for all time series functions is in Chapter 5.

2.10.1 Extraction, Retrieval, and Trim Functions

Time series extraction, retrieval, and trim functions operate on any time series type. Extraction functions return one or more time series rows, while retrieval and trim functions return a time series.

Table 2–6 lists the extraction functions.

Table 2–6 Extraction Functions

Function	Description
DeriveExceptions	Returns a calendar populated with exceptions derived from either a calendar and a table of dates or two time series.
ExtractCal	Returns a calendar that is the same as the calendar on which the time series is based.
ExtractDate	Gets the date from an element in a time series.
ExtractTable	Returns the time series table (ORDTNumTab or ORDTVarchar2Tab) associated with a time series.
ExtractValue	Gets the value stored in an element in a time series.
First	Gets the first element in a time series.
GetDatedElement	Gets the element of a time series at a supplied date.
GetNthElement	Gets the Nth element of a time series.
Last	Gets the last element in a time series.

Table 2–7 lists the retrieval and trim functions.

Table 2–7 Retrieval and Trim Functions

Function	Description
FirstN	Gets the first *n* elements in a time series.
GetSeries	Returns the entire time series.
LastN	Gets the last *n* elements in a time series.
TrimSeries	Returns the time series data between the supplied dates.

2.10.2 Shift Functions

Shift functions (listed in Table 2–8) lead or lag a time series by a specified number of units, where units reflects the frequency of the calendar for the time series.

Table 2–8 Shift Functions

Function	Description
Lead	Leads a time series by the specified number of units.
Lag	Lags a time series by the specified number of units.

2.10.3 SQL Formatting Functions

When called from a SQL SELECT expression, a time series function returns an instance of a time series data type, which cannot be displayed. The SQL formatting functions (listed in Table 2–9) facilitate format conversions that allow time series to be displayed.

Table 2–9 SQL Formatting Functions

Function	Description
ExtractCal	Given a time series, returns a calendar that is the same as the calendar on which the time series is based.
ExtractDate	Given an element in a time series, returns the date.
ExtractTable	Given a time series, returns the time series table (ORDTNumTab or ORDTVarchar2Tab) associated with the time series.
ExtractValue	Given an element in a time series, returns the value stored in it.

2.10.4 Aggregate Functions

Aggregate functions (listed in Table 2–10) return scalar or ORDTNumTab values. Each aggregate function can be used in either of the following ways:

- The function accepts a numeric time series, ORDTNumSeries, and operates on all elements of the collection.
- The function accepts a numeric time series, ORDTNumSeries, and a date range, bounded by date1 and date2. The function is computed on the time series defined by the date range.

Thus, each aggregate function is of the form:

```
f(ts ORTDNumSeries, [date1 DATE, date2 DATE])
```

Table 2–10 Aggregate Functions

Function	Returns
TSAvg	Average (mean) of a time series
TSCount	Number of elements in a time series
TSMax	Maximum value of a time series
TSMaxN	Specified number of top (highest) values in a time series
TSMedian	Middle element of a time series
TSMin	Minimum value of a times series
TSMinN	Specified number of bottom (lowest) values in a time series
TSProd	Product of the elements of a time series
TSStddev	Standard deviation (square root of VAR)
TSSum	Sum of the elements of a time series
TSVariance	Variance (analogous to the SQL group function VAR)

2.10.5 Arithmetic Functions

Arithmetic functions (listed in Table 2–11) accept two time series (ORDTNumSeries1,ORDTNumSeries2) or a time series and a constant (ORDTNumSeries1, Const), and perform a pairwise arithmetic operation on each element of the time series. This operation determines the value of each element of the returned time series:

```
Algorithm for f(ts1, ts2)
ForAll i, tsRet(i) = ts1(i) op ts2(i);
```

Table 2–11 Arithmetic Functions

Function	Description
TSAdd	Time series addition
TSDivide	Time series division
TSMultiply	Time series multiplication
TSSubtract	Time series subtraction

2.10.6 Cumulative Sequence Functions

Cumulative sequence functions (listed in Table 2–12) operate on successive elements of a time series, accumulating the result into the current element of the output time series. For example, CSUM((1,2,3,4,5)) => (1,3,6,10,15). In this example, the result time series (f(i)), is computed from the input time series (I(i)) as follows:

```
f(1) = I(1)
ForAll i > 1, f(i) = f(i - 1) + I(i)
```

Table 2–12 Cumulative Sequence Functions

Function	Returns
Cavg	Cumulative average
Cmax	Cumulative maximum
Cmin	Cumulative minimum
Cprod	Cumulative product
Csum	Cumulative sum

2.10.7 Moving Average and Sum Functions

The Moving Average (Mavg) function returns a time series that contains the averages of values from each successive timestamp for a specified interval over a range of dates. For example, the 30-day moving average for a stock is the average of the closing price for the specified date and the 29 trading days preceding it.

The Moving Sum (Msum) function returns a sum of values from each successive timestamp for a specified interval over a range of dates. For example, the 30-day moving sum of trading volumes for a stock is the sum of the volume for the specified date and for 29 trading days preceding it.

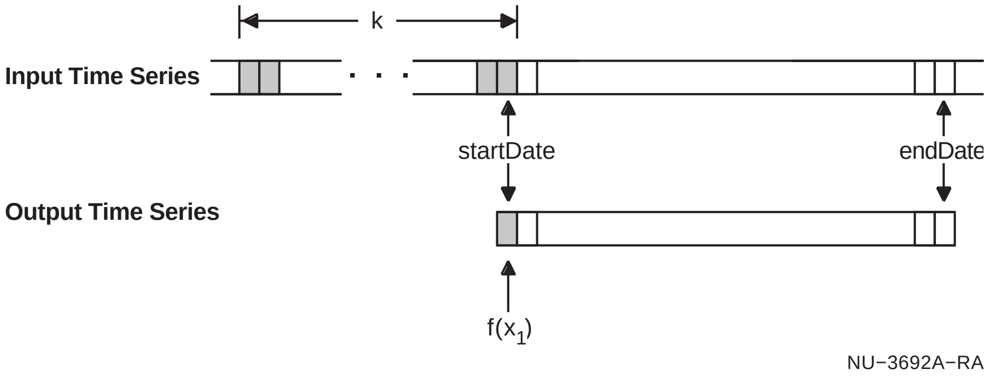
Table 2–13 lists the moving average and sum functions.

Table 2–13 Moving Average and Sum Functions

Function	Returns
Mavg	Moving average
Msum	Moving sum

The relationship between the input and output time series in the computation of a moving average or sum is illustrated in Figure 2–5. The figure focuses on the common invocation of moving average or sum, where *k* is the number of timestamps in the lookback window (for example, 30) and a date range (*startDate* and *endDate*) is supplied. (For more information about the parameters, see the Mavg function description in Chapter 5.)

Figure 2–5 Relationship of Input and Output Time Series in Moving Average/Sum



2.10.8 Conversion Functions

Conversion functions (see Table 2–14) convert a time series by filling in missing timestamp-value pairs. With the Fill function, any timestamps that are valid calendar timestamps but missing from the time series are inserted into the time series. This function is especially useful for converting a time series from one calendar to another.

Table 2–14 Conversion Functions

Function	Description
Fill	Fills a time series based on the calendar and fill type.

2.11 Time Scaling Functions

Oracle8*i* Time Series provides functions to scale time series data:

- **Scaleup** functions produce summary information from finer granularity information. For example, monthly data can be derived based on daily data. Scaleup is also known as *rollup*.

- **Scaledown** functions generate finer-granularity information from coarser-granularity information. For example, quarterly data can be converted to a daily time series. Scaledown is also known as *distribution*.

The relationship between the input and output time series in a scaleup operation is illustrated in Figure 2–6, which shows a mapping when scaling from a daily frequency to a monthly frequency.

Figure 2–6 Time Scaling from Daily to Monthly Frequency

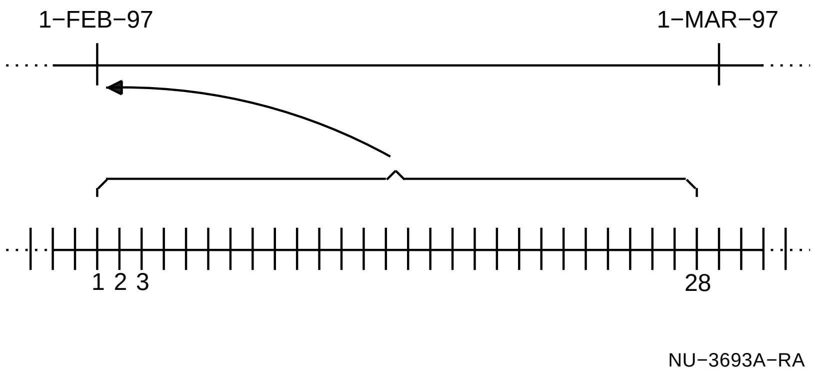


Figure 2–6 shows all days in February being mapped to the month of February. This mapping also suggests the importance of the precision of timestamps of different frequencies. In the example shown in this figure:

- The month timestamp for February 1997 is represented as 1-FEB-97 00:00:00.
- The day timestamps for February 1997 are represented as 1-FEB-97 00:00:00, 2-FEB-97 00:00:00,... 28-FEB-97 00:00:00.

Time scaling is permitted only when the calendar for the target time series is an integral multiple of the calendar for the data to be scaled. For example, weekly data (data associated with a calendar with a *week* frequency) cannot be scaled up to a calendar with a frequency of *month*, *quarter*, *half year*, or *year* because a week does not divide evenly into any of these time periods. However, monthly data can be scaled up to a calendar with a frequency of *quarter*, *half year*, or *year*.

Table 2–15 provides a scaling compatibility matrix that shows for each frequency the frequencies to which you can scale up and scale down data. For each cell in the matrix, a Y means that scaling is permitted and a blank means that scaling is not permitted. For scaleup operations, go down the left column to find the desired scale-from frequency, then go across that row to see if scaling is permitted for the desired scale-to frequency. For scaledown operations, go across the top row to find the desired scale-from frequency, then go down that column to see if scaling is permitted for the desired scale-to frequency.

Table 2–15 Scaling Compatibility Matrix

	Day	Week	10-day	Semi-month	Month	Quarter	Semi-annual	Annual
Day	Y	Y	Y	Y	Y	Y	Y	Y
Week		Y						
10-day			Y		Y	Y	Y	Y
Semi-month				Y	Y	Y	Y	Y
Month					Y	Y	Y	Y
Quarter						Y	Y	Y
Semi-annual							Y	Y
Annual								Y

If the calendar for the target time series has a zero ('0') pattern and one or more on-exceptions that are precise with respect to the calendar frequency, the time scaling functions use the on-exceptions to perform scaling. For example, if quarterly dividend payment dates are defined as on-exceptions with a *day* frequency calendar that has a zero pattern (ORDSYS.ORDTPatternBits(0)), the ScaledownRepeat function could be used to insert the current quarterly dividend rate in each daily timestamp.

The collection-based interface (operations on collections) for time scaling is discussed in Section 2.11.1.

Reference information for all time scaling functions is in Chapter 6.

2.11.1 Time Scaling on Collections

The scaleup and scaledown functions accept as input a numeric time series and a destination calendar. A numeric time series is returned, which is scaled based on the destination calendar.

For example, the following statement returns the last closing prices for stock SAMCO for the months of October, November, and December of 1996:

```
select * from the
  (select cast(ORDSYS.TimeSeries.ExtractTable(
                 ORDSYS.TimeSeries.ScaleupLast(
                        ts.close,
                        sc.calendar,
                        to_date('01-OCT-1996','DD-MON-YYYY'),
                        to_date('01-JAN-1997','DD-MON-YYYY')
                        )
              ) as ORDSYS.ORDTNumTab)
       from tsdev.stockdemo_ts ts, tsdev.scale sc
       where ts.ticker='SAMCO' and
             sc.name  ='MONTHLY');
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-OCT-96     42.375
01-NOV-96      38.25
01-DEC-96      39.75
3 rows selected.
```

Note that each timestamp reflects the first date of the month in the calendar (following the convention illustrated in Table 2–3), and each value in this case reflects the closing price on the last date for that month in the calendar.

Scaleup functions ignore nulls. For example, ScaleupAvg returns a time series reflecting the average value of each scaled group of non-null values.

Table 2–16 lists the scaleup functions, and Table 2–17 lists the scaledown functions.

Table 2–16 Scaleup Functions for Collections

Function	Description
ScaleupAvg	Returns the average value of each group.
ScaleupAvgX	Returns the average value of the sum of each group and the immediately preceding source period (for example, a first-quarter average computed as the average for the months of December, January, February, and March).
ScaleupCount	Returns the count of timestamps in each group.
ScaleupGMean	Returns the geometric mean of each group.

Table 2–16 Scaleup Functions for Collections (Cont.)

Function	Description
ScaleupSum	Returns the sum of each group.
ScaleupSumAnnual	Returns the sum of each group multiplied by a factor to state the resulting time series at annual rates.
ScaleupMin	Returns the minimum of each group.
ScaleupMax	Returns the maximum of each group.
ScaleupFirst	Returns the first value of each group.
ScaleupLast	Returns the last value of each group.

Table 2–17 Scaledown Functions for Collections

Function	Description
ScaledownInterpolate	Returns missing values by interpolating between the values of the input time series.
ScaledownRepeat	Returns missing values by repeating the value of the input time series.
ScaledownSplit	Returns missing values by splitting (dividing) the value in the input time series evenly.

2.11.2 Scaleup Options: IgnoreNulls and DiscardError

All scaleup functions allow you to specify either or both of the following options:

- IgnoreNulls (default) or IgnoreNullsOFF
- DiscardError (default) or DiscardErrorOFF

These options do not apply to scaledown functions.

IgnoreNulls controls the behavior with respect to null values. If IgnoreNulls is in effect, nulls in the input time series are not be included in the aggregation being performed. For example, if ScaleupAvg is operating on a group with 12 values, 3 of which are null, only the 9 non-null values are averaged. If IgnoreNullsOFF is specified, then any calculation involving one or more nulls results in a null. For example, if IgnoreNullsOFF is specified and ScaleupAvg is operating on a group with 12 values, 3 of which are null, a null is returned.

DiscardError controls the behavior with respect to gaps in the target calendar. If DiscardError (the default setting) is in effect, then whenever data from the source time series has no corresponding interval in the target time series (which can result from zeros in the pattern bits of the target calendar), an exception is raised. An example of this condition is scaling up daily data to a monthly calendar for the months January through March (anchor date of 01-Jan and pattern of '1,1,1,0,0,0,0,0,0,0,0,0'). Using this example, the default behavior (DiscardError) raises an exception if any input timestamps are from April through December; however, DiscardErrorOFF performs the scaling for January through March and ignores any input timestamps from April through December. The default behavior ensures that certain types of incompatible calendars are not inadvertently used in scaling.

2.11.2.1 Syntax Options: Names and Numbers

You can express the IgnoreNulls and DiscardError options syntactically using either names or a number. Using names, you can specify one or both of the following as the final parameter or parameters of a scaleup function call:

- ORDSYS.TimeScale.IgnoreNulls or ORDSYS.TimeScale.IgnoreNullsOFF
- ORDSYS.TimeScale.DiscardError or ORDSYS.TimeScale.DiscardErrorOFF

Instead of using names, you can use a one-digit or two-digit number from Table 2–18:

Table 2–18 IgnoreNulls and DiscardError Syntax Options

		IgnoreNulls	
		ON	OFF
DiscardError	ON	0	1
	OFF	10	11

The following examples show the use of names and a number to specify the same options (IgnoreNullsOFF and DiscardErrorOFF):

```
ORDSYS.TimeScale.ScaleUpAvg('My Timeseries', myTS, targetCal,
                         ORDSYS.TimeScale.IgnoreNullsOFF,
                         ORDSYS.TimeScale.DiscardErrorOFF);
ORDSYS.TimeScale.ScaleUpAvg('My Timeseries', myTS, targetCal,
                         11);
```

Names and numbers for the options cannot be used in the same function call. For example, the following is *not* valid:

```
ORDSYS.TimeScale.ScaleUpAvg('My Timeseries', myTS, targetCal,
                            1, ORDSYS.TimeScale.DiscardErrorOFF);
```

An exception is raised if an option is specified twice or if conflicting options are specified (for example, specifying IgnoreNulls and IgnoreNullsOff in the same call). These options can be specified in any order, but they must appear after any other parameters to the function.

2.12 Administrative Tools Procedures

Oracle8*i* Time Series provides procedures that simplify the creation of time series schema objects. The quick-start demo (described in Section 1.6.1) illustrates the use of several of these administrative tools procedures.

Table 2–19 lists the administrative tools procedures. Reference information for these procedures is in Chapter 7.

Table 2–19 Administrative Tools Procedures

Procedure	Description
Add_Existing_Column	Adds a column attribute from an existing flat table to a time series.
Add_Integer_Column	Adds an integer column attribute to an ongoing flat time series creation specification.
Add_Number_Column	Adds a number column attribute to an ongoing flat time series creation specification.
Add_Varchar2_Column	Adds a VARCHAR2 column attribute to an ongoing flat time series creation specification.
Begin_Create_TS_Group	Initiates the context for creating a time series group (the schema objects for a time series).
Cancel_Create_TS_Group	Cancels the creation of a time series group, that is, cancels the context initiated by the Begin_Create_TS_Group procedure.
Close_Log	Closes the log file that had been opened by the Open_Log procedure.
Display_Attributes	Displays information about the time series schema being created.

Table 2–19 Administrative Tools Procedures (Cont.)

Procedure	Description
Drop_TS_Group	Deletes the time series definition and views associated with it. However, the underlying tables (calendar tables, detail data tables, and so on) are not deleted.
Drop_TS_Group_All	Deletes the time series definition and all tables, views, indexes, constraints, and triggers associated with it.
End_Create_TS_Group	Closes the context established by the Begin_Create_TS_Group procedure and creates all appropriate schema objects.
Get_Flat_Attributes	Retrieves the attributes of a flat time series.
Get_Object_Attributes	Retrieves the attributes of an object-model time series.
Get_Status	Checks to see if a time series creation sequence is in progress.
Open_Log	Opens a log file that will contain the data definition language (DDL) statements generated by the Time Series administrative tools procedures.
Set_Flat_Attributes	Sets the attributes of a flat time series.
Set_Object_Attributes	Sets the attributes of an object-model time series.
Trace_Off	Disables debugging for Oracle8*i* Time Series administrative tools procedures. Any data definition language (DDL) statements and errors encountered when generating DDL statements will not be logged to SERVEROUTPUT.
Trace_On	Enables debugging for Oracle8*i* Time Series administrative tools procedures. Any data definition language (DDL) statements and errors encountered when generating DDL statements will be logged to SERVEROUTPUT.

2.12.1 Role Requirement for Administrative Tools Procedures

To create, delete, and modify schema objects using the Oracle8*i* Time Series administrative tools procedures, you must have been granted one or more of the following roles:

- DBA
- TIMESERIES_DBA
- TIMESERIES_DEVELOPER (deletion restricted to current user)

For deletion of time series schema objects, the DBA and TIMESERIES_DBA roles let you delete objects that belong to any user (schema), but the

TIMESERIES_DEVELOPER role lets you delete only objects that belong to the current user.

2.12.2 Other Requirements for Administrative Tools Procedures

Logging, which is controlled by the Open_Log and Close_Log procedures, relies on the PL/SQL file I/O procedure UTL_FILE, which is documented in the *Oracle8i Application Developer's Reference* **-** *Packages* manual.

To use logging, one or more directories for UTL_FILE output must be defined using the UTL_FILE_DIR parameter in the Oracle initialization file. For information about the UTL_FILE_DIR parameter, see the *Oracle8i Reference* manual.

3

Time Series Usage

This chapter explains important procedures related to using Oracle8*i* Time Series. It contains the following major sections:

- Section 3.1, "Creating a Time Series Group"
- Section 3.2, "Creating a Calendar" (also validating the calendar)
- Section 3.3, "Maintaining a Map Table"
- Section 3.4, "Populating the Detail Table Using SQL*Loader" (loading time series data)
- Section 3.5, "Retrofitting Existing Tables" ("retrofitting" existing detail, calendar, and map tables by using administrative tools procedures to generate schema objects)
- Section 3.6, "Validating Time Series Consistency"
- Section 3.7, "Formulating Time Series Queries"
- Section 3.8, "Deriving Calendar Exceptions" (deriving exceptions from time series data)
- Section 3.9, "Using Product-Developer Functions"

For detailed explanations of Oracle8*i* Time Series concepts and terminology, see Chapter 2.

3.1 Creating a Time Series Group

You can use the administrative tools procedures to create a time series group (all the necessary time series schema objects), accepting default values for most object names. These procedures provide a convenient, simple way to create time series schema objects, and they are recommended for most users. These procedures are used in the quick-start demo (see Section 1.6.1) and the usage demo (see Section 1.6.2).

The following example shows the use of administrative tools procedures to create all the necessary schema objects:

```
DECLARE

BEGIN

--
-- Establish 'tsquick' as the time series group name for purposes of the
-- administrative tools procedures. Columns will automatically be created
-- for the time series name (which will be set to 'ticker') and a
-- timestamp. The columns for the opening, high, low, and closing prices
-- and the trading volume will be explicitly defined.
--

  ORDSYS.TSTools.Begin_Create_TS_Group('tsquick','flat');

-- Set 'ticker' as the name of the time series for functions.
-- Sample values for specific tickers include 'ACME', 'AONE', and 'XCORP'.

  ORDSYS.TSTools.Set_Flat_Attributes(tsname_colname => 'ticker');
  ORDSYS.TSTools.Set_Flat_Attributes(tsname_length => 10);

-- Define numeric columns for prices.

  ORDSYS.TSTools.Add_Number_Column('open');
  ORDSYS.TSTools.Add_Number_Column('high');
  ORDSYS.TSTools.Add_Number_Column('low');
  ORDSYS.TSTools.Add_Number_Column('close');

-- Define an integer column for trading volume (number of shares
-- traded on a given day).

  ORDSYS.TSTools.Add_Integer_Column('volume');

-- End the specification of schema objects and create the objects.
--

  ORDSYS.TSTools.End_Create_TS_Group;

  exception
    when others then
       begin
```

```
            ORDSYS.TSTools.Cancel_Create_TS_Group;
            raise;
        end;

END;
/
```

The preceding call to End_Create_TS_Group causes many schema objects to be created. Among them are:

- TSQUICK - object view. Use this when using Time Series functions.

  Example: SELECT TimeSeries.<function>(ts.OPEN) FROM TSQUICK ts;
- TSQUICK_RVW - relational view. Use this for protected insert, update, and delete operations. Uses an INSTEAD OF trigger.
- TSQUICK_TAB - flat table for detail data.
- TSQUICK_MAP - mapping (metadata) table. The quick-start demo associates a null calendar with each ticker.
- TSQUICK_CAL - table for calendar definitions. The quick-start demo defines a monthly calendar for use with scaleup operations.

3.2 Creating a Calendar

Calendars are needed if one or more of the following conditions apply:

- You are using any regular time series.
- You need to perform time scaling.

You have several options for creating a calendar, including:

- Create the calendar dynamically in the context of a query, using one of the calendar-creation functions. See the information on the Month function in Chapter 4 for an example.
- Create the calendar by inserting its definition in a table of calendars. If the table of calendars does not already exist, create it first.

Calendars for a regular time series are stored in the calendar table associated with that time series group. The calendar table typically has a name in the format *groupname_CAL* (for example, *tsquick_cal* for the quick-start demo). Calendars to be used for scaling can be stored in the group calendar table or in a calendar table that is separately created and managed.

Calendars are based on the system-defined data type ORDTCalendar, which is supplied with Oracle8*i* Time Series. ORDTCalendar has the following definition:

```
/* System-Defined Calendar Data Type */

CREATE TYPE ORDSYS.ORDTCalendar AS OBJECT (
  caltype INTEGER,
  name VARCHAR2(256),
  frequency INTEGER,
  pattern ORDSYS.ORDTPattern,
  minDate DATE,
  maxDate DATE,
  offExceptions ORDSYS.ORDTExceptions,
  onExceptions ORDSYS.ORDTExceptions);
```

Example 3–1 creates a table named *my_calendars* and defines a calendar named *BusinessDays-97*. The *BusinessDays-97* calendar includes Mondays through Fridays in 1997, but excludes 04-Jul-1997 and 25-Dec-1997. Explanatory notes follow the example.

Example 3–1 Create a Calendar of Business Days

```
CREATE TABLE my_calendars of ORDSYS.ORDTCalendar;

INSERT INTO my_calendars ❶
VALUES(
   ORDSYS.ORDTCalendar(
       0, ❷
       'BusinessDays-97', ❸
       4, ❹
      ORDSYS.ORDTPattern(ORDSYS.ORDTPatternBits(0,1,1,1,1,1,0), ❺
         (to_date('01-05-97','MM-DD-YY'))),
      to_date('01-01-97','MM-DD-YY'), ❻
      to_date('01-01-98','MM-DD-YY'),
       ORDSYS.ORDTExceptions(to_date('07-04-97','MM-DD-YY'), ❼
         to_date('12-25-97','MM-DD-YY')),
      NULL)); ❽
```

Notes on Example 3–1:

❶ *my_calendars* is a table of ORDSYS.ORDTCalendar objects. The ORDTCalendar data type is described in Section 2.3.1.

❷ *0* (zero) for calendar type (*caltype*) indicates that this is an exception-based calendar. (This is the only calendar type currently supported.)

❸ *BusinessDays-97* is the name of this calendar.

❹ *4* is the frequency code for *day*.

❺ The pattern is defined as an excluded occurrence followed by five included occurrences followed by an excluded occurrence (`0,1,1,1,1,1,0`). Because the frequency is daily and because the anchor date (05-Jan-1997) is a Sunday, Sundays are excluded, Mondays through Fridays are included, and Saturdays are excluded.

❻ The calendar begins at the start of 01-Jan-1997 and ends at the start of 01-Jan-1998.

❼ 04-Jul-1997 and 25-Dec-1997 are off-exceptions (that is, excluded from the calendar).

❽ NULL indicates that there are no on-exceptions (that is, no Saturday or Sunday dates to be included in the calendar).

After you create the calendar, you should validate it to ensure that you have not made any mistakes in the calendar definition. The following example validates the *BusinessDays-97* calendar created by Example 3–1.

```
DECLARE
        tstCal ordsys.ordtcalendar;
        dummyVal integer;
        validFlag integer;
BEGIN

   -- Select a calendar into tstCal from my_calendars.
   select value(cal) into tstCal
   from my_calendars cal
   where cal.name = 'BusinessDays-97';

   -- Display the calendar
   select ordsys.timeseries.display(tstCal) into dummyVal from dual;
   dbms_output.new_line;

   validFlag := ORDSYS.CALENDAR.IsValidCal(tstCal);

   if (validFlag = 1) then
      dbms_output.put_line('BusinessDays-97 calendar is valid.');
   else
      dbms_output.put_line('BusinessDays-97 calendar is NOT valid.');
      dbms_output.put_line('Use ValidateCal to determine inconsistency.');
   end if;
```

```
END;
/
Statement processed.

Calendar Name = BusinessDays-97
 Frequency = 4 (day)
 MinDate = 01-JAN-97
 MaxDate = 01-JAN-98
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 05-JAN-97
 onExceptions  :     Atomic NULL

 offExceptions :
     04-JUL-97     25-DEC-97

BusinessDays-97 calendar is valid.
```

3.3 Maintaining a Map Table

A map table maintains the mapping, or coupling, between a time series (such as a specific stock ticker) and a calendar. When you create a time series group, the map table by default has a name in the form *groupname_MAP*. (In the quick-start demo, the map table is named *tsquick_map*; in the usage demo, the map table is named *stockdemo_metadata*.) The map table has two VARCHAR2 columns:

- Time series name that by default matches the *tsname_colname* value when the time series group was created (for example, *ticker*)
- Calendar name

The following example creates a map table named *my_calendars_map*:

```
CREATE TABLE my_calendars_map (
   ticker   VARCHAR2(5),
   calname VARCHAR2(256),
   CONSTRAINT  pk_my_calendars_map PRIMARY KEY (ticker));
```

For each row in the map table, the calendar name can be null or can contain the name of a calendar:

- If the calendar name column is null, no calendar will be used for Time Series functions (that is, the input time series will be treated as an irregular time series). The following example creates a row in the *my_calendars_map* table for each ticker in the *tsquick_tab* table and leaves the *calname* column null:

```
INSERT INTO my_calendars_map (ticker)
     SELECT DISTINCT ticker FROM tsdev.tsquick_tab;
```

- If the calendar name column contains the name of a calendar, that calendar is used for Time Series functions specifying that time series. The following example creates two rows in the *my_calendars_map* table, associating two tickers with the *BusinessDays-97* calendar:

  ```
  INSERT INTO my_calendars_map VALUES('ACME', 'BusinessDays-97');
  INSERT INTO my_calendars_map VALUES('SAMCO', 'BusinessDays-97');
  ```

For rows where a calendar name is specified, you can adopt one of the following strategies, depending on which calendars apply to which time series:

- Use the same calendar for all time series (the "shared calendar" approach). For example, map all tickers to a single calendar of stock trading days.
- Use a separate calendar for each time series. For example, create an *acme* calendar for the ACME ticker, a *samco* calendar for the SAMCO ticker, and so on.
- Use a combination of approaches: use some calendars for multiple time series, and perhaps some calendars for only one time series each. For example, some stocks might trade on exchanges with different holidays, or some stocks might have had trading suspended on certain days.

3.4 Populating the Detail Table Using SQL*Loader

To populate the underlying data storage table or tables, perform a bulk load of the time series data, preferably using the SQL*Loader utility if you have a large amount of data. For example, the tsquick.sql procedure uses SQL*Loader with hypothetical stock market data, as follows:

```
sqlldr userid=tsdev/tsdev control=tsquick.ctl log=tsquick.log bad=tsquick.bad
skip=15 errors=1000
```

To update the time series data, perform incremental loads as needed.

To ensure the consistency of time series data during loading, you must choose one of the approaches described in Section 2.8.3:

- Adjust calendars to be consistent with the time series, if you are sure that all timestamps are correct.

This strategy is normally appropriate when there is a unique calendar per time series.

- Validate that each time series is consistent with the calendar, if you expect time series data to adhere to a predefined calendar.

  This approach is particularly useful if the same calendar is used for all time series data being loaded.

This section describes how to perform bulk loading using these two approaches, and it also describes how to perform incremental loading.

The loading of time series data is usually performed under controlled circumstances, so it is safe to perform these loads directly to an underlying table instead of to a relational view.

3.4.1 Bulk Loading

After you create an index-organized table (IOT) to hold time series data (such as for the *stockdemo* demo database), you must populate the table with data. For a database of stock information, you may need to load millions of rows of daily summary information into the IOT.

SQL*Loader is recommended for loading large amounts of time series data. The following example shows a SQL*Loader script, with an excerpt from the sample data (stockdat.dat) and the SQL*Loader control file (stockdat.ctl). For complete information about SQL*Loader, see the *Oracle8i Utilities* manual.

The SQL*Loader script contains the following:

```
% sqlldr userid=tsdev/tsdev control=stockdat.ctl
       log=stockdat.log bad=stockdat.bad errors=1000
```

The stockdat.dat sample data file includes the following:

```
ACME  01-NOV-96  59.00  60.00  58.00  59.00  1000
ACME  04-NOV-96  60.00  61.00  59.00  60.00  1000
ACME  05-NOV-96  61.00  62.00  60.00  61.00  1000
         ...
```

The stockdat.ctl file contains the following

```
options (direct=true)
unrecoverable
load data
infile 'stockdat.dat'
replace
into table stockdemo
```

```
sorted indexes (StockTabx)
fields terminated by whitespace
(ticker, tstamp DATE(13) "DD-MON-YY", open, high, low, close, volume)
```

SQL*Loader can handle many file formats and delimiters, as documented in the *Oracle8i Utilities* manual.

After the load has completed, you may want to choose one of the following approaches for ensuring calendar consistency:

- Adjust calendars to conform to time series data (see Section 3.4.1.1).
- Validate that the time series conforms to the calendar (see Section 3.4.1.2).

In either case, you may need to update the exception lists of your calendars.

3.4.1.1 Adjusting Calendars to Conform to Time Series Data

Often you will want to create calendars that conform to the time series data that you are receiving. In this case, you usually know the frequency and the pattern of a calendar, but not the specific on- or off-exceptions. You can extract these exceptions from the data by using the DeriveExceptions function.

3.4.1.2 Validating That the Time Series Conforms to the Calendar

Often you will want to ensure that the time series data extracted from the incoming data conforms to a predefined calendar. To do this, insert the exceptions either when you create the calendar or afterward with the InsertExceptions functions (or do both, creating the calendar with some exceptions and then adding others); then use the IsValidTimeSeries function to check that the time series is consistent with the calendar.

You can insert exceptions when you define the calendar. For example, the following statement specifies 28-Nov-1996 and 25-Dec-1996 as off-exceptions in the calendar named *BUSINESS-96*:

```
INSERT INTO stockdemo_calendars VALUES(
       ORDSYS.ORDTCalendar(
             0,
             'BUSINESS-96',
             4,
             ORDSYS.ORDTPattern(
                      ORDSYS.ORDTPatternBits(0,1,1,1,1,1,0),
                           TO_DATE('01-JAN-1995','DD-MON-YYYY')),
             TO_DATE('01-JAN-1990','DD-MON-YYYY'),
             TO_DATE('01-JAN-2001','DD-MON-YYYY'),
             ORDSYS.ORDTExceptions(
                             TO_DATE('28-NOV-1996','DD-MON-YYYY'),
                             TO_DATE('25-DEC-1996','DD-MON-YYYY')),
             ORDSYS.ORDTExceptions()
             ));
```

You can also add exceptions after the calendar is defined by using the InsertExceptions function. For example, the following statement adds 01-Jan-1997, 17-Feb-1997, and 26-May-1997 as off-exceptions:

```
UPDATE stockdemo_calendars cal
   SET cal = (SELECT ORDSYS.Calendar.InsertExceptions(
                       VALUE(cal),
                       ORDSYS.ORDTDateTab(
                             to_date('01-JAN-97','DD-MON-YY'),
                             to_date('17-FEB-97','DD-MON-YY'),
                             to_date('26-MAY-97','DD-MON-YY')))
          FROM dual)
WHERE cal.name = 'BUSINESS-96';
```

After you have defined the calendar and populated the exception lists, you can use the IsValidTimeSeries function to check that the time series is consistent with the calendar.

3.4.2 Incremental Loading

After you have performed the bulk load of time series data and have started using Oracle8*i* Time Series, you will probably want to add data periodically. For example, every trading day after the stock exchange closes, that day's data for each ticker becomes available.

As with bulk loading, incremental loading is typically done in a controlled environment. You know which timestamps will become off-exceptions, and you can explicitly update the exception lists of the appropriate calendars. The following example demonstrates such an update:

```
UPDATE stockdemo_calendars cal
 SET cal = (SELECT ORDSYS.Calendar.InsertExceptions(
                        VALUE(cal),
                        to_date('01-JAN-97','DD-MON-YY'))
          FROM dual)
        WHERE cal.name = 'XCORP';
```

The SQL*Loader utility is recommended for performing an incremental load of such additional data. The following example shows a SQL*Loader script, with an excerpt from the sample daily data (stockinc.dat) and the SQL*Loader control file (stockinc.ctl).

The SQL*Loader script contains the following:

```
sqlldr userid=tsdev/tsdev control=stockinc.ctl
        log=stockinc.log bad=stockinc.bad errors=1000
```

The stockinc.dat sample data file includes the following:

```
ACME  02-JAN-97 100.00 101.00  99.00 100.00  1000
FUNCO 02-JAN-97  25.00  25.00  25.00  25.00  2000
SAMCO 02-JAN-97  39.00  40.00  38.00  39.50 30000
        ...
```

The stockinc.ctl file contains the following:

```
load data
infile ’stockinc.dat’
append
into table stockdemo
fields terminated by whitespace
(ticker, tstamp DATE(13) "DD-MON-YY", open, high, low, close, volume)
```

Note the following differences in the control file for incremental loading as opposed to bulk loading:

- The *conventional path* is used instead of the direct path. That is, the control file for incremental loading does *not* contain the line *options (direct=true)*.

  The conventional path is better for incremental loading because the amount of new data (daily stock information) is small relative to the total amount of data. For an explanation of conventional and direct paths, including situations in which the conventional path is necessary or preferable, see the SQL*Loader documentation in the *Oracle8i Utilities* manual.

- The APPEND keyword is specified, so that the new data is appended to the existing tabular data.

3.5 Retrofitting Existing Tables

You can use the administrative tools procedures to "retrofit" existing tables (that is, generate schema objects using existing detail, calendar, and map tables). The retrofit demo uses this approach, and the statements and comments in the retrofit.sql file reflect the approach described in this section. (The existing tables that are retrofitted are created in the tables.sql procedure, which the usage demo invokes before the retrofit.sql procedure.)

To use the administrative tools procedures to retrofit existing tables:

1. Create the time series schema, specifying that the tables already exist and using the Add_Existing_Column procedure to identify each existing column to be included in the time series schema objects. For example:

   ```
   DECLARE

   BEGIN

   -- Establish 'stockdemo_ts' as the time series group name for purposes
   -- of the administrative tools procedures.

     ORDSYS.TSTools.Begin_Create_TS_Group('stockdemo_ts','flat');

     -- Assert that the detail, map, and calendar tables exist,
     -- and define the names for these tables.
     -- Explicitly set the name of the relational view.
     -- Explicitly set the names of the timestamp and time series name
     -- columns.

     ordsys.tstools.set_flat_attributes(
           detail_table_name    => 'stockdemo',
           detail_table_exists  => 1,
           map_table_name       => 'stockdemo_metadata',
           map_table_exists     => 1,
           cal_table_name       => 'stockdemo_calendars',
           cal_table_exists     => 1,
           tstamp_colname       => 'tstamp',
           tsname_colname       => 'ticker',
           rel_view_name        => 'stockdemo_sv');

     -- Tell TSTools the names of existing time series columns
     -- (as defined for the table stockdemo)

     ORDSYS.TSTools.Add_Existing_Column('open');
     ORDSYS.TSTools.Add_Existing_Column('high');
     ORDSYS.TSTools.Add_Existing_Column('low');
     ORDSYS.TSTools.Add_Existing_Column('close');
     ORDSYS.TSTools.Add_Existing_Column('volume');

   -- End the specification of schema objects and create the objects.
   ```

```
  ORDSYS.TSTools.End_Create_TS_Group;

  exception
    when others then
       begin
          ORDSYS.TSTools.Cancel_Create_TS_Group;
          raise;
       end;

END;
/
```

2. Grant specific privileges on the views to intended users. For example:

```
-- Grant SELECT privileges on the object view.
GRANT SELECT ON stockdemo_ts TO tsuser;

-- Grant SELECT, UPDATE, DELETE privileges on the relational view.
GRANT SELECT,INSERT,UPDATE,DELETE on stockdemo_sv TO tsuser;
GRANT RESOURCE TO tsuser;
```

3.6 Validating Time Series Consistency

Choose one of the following approaches to ensuring the consistency of time series data, using the guidelines in Section 2.8.3:

- Adjust calendars to be consistent with the time series.

  Use the DeriveExceptions function in adjusting a calendar to be consistent with the time series. See Section 2.2.5 for more information about this approach.

- Validate that each time series is consistent with the calendar.

  Use the IsValidTS function to check that the time series is consistent with the calendar. See the IsValidTS function reference information in Chapter 5.

3.7 Formulating Time Series Queries

Formulating time series queries involves invoking time series or time scaling functions, or both. Example 3–2 uses the Mavg time series function to obtain 10-day moving average of the closing price for stock ACME for December 1996, and it uses the ScaleupSum time scaling function to obtain monthly trading volumes for stock ACME. (The results shown in the example reflect sample data for the Oracle8*i* Time Series usage demo.)

Example 3–2   Formulate Time Series Queries

```
SELECT to_char(tstamp) tstamp, value
FROM stockdemo_ts ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Mavg(ts.close,to_date('01-DEC-96','DD-MON-YY'),
             to_date('31-DEC-96','DD-MON-YY'),10)
       ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';

TSTAMP    VALUE
--------- ----------
02-DEC-96       74.5
03-DEC-96       75.5
04-DEC-96       76.5
05-DEC-96       77.5
06-DEC-96       78.5
09-DEC-96       79.5
10-DEC-96       80.5
11-DEC-96       81.5
12-DEC-96       82.5
13-DEC-96       83.5
16-DEC-96       84.5
17-DEC-96       85.5
18-DEC-96       86.5
19-DEC-96       87.5
20-DEC-96       88.5
23-DEC-96       89.5
24-DEC-96       90.5
26-DEC-96       91.5
27-DEC-96       92.5
30-DEC-96       93.5
31-DEC-96       94.5
21 rows selected.

SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupSum(ts.volume,
                   VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='ACME' and cal.name='Monthly';

TSTAMP    VALUE
--------- ----------
01-NOV-96      20000
```

```
01-DEC-96      21000
2 rows selected.
```

3.8 Deriving Calendar Exceptions

This section explains in greater detail the approaches to deriving calendar exceptions from time series data. These approaches were introduced in Section 2.2.5; see that section for information on concepts related to exceptions and the reasons for choosing a particular approach.

3.8.1 Deriving Exceptions Using a Time Series (Approach 1)

This approach to deriving exceptions takes a time series and optionally a date range as input parameters, using the following form of the function:

```
DeriveExceptions(inputTS ORDTNumSeriesIOTRef
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

or

```
DeriveExceptions(inputTS ORDTVarchar2SeriesIOTRef
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

The input time series (*inputTS*) has an associated calendar and data for all the desired timestamps (for example, daily closing prices for stock XYZ for all trading days during the period for the time series or the date range bounded by *startDate* and *endDate*). A calendar is returned that is defined on the same pattern and frequency as the input calendar, and the exception lists of the returned calendar are populated to be consistent with the time series data. The exception lists are updated based on finding timestamps that are in the calendar pattern or in the time series, but not in both. (A timestamp is in the calendar pattern if it is within the date range of the calendar and maps to an *on* (1) bit in the pattern.)

The returned calendar's on- and off- exceptions are populated based on the calendar pattern and the time series, as follows:

- All timestamps that are in the calendar pattern but not in the time series become off-exceptions.

For example, 04-Jul-1997 (Friday) is in the pattern of a stock trading calendar, but it is not a date on which U.S. stocks were traded.

- All timestamps that are in the time series but are not in the calendar pattern become on-exceptions.

The following example populates a calendar named Quarterly with exceptions based on the actual data in the *unemployment_rate* time series (in which data for some quarters is missing):

```
UPDATE myts_cal cal
  SET  cal =
  (SELECT ORDSYS.TimeSeries.DeriveExceptions(ts.unemployment_rate)
              FROM   myts ts
              WHERE  ts.region = '1')
  WHERE cal.name = 'Quarterly';
```

3.8.2 Deriving Exceptions Using a Calendar and Table of Dates (Approach 1A)

This approach to deriving exceptions takes a calendar and an ORDTDateTab (that is, a table of dates) as input parameters, using the following form of the function:

DeriveExceptions(cal ORDTCalendar, DateTab ORDTDateTab

[, startDate DATE [, endDate DATE]]

) RETURN ORDSYS.ORDTCalendar;

The table of dates (*DateTab* parameter) includes all dates in the time series, for example, all dates on which stock XYZ traded. A calendar is returned that is defined on the same pattern and frequency as the input calendar, and the exception lists of the returned calendar are populated to be consistent with the time series data in *DateTab*. The exception lists are updated based on finding timestamps that are in the calendar pattern or in the table of dates, but not in both. (A timestamp is in the calendar pattern if it is within the date range of the calendar and maps to an *on* (1) bit in the pattern.)

The returned calendar's on- and off- exceptions are populated based on the calendar pattern and the table of dates, as follows:

- All timestamps that are in the calendar pattern but not in the table of dates become off-exceptions.

  For example, 04-Jul-1997 (Friday) is in the pattern of a stock trading calendar, but it is not a date on which U.S. stocks were traded.

- All timestamps that are in the table of dates but are not in the calendar pattern become on-exceptions.

The following example derives the exceptions for all time series in the *stockdemo* table and updates the corresponding calendars in the *stockdemo_calendars* table:

```
UPDATE stockdemo_calendars cal
  SET cal = (SELECT ORDSYS.Calendar.DeriveExceptions(
                VALUE(cal),
                CAST(multiset(
                  SELECT s.tstamp
                  FROM stockdemo s
                  WHERE cal.name = s.ticker) AS ORDSYS.ORDTDateTab))
            FROM dual);
```

This approach (Approach 1A) to deriving calendar exceptions has the following requirements:

- The input table of dates must be sorted in ascending timestamp order before the call to the DeriveExceptions function.
- The precision of the timestamps of the dates in the table must conform to the frequency of the input calendar.

3.8.3 Deriving Exceptions Using Two Time Series Parameters (Approach 2)

This approach to deriving exceptions takes two time series references as input parameters, using the following form of the function:

```
DeriveExceptions(series1 ORDTNumSeriesIOTRef,
    series2 ORDTNumSeriesIOTRef)
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

or

```
DeriveExceptions(series1 ORDTVarchar2SeriesIOTRef,
    series2 ORDTVarchar2SeriesIOTRef)
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

This overloading of the DeriveExceptions function allows the input parameters to be time series REFs (either two ORDTNumSeriesIOTRef parameters or two ORDTVarchar2SeriesIOTRef parameters).

Before calling DeriveExceptions, you must construct a time series based on a reference calendar. This time series will contain all the timestamps within the date range (*minDate* through *maxDate*) of the calendar.

The following example builds a reference time series based on a calendar named *PATTERN-ONLY*. An INSERT statement populates the time series named *PATTERN-ONLY* with the valid timestamps between the starting and ending dates of the calendar.

```
INSERT INTO stocks(ticker,tstamp)
SELECT 'PATTERN-ONLY',
       t1.c1
       FROM
       (SELECT column_value c1 FROM the
        (SELECT CAST(ORDSYS.Calendar.TimeStampsBetween(VALUE(cal),
                                        cal.mindate,
                                        cal.maxdate)
               AS ORDSYS.ORDTDateTab)
        FROM stock_calendars cal
        WHERE cal.name = 'PATTERN-ONLY')) t1;
```

The insertion is made directly into the underlying table, not into the relational view. Using the underlying table is safe here because the time series is presumed to be correct, so the mechanisms for ensuring consistency between the time series and the calendar provided by the relational view are not needed in this case.

The *PATTERN-ONLY* calendar should have no exceptions. If this calendar has any exceptions, the resulting time series will have exception lists that are not null, which will cause the DeriveExceptions function to report an error.

After you create the reference time series, call the DeriveExceptions function with the reference time series as the first parameter (*series1*). DeriveExceptions compares the dates in *series1* with the dates in *series2*, and it returns the calendar of *series2* with the exceptions created as follows:

- All timestamps that are in *series1* but not in *series2* become off-exceptions.

  For example, if *series2* contains dates on which stock XYZ traded and 04-Jul-1997 (Friday) is not in that time series, then 04-Jul-1997 is added to the calendar as an off-exception.

- All timestamps that are in *series2* but not in *series1* become on-exceptions.

The following example uses the reference time series created in the preceding statement to update the exception lists of every other calendar in the *stockdemo_calendars* table, with the exceptions for each calendar derived from the timestamps in the associated time series. (This example assumes that each calendar maps to a time series with the same name.)

```
UPDATE stockdemo_calendars cal
 SET cal = (SELECT ORDSYS.TimeSeries.DeriveExceptions(ts1.open,ts2.open)
           FROM stockdemo_ts ts1, stockdemo_ts ts2
             WHERE ts1.ticker = 'PATTERN-ONLY' and ts2.ticker = cal.name)
WHERE cal.name <> 'PATTERN-ONLY';
```

This approach (Approach 2) to deriving calendar exceptions has the following requirements:

- The input parameters to the DeriveExceptions function must be either two ORDTNumSeriesIOTRef parameters or two ORDTVarchar2SeriesIOTRef parameters. ORDTNumSeries and ORDTVarchar2Series variants are not supported for this function.
- Calendars of the time series input parameters must have the same frequency and pattern.
- The first time series parameter (*PATTERN-ONLY* time series) must have no exceptions.
- The starting date (*minDate*) of the calendar of the second time series must be greater (later) than or equal to the starting date of the calendar of the first time series.
- The ending date (*maxDate*) of the calendar of the second time series must be less (earlier) than or equal to the ending date of the calendar of the first time series.

3.9 Using Product-Developer Functions

Product-developer functions, described in Section 2.9.2, let you modify and expand the Oracle8*i* Time Series capabilities. For example, an ISV could develop additional time series analysis functions by calling product-developer functions.

The following example shows the use of the IsValidDate, NumTstampsBetween, and OffsetDate product-developer functions in a PL/SQL implementation of the Lead function. The Lead function accepts an input time series and a *lead_date*, and returns a time series where the starting timestamp is the *lead_date*. (Note that to simplify the presentation, some error checking has been omitted.)

```
create function Lead (ts ORDSYS.ORDTNumSeries, lead_date date)
     return ORDSYS.ORDTNumSeries is
i integer;
outts ORDSYS.ORDTNumSeries; /* Temporary Storage for Result */
new_tstamp date;   /* Changeable version of lead_date */
last_lead_date date; /* Last timestamp of the output time series*/
first_tstamp date; /* First timestamp of
                      the input time series */
last_index integer; /* Last index of the input time series */
last_tstamp date;       /* Last timestamp of the input time series */
units integer;      /* Number of timestamps between input and
                       output time series */

ERR_LEAD_TSTAMP_BOUNDS constant integer := 20540;
ERR_LEAD_TSTAMP_BOUNDS_MSG constant varchar2(100) :=
         'Projected lead timestamp beyond calendar bounds';

begin
     first_tstamp :=ts.series(1).tstamp;
     last_index :=ts.series.last;
     last_tstamp :=ts.series(last_index).tstamp;

     if ORDSYS.Calendar.IsValidDate(ts.cal, lead_date) = 0 then
           Raise_Application_Error(ERR_LEAD_TSTAMP_BOUNDS,
                   ERR_LEAD_TSTAMP_BOUNDS_MSG);
     end if;

     /* units is the number of timestamps between the first timestamp of
     the input time series and lead_date. */
     units := ORDSYS.Calendar.NumTimeStampsBetween(ts.cal, first_tstamp,
              lead_date);

     last_lead_date := ORDSYS.Calendar.OffsetDate(ts.cal, last_tstamp,
                     units);
     if last_lead_date is null then
           Raise_Application_Error(ERR_LEAD_TSTAMP_BOUNDS,
                    ERR_LEAD_TSTAMP_BOUNDS_MSG);
     end if;

     /* Instantiate output time series. */
     outts := ORDSYS.ORDTNumSeries('Lead Result', ts.cal, ORDSYS.ORDTNumTab());
     outts.series.extend(last_index);

     /* Assign the first timestamp of the output time series to
     first_lead_date. Copy value from input time series to output
     time series. */
     new_tstamp := lead_date;
     outts.series(1) := ORDSYS.ORDTNumCell(new_tstamp, ts.series(1).value);

     /* Assign subsequent timestamps by calling OffsetDate with the
     previous date and an offset of 1. */
     for i in 2..outts.series.last loop
          new_tstamp := ORDSYS.Calendar.OffsetDate(ts.cal,
            outts.series(i-1).tstamp, 1);
          outts.series(i) := ORDSYS.ORDTNumCell(new_tstamp,
            ts.series(i).value);
     end loop;

     return(outts);
end;
```

For other examples of using product-developer functions, see the files for the advanced-developer demo (described briefly in Table 1–1 in Section 1.6).

4 Calendar Functions: Reference

The Oracle8*i* Time Series library consists of:

- Data types (described in Section 2.3)
- Calendar functions (described in this chapter)
- Time series functions (described in Chapter 5)
- Time scaling functions (described in Chapter 6)
- Administrative tools procedures for creating time series schema objects (described in Chapter 7)

Calendar functions are mainly used by product developers, such as ISVs, to develop new time series functions and to administer and modify calendars.

Time series and time scaling functions and the administrative tools procedures are used mainly by application developers.

Syntax notes:

- The ORDSYS schema name and the package name must be used with the function name, although public synonyms can be created to eliminate the need for specifying the schema name (see Section 1.5). Each function is included in a PL/SQL package, such as Calendar, TimeSeries, or TimeScale. The ORDSYS schema name and the package name are included in the Format and in any examples.
- Function calls are not case sensitive, except for any quoted literal values. For example, the following code line excerpts are valid and semantically identical:

```
select CAST(TimeSeries.ExtractTable(close) AS ORDTNumTab)
select cast(TIMESERIES.extracttable(close) as ordtnumtab)
select cast(TiMeSeRiEs.eXtRaCtTaBlE(ClosE) As ordtNUMtab)
```

- The syntax and examples show the reference-based interface (types ORDTNumSeriesIOTRef and ORDTVarchar2SeriesIOTRef).

CombineCals

Format

```
ORDSYS.Calendar.CombineCals(
    cal1 ORDSYS.ORDTCalendar,
    cal2 ORDSYS.ORDTCalendar,
    [startDate DATE,
    endDate DATE,]
    equalFlag OUT INTEGER
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Combines two calendars. The CombineCals function is provided primarily for use in developing functions that operate on two time series (such as the TSAdd function).

Parameters

cal1
The first calendar to be combined.

cal2
The second calendar to be combined.

startDate
Starting date for the resulting calendar. If *startDate* is not specified, the starting date is the starting date for the calendars, or the higher (later) of the starting dates if they are different.

endDate
Ending date for the resulting calendar. If *endDate* is not specified, the ending date is the ending date for the calendars, or the lower (earlier) of the ending dates if they are different.

equalFlag
Contains 1 if the input calendars are equal, and 0 if the input calendars are not equal.

Usage

If the frequencies of the two calendars are not equal, the function returns NULL.

If the aligned patterns of the two calendars are not equal, the function returns NULL.

If *startDate* is not specified, the starting date of the resulting calendar is the later of the starting dates of the two calendars, that is, resulting minDate = max(minDate1, minDate2).

If *endDate* is not specified, the ending date of the resulting calendar is the earlier of the ending dates of the two calendars, that is, resulting maxDate = min(maxDate1, maxDate2).

The function intersects the on-exception lists of the two calendars. For example, if *cal1* has 30-Mar and 29-Jun as on-exceptions and *cal2* has 29-Jun and 28-Sep as on-exceptions, the resulting calendar has only 29-Jun as an on-exception.

The function performs a union of the off-exceptions of the two calendars. For example, if *cal1* has 01-Jan and 04-Jul as off-exceptions and *cal2* has 01-Jan and 14-Jul as off-exceptions, the resulting calendar has 01-Jan, 04-Jul, and 14-Jul as off-exceptions.

CombineCals and IntersectCals differ as follows:

- CombineCals requires the frequencies and the aligned patterns of the two calendars to be equal, whereas IntersectCals requires only that the frequencies be equal. However, IntersectCals does require that the patterns be of the same length.
- CombineCals lets you specify starting and ending dates for the resulting calendar, whereas IntersectCals does not let you specify starting and ending dates.

Example

Combine two calendars (*GENERIC-CAL1* and *GENERIC-CAL2*), then intersect the two calendars:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';
```

```
DECLARE
tstCal1 ORDSYS.ORDTCalendar;
tstCal2 ORDSYS.ORDTCalendar;
resultCal ORDSYS.ORDTCalendar;
equalFlag INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select the calendars GENERIC-CAL1 into tstCal1
 -- and GENERIC-CAL2 into tstCal2
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal1
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = ’GENERIC-CAL1’;
 SELECT value(cal) INTO tstCal2
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = ’GENERIC-CAL2’;

 -- Display the calendars tstCal1 and tstCal2.
 SELECT ORDSYS.TimeSeries.Display(tstCal1) INTO dummyVal FROM dual;
 SELECT ORDSYS.TimeSeries.Display(tstCal2) INTO dummyVal FROM dual;

 -- Combine tstCal1 and tstCal2
 resultCal := ORDSYS.Calendar.CombineCals(tstCal1, tstCal2, equalFlag);
 SELECT ORDSYS.TimeSeries.Display(resultCal, ’result of CombineCals’)
 INTO dummyVal
 FROM dual;
 DBMS_OUTPUT.PUT_LINE(’equalFlag = ’ || equalFlag);

 -- Intersect tstCal1 and tstCal2
 resultCal := ORDSYS.Calendar.IntersectCals(tstCal1, tstCal2);
 SELECT ORDSYS.TimeSeries.Display(resultCal, ’result of IntersectCals’)
 INTO dummyVal
 FROM dual;

END;
/
```

This example might produce the following output:

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01-JAN-96
 MaxDate = 31-DEC-96
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 07-JAN-96
 onExceptions  :
     21-JAN-96     03-FEB-96     24-MAR-96
     27-APR-96     19-MAY-96     23-JUN-96
     07-JUL-96     04-AUG-96     15-SEP-96
 offExceptions :
     08-JAN-96     02-FEB-96     05-MAR-96
     04-APR-96     08-MAY-96     25-JUN-96
     09-JUL-96

Calendar Name = GENERIC-CAL2
 Frequency = 4 (day)
 MinDate = 01-JAN-96
 MaxDate = 31-DEC-97
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 08-JAN-96
 onExceptions  :
     07-JUL-96     04-AUG-96     15-SEP-96
     13-OCT-96     10-NOV-96     14-DEC-96
     04-JAN-97     09-FEB-97     08-MAR-97
     05-APR-97     11-MAY-97     08-JUN-97
 offExceptions :
     09-JUL-96     05-AUG-96     10-SEP-96
     23-OCT-96     19-NOV-96     12-DEC-96
     01-JAN-97     12-FEB-97     04-MAR-97
     07-APR-97     05-MAY-97     09-JUN-97

result of CombineCals :

 Frequency = 4 (day)
 MinDate = 01-JAN-96
 MaxDate = 31-DEC-96
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 07-JAN-96
 onExceptions  :
     07-JUL-96     04-AUG-96     15-SEP-96
 offExceptions :
     08-JAN-96     02-FEB-96     05-MAR-96
     04-APR-96     08-MAY-96     25-JUN-96
     09-JUL-96     05-AUG-96     10-SEP-96
     23-OCT-96     19-NOV-96     12-DEC-96
equalFlag = 0

result of IntersectCals :

 Frequency = 4 (day)
 MinDate = 01-JAN-96
 MaxDate = 31-DEC-96
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 08-JAN-96
 onExceptions  :
     07-JUL-96     04-AUG-96     15-SEP-96
 offExceptions :
     08-JAN-96     02-FEB-96     05-MAR-96
     04-APR-96     08-MAY-96     25-JUN-96
     09-JUL-96     05-AUG-96     10-SEP-96
     23-OCT-96     19-NOV-96     12-DEC-96
```

Day

Format

```
ORDSYS.Calendar.Day(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *day*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Example

Insert into the *stockdemo_calendars* table a calendar of *day* frequency with a calendar name of *Daily* and an anchor date of 01-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Day(
       ’Daily’,
       (to_date(’01-01-97’,’MM-DD-YY’))));
```

DeleteExceptions

Format

```
ORDSYS.Calendar.DeleteExceptions(
    inputCal IN ORDSYS.ORDTCalendar,
    delExcDate IN DATE
    ) RETURN ORDSYS.ORDTCalendar;
```

or

```
ORDSYS.Calendar.DeleteExceptions(
    inputCal IN ORDSYS.ORDTCalendar,
    delExcTab IN ORDSYS.ORDTDateTab
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Deletes from the specified calendar all exceptions that either match a specified date (*delExcDate*) or are included in a table of dates (*delExcTab*), and returns the resulting calendar.

Parameters

inputCal
The calendar from which one or more exceptions are to be deleted.

delExcDate
The date to be deleted from the exceptions of the calendar.

delExcTab
A table of dates to be deleted from the exceptions of the calendar.

Usage

If a date to be deleted is in either the on-exception list or off-exception list of the calendar, the function deletes the date from the appropriate list.

If *delExcDate* is not in either the on-exception list or off-exception list of the calendar, the function returns the input calendar with no changes.

For any date in *delExcTab* that is not in either the on-exception list or off-exception list of the calendar, the function ignores the date. If no date in *delExcTab* is in either the on-exception list or off-exception list of the calendar, the function returns the input calendar with no changes.

Example

Delete some exceptions from a calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDTab ORDSYS.ordtDateTab;
resultCal ORDSYS.ORDTCalendar;
dummyVal INTEGER;
relOffset INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Delete some exceptions in tstCal.
 tstDTab := ORDSYS.ORDTDateTab(
                  '01/21/1996', -- ON  Exception
                  '05/08/1996', -- OFF Exception
                  '08/04/1996', -- ON  Exception
                  '07/09/1996');-- OFF Exception
 SELECT ORDSYS.TimeSeries.Display(tstDTab, 'Input DateTab')
 INTO dummyVal
 FROM dual;
 resultCal := ORDSYS.Calendar.DeleteExceptions(tstCal, tstDTab);
 SELECT ORDSYS.TimeSeries.Display(resultCal) INTO dummyVal
 FROM dual;

END;
/
```

This example might produce the following output. The second display of information about *GENERIC-CAL1* does not include the deleted on-exceptions and off-exceptions.

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

Input DateTab :

     01/21/1996 00:00:00     05/08/1996 00:00:00     08/04/1996 00:00:00
     07/09/1996 00:00:00

Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     02/03/1996 00:00:00     03/24/1996 00:00:00     04/27/1996 00:00:00
     05/19/1996 00:00:00     06/23/1996 00:00:00     07/07/1996 00:00:00
     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     06/25/1996 00:00:00
```

DisplayValCal Procedure

Format

```
ORDSYS.Calendar.DisplayValCal(
    validFlag IN INTEGER,
    outMessage IN VARCHAR2,
    invOnExc IN ORDSYS.ORDTDateTab,
    invOffExc IN ORDSYS.ORDTDateTab,
    impOnExc IN ORDSYS.ORDTDateTab,
    impOffExc IN ORDSYS.ORDTDateTab,
    inputCal IN ORDSYS.ORDTCalendar,
    mesg IN VARCHAR2
    );
```

Description

Displays the results returned by the ValidateCal function.

Note: DisplayValCal is a procedure, not a function. Procedures do not return values.

Parameters

validFlag
The return value from the ValidateCal function call:

Value	Meaning
0	The calendar is valid. No errors were found.
1	Correctable errors were found and corrected. The resulting calendar is valid.
-1	Uncorrectable errors were found. The calendar is not valid.

outMessage
Message output by ValidateCal describing how the calendar was repaired (if the return value = 1) or why the calendar could not be repaired (if the return value = -1).

invOnExc
Table of the invalid on-exceptions found in the calendar.

invOffExc
Table of the invalid off-exceptions found in the calendar.

impOnExc
Table of the imprecise on-exceptions found in the calendar.

impOffExc
Table of the imprecise off-exceptions found in the calendar.

inputCal
The calendar returned by ValidateCal (repaired if necessary).

mesg
Optional message.

Usage

This procedure is intended to be used with the ValidateCal function. See the information on ValidateCal in this chapter.

Example

Use the IsValidCal and ValidateCal functions and the DisplayValCal procedure with an invalid calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
outMessage  varchar2(32750);
invOnExc    ORDSYS.ORDTDateTab;
invOffExc   ORDSYS.ORDTDateTab;
impOnExc    ORDSYS.ORDTDateTab;
impOffExc   ORDSYS.ORDTDateTab;
dummyval     integer;
validFlag     integer;
```

```
tstCal1     ORDSYS.ORDTCalendar :=
                  ORDSYS.ORDTCalendar(
                        0,
                        'CALENDAR MYCAL',
                        4,
                        ORDSYS.ORDTPattern(ORDSYS.ORDTPatternBits(1,1,1,1,1,0,0),
                                                        TO_DATE('01-08-1996 01:01:01')),
                        TO_DATE('01-01-1975'),
   TO_DATE('01-01-1999'),
                        ORDSYS.ORDTExceptions(
              TO_DATE('02-03-1969'), -- Date < minDate,
              TO_DATE('02-14-1969'), -- Date < minDate,
              TO_DATE('02-03-1999'), -- Date > maxDate,
              TO_DATE('02-17-1999'), -- Date > maxDate,
              TO_DATE('12-31-1995'), -- Maps to 0 in pattern (Sunday)
              TO_DATE('01-13-1996'), -- Maps to 0 in pattern (Saturday)
              TO_DATE('02-24-1996'), -- Maps to 0 in pattern (Saturday)
              TO_DATE('03-30-1996'), -- Maps to 0 in pattern (Saturday)
              TO_DATE('02-02-1996 01:01:01'), -- Imprecise
              TO_DATE('03-04-1996 01:01:01'), -- Imprecise
              TO_DATE('04-05-1996 02:02:02'), -- Imprecise
              TO_DATE('03-25-1996'), -- Valid off-exception
              TO_DATE('01-22-1996'), -- Valid, but out of sequence
              TO_DATE('02-12-1996'),
              TO_DATE('04-30-1996'),
              NULL,                  -- Null date
              TO_DATE('02-12-1996'), -- Duplicate date within OFFs
              NULL,                  -- Null date
              TO_DATE('04-30-1996'), -- Duplicate off-exception
              NULL,                  -- Null date
              TO_DATE('03-25-1996'), -- Duplicate off-exception
              TO_DATE('01-22-1996'), -- Duplicate off-exception
              TO_DATE('01-17-1996'), -- Added to on- and off-exceptions
              TO_DATE('05-28-1996'), -- Added to on- and off-exceptions
              TO_DATE('06-18-1996'), -- Added to on- and off-exceptions
              TO_DATE('04-23-1996'), -- Added to on- and off-exceptions
              TO_DATE('02-02-1996'),
              TO_DATE('03-04-1996'),
              TO_DATE('05-06-1997')),
     ORDSYS.ORDTExceptions(
              TO_DATE('02-08-1969'), -- Date < minDate,
              TO_DATE('02-15-1969'), -- Date < minDate,
              TO_DATE('02-13-1999'), -- Date > maxDate,
              TO_DATE('02-20-1999'), -- Date > maxDate,
              TO_DATE('01-03-1996'), -- Maps to 1 in pattern (Wednesday)
```

```
                 TO_DATE('02-19-1996'), -- Maps to 1 in pattern (Monday)
                 TO_DATE('03-18-1996'), -- Maps to 1 in pattern (Monday)
                 TO_DATE('05-27-1996'), -- Maps to 1 in pattern (Monday)
                 TO_DATE('03-23-1996 01:01:01'), -- Imprecise
                 TO_DATE('02-18-1996 01:01:01'), -- Imprecise
                 TO_DATE('05-26-1996 01:01:01'), -- Imprecise
                 TO_DATE('01-13-1996'), -- Valid on-exception
                 TO_DATE('01-14-1996'), -- Valid on-exception
                 NULL,                  -- Null date
                 NULL,                  -- Null date
                 TO_DATE('02-24-1996'), -- Valid on-exception
                 TO_DATE('03-23-1996'), -- Valid on-exception
                 TO_DATE('01-13-1996'), -- Duplicate on-exception
                 TO_DATE('01-14-1996'), -- Duplicate on-exception
                 TO_DATE('02-24-1996'), -- Duplicate on-exception
                 TO_DATE('03-23-1996'), -- Duplicate on-exception
                 TO_DATE('01-17-1996'), -- Added to on- and off-exceptions
                 TO_DATE('05-28-1996'), -- Added to on- and off-exceptions
                 TO_DATE('06-18-1996'), -- Added to on- and off-exceptions
                 TO_DATE('04-23-1996'), -- Added to on- and off-exceptions
                 TO_DATE('01-06-1996'), -- Valid, but out of sequence
                 TO_DATE('02-03-1996'),
                 TO_DATE('05-04-1997'))
                        );
BEGIN
   SELECT ORDSYS.TIMESERIES.Display(tstCal1, 'tstCal1') INTO dummyval
   FROM dual;
   validFlag := ORDSYS.CALENDAR.IsValidCal(tstCal1);
   IF(validFlag = 0)
   THEN
       validFlag := ORDSYS.CALENDAR.ValidateCal(
                tstCal1, outMessage, invOnExc, invOffExc, impOnExc, impOffExc
                );

       ORDSYS.TIMESERIES.DisplayValCal(
             validFlag,
             outMessage,
             invOnExc,
             invOffExc,
             impOnExc,
             impOffExc,
             tstCal1,
             'Your Message'
             );
   END IF;
```

```
END;
/
```

This example might produce the following output:

```
tstCal1 :

Calendar Name = CALENDAR MYCAL
 Frequency = 4 (day)
 MinDate = 01/01/1975 00:00:00
 MaxDate = 01/01/1999 00:00:00
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 01/08/1996 01:01:01
 onExceptions  :
     02/08/1969 00:00:00     02/15/1969 00:00:00     02/13/1999 00:00:00
     02/20/1999 00:00:00     01/03/1996 00:00:00     02/19/1996 00:00:00
     03/18/1996 00:00:00     05/27/1996 00:00:00     03/23/1996 01:01:01
     02/18/1996 01:01:01     05/26/1996 01:01:01     01/13/1996 00:00:00
     01/14/1996 00:00:00
     02/24/1996 00:00:00     03/23/1996 00:00:00     01/13/1996 00:00:00
     01/14/1996 00:00:00     02/24/1996 00:00:00     03/23/1996 00:00:00
     01/17/1996 00:00:00     05/28/1996 00:00:00     06/18/1996 00:00:00
     04/23/1996 00:00:00     01/06/1996 00:00:00     02/03/1996 00:00:00
     05/04/1997 00:00:00
 offExceptions :
     02/03/1969 00:00:00     02/14/1969 00:00:00     02/03/1999 00:00:00
     02/17/1999 00:00:00     12/31/1995 00:00:00     01/13/1996 00:00:00
     02/24/1996 00:00:00     03/30/1996 00:00:00     02/02/1996 01:01:01
     03/04/1996 01:01:01     04/05/1996 02:02:02     03/25/1996 00:00:00
     01/22/1996 00:00:00     02/12/1996 00:00:00     04/30/1996 00:00:00
          02/12/1996 00:00:00
     04/30/1996 00:00:00          03/25/1996 00:00:00
     01/22/1996 00:00:00     01/17/1996 00:00:00     05/28/1996 00:00:00
     06/18/1996 00:00:00     04/23/1996 00:00:00     02/02/1996 00:00:00
     03/04/1996 00:00:00     05/06/1997 00:00:00

DisplayValCal Your Message:

TS-WRN: the input calendar has rectifiable errors. See the message for details

message output by validateCal:

TS-WRN: fixed precision of the pattern anchor date
TS-WRN: removed superfluous dates in the on exception list (refer invalidOnExc)
```

```
TS-WRN: fixed imprecise dates in the on exception list (refer impreciseOnExc)
TS-WRN: removed null dates in the on exception list
TS-WRN: sorted the on exceptions list
TS-WRN: removed duplicate dates in the on exceptions list
TS-WRN: removed superfluous dates in off exceptions list (refer invalidOffExc)
TS-WRN: fixed imprecise dates in the off exception list (refer impreciseOffExc)
TS-WRN: removed null dates in the off exception list
TS-WRN: sorted the off exceptions list
TS-WRN: removed duplicate dates in the off exceptions list
TS-WRN: the on exceptions list was trimmed between calendar minDate & maxDate
TS-WRN: the off exceptions list was trimmed between calendar minDate & maxDate

list of invalid on exceptions :

     01/03/1996 00:00:00     02/19/1996 00:00:00     03/18/1996 00:00:00
     05/27/1996 00:00:00     01/17/1996 00:00:00     05/28/1996 00:00:00
     06/18/1996 00:00:00     04/23/1996 00:00:00

list of invalid off exceptions :

     12/31/1995 00:00:00     01/13/1996 00:00:00     02/24/1996 00:00:00
     03/30/1996 00:00:00

list of imprecise on exceptions :

     03/23/1996 01:01:01     02/18/1996 01:01:01     05/26/1996 01:01:01

list of imprecise off exceptions :

     02/02/1996 01:01:01     03/04/1996 01:01:01     04/05/1996 02:02:02

the validated calendar :

Calendar Name = CALENDAR MYCAL
 Frequency = 4 (day)
 MinDate = 01/01/1975 00:00:00
 MaxDate = 01/01/1999 00:00:00
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 01/08/1996 00:00:00
 onExceptions  :
     01/06/1996 00:00:00     01/13/1996 00:00:00     01/14/1996 00:00:00
     02/03/1996 00:00:00     02/18/1996 00:00:00     02/24/1996 00:00:00
     03/23/1996 00:00:00     05/26/1996 00:00:00     05/04/1997 00:00:00
 offExceptions :
```

```
01/17/1996 00:00:00     01/22/1996 00:00:00     02/02/1996 00:00:00
02/12/1996 00:00:00     03/04/1996 00:00:00     03/25/1996 00:00:00
04/05/1996 00:00:00     04/23/1996 00:00:00     04/30/1996 00:00:00
05/28/1996 00:00:00     06/18/1996 00:00:00     05/06/1997 00:00:00
```

EqualCals

Format

```
ORDSYS.Calendar.EqualCals(
    cal1 ORDSYS.ORDTCalendar,
    cal2 ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    ) RETURN BINARY_INTEGER;
```

Description

Checks if two calendars (completely or within a specified date range) are equal.

Parameters

cal1
The first calendar to be checked.

cal2
The second calendar to be checked.

startDate
Starting date for the checking. If *startDate* is not specified, the starting date is the starting date for the calendars, or the higher (later) of the starting dates if they are different.

endDate
Ending date for the checking. If *endDate* is not specified, the ending date is the ending date for the calendars, or the lower (earlier) of the ending dates if they are different.

Usage

The function checks if the frequencies, off-exceptions, on-exceptions, and aligned patterns are the same for the two calendars. If they are all the same, the function returns 1; if they are not all the same, the function returns 0.

The function does not require the calendars to have the same starting and ending dates.

Example

Check if two calendars (*GENERIC-CAL1* and *GENERIC-CAL2*) are equal:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal1 ORDSYS.ORDTCalendar;
tstCal2 ORDSYS.ORDTCalendar;
resultCal ORDSYS.ORDTCalendar;
equalFlag INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select the calendars GENERIC-CAL1 into tstCal1
 -- and GENERIC-CAL2 into tstCal2
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal1
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';
 SELECT value(cal) INTO tstCal2
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL2';

 -- Display the calendars tstCal1 and tstCal2.
 SELECT ORDSYS.TimeSeries.Display(tstCal1) INTO dummyVal FROM dual;
 SELECT ORDSYS.TimeSeries.Display(tstCal2) INTO dummyVal FROM dual;

 -- Compare tstCal1 and tstCal2 for equality.
 DBMS_OUTPUT.NEW_LINE;
 equalFlag := ORDSYS.Calendar.EqualCals(tstCal1, tstCal2);
 DBMS_OUTPUT.PUT_LINE('EqualCals(GENERIC-CAL1, GENERIC-CAL2) = ' || equalFlag);

END;
/
```

This example might display the following output. In this example, the returned value of 0 indicates that the calendars are not equal.

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

Calendar Name = GENERIC-CAL2
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1997 00:00:00
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 01/08/1996 00:00:00
 onExceptions  :
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
     10/13/1996 00:00:00     11/10/1996 00:00:00     12/14/1996 00:00:00
     01/04/1997 00:00:00     02/09/1997 00:00:00     03/08/1997 00:00:00
     04/05/1997 00:00:00     05/11/1997 00:00:00     06/08/1997 00:00:00
 offExceptions :
     07/09/1996 00:00:00     08/05/1996 00:00:00     09/10/1996 00:00:00
     10/23/1996 00:00:00     11/19/1996 00:00:00     12/12/1996 00:00:00
     01/01/1997 00:00:00     02/12/1997 00:00:00     03/04/1997 00:00:00
     04/07/1997 00:00:00     05/05/1997 00:00:00     06/09/1997 00:00:00

EqualCals(GENERIC-CAL1, GENERIC-CAL2) = 0
```

GenDateRangeTab

Format

```
ORDSYS.Calendar.GenDateRangeTab(
    inputCal ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTDateRangeTab;
```

Description

Given an input calendar, returns a table of date ranges that represent all of the valid intervals in the calendar (or from *startDate* through *endDate*).

Parameters

inputCal
The input calendar.

startDate
Starting date for returning date ranges. If *startDate* is not specified, the starting date is the starting date for the calendar (*minDate*).

endDate
Ending date for returning date ranges. The returned ending date is actually the first valid timestamp after *endDate*. If *endDate* is not specified, the ending date is the ending date for the calendar (*maxDate*).

Usage

The function can be used to perform time scaling against any table with a DATE column. It is used in a TABLE construct in the FROM clause of a SQL statement, and it generates a table of intervals based on *inputCal*. By joining the output of this function with a table containing a DATE column, you can use GROUP BY semantics to aggregate by the generated intervals.

For example, if you specify a monthly calendar starting on 01-Jan-1999 and ending on 31-Mar-1999, with standard U.S. holidays (including 01-Jan), the function returns the following timestamps:

02-Jan-1999	01-Feb-1999
01-Feb-1999	01-Mar-1999
01-Mar-1999	01-Apr-1999

The scope of the date ranges returned is adjusted, if necessary, as follows:

- The first date range reflects the first whole date range interval that it can cover after *startDate*.
- The last date range reflects the full date range that includes *endDate*.

For example, assume a monthly calendar with a '1' pattern (no off days), no exceptions, and starting on the first day of the month. If *startDate* is 15-Jan-1999 and *endDate* is 15-Dec-1999, the returned date ranges are from February through December of 1999.

For best performance, especially with large data sets, always follow these guidelines when constructing a date range to be joined with time series data:

- Perform the date generation before specifying other tables for the join operation.
- Use the /*+ ORDERED */ optimizer hint to ensure that the TABLE clause is the innermost table.
- Index the fields used in the table for the join operation.

If the calendar does not include date bounds (a *minDate* and *maxDate*), you must specify *startDate* and *endDate*. (The date range table cannot be infinite.)

If *startDate* is greater (later) than *endDate*, an exception is raised.

Examples

Create a date range table of 10-day cycles (using the *10-day* frequency, described in Table 2–2 in Section 2.2.1) for 1990 through 1993:

```
SELECT to_char(t.startdate,'DAY'),
       to_char(t.startdate,'DD-MON-YYYY HH24:MI:SS'),
       to_char(t.enddate,'DAY'),
       to_char(t.enddate,  'DD-MON-YYYY HH24:MI:SS')
```

```
   FROM TABLE(cast ( ORDSYS.Calendar.GenDateRangeTab(
              ORDSYS.ORDTCalendar(
                0,
                '10-Day',
                10,
                ORDSYS.ORDTPattern(
                  ORDSYS.ORDTPatternBits(1),
                    TO_DATE('01-JAN-1998','DD-MON-YYYY')),
                TO_DATE('01-JAN-1990','DD-MON-YYYY'),
                TO_DATE('31-DEC-1993','DD-MON-YYYY'),
                ORDSYS.ORDTExceptions(),
                ORDSYS.ORDTExceptions()
              )) as ORDSYS.ORDTDateRangeTab)) t;
```

This example might display the following output:

```
TO_CHAR(T TO_CHAR(T.STARTDATE, TO_CHAR(T TO_CHAR(T.ENDDATE,'D
--------- -------------------- --------- --------------------
MONDAY    01-JAN-1990 00:00:00 THURSDAY  11-JAN-1990 00:00:00
THURSDAY  11-JAN-1990 00:00:00 SUNDAY    21-JAN-1990 00:00:00
SUNDAY    21-JAN-1990 00:00:00 THURSDAY  01-FEB-1990 00:00:00
THURSDAY  01-FEB-1990 00:00:00 SUNDAY    11-FEB-1990 00:00:00
SUNDAY    11-FEB-1990 00:00:00 WEDNESDAY 21-FEB-1990 00:00:00
WEDNESDAY 21-FEB-1990 00:00:00 THURSDAY  01-MAR-1990 00:00:00
  ...          ...                 ...          ...
WEDNESDAY 01-DEC-1993 00:00:00 SATURDAY  11-DEC-1993 00:00:00
SATURDAY  11-DEC-1993 00:00:00 TUESDAY   21-DEC-1993 00:00:00
TUESDAY   21-DEC-1993 00:00:00 SATURDAY  01-JAN-1994 00:00:00
144 rows selected.
```

Return the count and the minimum, maximum, and average values of closing prices (for all stock tickers, not broken down by ticker) from the *tsquick_tab* table for 01-Oct-1996 through 31-Dec-1996, using a weekly business-day calendar generated by the GenDateRangeTab function:

```
select /*+ ORDERED */ to_char(t.startdate,'DAY') "day",
           to_char(t.startdate,'DD-MON-YYYY HH24:MI:SS') "tstamp",
           count(s.close) "count",
           min(s.close) "min",
           max(s.close) "max",
           avg(s.close) "avg"
    from TABLE(cast ( ORDSYS.Calendar.GenDateRangeTab(
 ORDSYS.ORDTCalendar(
                 0,
                 'BusinessWeek',
                 4,
                 ORDSYS.ORDTPattern(
                          ORDSYS.ORDTPatternBits(0,5,0),
                          TO_DATE('15-DEC-1996','DD-MON-YYYY')),
                 TO_DATE('01-OCT-1996','DD-MON-YYYY'),
                 TO_DATE('31-DEC-1996','DD-MON-YYYY'),
                 ORDSYS.ORDTExceptions(),
                 ORDSYS.ORDTExceptions()
                 )) as ORDSYS.ORDTDateRangeTab)) t,
          tsquick_tab s
    where s.tstamp >= t.startdate and s.tstamp < t.enddate
    group by t.startdate
    order by t.startdate;
```

Note that this example follows the guidelines in the Usage section for this function, including the use of the /*+ ORDERED */ optimizer hint.

This example might produce the following output:

```
day       tstamp              count      min        max        avg
--------- ------------------- ---------- ---------- ---------- ----------
MONDAY    28-OCT-1996 00:00:00          6      23.69     79.688 63.7818333
MONDAY    04-NOV-1996 00:00:00         20      23.72      83.25   52.64925
MONDAY    11-NOV-1996 00:00:00         20      23.84     85.813    53.5503
MONDAY    18-NOV-1996 00:00:00         20      23.82     88.938    55.2897
MONDAY    25-NOV-1996 00:00:00         15      23.71      88.75 54.5533333
MONDAY    02-DEC-1996 00:00:00         20      23.75     89.875    57.8124
MONDAY    09-DEC-1996 00:00:00         20       23.4     94.375   60.12525
MONDAY    16-DEC-1996 00:00:00         19      23.36     95.875 59.6052632
MONDAY    23-DEC-1996 00:00:00         15      23.93         97 61.1606667
MONDAY    30-DEC-1996 00:00:00          8      24.11         99  63.951875
10 rows selected.
```

GetIntervalEnd

Format

ORDSYS.TimeSeries.GetIntervalEnd(

inputCal IN ORDSYS.ORDTCalendar,

inputDate IN DATE

) RETURN DATE;

Description

Given a Calendar and an input timestamp (*inputDate*), returns the end of the interval that includes the input timestamp.

Parameters

inputCal
The input calendar.

inputDate
Timestamp for which the end of the interval is to be returned.

Usage

If *inputDate* is a valid timestamp, the function returns a date. Otherwise, the function returns a null.

An exception is returned if *inputCal* is null.

Example

Return the end of the interval for several timestamps:

```
DECLARE
inputCal ORDSYS.ORDTCalendar;
tstDate  DATE;
retDate  DATE;
tstDtTab ordsys.ordtdatetab;
BEGIN

  -- Select a Calendar into a local variable
```

```
     SELECT value(cal)
     INTO   inputCal
     FROM   TSDEV.stockdemo_calendars cal
     WHERE  cal.name = 'BIWEEKLY';

  -- Display the input Calendar
     ORDSYS.TimeSeries.Display(inputCal);

     DBMS_OUTPUT.PUT_LINE('');

  -- GetIntervalEnd of a Valid timestamp
     tstDate  := TO_DATE('01-JAN-1996','DD-MON-YYYY');
     retDate  := ORDSYS.Calendar.GetIntervalEnd(inputCal, tstDate);

     DBMS_OUTPUT.PUT_LINE('GetIntervalEnd (' ||
                          TO_CHAR(tstDate, 'MM-DD-YYYY')  ||
                                           ') = '              ||
                          TO_CHAR(retDate, 'MM-DD-YYYY')  );

  -- GetIntervalEnd of an InValid timestamp - returns NULL
     tstDate  := TO_DATE('01-JUL-1996','DD-MON-YYYY');
     retDate  := ORDSYS.Calendar.GetIntervalEnd(inputCal, tstDate);

     DBMS_OUTPUT.PUT_LINE('GetIntervalEnd (' ||
                          TO_CHAR(tstDate, 'MM-DD-YYYY')  ||
                                           ') = '              ||
                          TO_CHAR(retDate, 'MM-DD-YYYY')  );

  -- GetIntervalEnd of a Covered timestamp
     tstDate  := TO_DATE('08-JAN-1996','DD-MON-YYYY');
     retDate  := ORDSYS.Calendar.GetIntervalEnd(inputCal, tstDate);

     DBMS_OUTPUT.PUT_LINE('GetIntervalEnd (' ||
                          TO_CHAR(tstDate, 'MM-DD-YYYY')  ||
                                           ') = '              ||
                          TO_CHAR(retDate, 'MM-DD-YYYY')  );

END;
/
```

This example might produce the following output:

```
Calendar Name = BIWEEKLY
 Frequency = 5 (week)
 MinDate is NULL
 MaxDate is NULL
 patBits: 2
 patAnchor = 01/01/1996 00:00:00
 onExceptions  :
 offExceptions :
     07/01/1996 00:00:00

GetIntervalEnd (01-01-1996) = 01-15-1996
GetIntervalEnd (07-01-1996) =
GetIntervalEnd (01-08-1996) = 01-15-1996
```

GetIntervalStart

Format

ORDSYS.TimeSeries.GetIntervalStart(

inputCal IN ORDSYS.ORDTCalendar,

inputDate IN DATE

) RETURN DATE;

Description

Given a Calendar and an input timestamp (*inputDate*), returns the start of the interval that includes the input timestamp.

Parameters

inputCal
The input calendar.

inputDate
Timestamp for which the start of the interval is to be returned.

Usage

If *inputDate* is a valid timestamp, the function returns a date. Otherwise, the function returns a null.

An exception is returned if *inputCal* is null.

Example

Return the start of the interval for several timestamps:

```
DECLARE
inputCal ORDSYS.ORDTCalendar;
tstDate  DATE;
retDate  DATE;
tstDtTab ordsys.ordtdatetab;
BEGIN

  -- Select a Calendar into a local variable
```

```
      SELECT value(cal)
      INTO   inputCal
      FROM   TSDEV.stockdemo_calendars cal
      WHERE  cal.name = 'BIWEEKLY';

   -- Display the input Calendar
      ORDSYS.TimeSeries.Display(inputCal);

      DBMS_OUTPUT.PUT_LINE('');

   -- GetIntervalStart of a Valid timestamp
      tstDate  := TO_DATE('01-JAN-1996','DD-MON-YYYY');
      retDate  := ORDSYS.Calendar.GetIntervalStart(inputCal, tstDate);

      DBMS_OUTPUT.PUT_LINE('GetIntervalStart (' ||
                           TO_CHAR(tstDate, 'MM-DD-YYYY')  ||
                                                   ') = '              ||
                           TO_CHAR(retDate, 'MM-DD-YYYY')  );

   -- GetIntervalStart of an InValid timestamp - returns NULL
      tstDate  := TO_DATE('01-JUL-1996','DD-MON-YYYY');
      retDate  := ORDSYS.Calendar.GetIntervalStart(inputCal, tstDate);

      DBMS_OUTPUT.PUT_LINE('GetIntervalStart (' ||
                           TO_CHAR(tstDate, 'MM-DD-YYYY')  ||
                                                   ') = '              ||
                           TO_CHAR(retDate, 'MM-DD-YYYY')  );

   -- GetIntervalStart of a Covered timestamp
      tstDate  := TO_DATE('08-JAN-1996','DD-MON-YYYY');
      retDate  := ORDSYS.Calendar.GetIntervalStart(inputCal, tstDate);

      DBMS_OUTPUT.PUT_LINE('GetIntervalStart (' ||
                           TO_CHAR(tstDate, 'MM-DD-YYYY')  ||
                                                   ') = '              ||
                           TO_CHAR(retDate, 'MM-DD-YYYY')  );

END;
/
```

This example might produce the following output:

```
Calendar Name = BIWEEKLY
 Frequency = 5 (week)
 MinDate is NULL
 MaxDate is NULL
 patBits: 2
 patAnchor = 01/01/1996 00:00:00
 onExceptions  :
 offExceptions :
     07/01/1996 00:00:00

GetIntervalStart (01-01-1996) = 01-01-1996
GetIntervalStart (07-01-1996) =
GetIntervalStart (01-08-1996) = 01-01-1996
```

GetOffset

Format

```
ORDSYS.TimeSeries.GetOffset(
    inputCal IN ORDSYS.ORDTCalendar,
    origin_date IN DATE,
    reference_date IN DATE
    ) RETURN INTEGER;
```

Description

Given a calendar, one date (*origin_date*), and another date (*reference_date*), returns the number of timestamps that the second date is offset from the first.

Parameters

inputCal
The input calendar.

origin_date
Date from which the offset is to be computed.

reference_date
Date whose offset from *origin_date* is to be returned.

Usage

The function considers the frequency, pattern, and exceptions of the calendar.

The returned integer is positive if *reference_date* is one or more timestamps in the future with respect to *origin_date*, and negative if it is in the past with respect to *origin_date*. For example, assume that the calendar includes Mondays through Fridays, that 04-Jul-1997 (Friday) is an off-exception, and that 03-Jul-1997 (Thursday) is the *origin_date*. If 10-Jul-1997 (Thursday) is the *reference_date*, the returned offset is 4; if the *reference_date* is 01-Jul-1997 (Monday), the returned offset is -2.

If *origin_date* and *reference_date* are the same, the function returns 0 (zero).

An exception is returned if the calendar has an empty or null pattern.

Example

Return the offset of 05-Jun-1996 from 04-Mar-1996 in the *GENERIC-CAL1* calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';
DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
tstDate2 DATE;
result INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Get offset of 05-JUN-1996 from 04-MAR-1996.
 tstDate1 := TO_DATE('04/03/1996');
 tstDate2 := TO_DATE('06/05/1996');
 result  := ORDSYS.Calendar.GetOffset(tstCal,tstDate1, tstDate2);
 DBMS_OUTPUT.PUT_LINE('GetOffset(' || tstDate1 ||' , ' || tstDate2
                                                  || ') = ' || result);
END;
/
```

This example might produce the following output. In this example, 05-Jun-1996 is 45 timestamps later than 04-Mar-1996.

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

GetOffset(04/03/1996 00:00:00 , 06/05/1996 00:00:00) = 45
```

Hour

Format

```
ORDSYS.Calendar.Hour(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *hour*, a pattern of ’1’ (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Example

Insert into the *stockdemo_calendars* table a calendar of *hour* frequency with a calendar name of *Hourly* and an anchor date of 01-Jan-1997 (at midnight). The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Hour(
       'Hourly',
       (to_date('01-01-97 01','MM-DD-YY HH'))));
```

InsertExceptions

Format

```
ORDSYS.Calendar.InsertExceptions(
    inputCal IN ORDSYS.ORDTCalendar,
    newExcDate IN DATE
    ) RETURN ORDSYS.ORDTCalendar;
```

or

```
ORDSYS.Calendar.InsertExceptions(
    inputCal IN ORDSYS.ORDTCalendar,
    newExcTab IN ORDSYS.ORDTDateTab
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Inserts into the specified calendar all exceptions that either match a specified date (*newExcDate*) or are included in a table of dates (*newExcTab*), and returns the resulting calendar.

Parameters

inputCal
The calendar into which one or more exceptions are to be inserted.

newExcDate
The date to be inserted as an exception in the calendar.

newExcTab
A table of dates to be inserted as exceptions in the calendar.

Usage

For each date to be inserted, the function inserts it in the appropriate list (off-exceptions or on-exceptions), according to the frequency and pattern of the calendar.

If a date to be inserted is already an exception in the calendar, the function ignores the request to insert the date.

If *newExcDate* or *newExcTab* is empty or null, or if all dates to be inserted already exist in the calendar as exceptions, the function returns the input calendar with no changes.

Example

Insert some exceptions into a calendar.

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDTab ORDSYS.ordtDateTab;
resultCal ORDSYS.ORDTCalendar;
dummyVal INTEGER;
relOffset INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Populate tstDTab with some on- and off-exceptions.
 tstDTab := ORDSYS.ORDTDateTab(
                 '02/10/1996', -- ON  Exception
                 '07/09/1996', -- OFF Exception
                 '03/17/1996', -- ON  Exception
                 '04/08/1996');-- OFF Exception
 SELECT ORDSYS.TimeSeries.Display(tstDTab, 'Input DateTab')
 INTO dummyVal
 FROM dual;
 -- Insert some exceptions in tstCal.
 resultCal := ORDSYS.Calendar.InsertExceptions(tstCal, tstDTab);
 SELECT ORDSYS.TimeSeries.Display(resultCal) INTO dummyVal
 FROM dual;

END;
/
```

This example might produce the following output. The second display of information about *GENERIC-CAL1* includes the added on-exceptions and off-exceptions.

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00


Input DateTab :

     02/10/1996 00:00:00     07/09/1996 00:00:00     03/17/1996 00:00:00
     04/08/1996 00:00:00

Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     02/10/1996 00:00:00
     03/17/1996 00:00:00     03/24/1996 00:00:00     04/27/1996 00:00:00
     05/19/1996 00:00:00     06/23/1996 00:00:00     07/07/1996 00:00:00
```

```
    08/04/1996 00:00:00     09/15/1996 00:00:00
offExceptions :
    01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
    04/04/1996 00:00:00     04/08/1996 00:00:00     05/08/1996 00:00:00
    06/25/1996 00:00:00     07/09/1996 00:00:00
```

IntersectCals

Format

```
ORDSYS.Calendar.IntersectCals(
    cal1 ORDSYS.ORDTCalendar,
    cal2 ORDSYS.ORDTCalendar
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Returns the intersection of two calendars.

Parameters

cal1
The first calendar to be intersected.

cal2
The second calendar to be intersected.

Usage

The function performs an intersection of the two input calendars, as follows:

- The starting date of the resulting calendar is the later of the starting dates of the two calendars, that is, resulting minDate = max(minDate1, minDate2).
- The ending date of the resulting calendar is the earlier of the ending dates of the two calendars, that is, resulting maxDate = min(maxDate1, maxDate2).
- The intersection of the aligned patterns is computed. For example, if both calendars have a *day* frequency with Sunday as the first day, and if *cal1* has a pattern of '0,1,1,1,1,1,0' and *cal2* has a pattern of '0,0,1,1,1,1,1', the resulting pattern is '0,0,1,1,1,1,0' (that is, the calendar includes only Tuesdays, Wednesdays, Thursdays, and Fridays).
- The intersection of the on-exception lists of the two calendars is computed. For example, if *cal1* has 30-Mar and 29-Jun as on-exceptions and *cal2* has 29-Jun and 28-Sep as on-exceptions, the resulting calendar has only 29-Jun as an on-exception.

- The union of the off-exceptions of the two calendars is computed. For example, if *cal1* has 01-Jan and 04-Jul as off-exceptions and *cal2* has 01-Jan and 14-Jul as off-exceptions, the resulting calendar has 01-Jan, 04-Jul, and 14-Jul as off-exceptions.

If the frequencies of the two calendars are not equal, the function returns NULL.

Contrast this function with UnionCals, which performs a union of two calendars.

IntersectCals and CombineCals differ as follows:

- CombineCals requires the frequencies and the aligned patterns of the two calendars to be equal, whereas IntersectCals requires only that the frequencies be equal. However, IntersectCals does require that the patterns be of the same length.
- CombineCals lets you specify starting and ending dates for the resulting calendar, whereas IntersectCals does not let you specify starting and ending dates.

Example

Combine two calendars (*GENERIC-CAL1* and *GENERIC-CAL2*), then intersect the two calendars:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal1 ORDSYS.ORDTCalendar;
tstCal2 ORDSYS.ORDTCalendar;
resultCal ORDSYS.ORDTCalendar;
equalFlag INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select the calendars GENERIC-CAL1 into tstCal1
 -- and GENERIC-CAL2 into tstCal2
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal1
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';
 SELECT value(cal) INTO tstCal2
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL2';

 -- Display the calendars tstCal1 and tstCal2.
 SELECT ORDSYS.TimeSeries.Display(tstCal1) INTO dummyVal FROM dual;
 SELECT ORDSYS.TimeSeries.Display(tstCal2) INTO dummyVal FROM dual;

 -- Combine tstCal1 and tstCal2.
 resultCal := ORDSYS.Calendar.CombineCals(tstCal1, tstCal2, equalFlag);
 SELECT ORDSYS.TimeSeries.Display(resultCal, 'result of CombineCals')
 INTO dummyVal
 FROM dual;
 DBMS_OUTPUT.PUT_LINE('equalFlag = ' || equalFlag);

 -- Intersect tstCal1 and tstCal2.
 resultCal := ORDSYS.Calendar.IntersectCals(tstCal1, tstCal2);
 SELECT ORDSYS.TimeSeries.Display(resultCal, 'result of IntersectCals')
 INTO dummyVal
 FROM dual;

END;
/
```

This example might produce the following output:

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

Calendar Name = GENERIC-CAL2
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1997 00:00:00
 patBits:
         1,1,1,1,1,0,0
 patAnchor = 01/08/1996 00:00:00
 onExceptions  :
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
     10/13/1996 00:00:00     11/10/1996 00:00:00     12/14/1996 00:00:00
     01/04/1997 00:00:00     02/09/1997 00:00:00     03/08/1997 00:00:00
     04/05/1997 00:00:00     05/11/1997 00:00:00     06/08/1997 00:00:00
 offExceptions :
     07/09/1996 00:00:00     08/05/1996 00:00:00     09/10/1996 00:00:00
     10/23/1996 00:00:00     11/19/1996 00:00:00     12/12/1996 00:00:00
     01/01/1997 00:00:00     02/12/1997 00:00:00     03/04/1997 00:00:00
     04/07/1997 00:00:00     05/05/1997 00:00:00     06/09/1997 00:00:00

result of CombineCals :

 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
         0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00     08/05/1996 00:00:00     09/10/1996 00:00:00
     10/23/1996 00:00:00     11/19/1996 00:00:00     12/12/1996 00:00:00
equalFlag = 0

result of IntersectCals :

 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
         1,1,1,1,1,0,0
 patAnchor = 01/08/1996 00:00:00
 onExceptions  :
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00     08/05/1996 00:00:00     09/10/1996 00:00:00
     10/23/1996 00:00:00     11/19/1996 00:00:00     12/12/1996 00:00:00
```

InvalidTimeStampsBetween

Format

```
ORDSYS.Calendar.InvalidTimeStampsBetween(
    inputCal IN ORDSYS.ORDTCalendar,
    startDate IN DATE,
    endDate IN DATE
    ) RETURN ORDSYS.ORDTDateTab;
```

Description

Given starting and ending input timestamps, returns a table (ORDTDateTab) containing the invalid timestamps within that range according to the specified calendar.

Parameters

inputCal
The calendar to be used to determine whether a timestamp is valid or invalid.

startDate
Starting date in the range to be checked for invalid timestamps.

endDate
Ending date in the range to be checked for invalid timestamps.

Usage

A timestamp is invalid if one or more of the following conditions are true:

- It is outside the date range of the calendar.
- It is an off-exception in the calendar.
- It is imprecise (for example, a timestamp of 02-Jul-1997 if the calendar frequency is *month*).
- It is null.

startDate and *endDate* are included in the check for invalid timestamps.

If there are no invalid timestamps in the date range, the function returns an empty ORDTDateTab.

If *startDate* is greater (later) than *endDate*, an exception is raised.

Contrast this function with TimeStampsBetween, which returns a table containing the valid timestamps in a date range.

Example

Return a table of invalid timestamps between 03-Mar-1996 and 03-Jun-1996 in the *GENERIC-CAL1* calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
tstDate2 DATE;
resultDTab ORDSYS.ordtDateTab;
dummyVal INTEGER;
relOffset INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Get all the invalid timestamps between 03-MAR-1996 and 03-JUN-1996.
 tstDate1 := TO_DATE('03/03/1996');
 tstDate2 := TO_DATE('06/03/1996');
 resultDTab := ORDSYS.Calendar.InvalidTimeStampsBetween
                                    (tstCal, tstDate1, tstDate2);
 SELECT ORDSYS.TimeSeries.Display(resultDTab, 'InValid timestamps')
 INTO dummyVal
 FROM dual;
```

```
END;
/
```

This example might produce the following output:

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

InValid timestamps :

     03/03/1996 00:00:00     03/05/1996 00:00:00     03/09/1996 00:00:00
     03/10/1996 00:00:00     03/16/1996 00:00:00     03/17/1996 00:00:00
     03/23/1996 00:00:00     03/30/1996 00:00:00     03/31/1996 00:00:00
     04/04/1996 00:00:00     04/06/1996 00:00:00     04/07/1996 00:00:00
     04/13/1996 00:00:00     04/14/1996 00:00:00     04/20/1996 00:00:00
     04/21/1996 00:00:00     04/28/1996 00:00:00     05/04/1996 00:00:00
     05/05/1996 00:00:00     05/08/1996 00:00:00     05/11/1996 00:00:00
     05/12/1996 00:00:00     05/18/1996 00:00:00     05/25/1996 00:00:00
     05/26/1996 00:00:00     06/01/1996 00:00:00     06/02/1996 00:00:00
```

IsValidCal

Format

```
ORDSYS.Calendar.IsValidCal(
    inputCal IN ORDSYS.ORDTCalendar
    ) RETURN BINARY_INTEGER
```

Description

Returns 1 if a calendar is valid and 0 if a calendar is not valid.

Parameters

inputCal
The calendar to be checked for validity.

Usage

A calendar is invalid (not valid) if it contains any errors. This function does not correct any errors or perform any repair operations on the calendar.

Contrast this function with the ValidateCal function, which checks the validity of the calendar and repairs any correctable errors. For detailed information on calendar errors, see the information on ValidateCal in this chapter.

If the IsValidCal function returns 0, you should do the following before you attempt to use the calendar:

1. Use the ValidateCal function to repair any correctable errors.
2. If there are any errors that ValidateCal cannot correct, correct these errors yourself.
3. Repeat steps 1 and 2 as often as necessary until the resulting calendar is valid.

Example

Use the IsValidCal and ValidateCal functions and the DisplayValCal procedure with an invalid calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
```

```
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
outMessage  varchar2(32750);
invOnExc    ORDSYS.ORDTDateTab;
invOffExc   ORDSYS.ORDTDateTab;
impOnExc    ORDSYS.ORDTDateTab;
impOffExc   ORDSYS.ORDTDateTab;
dummyval     integer;
validFlag     integer;
tstCal1     ORDSYS.ORDTCalendar :=
               ORDSYS.ORDTCalendar(
                  0,
                  'CALENDAR MYCAL',
                  4,
                  ORDSYS.ORDTPattern(ORDSYS.ORDTPatternBits(1,1,1,1,1,0,0),
                                            TO_DATE('01-08-1996 01:01:01')),
                  TO_DATE('01-01-1975'),
  TO_DATE('01-01-1999'),
                  ORDSYS.ORDTExceptions(
           TO_DATE('02-03-1969'), -- Date < minDate,
           TO_DATE('02-14-1969'), -- Date < minDate,
           TO_DATE('02-03-1999'), -- Date > maxDate,
           TO_DATE('02-17-1999'), -- Date > maxDate,
           TO_DATE('12-31-1995'), -- Maps to 0 in pattern (Sunday)
           TO_DATE('01-13-1996'), -- Maps to 0 in pattern (Saturday)
           TO_DATE('02-24-1996'), -- Maps to 0 in pattern (Saturday)
           TO_DATE('03-30-1996'), -- Maps to 0 in pattern (Saturday)
           TO_DATE('02-02-1996 01:01:01'), -- Imprecise
           TO_DATE('03-04-1996 01:01:01'), -- Imprecise
           TO_DATE('04-05-1996 02:02:02'), -- Imprecise
           TO_DATE('03-25-1996'), -- Valid off-exception
           TO_DATE('01-22-1996'), -- Valid, but out of sequence
           TO_DATE('02-12-1996'),
           TO_DATE('04-30-1996'),
           NULL,                  -- Null date
           TO_DATE('02-12-1996'), -- Duplicate date within OFFs
           NULL,                  -- Null date
           TO_DATE('04-30-1996'), -- Duplicate off-exception
           NULL,                  -- Null date
           TO_DATE('03-25-1996'), -- Duplicate off-exception
           TO_DATE('01-22-1996'), -- Duplicate off-exception
           TO_DATE('01-17-1996'), -- Added to on- and off-exceptions
           TO_DATE('05-28-1996'), -- Added to on- and off-exceptions
           TO_DATE('06-18-1996'), -- Added to on- and off-exceptions
```

```
               TO_DATE('04-23-1996'), -- Added to on- and off-exceptions
               TO_DATE('02-02-1996'),
               TO_DATE('03-04-1996'),
               TO_DATE('05-06-1997')),
        ORDSYS.ORDTExceptions(
               TO_DATE('02-08-1969'), -- Date < minDate,
               TO_DATE('02-15-1969'), -- Date < minDate,
               TO_DATE('02-13-1999'), -- Date > maxDate,
               TO_DATE('02-20-1999'), -- Date > maxDate,
               TO_DATE('01-03-1996'), -- Maps to 1 in pattern (Wednesday)
               TO_DATE('02-19-1996'), -- Maps to 1 in pattern (Monday)
               TO_DATE('03-18-1996'), -- Maps to 1 in pattern (Monday)
               TO_DATE('05-27-1996'), -- Maps to 1 in pattern (Monday)
               TO_DATE('03-23-1996 01:01:01'), -- Imprecise
               TO_DATE('02-18-1996 01:01:01'), -- Imprecise
               TO_DATE('05-26-1996 01:01:01'), -- Imprecise
               TO_DATE('01-13-1996'), -- Valid on-exception
               TO_DATE('01-14-1996'), -- Valid on-exception
               NULL,                  -- Null date
               NULL,                  -- Null date
               TO_DATE('02-24-1996'), -- Valid on-exception
               TO_DATE('03-23-1996'), -- Valid on-exception
               TO_DATE('01-13-1996'), -- Duplicate on-exception
               TO_DATE('01-14-1996'), -- Duplicate on-exception
               TO_DATE('02-24-1996'), -- Duplicate on-exception
               TO_DATE('03-23-1996'), -- Duplicate on-exception
               TO_DATE('01-17-1996'), -- Added to on- and off-exceptions
               TO_DATE('05-28-1996'), -- Added to on- and off-exceptions
               TO_DATE('06-18-1996'), -- Added to on- and off-exceptions
               TO_DATE('04-23-1996'), -- Added to on- and off-exceptions
               TO_DATE('01-06-1996'), -- Valid, but out of sequence
               TO_DATE('02-03-1996'),
               TO_DATE('05-04-1997'))
                      );
BEGIN
   SELECT ORDSYS.TIMESERIES.Display(tstCal1, 'tstCal1') INTO dummyval
   FROM dual;
   validFlag := ORDSYS.CALENDAR.IsValidCal(tstCal1);
   IF(validFlag = 0)
   THEN
       validFlag := ORDSYS.CALENDAR.ValidateCal(
                tstCal1, outMessage, invOnExc, invOffExc, impOnExc, impOffExc
                );

       ORDSYS.TIMESERIES.DisplayValCal(
              validFlag,
              outMessage,
              invOnExc,
              invOffExc,
              impOnExc,
              impOffExc,
              tstCal1,
              ’Your Message’
              );
   END IF;
END;
/
```

This example might produce the following output:

```
tstCal1 :

Calendar Name = CALENDAR MYCAL
 Frequency = 4 (day)
 MinDate = 01/01/1975 00:00:00
 MaxDate = 01/01/1999 00:00:00
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 01/08/1996 01:01:01
 onExceptions  :
     02/08/1969 00:00:00     02/15/1969 00:00:00     02/13/1999 00:00:00
     02/20/1999 00:00:00     01/03/1996 00:00:00     02/19/1996 00:00:00
     03/18/1996 00:00:00     05/27/1996 00:00:00     03/23/1996 01:01:01
     02/18/1996 01:01:01     05/26/1996 01:01:01     01/13/1996 00:00:00
     01/14/1996 00:00:00
     02/24/1996 00:00:00     03/23/1996 00:00:00     01/13/1996 00:00:00
     01/14/1996 00:00:00     02/24/1996 00:00:00     03/23/1996 00:00:00
     01/17/1996 00:00:00     05/28/1996 00:00:00     06/18/1996 00:00:00
     04/23/1996 00:00:00     01/06/1996 00:00:00     02/03/1996 00:00:00
     05/04/1997 00:00:00
 offExceptions :
     02/03/1969 00:00:00     02/14/1969 00:00:00     02/03/1999 00:00:00
     02/17/1999 00:00:00     12/31/1995 00:00:00     01/13/1996 00:00:00
     02/24/1996 00:00:00     03/30/1996 00:00:00     02/02/1996 01:01:01
     03/04/1996 01:01:01     04/05/1996 02:02:02     03/25/1996 00:00:00
     01/22/1996 00:00:00     02/12/1996 00:00:00     04/30/1996 00:00:00
          02/12/1996 00:00:00
     04/30/1996 00:00:00          03/25/1996 00:00:00
     01/22/1996 00:00:00     01/17/1996 00:00:00     05/28/1996 00:00:00
     06/18/1996 00:00:00     04/23/1996 00:00:00     02/02/1996 00:00:00
     03/04/1996 00:00:00     05/06/1997 00:00:00

DisplayValCal Your Message:

TS-WRN: the input calendar has rectifiable errors. See the message for details

message output by validateCal:

TS-WRN: fixed precision of the pattern anchor date
TS-WRN: removed superfluous dates in the on exception list (refer invalidOnExc)
TS-WRN: fixed imprecise dates in the on exception list (refer impreciseOnExc)
TS-WRN: removed null dates in the on exception list
TS-WRN: sorted the on exceptions list
TS-WRN: removed duplicate dates in the on exceptions list
TS-WRN: removed superfluous dates in off exceptions list (refer invalidOffExc)
TS-WRN: fixed imprecise dates in the off exception list (refer impreciseOffExc)
TS-WRN: removed null dates in the off exception list
TS-WRN: sorted the off exceptions list
TS-WRN: removed duplicate dates in the off exceptions list
TS-WRN: the on exceptions list was trimmed between calendar minDate & maxDate
TS-WRN: the off exceptions list was trimmed between calendar minDate & maxDate

list of invalid on exceptions :

     01/03/1996 00:00:00     02/19/1996 00:00:00     03/18/1996 00:00:00
     05/27/1996 00:00:00     01/17/1996 00:00:00     05/28/1996 00:00:00
     06/18/1996 00:00:00     04/23/1996 00:00:00

list of invalid off exceptions :

     12/31/1995 00:00:00     01/13/1996 00:00:00     02/24/1996 00:00:00
     03/30/1996 00:00:00

list of imprecise on exceptions :

     03/23/1996 01:01:01     02/18/1996 01:01:01     05/26/1996 01:01:01

list of imprecise off exceptions :

     02/02/1996 01:01:01     03/04/1996 01:01:01     04/05/1996 02:02:02

the validated calendar :

Calendar Name = CALENDAR MYCAL
 Frequency = 4 (day)
```

```
MinDate = 01/01/1975 00:00:00
MaxDate = 01/01/1999 00:00:00
patBits:
         1,1,1,1,1,0,0
patAnchor = 01/08/1996 00:00:00
onExceptions  :
    01/06/1996 00:00:00     01/13/1996 00:00:00     01/14/1996 00:00:00
    02/03/1996 00:00:00     02/18/1996 00:00:00     02/24/1996 00:00:00
    03/23/1996 00:00:00     05/26/1996 00:00:00     05/04/1997 00:00:00
offExceptions :
    01/17/1996 00:00:00     01/22/1996 00:00:00     02/02/1996 00:00:00
    02/12/1996 00:00:00     03/04/1996 00:00:00     03/25/1996 00:00:00
    04/05/1996 00:00:00     04/23/1996 00:00:00     04/30/1996 00:00:00
    05/28/1996 00:00:00     06/18/1996 00:00:00     05/06/1997 00:00:00
```

IsValidDate

Format

```
ORDSYS.Calendar.IsValidDate(
    inputCal IN ORDSYS.ORDTCalendar,
    checkDate IN DATE
    ) RETURN BINARY_INTEGER;
```

Description

Checks whether an input date is valid or invalid according to the specified calendar.

Parameters

inputCal
The calendar to be used to determine whether the input timestamp is valid or invalid.

checkDate
The timestamp to be checked for validity according to the calendar.

Usage

If *checkDate* is valid, the function returns 1; if *checkDate* is invalid, the function returns 0.

A timestamp is invalid if one or more of the following conditions are true:

- It is outside the date range of the calendar.
- It is an off-exception in the calendar.
- It is not sufficiently precise (for example, a timestamp of 01-Jul-1997 if the calendar has a frequency of year).
- It is null.

Example

Check if 02-Jan-1996 is a valid timestamp for a calendar (*GENERIC-CAL1*):

```
CONNECT TSUSER/TSUSER
```

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';
DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
result INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Verify if 02-JAN-1996 (a Monday) is a valid date and display the result.
 tstDate1 := TO_DATE('01/02/1996');
 result  := ORDSYS.Calendar.IsValidDate(tstCal,tstDate1);
 DBMS_OUTPUT.PUT_LINE('IsValidDate(' || tstDate1 || ') = ' || result);

END;
/
```

This example might produce the following output. In this example, the returned value of 1 indicates that 02-Jan-1996 is a valid timestamp for the *BUSINESS-96* calendar.

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
```

```
          04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
          07/09/1996 00:00:00

     IsValidDate(01/02/1996 00:00:00) = 1
```

Minute

Format

```
ORDSYS.Calendar.Minute(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *minute*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Example

Insert into the *stockdemo_calendars* table a calendar of *minute* frequency with a calendar name of *Minute* and an anchor date of 01-Jan-1997 (at midnight). The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Minute(
       'Minute',
       (to_date('01-01-97','MM-DD-YY'))));
```

Month

Format

```
ORDSYS.Calendar.Month(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *month*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Examples

Insert into the *stockdemo_calendars* table a calendar of *month* frequency with a calendar name of *Monthly* and an anchor date of 01-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Month(
       'Monthly',
       (to_date('01-01-97','MM-DD-YY'))));
```

Return the sum of the daily trade volume for stock SAMCO for each month in the entire time series. For scaling, use a monthly calendar with a null name, an anchor date of 01-Jan-2001 (the default), no date boundaries (*minDate* or *maxDate*), and no exceptions. This example generates a calendar within the statement, and thus eliminates the need to specify a stored calendar that has the desired frequency.

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupSum(ts.volume,
                   ORDSYS.Calendar.Month())
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
11/01/96    10207000
12/01/96     3719450
2 rows selected.
```

NumInvalidTimeStampsBetween

Format

```
ORDSYS.Calendar.NumInvalidTimeStampsBetween(
    inputCal IN ORDSYS.ORDTCalendar,
    startDate IN DATE,
    endDate IN DATE
    ) RETURN INTEGER;
```

Description

Given starting and ending input timestamps, returns the number of invalid timestamps within that range according to the specified calendar.

Parameters

inputCal
The calendar to be used to determine whether a timestamp is valid or invalid.

startDate
Starting date in the range to be checked for invalid timestamps.

endDate
Ending date in the range to be checked for invalid timestamps.

Usage

A timestamp is invalid if one or more of the following conditions are true:

- It is outside the date range of the calendar.
- It is an off-exception in the calendar.
- It is not sufficiently precise (for example, a timestamp of 01-Jul-1997 if the calendar has a frequency of year).
- It is null.

startDate and *endDate* are included in the check for invalid timestamps.

If there are no invalid timestamps in the date range, the function returns 0 (zero).

If *startDate* is greater (later) than *endDate*, an exception is raised.

Contrast this function with NumTimeStampsBetween, which returns the number of valid timestamps in a date range.

Example

Return the number of invalid timestamps between 03-Feb-1996 and 16-May-1996 in the *GENERIC-CAL1* calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
tstDate2 DATE;
result INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Get the number of invalid timestamps between 03-FEB-1996 and 16-MAY-1996.
 tstDate1 := TO_DATE('02/03/1996');
 tstDate2 := TO_DATE('05/16/1996');
 result  := ORDSYS.Calendar.NumInvalidTimeStampsBetween(
                                                tstCal,tstDate1, tstDate2);
 DBMS_OUTPUT.PUT_LINE('NumInvalidTimeStampsBetween(' || tstDate1 ||' , ' ||
                                               tstDate2|| ') = ' || result);
END;
/
```

This example might produce the following output. In this example, there are 30 invalid timestamps in the specified date range.

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

NumInvalidTimeStampsBetween(02/03/1996 00:00:00 , 05/16/1996 00:00:00) = 30
```

NumOffExceptions

Format

```
ORDSYS.Calendar.NumOffExceptions(
    inputCal  IN ORDSYS.ORDTCalendar,
    startDate IN DATE,
    endDate IN DATE
    ) RETURN INTEGER;
```

Description

Given starting and ending input timestamps, returns the number of off-exceptions within that range according to the specified calendar.

Parameters

inputCal
The calendar to be used in computing the number of off-exceptions.

startDate
Starting date in the range to be checked for off-exceptions.

endDate
Ending date in the range to be checked for off-exceptions.

Usage

startDate and *endDate* are included in the check for off-exceptions. (For an explanation of off-exceptions and on-exceptions, see Section 2.2.)

If *startDate* is greater (later) than *endDate*, an exception is raised.

Example

Return the number of off-exceptions between 02-Feb-1996 and 07-Jul-1996 in the *GENERIC-CAL1* calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
```

```
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
tstDate2 DATE;
result INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Get the number of off-exceptions between 02-FEB-1996 and 07-JUL-1996.
 tstDate1 := TO_DATE('02/02/1996');
 tstDate2 := TO_DATE('07/07/1996');
 result  := ORDSYS.Calendar.NumOffExceptions(tstCal,tstDate1, tstDate2);
 DBMS_OUTPUT.PUT_LINE('NumOffExceptions(' || tstDate1 ||' , ' || tstDate2
                                                  || ') = ' || result);
END;
/
```

This example might produce the following output. As the last line of the output indicates, there are five off-exceptions in the specified date range (02-Feb-1996 through 07-Jul-1996).

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

NumOffExceptions(02/02/1996 00:00:00 , 07/07/1996 00:00:00) = 5
```

NumOnExceptions

Format

```
ORDSYS.Calendar.NumOnExceptions(
    inputCal  IN ORDSYS.ORDTCalendar,
    startDate IN DATE,
    endDate IN DATE
    ) RETURN INTEGER;
```

Description

Given starting and ending input timestamps, returns the number of on-exceptions within that range according to the specified calendar.

Parameters

inputCal
The calendar to be used in computing the number of on-exceptions.

startDate
Starting date in the range to be checked for on-exceptions.

endDate
Ending date in the range to be checked for on-exceptions.

Usage

startDate and *endDate* are included in the check for on-exceptions. (For an explanation of off-exceptions and on-exceptions, see Section 2.2.)

If *startDate is* greater (later) than *endDate*, an exception is raised.

Example

Return the number of on-exceptions between 02-Feb-1996 and 07-Jul-1996 in the *GENERIC-CAL1* calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';
```

```
DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
tstDate2 DATE;
result INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = ’GENERIC-CAL1’;

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Get the number of ON Exceptions between 02-FEB-1996 and 07-JUL-1996.
 tstDate1 := TO_DATE(’02/02/1996’);
 tstDate2 := TO_DATE(’07/07/1996’);
 result  := ORDSYS.Calendar.NumOnExceptions(tstCal,tstDate1, tstDate2);
 DBMS_OUTPUT.PUT_LINE(’NumOnExceptions(’ || tstDate1 ||’ , ’ || tstDate2
                                                   || ’) = ’ || result);
END;
/
```

This example might produce the following output. As the last line of the output indicates, there are six on-exceptions in the specified date range (02-Feb-1996 through 07-Jul-1996).

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
```

```
          01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
          04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
          07/09/1996 00:00:00

     NumOnExceptions(02/02/1996 00:00:00 , 07/07/1996 00:00:00) = 6
```

NumTimeStampsBetween

Format

```
ORDSYS.Calendar.NumTimeStampsBetween(
    inputCal  IN ORDSYS.ORDTCalendar,
    startDate IN DATE,
    endDate IN DATE
    ) RETURN INTEGER;
```

Description

Given starting and ending input timestamps, returns the number of valid timestamps within that range according to the specified calendar.

Parameters

inputCal
The calendar to be used to determine whether a timestamp is valid or invalid.

startDate
Starting date in the range to be checked for invalid timestamps.

endDate
Ending date in the range to be checked for invalid timestamps.

Usage

A timestamp is invalid if one or more of the following conditions are true:

- It is outside the date range of the calendar.
- It is an off-exception in the calendar.
- It is not sufficiently precise (for example, a timestamp of 01-Jul-1997 if the calendar has a frequency of year).
- It is null.

startDate and *endDate* are included in the check for valid timestamps.

If there are no valid timestamps in the date range, the function returns 0 (zero).

If *startDate* is greater (later) than *endDate*, an exception is raised.

Contrast this function with NumInvalidTimeStampsBetween, which returns the number of invalid timestamps in a date range.

Example

Return the number of valid timestamps between 03-Feb-1996 and 16-May-1996 in the *GENERIC-CAL1* calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
tstDate2 DATE;
result INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Get the number of Valid timestamps between 03-FEB-1996 and 16-MAY-1996.
 tstDate1 := TO_DATE('02/03/1996');
 tstDate2 := TO_DATE('05/16/1996');
 result  := ORDSYS.Calendar.NumTimeStampsBetween(tstCal,tstDate1, tstDate2);
 DBMS_OUTPUT.PUT_LINE('NumTimeStampsBetween(' || tstDate1 ||' , ' || tstDate2
                                                 || ') = ' || result);
END;
/
```

This example might produce the following output. In this example, there are 74 valid timestamps in the specified date range.

```
Calendar Name = GENERIC-CAL1
```

```
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

NumTimeStampsBetween(02/03/1996 00:00:00 , 05/16/1996 00:00:00) = 74
```

OffsetDate

Format

```
ORDSYS.Calendar.OffsetDate(
    inputCal  IN ORDSYS.ORDTCalendar,
    origin IN DATE,
    relOffset IN INTEGER
    ) RETURN DATE;
```

Description

Given a reference date (*origin*) and an offset with respect to the origin (*relOffset*), returns the timestamp corresponding to the offset input.

Parameters

inputCal
Calendar from which the date is to be returned.

origin
The date to which the offset value (relOffset) is to be applied in computing the returned date.

relOffset
The relative offset of the returned date with respect to the origin.

Usage

The function returns the date of the timestamp at the *relOffset* number of timestamps from the *origin* date. If *relOffset* is positive, the returned date is later than *origin*; if *relOffset* is negative, the returned date is earlier than *origin*. If *relOffset* is zero (0), the returned date is *origin* if *origin* is a valid date; however, if *relOffset* is zero (0) and *origin* is not a valid date, the function returns NULL.

For example, assume a Monday through Friday business day calendar for 1997 with 04-Jul-1997 (Friday) defined as an off-exception, and assume that *origin* is 02-Jul-1997 (Wednesday):

- If *relOffset* = 2, the returned date is 07-Jul-1997 (Monday).

- If *relOffset* = -2, the returned date is 30-Jun-1997 (Monday).
- If *relOffset* = 0, the returned date is 02-Jul-1997 (Wednesday).

If the *origin* date is not in the calendar (*inputCal*), the next later date is used if *relOffset* is positive or zero, and the next earlier date is used if *relOffset* is negative. Using the calendar in the preceding example, if *origin* is specified as 04-Jul-1997 and if *relOffset* = 2, then 07-Jul-1997 (Monday, the next business day) is used as *origin*, and the returned date is 09-Jul-1997 (Wednesday).

If the calendar pattern is empty or null, an exception is raised.

Example

Get the dates 20 timestamps later and 20 timestamps earlier than 03-Mar-1996 in the GENERIC-CAL1 calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
resultDate date;
dummyVal INTEGER;
relOffset INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Offset 03-MAR-1996 by 20.
 tstDate1 := TO_DATE('03/03/1996');
 relOffset := 20;
 resultDate := ORDSYS.Calendar.OffsetDate(tstCal, tstDate1, relOffset);
 DBMS_OUTPUT.PUT_LINE('OffsetDate(' || tstDate1 || ' , ' || relOffset
                                                 || ') = ' || resultDate);
 DBMS_OUTPUT.NEW_LINE;

 -- Offset 03-MAR-1996 by -20.
 tstDate1 := TO_DATE('03/03/1996');
 relOffset := -20;
 resultDate := ORDSYS.Calendar.OffsetDate(tstCal, tstDate1, relOffset);
 DBMS_OUTPUT.PUT_LINE('OffsetDate(' || tstDate1 || ' , ' || relOffset
                                                || ') = ' || resultDate);
 DBMS_OUTPUT.NEW_LINE;

END;
/
```

This example might produce the following output. In this example, 29-Mar-1996 is 20 timestamps later than 03-Mar-1996, and 05-Feb-1996 is 20 timestamps earlier than 03-Mar-1996.

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 12/31/1996 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
     01/21/1996 00:00:00     02/03/1996 00:00:00     03/24/1996 00:00:00
     04/27/1996 00:00:00     05/19/1996 00:00:00     06/23/1996 00:00:00
     07/07/1996 00:00:00     08/04/1996 00:00:00     09/15/1996 00:00:00
 offExceptions :
     01/08/1996 00:00:00     02/02/1996 00:00:00     03/05/1996 00:00:00
     04/04/1996 00:00:00     05/08/1996 00:00:00     06/25/1996 00:00:00
     07/09/1996 00:00:00

OffsetDate(03/03/1996 00:00:00 , 20) = 03/29/1996 00:00:00

OffsetDate(03/03/1996 00:00:00 , -20) = 02/05/1996 00:00:00
```

Quarter

Format

```
ORDSYS.Calendar.Quarter(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *quarter*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Example

Insert into the *stockdemo_calendars* table a calendar of *quarter* frequency with a calendar name of *Quarterly* and an anchor date of 01-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Quarter(
       'Quarterly',
       (to_date('01-01-97','MM-DD-YY'))));
```

Second

Format

```
ORDSYS.Calendar.Second(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *second*, a pattern of ’1’ (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Example

Insert into the *stockdemo_calendars* table a calendar of *second* frequency with a calendar name of *Second* and an anchor date of 01-Jan-1997 (at midnight). The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Second(
       'Second',
       (to_date('01-01-97','MM-DD-YY'))));
```

Semi_annual

Format

```
ORDSYS.Calendar.Semi_annual(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *semi_annual*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Example

Insert into the *stockdemo_calendars* table a calendar of *semi_annual* frequency with a calendar name of *Semi_annual* and an anchor date of 01-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Semi_annual(
       ’Semi_annual’,
       (to_date(’01-01-97’,’MM-DD-YY’))));
```

Semi_monthly

Format

```
ORDSYS.Calendar.Semi_monthly(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *semi_monthly*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. Must be the 1st or 16th day of a month. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Examples

Insert into the *stockdemo_calendars* table a calendar of *semi_monthly* frequency with a calendar name of *Semi_monthly* and an anchor date of 01-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Semi_monthly(
       'Semi_monthly',
       (to_date('01-01-97','MM-DD-YY'))));
```

Return the sum of the daily trade volume for stock SAMCO for each semimonthly period in the entire time series. For scaling, use a semimonthly calendar with a null name, an anchor date of 01-Jan-2001 (the default), no date boundaries (*minDate* or *maxDate*), and no exceptions.

```
SELECT * FROM THE
  (SELECT CAST(ORDSYS.TimeSeries.ExtractTable(
                ORDSYS.TimeScale.ScaleupSum(
                  ts.volume,
                  ORDSYS.Calendar.Semi_monthly() ))
             AS ORDSYS.ORDTNumTab)
       FROM TSDEV.stockdemo_ts ts
       WHERE ts.ticker='SAMCO');
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
11/01/96     6403150
11/16/96     3803850
12/01/96     1894200
12/16/96     1825250
4 rows selected.
```

SetPrecision

Format

```
ORDSYS.Calendar.SetPrecision(
    cal ORDSYS.ORDTCalendar
    timestamp IN DATE,
    ) RETURN DATE;
```

Description

Given a calendar and a timestamp, returns a timestamp that reflects the level of precision implied by the frequency of the specified calendar.

Parameters

cal
Calendar whose frequency is to be applied in setting the precision.

timestamp
Timestamp whose precision is to be set.

Usage

The returned timestamp reflects the precision implied by the frequency, as explained in Section 2.2.2. For example, if the input timestamp is 29-Dec-1997 12:45:00 and the frequency is 6 (*month*), the returned timestamp is 01-Dec-1997 00:00:00. Table 4–1 shows the frequencies, their precision conventions, and the resulting precision if an input timestamp of 19-Sep-1997 09:09:09 is supplied.

Table 4–1 SetPrecision and Timestamp of 19-Sep-1997 09:09:09

Frequency (Every:)	Precision Convention	Result
second	MM-DD-YYYY HH24:MI:SS	09-19-1997 09:09:09
minute	MM-DD-YYYY HH24:MI:00	09-19-1997 09:09:00

Table 4–1 SetPrecision and Timestamp of 19-Sep-1997 09:09:09 (Cont.)

Frequency (Every:)	Precision Convention	Result
hour	MM-DD-YYYY HH24:00:00	09-19-1997 09:00:00
day	MM-DD-YYYY 00:00:00 (midnight)	09-19-1997 00:00:00
month	MM-01-YYYY 00:00:00 (midnight of first day of month)	09-01-1997 00:00:00
year	01-01-YYYY 00:00:00 (midnight of first day of year)	01-01-1997 00:00:00

If the frequency is not valid, an exception is raised.

Note: The Release 8.0.4 SetPrecision syntax specifying a timestamp and a frequency (timestamp IN INTEGER, frequency IN INTEGER) is still supported, but will not be supported in a future release.

Example

Set the precision of an imprecise timestamp (here, a timestamp containing hour, minute, and second values where the calendar has a *day* frequency):

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
resultDate date;
dummyVal INTEGER;
relOffset INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';
 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

-- Set the precision of an imprecise date.
 tstDate1 := TO_DATE('03/03/1996 01:01:01');
 resultDate := ORDSYS.Calendar.SetPrecision(tstcal, tstDate1);
 DBMS_OUTPUT.PUT_LINE('SetPrecision with timestamp ' ||
                         TO_CHAR(tstDate1) ||
                         ' and frequency ' || tstCal.frequency);
 DBMS_OUTPUT.PUT_LINE(' returns: ' || TO_CHAR(resultDate) );
END;
/
```

This example might produce the following output. In this example, the hour, minute, and second components of the timestamp are set to zeroes because the calendar frequency is 4 (*day*).

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/96 00:00:00
 MaxDate = 12/31/96 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/96 00:00:00
 onExceptions  :
     01/21/96 00:00:00     02/03/96 00:00:00     03/24/96 00:00:00
     04/27/96 00:00:00     05/19/96 00:00:00     06/23/96 00:00:00
     07/07/96 00:00:00     08/04/96 00:00:00     09/15/96 00:00:00
 offExceptions :
     01/08/96 00:00:00     02/02/96 00:00:00     03/05/96 00:00:00
     04/04/96 00:00:00     05/08/96 00:00:00     06/25/96 00:00:00
     07/09/96 00:00:00

SetPrecision with timestamp 03/03/96 01:01:01 and frequency 4
 returns: 03/03/96 00:00:00
```

Ten_day

Format

```
ORDSYS.Calendar.Ten_day(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *10-day*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. Must be the 1st, 11th, or 21st day of a month. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Examples

Insert into the *stockdemo_calendars* table a calendar of *10-day* frequency with a calendar name of *Ten_day* and an anchor date of 01-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Ten_day(
       'Ten_day',
       (to_date('01-01-97','MM-DD-YY'))));
```

Return the sum of the daily trade volume for stock SAMCO for each 10-day period in the entire time series. For scaling, use a 10-day calendar with a null name, an anchor date of 01-Jan-2001 (the default), no date boundaries (*minDate* or *maxDate*), and no exceptions.

```
SELECT * FROM THE
  (SELECT CAST(ORDSYS.TimeSeries.ExtractTable(
                 ORDSYS.TimeScale.ScaleupSum(
                   ts.volume,
                   ORDSYS.Calendar.Ten_day() ))
              AS ORDSYS.ORDTNumTab)
       FROM TSDEV.stockdemo_ts ts
       WHERE ts.ticker='SAMCO');
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
11/01/96      361600
11/11/96     7281200
11/21/96     2564200
12/01/96     1433850
12/11/96     1437800
12/21/96      847800
6 rows selected.
```

TimeStampsBetween

Format

```
ORDSYS.Calendar.TimeStampsBetween(
    inputCal IN ORDSYS.ORDTCalendar,
    startDate IN DATE,
    endDate IN DATE
    ) RETURN ORDSYS.ORDTDateTab;
```

Description

Given starting and ending input timestamps, returns a table (ORDTDateTab) containing the valid timestamps within that range according to the specified calendar.

Parameters

inputCal
The calendar to be used to determine whether a timestamp is valid or invalid.

startDate
Starting date in the range to be checked for valid timestamps.

endDate
Ending date in the range to be checked for valid timestamps.

Usage

A timestamp is invalid if one or more of the following conditions are true:

- It is outside the date range of the calendar.
- It is an off-exception in the calendar.
- It is not sufficiently precise (for example, a timestamp of 01-Jul-1997 if the calendar has a frequency of year).
- It is null.

startDate and *endDate* are included in the check for valid timestamps.

If there are no valid timestamps in the date range, the function returns an empty ORDTDateTab.

If *startDate* is greater (later) than *endDate*, an exception is raised.

Contrast this function with InvalidTimeStampsBetween, which returns a table containing the invalid timestamps in a date range.

Example

Return a table of valid timestamps between 03-Mar-1996 and 03-Jun-1996 in the *GENERIC-CAL1* calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
tstDate1 DATE;
tstDate2 DATE;
resultDTab ORDSYS.ordtDateTab;
dummyVal INTEGER;
relOffset INTEGER;

BEGIN

 -- Select a calendar (say, GENERIC-CAL1) into tstCal
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';

 -- Display the calendar.
 SELECT ORDSYS.TimeSeries.Display(tstCal) INTO dummyVal FROM dual;
 DBMS_OUTPUT.NEW_LINE;

 -- Get all the valid timestamps between 03-MAR-1996 and 03-JUN-1996.
 tstDate1 := TO_DATE('03/03/1996');
 tstDate2 := TO_DATE('06/03/1996');
 resultDTab := ORDSYS.Calendar.TimeStampsBetween(tstCal, tstDate1, tstDate2);
 SELECT ORDSYS.TimeSeries.Display(resultDTab, 'Valid timestamps')
 INTO dummyVal
 FROM dual;

END;
```

```
/
```

This example might produce the following output:

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/96 00:00:00
 MaxDate = 12/31/96 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/96 00:00:00
 onExceptions  :
     01/21/96 00:00:00     02/03/96 00:00:00     03/24/96 00:00:00
     04/27/96 00:00:00     05/19/96 00:00:00     06/23/96 00:00:00
     07/07/96 00:00:00     08/04/96 00:00:00     09/15/96 00:00:00
 offExceptions :
     01/08/96 00:00:00     02/02/96 00:00:00     03/05/96 00:00:00
     04/04/96 00:00:00     05/08/96 00:00:00     06/25/96 00:00:00
     07/09/96 00:00:00


Valid timestamps :

     03/04/96 00:00:00     03/06/96 00:00:00     03/07/96 00:00:00
     03/08/96 00:00:00     03/11/96 00:00:00     03/12/96 00:00:00
     03/13/96 00:00:00     03/14/96 00:00:00     03/15/96 00:00:00
     03/18/96 00:00:00     03/19/96 00:00:00     03/20/96 00:00:00
     03/21/96 00:00:00     03/22/96 00:00:00     03/24/96 00:00:00
     03/25/96 00:00:00     03/26/96 00:00:00     03/27/96 00:00:00
     03/28/96 00:00:00     03/29/96 00:00:00     04/01/96 00:00:00
     04/02/96 00:00:00     04/03/96 00:00:00     04/05/96 00:00:00
     04/08/96 00:00:00     04/09/96 00:00:00     04/10/96 00:00:00
     04/11/96 00:00:00     04/12/96 00:00:00     04/15/96 00:00:00
     04/16/96 00:00:00     04/17/96 00:00:00     04/18/96 00:00:00
     04/19/96 00:00:00     04/22/96 00:00:00     04/23/96 00:00:00
     04/24/96 00:00:00     04/25/96 00:00:00     04/26/96 00:00:00
     04/27/96 00:00:00     04/29/96 00:00:00     04/30/96 00:00:00
     05/01/96 00:00:00     05/02/96 00:00:00     05/03/96 00:00:00
     05/06/96 00:00:00     05/07/96 00:00:00     05/09/96 00:00:00
     05/10/96 00:00:00     05/13/96 00:00:00     05/14/96 00:00:00
     05/15/96 00:00:00     05/16/96 00:00:00     05/17/96 00:00:00
     05/19/96 00:00:00     05/20/96 00:00:00     05/21/96 00:00:00
     05/22/96 00:00:00     05/23/96 00:00:00     05/24/96 00:00:00
     05/27/96 00:00:00     05/28/96 00:00:00     05/29/96 00:00:00
     05/30/96 00:00:00     05/31/96 00:00:00     06/03/96 00:00:00
```

Section 3.8.3 contains an example showing the use of TimeStampsBetween to create a time series for use with the DeriveExceptions function.

UnionCals

Format

```
ORDSYS.Calendar.UnionCals(
    cal1 ORDSYS.ORDTCalendar,
    cal2 ORDSYS.ORDTCalendar
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Returns a calendar that is the union of two input calendars.

Parameters

cal1
The first calendar on which the union operation is to be performed.

cal2
The second calendar on which the union operation is to be performed.

Usage

The function performs a union of the two input calendars, as follows:

- The starting date of the resulting calendar is the later of the starting dates of the two calendars, that is, resulting minDate = max(minDate1, minDate2).
- The ending date of the resulting calendar is the earlier of the ending dates of the two calendars, that is, resulting maxDate = min(maxDate1, maxDate2).
- The union of the aligned patterns is computed. For example, if both calendars have a *day* frequency with Sunday as the first day, and if *cal1* has a pattern of '0,1,1,1,1,1,0' and *cal2* has a pattern of '0,0,1,1,1,1,1', the resulting pattern is '0,1,1,1,1,1,1' (that is, the calendar includes Mondays through Saturdays).
- The union of the on-exception lists is computed. For example, if *cal1* has 30-Mar and 29-Jun as on-exceptions and *cal2* has 29-Jun and 28-Sep as on-exceptions, the resulting calendar has 30-Mar, 29-Jun, and 28-Sep on-exceptions.

- The intersection of the off-exception lists is computed. For example, if *cal1* has 01-Jan and 04-Jul as off-exceptions and *cal2* has 01-Jan and 14-Jul as off-exceptions, the resulting calendar has only 01-Jan as an off-exception.

If the frequencies of the two calendars are not equal, the function returns NULL.

Contrast this function with IntersectCals, which intersects two calendars.

Example

Perform a union of two calendars:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal1 ORDSYS.ORDTCalendar;
tstCal2 ORDSYS.ORDTCalendar;
resultCal ORDSYS.ORDTCalendar;
equalFlag INTEGER;
dummyVal INTEGER;

BEGIN

 -- Select the calendars GENERIC-CAL1 into tstCal1
 -- and GENERIC-CAL2 into tstCal2
 -- from stockdemo_calendars.
 SELECT value(cal) INTO tstCal1
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL1';
 SELECT value(cal) INTO tstCal2
 FROM TSDEV.stockdemo_calendars cal
 WHERE cal.name = 'GENERIC-CAL2';

 -- Display the calendars tstCal1 and tstCal2.
 SELECT ORDSYS.TimeSeries.Display(tstCal1) INTO dummyVal FROM dual;
 SELECT ORDSYS.TimeSeries.Display(tstCal2) INTO dummyVal FROM dual;

 -- Union tstCal1 and tstCal2.
 resultCal := ORDSYS.Calendar.Unioncals(tstCal1, tstCal2);
 SELECT ORDSYS.TimeSeries.Display(resultCal, 'result of UnionCals')
 INTO dummyVal
 FROM dual;

END;
```

```
/
```

This example might produce the following output:

```
Calendar Name = GENERIC-CAL1
 Frequency = 4 (day)
 MinDate = 01/01/96 00:00:00
 MaxDate = 12/31/96 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/96 00:00:00
 onExceptions  :
     01/21/96 00:00:00     02/03/96 00:00:00     03/24/96 00:00:00
     04/27/96 00:00:00     05/19/96 00:00:00     06/23/96 00:00:00
     07/07/96 00:00:00     08/04/96 00:00:00     09/15/96 00:00:00
 offExceptions :
     01/08/96 00:00:00     02/02/96 00:00:00     03/05/96 00:00:00
     04/04/96 00:00:00     05/08/96 00:00:00     06/25/96 00:00:00
     07/09/96 00:00:00

Calendar Name = GENERIC-CAL2
 Frequency = 4 (day)
 MinDate = 01/01/96 00:00:00
 MaxDate = 12/31/97 00:00:00
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 01/08/96 00:00:00
 onExceptions  :
     07/07/96 00:00:00     08/04/96 00:00:00     09/15/96 00:00:00
     10/13/96 00:00:00     11/10/96 00:00:00     12/14/96 00:00:00
     01/04/97 00:00:00     02/09/97 00:00:00     03/08/97 00:00:00
     04/05/97 00:00:00     05/11/97 00:00:00     06/08/97 00:00:00
 offExceptions :
     07/09/96 00:00:00     08/05/96 00:00:00     09/10/96 00:00:00
     10/23/96 00:00:00     11/19/96 00:00:00     12/12/96 00:00:00
     01/01/97 00:00:00     02/12/97 00:00:00     03/04/97 00:00:00
     04/07/97 00:00:00     05/05/97 00:00:00     06/09/97 00:00:00

result of UnionCals :

 Frequency = 4 (day)
 MinDate = 01/01/96 00:00:00
 MaxDate = 12/31/96 00:00:00
 patBits:
          1,1,1,1,1,0,0
```

```
patAnchor = 01/08/96 00:00:00
onExceptions  :
    01/21/96 00:00:00     02/03/96 00:00:00     03/24/96 00:00:00
    04/27/96 00:00:00     05/19/96 00:00:00     06/23/96 00:00:00
    07/07/96 00:00:00     08/04/96 00:00:00     09/15/96 00:00:00
    10/13/96 00:00:00     11/10/96 00:00:00     12/14/96 00:00:00
offExceptions :
    07/09/96 00:00:00
```

ValidateCal

Format

```
ORDSYS.Calendar.ValidateCal(
    cal INOUT ORDSYS.ORDTCalendar,
    outMessage OUT VARCHAR2,
    invOnExc OUT ORDTDateTab,
    invOffExc OUT ORDTDateTab,
    impOnExc OUT ORDTDateTab,
    impOffExc OUT ORDTDateTab
    ) RETURN BINARY_INTEGER;
```

Description

Validates a calendar and, if necessary, repairs the calendar and generates information related to the problems and repairs.

Parameters

cal
The calendar to be validated and (if necessary) repaired.

outMessage
Message describing how the calendar was repaired (if the return value = 1) or why the calendar could not be repaired (if the return value = -1).

invOnExc
Table of the invalid on-exceptions found in the calendar.

invOffExc
Table of the invalid off-exceptions found in the calendar.

impOnExc
Table of the imprecise on-exceptions found in the calendar.

impOffExc
Table of the imprecise off-exceptions found in the calendar.

Usage

This function returns one of the following values:

Value	Meaning
0	The calendar is valid. No errors were found.
1	Correctable errors were found and corrected. The resulting calendar is valid.
-1	Uncorrectable errors were found. The calendar is not valid.

Errors in the input calendar make it invalid. Depending on the error, it may be correctable or uncorrectable. Correctable errors are repaired by the ValidateCal function. If all errors are correctable, the resulting calendar is valid.

For a calendar to be valid, all timestamps in the off-exception and on-exception lists must be consistent with the defined pattern for the calendar. If one or more exception timestamps are not consistent with the pattern, the calendar is invalid. For example, if 04-Jan-1997 (Saturday) is in the off-exception list of a calendar whose pattern includes only Mondays through Fridays as normal business days, 04-Jan-1997 is an invalid off-exception (because as a Saturday it would normally be an "off" day).

Imprecise exception timestamps are repaired. For an explanation of precision, see Section 2.2.2.

Table 4–2 lists correctable errors and the repair actions taken by the ValidateCal function.

Table 4–2 Errors Repaired by ValidateCal

Error	Repair Action
Imprecise anchor date	The precision is adjusted.
Character other than 1 or 0 in the pattern	All pattern characters other than 0 or 1 are set to 1.
Imprecise date	The precision is adjusted.
Superfluous date (for example, a regular valid date in the on-exceptions list)	The date is removed from the exceptions list.
Null date	The date is removed from the calendar.
Unsorted dates	The dates are sorted.
Duplicate dates in the on-exceptions or off-exceptions list	Duplicates are removed; the date appears only once in the list.

Table 4–2 Errors Repaired by ValidateCal (Cont.)

Error	**Repair Action**
Date appearing in both the on-exceptions and off-exceptions lists	The date is removed from the inappropriate list, depending on the pattern and the anchor date.
Date outside the date range of the calendar	The date is removed from the exceptions list.

The following errors are not correctable. The function returns -1 if one or more of these errors are found:

- The frequency is not valid.
- The starting date is later than the ending date.
- The pattern is null or empty.
- All pattern bits are empty.
- One or more pattern bits are null.
- The anchor date is null and the pattern is not "all ones" or "all zeroes" (for example, a pattern of ’0,1,1,1,1,1,0’ but no anchor date specified).

If the function returns -1, you should not use the calendar until you have fixed the errors that ValidateCal could not fix. Then use ValidateCal again, and use the calendar only if the function returns 0 or 1.

You can use the DisplayValCal procedure to display the information returned by the ValidateCal function. See the information on DisplayValCal in this chapter.

The IsValidCal function (described in this chapter) checks the validity of the calendar but does not perform any repair operations.

Example

Use the IsValidCal and ValidateCal functions and the DisplayValCal procedure with an invalid calendar:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = ’MM/DD/YYYY HH24:MI:SS’;

DECLARE
outMessage  varchar2(32750);
invOnExc    ORDSYS.ORDTDateTab;
invOffExc   ORDSYS.ORDTDateTab;
```

```
impOnExc    ORDSYS.ORDTDateTab;
impOffExc   ORDSYS.ORDTDateTab;
dummyval     integer;
validFlag     integer;
tstCal1     ORDSYS.ORDTCalendar :=
               ORDSYS.ORDTCalendar(
                  0,
                  'CALENDAR MYCAL',
                  4,
                  ORDSYS.ORDTPattern(ORDSYS.ORDTPatternBits(1,1,1,1,1,0,0),
                                            TO_DATE('01-08-1996 01:01:01')),
                  TO_DATE('01-01-1975'),
  TO_DATE('01-01-1999'),
                  ORDSYS.ORDTExceptions(
           TO_DATE('02-03-1969'), -- Date < minDate,
           TO_DATE('02-14-1969'), -- Date < minDate,
           TO_DATE('02-03-1999'), -- Date > maxDate,
           TO_DATE('02-17-1999'), -- Date > maxDate,
           TO_DATE('12-31-1995'), -- Maps to 0 in pattern (Sunday)
           TO_DATE('01-13-1996'), -- Maps to 0 in pattern (Saturday)
           TO_DATE('02-24-1996'), -- Maps to 0 in pattern (Saturday)
           TO_DATE('03-30-1996'), -- Maps to 0 in pattern (Saturday)
           TO_DATE('02-02-1996 01:01:01'), -- Imprecise
           TO_DATE('03-04-1996 01:01:01'), -- Imprecise
           TO_DATE('04-05-1996 02:02:02'), -- Imprecise
           TO_DATE('03-25-1996'), -- Valid off-exception
           TO_DATE('01-22-1996'), -- Valid, but out of sequence
           TO_DATE('02-12-1996'),
           TO_DATE('04-30-1996'),
           NULL,                  -- Null date
           TO_DATE('02-12-1996'), -- Duplicate date within OFFs
           NULL,                  -- Null date
           TO_DATE('04-30-1996'), -- Duplicate off-exception
           NULL,                  -- Null date
           TO_DATE('03-25-1996'), -- Duplicate off-exception
           TO_DATE('01-22-1996'), -- Duplicate off-exception
           TO_DATE('01-17-1996'), -- Added to on- and off-exceptions
           TO_DATE('05-28-1996'), -- Added to on- and off-exceptions
           TO_DATE('06-18-1996'), -- Added to on- and off-exceptions
           TO_DATE('04-23-1996'), -- Added to on- and off-exceptions
           TO_DATE('02-02-1996'),
           TO_DATE('03-04-1996'),
           TO_DATE('05-06-1997')),
    ORDSYS.ORDTExceptions(
           TO_DATE('02-08-1969'), -- Date < minDate,
           TO_DATE('02-15-1969'), -- Date < minDate,
           TO_DATE('02-13-1999'), -- Date > maxDate,
           TO_DATE('02-20-1999'), -- Date > maxDate,
           TO_DATE('01-03-1996'), -- Maps to 1 in pattern (Wednesday)
           TO_DATE('02-19-1996'), -- Maps to 1 in pattern (Monday)
           TO_DATE('03-18-1996'), -- Maps to 1 in pattern (Monday)
           TO_DATE('05-27-1996'), -- Maps to 1 in pattern (Monday)
           TO_DATE('03-23-1996 01:01:01'), -- Imprecise
           TO_DATE('02-18-1996 01:01:01'), -- Imprecise
           TO_DATE('05-26-1996 01:01:01'), -- Imprecise
           TO_DATE('01-13-1996'), -- Valid on-exception
           TO_DATE('01-14-1996'), -- Valid on-exception
           NULL,                  -- Null date
           NULL,                  -- Null date
           TO_DATE('02-24-1996'), -- Valid on-exception
           TO_DATE('03-23-1996'), -- Valid on-exception
           TO_DATE('01-13-1996'), -- Duplicate on-exception
           TO_DATE('01-14-1996'), -- Duplicate on-exception
           TO_DATE('02-24-1996'), -- Duplicate on-exception
           TO_DATE('03-23-1996'), -- Duplicate on-exception
           TO_DATE('01-17-1996'), -- Added to on- and off-exceptions
           TO_DATE('05-28-1996'), -- Added to on- and off-exceptions
           TO_DATE('06-18-1996'), -- Added to on- and off-exceptions
           TO_DATE('04-23-1996'), -- Added to on- and off-exceptions
           TO_DATE('01-06-1996'), -- Valid, but out of sequence
           TO_DATE('02-03-1996'),
           TO_DATE('05-04-1997'))
                  );
BEGIN
   SELECT ORDSYS.TIMESERIES.Display(tstCal1, 'tstCal1') INTO dummyval
   FROM dual;
   validFlag := ORDSYS.CALENDAR.IsValidCal(tstCal1);
   IF(validFlag = 0)
   THEN
       validFlag := ORDSYS.CALENDAR.ValidateCal(
                tstCal1, outMessage, invOnExc, invOffExc, impOnExc, impOffExc
                );

       ORDSYS.TIMESERIES.DisplayValCal(
             validFlag,
             outMessage,
             invOnExc,
             invOffExc,
             impOnExc,
             impOffExc,
          tstCal1,
          'Your Message'
          );
   END IF;
END;
/
```

This example might produce the following output:

```
tstCal1 :

Calendar Name = CALENDAR MYCAL
 Frequency = 4 (day)
 MinDate = 01/01/1975 00:00:00
 MaxDate = 01/01/1999 00:00:00
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 01/08/1996 01:01:01
 onExceptions  :
     02/08/1969 00:00:00     02/15/1969 00:00:00     02/13/1999 00:00:00
     02/20/1999 00:00:00     01/03/1996 00:00:00     02/19/1996 00:00:00
     03/18/1996 00:00:00     05/27/1996 00:00:00     03/23/1996 01:01:01
     02/18/1996 01:01:01     05/26/1996 01:01:01     01/13/1996 00:00:00
     01/14/1996 00:00:00
     02/24/1996 00:00:00     03/23/1996 00:00:00     01/13/1996 00:00:00
     01/14/1996 00:00:00     02/24/1996 00:00:00     03/23/1996 00:00:00
     01/17/1996 00:00:00     05/28/1996 00:00:00     06/18/1996 00:00:00
     04/23/1996 00:00:00     01/06/1996 00:00:00     02/03/1996 00:00:00
     05/04/1997 00:00:00
 offExceptions :
     02/03/1969 00:00:00     02/14/1969 00:00:00     02/03/1999 00:00:00
     02/17/1999 00:00:00     12/31/1995 00:00:00     01/13/1996 00:00:00
     02/24/1996 00:00:00     03/30/1996 00:00:00     02/02/1996 01:01:01
     03/04/1996 01:01:01     04/05/1996 02:02:02     03/25/1996 00:00:00
     01/22/1996 00:00:00     02/12/1996 00:00:00     04/30/1996 00:00:00
          02/12/1996 00:00:00
     04/30/1996 00:00:00          03/25/1996 00:00:00
     01/22/1996 00:00:00     01/17/1996 00:00:00     05/28/1996 00:00:00
     06/18/1996 00:00:00     04/23/1996 00:00:00     02/02/1996 00:00:00
     03/04/1996 00:00:00     05/06/1997 00:00:00

DisplayValCal Your Message:

TS-WRN: the input calendar has rectifiable errors. See the message for details
```

```
message output by validateCal:

TS-WRN: fixed precision of the pattern anchor date
TS-WRN: removed superfluous dates in the on exception list (refer invalidOnExc)
TS-WRN: fixed imprecise dates in the on exception list (refer impreciseOnExc)
TS-WRN: removed null dates in the on exception list
TS-WRN: sorted the on exceptions list
TS-WRN: removed duplicate dates in the on exceptions list
TS-WRN: removed superfluous dates in off exceptions list (refer invalidOffExc)
TS-WRN: fixed imprecise dates in the off exception list (refer impreciseOffExc)
TS-WRN: removed null dates in the off exception list
TS-WRN: sorted the off exceptions list
TS-WRN: removed duplicate dates in the off exceptions list
TS-WRN: the on exceptions list was trimmed between calendar minDate & maxDate
TS-WRN: the off exceptions list was trimmed between calendar minDate & maxDate

list of invalid on exceptions :

     01/03/1996 00:00:00     02/19/1996 00:00:00     03/18/1996 00:00:00
     05/27/1996 00:00:00     01/17/1996 00:00:00     05/28/1996 00:00:00
     06/18/1996 00:00:00     04/23/1996 00:00:00

list of invalid off exceptions :

     12/31/1995 00:00:00     01/13/1996 00:00:00     02/24/1996 00:00:00
     03/30/1996 00:00:00

list of imprecise on exceptions :

     03/23/1996 01:01:01     02/18/1996 01:01:01     05/26/1996 01:01:01

list of imprecise off exceptions :

     02/02/1996 01:01:01     03/04/1996 01:01:01     04/05/1996 02:02:02

the validated calendar :

Calendar Name = CALENDAR MYCAL
 Frequency = 4 (day)
 MinDate = 01/01/1975 00:00:00
 MaxDate = 01/01/1999 00:00:00
 patBits:
          1,1,1,1,1,0,0
 patAnchor = 01/08/1996 00:00:00
 onExceptions  :
    01/06/1996 00:00:00     01/13/1996 00:00:00     01/14/1996 00:00:00
    02/03/1996 00:00:00     02/18/1996 00:00:00     02/24/1996 00:00:00
    03/23/1996 00:00:00     05/26/1996 00:00:00     05/04/1997 00:00:00
offExceptions :
    01/17/1996 00:00:00     01/22/1996 00:00:00     02/02/1996 00:00:00
    02/12/1996 00:00:00     03/04/1996 00:00:00     03/25/1996 00:00:00
    04/05/1996 00:00:00     04/23/1996 00:00:00     04/30/1996 00:00:00
    05/28/1996 00:00:00     06/18/1996 00:00:00     05/06/1997 00:00:00
```

Week

Format

```
ORDSYS.Calendar.Week(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *week*, a pattern of ’1’ (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Examples

Insert into the *stockdemo_calendars* table a calendar of *week* frequency with a calendar name of *Weekly* and an anchor date of 05-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Week(
       'Weekly',
       (to_date('01-05-97','MM-DD-YY'))));
```

Return the sum of the daily trade volume for stock SAMCO for each week in the entire time series. For scaling, use a weekly calendar with a null name, an anchor date of 01-Jan-2001 (the default), no date boundaries (*minDate* or *maxDate*), and no exceptions.

```
SELECT * FROM THE
  (SELECT CAST(ORDSYS.TimeSeries.ExtractTable(
                 ORDSYS.TimeScale.ScaleupSum(
                   ts.volume,
                   ORDSYS.Calendar.Week() ))
              AS ORDSYS.ORDTNumTab)
       FROM TSDEV.stockdemo_ts ts
       WHERE ts.ticker='SAMCO');
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
10/28/96       41550
11/04/96      320050
11/11/96     6041550
11/18/96     1909850
11/25/96     1894000
12/02/96     1051350
12/09/96      842850
12/16/96      977450
12/23/96      430800
12/30/96      417000
10 rows selected.
```

Year

Format

```
ORDSYS.Calendar.Year(
    [calname VARCHAR2]
    [, anchorDate DATE]
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Creates a calendar with a frequency of *year*, a pattern of '1' (all timestamps included), no lower or upper boundary dates (*minDate* or *maxDate*), no off-exceptions or on-exceptions, a specified or default (null) name, and a specified or default anchor date.

Parameters

calname
The name of the calendar. If *calname* is not specified, the calendar name is null.

anchorDate
The anchor date for the calendar pattern. If *anchorDate* is not specified, the anchor date is 01-Jan-2001 (a Monday).

Usage

This function provides a convenient alternative to providing a complete calendar definition when you are creating a calendar. If you need to modify the definition later, you can do so (for example, using the InsertExceptions function to specify exceptions).

For an explanation of calendar concepts (such as frequency, pattern, anchor date, and exceptions), see Section 2.2.

The following functions create a calendar with a frequency corresponding to the function name: Day, Hour, Minute, Month, Quarter, Second, Semi_annual, Semi_monthly, Ten_day, Week, and Year.

Example

Insert into the *stockdemo_calendars* table a calendar of *year* frequency with a calendar name of *Yearly* and an anchor date of 01-Jan-1997. The calendar has no date boundaries (*minDate* or *maxDate*) or exceptions.

```
INSERT INTO stockdemo_calendars
VALUES(
   ORDSYS.Calendar.Year(
       'Yearly',
       (to_date('01-01-97','MM-DD-YY'))));
```

5 Time Series Functions: Reference

The Oracle8*i* Time Series library consists of:

- Data types (described in Section 2.3)
- Calendar functions (described in Chapter 4)
- Time series functions (described in this chapter)
- Time scaling functions (described in Chapter 6)
- Administrative tools procedures for creating time series schema objects (described in Chapter 7)

Calendar functions are mainly used by product developers, such as ISVs, to develop new time series functions and to administer and modify calendars.

Time series and time scaling functions and the administrative tools procedures are used mainly by application developers.

Syntax notes:

- The ORDSYS schema name and the package name must be used with the function name, although public synonyms can be created to eliminate the need for specifying the schema name (see Section 1.5). Each function is included in a PL/SQL package, such as Calendar, TimeSeries, or TimeScale. The ORDSYS schema name and the package name are included in the Format and in any examples.
- Function calls are not case sensitive, except for any quoted literal values. For example, the following code line excerpts are valid and semantically identical:

```
select CAST(TimeSeries.ExtractTable(close) AS ORDTNumTab)
select cast(TIMESERIES.extracttable(close) as ordtnumtab)
select cast(TiMeSeRiEs.eXtRaCtTaBlE(ClosE) As ordtNUMtab)
```

- The syntax and examples show the reference-based interface (types ORDTNumSeriesIOTRef and ORDTVarchar2SeriesIOTRef).

All time series and time scaling functions accept both references and instances as parameters. (For example, an ORDTNumSeriesIOTRef parameter could also be ORDTNumSeries.) All time series functions return instances. Thus, if you nest functions, such as *Cmax(Cmax(...), ...)*, the innermost nesting accepts a reference and returns an instance, and any other functions in the nesting accept an instance and return an instance.

For an explanation of the reference-based interface, see Section 2.7.2.

Cavg

Format

```
ORDSYS.TimeSeries.Cavg(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, returns an ORDTNumSeries with each element containing the cumulative average up to and including the corresponding element in the input ORDTNumSeries.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

startDate
Starting date within the time series for which the cumulative average is to be computed. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the cumulative average is to be computed. If *endDate* is specified, *startDate* must also be specified.

Usage

Only non-null values are considered in computing the cumulative average.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

If *startDate* and *endDate* are specified, the time series is trimmed to the date range before the cumulative average is computed.

Example

Return the cumulative average of the closing price of stock ACME for November 1996:

```
SELECT to_char(tstamp) tstamp, value
FROM tsquick ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Cavg(ts.close,to_date('01-NOV-96','DD-MON-YY'),
             to_date('30-NOV-96','DD-MON-YY'))
       ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96         59
04-NOV-96       59.5
05-NOV-96         60
06-NOV-96       60.5
07-NOV-96         61
08-NOV-96       61.5
11-NOV-96         62
12-NOV-96       62.5
13-NOV-96         63
14-NOV-96       63.5
15-NOV-96         64
18-NOV-96       64.5
19-NOV-96         65
20-NOV-96       65.5
21-NOV-96         66
22-NOV-96       66.5
25-NOV-96         67
26-NOV-96       67.5
27-NOV-96         68
29-NOV-96       68.5
20 rows selected.
```

Cmax

Format

```
ORDSYS.TimeSeries.Cmax(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, returns an ORDTNumSeries with each element containing the cumulative maximum up to and including the corresponding element in the input ORDTNumSeries.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

startDate
Starting date within the time series for which the cumulative maximum is to be returned. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the cumulative maximum is to be returned. If *endDate* is specified, *startDate* must also be specified.

Usage

Only non-null values are considered in determining the cumulative maximum.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

If *startDate* and *endDate* are specified, the time series is trimmed to the date range before the cumulative maximum is computed.

Example

Return the cumulative maximum of the closing price of stock ACME for November 1996:

```
SELECT to_char(tstamp) tstamp, value
FROM tsquick ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Cmax(ts.close,to_date('01-NOV-96','DD-MON-YY'),
             to_date('30-NOV-96','DD-MON-YY'))
       ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output. (Note that this output reflects the simplified artificial data in the usage demo database, where the closing price rises one point each day.)

```
TSTAMP    VALUE
--------- ----------
01-NOV-96         59
04-NOV-96         60
05-NOV-96         61
06-NOV-96         62
07-NOV-96         63
08-NOV-96         64
11-NOV-96         65
12-NOV-96         66
13-NOV-96         67
14-NOV-96         68
15-NOV-96         69
18-NOV-96         70
19-NOV-96         71
20-NOV-96         72
21-NOV-96         73
22-NOV-96         74
25-NOV-96         75
26-NOV-96         76
27-NOV-96         77
```

```
29-NOV-96         78
20 rows selected.
```

Cmin

Format

```
ORDSYS.TimeSeries.Cmin(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, returns an ORDTNumSeries with each element containing the cumulative minimum up to and including the corresponding element in the input ORDTNumSeries.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

startDate
Starting date within the time series for which the cumulative minimum is to be returned. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the cumulative minimum is to be returned. If *endDate* is specified, *startDate* must also be specified.

Usage

Only non-null values are considered in determining the cumulative minimum.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

If *startDate* and *endDate* are specified, the time series is trimmed to the date range before the cumulative minimum is computed.

Example

Return the cumulative minimum of the closing price of stock ACME for November 1996:

```
SELECT to_char(tstamp) tstamp, value
FROM tsquick ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Cmin(ts.close,to_date('01-NOV-96','DD-MON-YY'),
             to_date('30-NOV-96','DD-MON-YY'))
       ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output. (Note that this output reflects the simplified artificial data in the usage demo database, where the closing price rises one point each day.)

```
TSTAMP    VALUE
--------- ----------
01-NOV-96         59
04-NOV-96         59
05-NOV-96         59
06-NOV-96         59
07-NOV-96         59
08-NOV-96         59
11-NOV-96         59
12-NOV-96         59
13-NOV-96         59
14-NOV-96         59
15-NOV-96         59
18-NOV-96         59
19-NOV-96         59
20-NOV-96         59
21-NOV-96         59
22-NOV-96         59
25-NOV-96         59
26-NOV-96         59
27-NOV-96         59
```

```
29-NOV-96         59
20 rows selected.
```

Cprod

Format

```
ORDSYS.TimeSeries.Cprod(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, returns an ORDTNumSeries with each element containing the cumulative product of multiplication up to and including the corresponding element in the input ORDTNumSeries.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

startDate
Starting date within the time series for which the cumulative product is to be computed. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the cumulative product is to be computed. If *endDate* is specified, *startDate* must also be specified.

Usage

Only non-null values are considered in computing the cumulative product.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

If *startDate* and *endDate* are specified, the time series is trimmed to the date range before the cumulative product is computed.

Example

Return the cumulative product of the daily volume of stock ACME for the first four trading days of November 1996. (This example is presented merely to illustrate the function; the results of this query have no practical value for financial analysis.)

```
SELECT to_char(tstamp) tstamp, value
FROM tsquick ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Cprod(ts.volume,to_date('01-NOV-96','DD-MON-YY'),
             to_date('06-NOV-96','DD-MON-YY'))
       ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96       1000
04-NOV-96    1000000
05-NOV-96 1000000000
06-NOV-96 1.0000E+12
4 rows selected.
```

Csum

Format

```
ORDSYS.TimeSeries.Csum(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, returns an ORDTNumSeries with each element containing the cumulative sum up to and including the corresponding element in the input ORDTNumSeries.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

startDate
Starting date within the time series for which the cumulative sum is to be computed. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the cumulative sum is to be computed. If *endDate* is specified, *startDate* must also be specified.

Usage

Only non-null values are considered in computing the cumulative sum.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

If *startDate* and *endDate* are specified, the time series is trimmed to the date range before the cumulative sum is computed.

Example

Return the cumulative sum of the daily volume of stock ACME for November 1996:

```
SELECT to_char(tstamp) tstamp, value
FROM tsquick ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Csum(ts.volume,to_date('01-NOV-96','DD-MON-YY'),
             to_date('30-NOV-96','DD-MON-YY'))
       ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96       1000
04-NOV-96       2000
05-NOV-96       3000
06-NOV-96       4000
07-NOV-96       5000
08-NOV-96       6000
11-NOV-96       7000
12-NOV-96       8000
13-NOV-96       9000
14-NOV-96      10000
15-NOV-96      11000
18-NOV-96      12000
19-NOV-96      13000
20-NOV-96      14000
21-NOV-96      15000
22-NOV-96      16000
25-NOV-96      17000
26-NOV-96      18000
27-NOV-96      19000
29-NOV-96      20000
20 rows selected.
```

DeriveExceptions

Format

Approach 1:

```
ORDSYS.TimeSeries.DeriveExceptions(
    inputTS ORDTNumSeriesIOTRef
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

or

```
ORDSYS.TimeSeries.DeriveExceptions(
    inputTS ORDTVarchar2SeriesIOTRef
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

Approach 1A:

```
ORDSYS.TimeSeries.DeriveExceptions(
    inputCal  IN ORDSYS.ORDTCalendar,
    DateTab IN ORDSYS.ORDTDateTab
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

Approach 2:

```
ORDSYS.TimeSeries.DeriveExceptions(
    series1 ORDTNumSeriesIOTRef,
    series2 ORDTNumSeriesIOTRef
    [, startDate DATE [, endDate DATE]]
    ) RETURN ORDSYS.ORDTCalendar;
```

or

```
ORDSYS.TimeSeries.DeriveExceptions(
series1 ORDTVarchar2SeriesIOTRef,
series2 ORDTVarchar2SeriesIOTRef
[, startDate DATE [, endDate DATE]]
) RETURN ORDSYS.ORDTCalendar;
```

Description

Derives calendar exceptions from a time series (Approach 1), a calendar and a table of dates (Approach 1A), or two time series (Approach 2).

Parameters

inputTS
The time series whose calendar is to be used as the basis for the returned calendar and whose timestamps are to be used to populate the off- and on-exceptions lists of the returned calendar.

startDate
Starting date within the time series for which the exceptions are to be derived. If *startDate* is not specified, it is the *minDate* of the calendar and *endDate* is the *maxDate* of the calendar.

endDate
Ending date within the time series for which the exceptions are to be derived. If *endDate* is specified, *startDate* must also be specified. If *startDate* is specified and *endDate* is not specified, *endDate* is the *maxDate* of the calendar.

inputCal
The calendar that contains no exceptions and for which exceptions are to be derived.

DateTab
The table of dates that includes all dates in the time series (for example, all dates on which stock XYZ traded).

series1
The "reference" time series that contains no exceptions and all valid timestamps from the calendar (for example, all Monday through Friday dates within the date range of the calendar).

series2
The time series that contains the timestamps to be used in deriving the exceptions for the resulting calendar (for example, all dates on which stock XYZ traded).

Usage

Approach 1 is the most convenient method. You specify a time series (for example, daily closing prices of stock XYZ) that has an associated calendar. A calendar is returned that is defined on the same pattern and frequency as the calendar for the input time series, and the exceptions lists of the returned calendar are populated to be consistent with the time series data.

Approach 1A is a variation of Approach 1 in which you specify a calendar and a table of the desired timestamps (for example, dates on which stock XYZ traded).

Approach 2 involves creating a time series (*series1*) that in effect functions as a calendar, and then using a second time series (*series2*) with desired timestamps to populate the exceptions lists. Approach 2 offers a performance advantage if you need to derive exceptions for many calendars based on many time series.

See Section 2.2.5 for a detailed explanation of the approaches to using this function.

Example

See Section 3.8 for examples of the approaches to using this function.

Display

Format

```
ORDSYS.TimeSeries.Display(
     ts  ORDSYS.[see parameter description]
     [,mesg VARCHAR2]
     ) RETURN INTEGER;
```

Description

Displays various information (see the description of the *ts* parameter) using DBMS_OUTPUT routines.

Parameters

ts
The object to be displayed. Because the function is overloaded, this parameter can be any of the following data types:

- ORDTNumSeriesIOTRef or ORDTNumSeries
- ORDTVarchar2SeriesIOTRef or ORDTVarcharSeries
- ORDTNumTab
- ORDTVarchar2Tab
- ORDTNumCell
- ORDTVarchar2Cell
- ORDTDateTab
- ORDTCalendar
- ORDTExceptions
- ORDTPattern

mesg
Optional message text to be included in the display heading ("Timeseries dump for <mesg>").

Usage

Use the SET SERVEROUTPUT ON statement to view the output of the Display function. However, the default display buffer of 2000 bytes is often too small to display a large time series. In such cases you must use the ENABLE procedure of the DBMS_OUTPUT package to specify a larger display buffer size. For example:

```
DBMS_OUTPUT.ENABLE(1000000);
```

You should use Display only for development and debugging. Specify a display buffer larger than 2000 only when necessary, because the display buffer uses shared system resources, and a large value might affect the performance of other users.

Because the Display function uses DBMS_OUTPUT routines, it is subject to the limitations of these routines. These limitations include the following:

- Output cannot exceed 1 megabyte.
- The Display function cannot be used with the OCI.
- SQL*Plus does not support DBMS_OUTPUT in the context of a SELECT statement, but it does support DBMS_OUTPUT for anonymous PL/SQL blocks.

Example

Display the output for a query that returns the 10 highest closing prices for stock ACME for the month of November 1996:

```
SET SERVEROUTPUT ON
DECLARE
     tmp INTEGER;
BEGIN
SELECT ORDSYS.TimeSeries.Display(
         ORDSYS.TimeSeries.TSMaxN(close,10,
            to_date('11011996','MMDDYYYY'),
            to_date('11301996','MMDDYYYY')))
         INTO tmp
FROM TSDEV.stockdemo_ts
WHERE ticker ='ACME';
END;
/
```

This example might produce the following output:

```
Tab Data:
 -----------------------------
 Date                    Value
 29-NOV-96      78
 27-NOV-96      77
 26-NOV-96      76
 25-NOV-96      75
 22-NOV-96      74
 21-NOV-96      73
 20-NOV-96      72
 19-NOV-96      71
 18-NOV-96      70
 15-NOV-96      69
 -----------------------------
```

The preceding example works from both SQL*Plus and the Server Manager (svrmgrl) prompt. The following version of the example works from the Server Manager prompt but not from SQL*Plus:

```
SET SERVEROUTPUT ON
SELECT ORDSYS.TimeSeries.Display(
         ORDSYS.TimeSeries.TSMaxN(close,10,
            to_date('11011996','MMDDYYYY'),
            to_date('11301996','MMDDYYYY')))
FROM TSDEV.stockdemo_ts
WHERE ticker ='ACME';
```

See the TSMaxN function for an example that returns the same information, but that uses a subquery instead of the Display function.

DisplayValTS Procedure

Format

```
ORDSYS.TimeSeries.DisplayValTS(
    validFlag IN INTEGER,
    outMessage IN VARCHAR2,
    loDateTab IN ORDSYS.ORDTDateTab,
    hiDateTab  IN ORDSYS.ORDTDateTab,
    impreciseDateTab IN ORDSYS.ORDTDateTab,
    duplicateDateTab IN ORDSYS.ORDTDateTab,
    extraDateTab IN ORDSYS.ORDTDateTab,
    missingDateTab IN ORDSYS.ORDTDateTab,
    mesg IN VARCHAR2
    );
```

Description

Displays the results returned by the ValidateTS function.

Note: DisplayValTS is a procedure, not a function. Procedures do not return values.

Parameters

validFlag
The return value from the ValidateTS function.

outMessage
The diagnostic returned by the ValidateTS function.

loDateTab
A table of dates before the starting date of the calendar associated with the time series.

hiDateTab
A table of dates after the starting date of the calendar associated with the time series.

impreciseDateTab
A table of the imprecise dates found in the time series.

duplicateDateTab
A table of the duplicate dates (dates that appear more than once in the time series).

extraDateTab
A table of dates that are included in the time series but that should be excluded based on the calendar definition (for example, a Saturday timestamp that is in a Monday-Friday calendar and that is not an on-exception).

missingDateTab
A table of dates that are excluded from the time series but that should be included based on the calendar definition (for example, a Wednesday date that is not a holiday in a Monday-Friday calendar and for which there is no data). Such dates can be considered as "holes" in the time series.

mesg
Optional message.

Usage

This procedure is intended to be used with the ValidateTS function. See the information on ValidateTS in this chapter.

The DisplayValTS procedure uses the DBMS_OUTPUT package. See the Usage information for the Display function for limitations relating to the use of DBMS_OUTPUT.

Example

Use the IsValidTS and ValidateTS functions and the DisplayValTS procedure with an invalid time series:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
 numTS  ORDSYS.ORDTNumSeries;
 tempVal integer;
 retIsValid  integer;
 retValTS    integer;
 loDateTab  ORDSYS.ORDTDateTab := NULL;
 hiDateTab  ORDSYS.ORDTDateTab := NULL;
 impDateTab ORDSYS.ORDTDateTab := NULL;
 dupDateTab ORDSYS.ORDTDateTab := NULL;
 extraDateTab ORDSYS.ORDTDateTab := NULL;
 missingDateTab ORDSYS.ORDTDateTab := NULL;
 outMesg varchar2(2000);

BEGIN

   --  Set the buffer size
   DBMS_OUTPUT.ENABLE(100000);

   --
   -- NOTE: Here, an instance of the time series is materialized
   -- so that it could be modified to generate an invalid time series.
   --
   SELECT ORDSYS.TIMESERIES.GetSeries(ts.open) INTO numTS
   FROM tsdev.stockdemo_ts ts
   WHERE ts.ticker = 'ACME';

   -- Example of validating a valid time series.
   SELECT ordsys.timeseries.display(numTS, 'A VALID TIME SERIES') INTO tempVal
   FROM dual;
   retIsValid := ORDSYS.TIMESERIES.IsValidTS(numTS);
   retValTS := ORDSYS.TIMESERIES.ValidateTS(numTS, outMesg, loDateTab,
                          hiDateTab, impDateTab, dupDateTab,
                          extraDateTab, missingDateTab);
   DBMS_OUTPUT.PUT_LINE('Value returned by IsValid  = ' || retIsValid);
   DBMS_OUTPUT.PUT_LINE('Value returned by ValidateTS  = ' || retValTS);
   ORDSYS.TIMESERIES.DisplayValTS(retValTS, outMesg, loDateTab, hiDateTab,
                       impDateTab, dupDateTab, extraDateTab, missingDateTab,
                       'Testing DisplayValTS');
   DBMS_OUTPUT.NEW_LINE;

   -- For illustration let us first create an invalid timeseries.
   --
   -- Here we are adjusting the calendar's minDate and maxDate to avoid
   -- getting a huge list of missing dates.
   --
   numTS.cal.minDate := TO_DATE('10/28/1996');
   numTS.cal.maxDate := TO_DATE('01/05/1997');
```

```
    -- Add Dates Before numTS.cal.minDate
    numTS.series(10).tstamp := numTS.cal.minDate - 1;
    numTS.series(11).tstamp := numTS.cal.minDate - 2;

    -- Add Dates Beyond numTS.cal.maxDate
    numTS.series(12).tstamp := numTS.cal.maxDate + 1;
    numTS.series(13).tstamp := numTS.cal.maxDate + 2;

    -- Add some null timestamps
    numTS.series(14).tstamp := NULL;
    numTS.series(15).tstamp := NULL;

    -- Add some imprecise dates (some are duplicated)
    numTS.series(17).tstamp := numTS.series(16).tstamp + 1/24;
    numTS.series(18).tstamp := numTS.series(16).tstamp + 15/24;

    -- Add some duplicate timestamps
    numTS.series(19).tstamp := numTS.series(18).tstamp;
    numTS.series(21).tstamp := numTS.series(20).tstamp;

    -- Add some extra dates in the middle
    numTS.series(37).tstamp := TO_DATE('12/28/1996');
    numTS.series(36).tstamp := TO_DATE('12/29/1996');

    -- Add some holes at the end
    numTS.series(numTS.series.count).tstamp := TO_DATE('01/04/1997');

    -- Example of validating an invalid time series.
    SELECT ordsys.timeseries.display(numTS, 'AN INVALID TIME SERIES')
    INTO tempVal FROM dual;
    retIsValid := ORDSYS.TIMESERIES.IsValidTS(numTS);
    retValTS := ORDSYS.TIMESERIES.ValidateTS(numTS, outMesg,
                          loDateTab, hiDateTab, impDateTab,
                          dupDateTab, extraDateTab, missingDateTab);
    DBMS_OUTPUT.PUT_LINE('Value returned by IsValid  = ' || retIsValid);
    DBMS_OUTPUT.PUT_LINE('Value returned by ValidateTS  = ' || retValTS);
    ORDSYS.TIMESERIES.DisplayValTS(retValTS, outMesg, loDateTab, hiDateTab,
                       impDateTab, dupDateTab, extraDateTab, missingDateTab,
                       'Testing DisplayValTS');
END;
/
```

This example might produce the following output:

```
A VALID TIME SERIES :
```

```
Name = OPEN ACME
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 11/14/1996 00:00:00      68
 11/15/1996 00:00:00      69
 11/18/1996 00:00:00      70
 11/19/1996 00:00:00      71
 11/20/1996 00:00:00      72
 11/21/1996 00:00:00      73
 11/22/1996 00:00:00      74
 11/25/1996 00:00:00      75
 11/26/1996 00:00:00      76
 11/27/1996 00:00:00      77
 11/29/1996 00:00:00      78
 12/02/1996 00:00:00      79
 12/03/1996 00:00:00      80
 12/04/1996 00:00:00      81
 12/05/1996 00:00:00      82
 12/06/1996 00:00:00      83
 12/09/1996 00:00:00      84
 12/10/1996 00:00:00      85
 12/11/1996 00:00:00      86
 12/12/1996 00:00:00      87
 12/13/1996 00:00:00      88
 12/16/1996 00:00:00      89
 12/17/1996 00:00:00      90
 12/18/1996 00:00:00      91
 12/19/1996 00:00:00      92
 12/20/1996 00:00:00      93
 12/23/1996 00:00:00      94
 12/24/1996 00:00:00      95
 12/26/1996 00:00:00      96
 12/27/1996 00:00:00      97
 12/30/1996 00:00:00      98
 12/31/1996 00:00:00      99
 -----------------------------

Value returned by IsValid  = 1
Value returned by ValidateTS  = 1

DisplayValTS: Testing DisplayValTS:

TS-SUC: the input time series is a valid time series



AN INVALID TIME SERIES :

Name = OPEN ACME
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 10/28/1996 00:00:00
 MaxDate = 01/05/1997 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                   Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 10/27/1996 00:00:00      68
 10/26/1996 00:00:00      69
 01/06/1997 00:00:00      70
 01/07/1997 00:00:00      71
       72
       73
 11/22/1996 00:00:00      74
 11/22/1996 01:00:00      75
 11/22/1996 15:00:00      76
 11/22/1996 15:00:00      77
 11/29/1996 00:00:00      78
 11/29/1996 00:00:00      79
 12/03/1996 00:00:00      80
 12/04/1996 00:00:00      81
 12/05/1996 00:00:00      82
 12/06/1996 00:00:00      83
 12/09/1996 00:00:00      84
 12/10/1996 00:00:00      85
 12/11/1996 00:00:00      86
 12/12/1996 00:00:00      87
 12/13/1996 00:00:00      88
 12/16/1996 00:00:00      89
 12/17/1996 00:00:00      90
 12/18/1996 00:00:00      91
 12/19/1996 00:00:00      92
 12/20/1996 00:00:00      93
 12/29/1996 00:00:00      94
 12/28/1996 00:00:00      95
 12/26/1996 00:00:00      96
 12/27/1996 00:00:00      97
 12/30/1996 00:00:00      98
 01/04/1997 00:00:00      99
 -----------------------------

Value returned by IsValid  = 0
Value returned by ValidateTS  = 0

DisplayValTS: Testing DisplayValTS:

TS-WRN: the input time series has errors. See the message for details
```

```
message output by validateTS:

TS-ERR: the input time series is unsorted
TS-ERR: the time series has null timestamps
TS-ERR: the time series has timestamps < calendar minDate (refer LoDateTab)
TS-ERR: the time series has timestamps > calendar maxDate (refer HiDateTab)
TS-ERR: the time series has imprecise timestamps (refer impreciseDateTab)
TS-ERR: the time series has duplicate timestamps (refer DuplicateDateTab)

list of dates < calendar minDate - lowDateTab :

     10/26/1996 00:00:00     10/27/1996 00:00:00

list of dates > calendar maxDate - hiDateTab :

     01/06/1997 00:00:00     01/07/1997 00:00:00

list of imprecise dates - impreciseDateTab :

     11/22/1996 01:00:00     11/22/1996 15:00:00

list of duplicate dates - duplicateDateTab :

     11/22/1996 15:00:00     11/29/1996 00:00:00

ExtraDateTab :

     12/28/1996 00:00:00     12/29/1996 00:00:00     01/04/1997 00:00:00

MissingDateTab :

     10/28/1996 00:00:00     10/29/1996 00:00:00     10/30/1996 00:00:00
     10/31/1996 00:00:00     11/14/1996 00:00:00     11/15/1996 00:00:00
     11/18/1996 00:00:00     11/19/1996 00:00:00     11/20/1996 00:00:00
     11/21/1996 00:00:00     11/25/1996 00:00:00     11/26/1996 00:00:00
     11/27/1996 00:00:00     12/02/1996 00:00:00     12/23/1996 00:00:00
     12/24/1996 00:00:00     12/31/1996 00:00:00     01/01/1997 00:00:00
     01/02/1997 00:00:00     01/03/1997 00:00:00
```

ExtractCal

Format

```
ORDSYS.TimeSeries.ExtractCal(
    ts ORDSYS.ORDTNumSeriesIOTRef
    ) RETURN ORDSYS.ORDTCalendar;
```

or

```
ORDSYS.TimeSeries.ExtractCal(
    ts ORDSYS.ORDTVarchar2SeriesIOTRef
    ) RETURN ORDSYS.ORDTCalendar;
```

Description

Given a time series, returns a calendar that is the same as the calendar on which the time series is based.

Parameters

ts
The input time series.

Usage

The function returns a calendar that has the same starting and ending timestamps, pattern, frequency, and exceptions (on- and off-) as the calendar on which the specified time series is based.

An exception is returned if the time series (*ts*) is null.

Example

Return a calendar that matches the one on which the time series for the ACME ticker is based:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
```

```
dummyval INTEGER;

BEGIN

 SELECT ORDSYS.TimeSeries.Display(
     ORDSYS.TimeSeries.ExtractCal(ts.open), 'ExtractCal Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts
 WHERE ts.ticker='ACME';

END;
/
```

This example might produce the following output:

```
ExtractCal Results :

Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
```

ExtractDate

Format

ORDSYS.TimeSeries.ExtractDate(

    cell ORDSYS.ORDTNumCell

) RETURN DATE;

or

ORDSYS.TimeSeries.ExtractDate(

    cell ORDSYS.ORDTVarchar2Cell

) RETURN DATE;

Description

Given an element in a time series, returns the date.

Parameters

cell
The time series element for which you want the date.

Usage

The time series element must first be identified, such as by using the GetNthElement function.

An exception is returned if the time series element (cell) is null.

Example

Return the date associated with the tenth element in a specified time series:

```
SELECT to_char( ORDSYS.TimeSeries.ExtractDate(
       ORDSYS.TimeSeries.GetNthElement(open, 10)),
            ’MM/DD/YYYY HH24:MI:SS’)
       FROM TSDEV.stockdemo_ts
       WHERE ticker = ’ACME’;
```

This example might produce the following output:

```
TO_CHAR(ORDSYS.TIME
-------------------
11/14/1996 00:00:00
1 row selected.
```

ExtractTable

Format

ORDSYS.TimeSeries.ExtractTable(

    ts ORDSYS.ORDTNumSeriesIOTRef

) RETURN ORDSYS.ORDTNumTab;

or

ORDSYS.TimeSeries.ExtractTable(

    ts ORDSYS.ORDTVarchar2SeriesIOTRef

) RETURN ORDSYS.ORDTVarchar2Tab;

Description

Given a time series, returns the time series table (ORDTNumTab or ORDTVarchar2Tab) associated with the time series.

Parameters

ts
The input time series.

Usage

The function returns the time series table (ORDTNumTab or ORDTVarchar2Tab) associated with the time series.

An exception is returned if the time series (ts) is null.

Example

Return the closing prices for stock ACME:

```
SELECT * FROM the
  (SELECT CAST(ORDSYS.TimeSeries.ExtractTable(ts.close)
               as ORDSYS.ORDTNumTab)
       FROM TSDEV.stockdemo_ts ts
       WHERE ts.ticker='ACME');
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96         59
04-NOV-96         60
05-NOV-96         61
   ...                    ...
31-DEC-96         99
41 rows selected.
```

ExtractValue

Format

ORDSYS.TimeSeries.ExtractValue(

    cell ORDSYS.ORDTNumCell

) RETURN NUMBER;

or

ORDSYS.TimeSeries.ExtractValue(

    cell ORDSYS.ORDTVarchar2Cell

) RETURN VARCHAR2;

Description

Given an element in a time series, returns the value stored in it.

Parameters

cell
The time series element for which you want the value.

Usage

The time series element must first be identified, such as by using the GetNthElement function.

An exception is returned if the time series element (cell) is null.

Example

Return the value of the tenth opening price in the *stockdemo_ts* table:

```
SELECT ORDSYS.TimeSeries.ExtractValue(
         ORDSYS.TimeSeries.GetNthElement(open, 10))
  FROM TSDEV.stockdemo_ts
  WHERE ticker = 'ACME';
```

This example might produce the following output:

```
ORDSYS.TIM
----------
        68
1 row selected.
```

Fill

Format

```
ORDSYS.TimeSeries.Fill(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef
    [, fill_type INTEGER]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series and optionally a fill type, returns a time series in which values for missing dates are inserted. A missing date is a date that is defined by the calendar and within the time series bounds, but that is not in the current time series.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

fill_type
One of the following integers indicating how missing values are to be filled:

- 0 = null: Insert nulls.
- 1 = forward repeat: Use the values from the preceding (most recent) timestamp.
- 2 = backward repeat: Use the values from the following (next in the future) timestamp.

If *fill_type* is omitted, 0 is assumed.

Usage

The function inserts timestamps and associated values for timestamps that are included in a calendar but for which no entries exist in the time series.

The *fill_type* parameter lets you choose the manner in which missing values will be defaulted. For example, assume that data for 30-Jan-1997 (Thursday) is missing from a time series and that it should be included because this date is within the calendar definition. Assume the following closing prices for stock XYZ:

- 49 on 29-Jan-1997
- 50 on 31-Jan-1997

The following table shows the closing price that would be inserted for 30-Jan-1997 with each of the *fill_type* parameter values:

fill_type	Closing Price for 30-Jan-1997
0	null
1	49
2	50

Some potential uses for this function include:

- Deriving the price of a stock for a nontrading day

  For example, you may want to compare prices for a stock that trades on several stock exchanges, where the exchanges have different trading days.

- Converting a quarterly time series to a daily time series

  For example, earnings per share (EPS) is computed quarterly, and stocks trade daily. To compute a price-earnings (PE) ratio, earnings per share is first converted to a daily time series using forward repeat. Then, the daily PE ratio is calculated by dividing the daily price time series value by the corresponding daily EPS time series value.

An exception is returned if the specified *fill_type* value is not 0, 1, or 2.

Example

Return a time series illustrating each *fill_type* value:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';
-- For illustrating Fill we need a time series with missing dates.
-- In the following example, the time series 'FOO' has some missing dates
-- (07-DEC-1996 and 08-DEC-1996). Also, note that the calendar associated
-- with 'FOO' has an 'all one' pattern.
--
```

```
DECLARE
tstCal ORDSYS.ORDTCalendar;
ts     ORDSYS.ordtnumseries :=
            ORDSYS.ordtnumseries(
                 'FOO',
                 ORDSYS.ORDTCalendar(
                                 0,
                                 'FOO CALENDAR',
                                 4,
                                 ORDSYS.ORDTPattern(
                                     ORDSYS.ORDTPatternBits(1,1,1,1,1,1,1),
                                        TO_DATE('01/07/1996')),
                                 TO_DATE('01/01/1996'),
                                 TO_DATE('01/01/1997'),
                ORDSYS.ORDTExceptions(),
                ORDSYS.ORDTExceptions()
                ),
              ORDSYS.ordtnumtab(
                       ORDSYS.ordtnumcell(TO_DATE('12/02/1996'), 1),
                       ORDSYS.ordtnumcell(TO_DATE('12/03/1996'), 2),
                       ORDSYS.ordtnumcell(TO_DATE('12/04/1996'), 3),
                       ORDSYS.ordtnumcell(TO_DATE('12/05/1996'), 4),
                       ORDSYS.ordtnumcell(TO_DATE('12/06/1996'), 5),
                       ORDSYS.ordtnumcell(TO_DATE('12/09/1996'), 6),
                       ORDSYS.ordtnumcell(TO_DATE('12/10/1996'), 7),
                       ORDSYS.ordtnumcell(TO_DATE('12/11/1996'), 8),
                       ORDSYS.ordtnumcell(TO_DATE('12/12/1996'), 9),
                       ORDSYS.ordtnumcell(TO_DATE('12/13/1996'), 10))
              );

dummyval INTEGER;

BEGIN

 -- Generate a time series by from XCORP's high (repeat forward).
 SELECT ORDSYS.TimeSeries.Display(
           ORDSYS.TimeSeries.Fill(ts, 1),
           'Fill Forward') INTO dummyval
 FROM dual;

 -- Generate a time series by from XCORP's high (repeat backward).
 SELECT ORDSYS.TimeSeries.Display(
           ORDSYS.TimeSeries.Fill(ts, 2),
           'Fill Backward') INTO dummyval
 FROM dual;
 -- Generate a time series by from XCORP's high (null fill).
 SELECT ORDSYS.TimeSeries.Display(
           ORDSYS.TimeSeries.Fill(ts, 0),
           'Null Fill') INTO dummyval
 FROM dual;

END;
/
```

This example might produce the following output:

```
Fill Forward :

Calendar Data:
Calendar Name = FOO CALENDAR
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 01/01/1997 00:00:00
 patBits:
          1,1,1,1,1,1,1
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
Series Data:
 -----------------------------
 Date                    Value
 12/02/1996 00:00:00      1
 12/03/1996 00:00:00      2
 12/04/1996 00:00:00      3
 12/05/1996 00:00:00      4
 12/06/1996 00:00:00      5
 12/07/1996 00:00:00      5
 12/08/1996 00:00:00      5
 12/09/1996 00:00:00      6
 12/10/1996 00:00:00      7
 12/11/1996 00:00:00      8
 12/12/1996 00:00:00      9
 12/13/1996 00:00:00      10
 -----------------------------

Fill Backward :

Calendar Data:
Calendar Name = FOO CALENDAR
```

```
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 01/01/1997 00:00:00
 patBits:
          1,1,1,1,1,1,1
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
Series Data:
 -----------------------------
 Date                    Value
 12/02/1996 00:00:00      1
 12/03/1996 00:00:00      2
 12/04/1996 00:00:00      3
 12/05/1996 00:00:00      4
 12/05/1996 00:00:00      4
 12/06/1996 00:00:00      5
 12/07/1996 00:00:00      6
 12/08/1996 00:00:00      6
 12/09/1996 00:00:00      6
 12/10/1996 00:00:00      7
 12/11/1996 00:00:00      8
 12/12/1996 00:00:00      9
 12/13/1996 00:00:00      10
 -----------------------------

Null Fill :

Calendar Data:
Calendar Name = FOO CALENDAR
 Frequency = 4 (day)
 MinDate = 01/01/1996 00:00:00
 MaxDate = 01/01/1997 00:00:00
 patBits:
          1,1,1,1,1,1,1
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
Series Data:
 -----------------------------
 Date                    Value
 12/02/1996 00:00:00      1
 12/03/1996 00:00:00      2
 12/04/1996 00:00:00      3
 12/05/1996 00:00:00      4
```

```
12/06/1996 00:00:00      5
12/07/1996 00:00:00
12/08/1996 00:00:00
12/09/1996 00:00:00      6
12/10/1996 00:00:00      7
12/11/1996 00:00:00      8
12/12/1996 00:00:00      9
12/13/1996 00:00:00      10
-----------------------------
```

First

Format

ORDSYS.TimeSeries.First(

ts ORDSYS.ORDTNumSeriesIOTRef

) RETURN ORDSYS.ORDTNumCell;

Description

Given a time series, returns the first element in it.

Parameters

ts
The input time series.

Usage

A null is returned if the time series (*ts*) is empty.

An exception is returned if the time series (*ts*) is null.

Example

Return the first timestamp and opening price for stock ACME in the *stockdemo_ts* time series:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
dummyval INTEGER;

BEGIN

 SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.First(ts.open), 'First Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts
 WHERE ts.ticker='ACME';

END;
```

```
/
```

This example might produce the following output:

```
First Results :

   Timestamp : 11/01/1996 00:00:00
       Value : 59
```

FirstN

Format

```
ORDSYS.TimeSeries.FirstN(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    NumValues NUMBER
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series and a number of elements (*NumValues*) to return, returns the first *NumValues* elements in the time series.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

NumValues
Number of elements from the beginning of the time series to be returned.

startDate
Starting date within the time series for which *NumValues* elements are to be returned. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which *NumValues* elements are to be returned. If *endDate* is specified, *startDate* must also be specified.

Usage

The function returns a time series populated with the first *NumValues* cells from the input time series (*ts*). The calendar of the output time series is the same as that of the input time series.

An exception is returned if the time series (*ts*) is null, if *NumValues* is zero (0) or negative, or if *endDate* is earlier than *startDate*.

If *startDate* and *endDate* are specified, the time series is trimmed to the date range before the first *NumValues* cells are returned.

Example

Return the first 10 timestamps and opening prices in the time series for stock ACME.:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
dummyval INTEGER;

BEGIN

 SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.FirstN(ts.open, 10), 'FirstN Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts
 WHERE ts.ticker='ACME';

END;
/
```

This example might produce the following output:

```
FirstN Results :

Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
```

```
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 11/14/1996 00:00:00      68
 -----------------------------
```

GetDatedElement

Format

ORDSYS.TimeSeries.GetDatedElement (

ts ORDSYS.ORDTNumSeriesIOTRef,

target_date date

) RETURN ORDSYS.ORDTNumCell;

Description

Given a time series and a date, returns the time series element for that date.

Parameters

ts
The input time series.

target_date
Positive integer specifying the date of the element to be returned.

Usage

The function returns the cell from the input time series (*ts*) at the specified date (*target_date*). If there is no data in *ts* at *target_date*, the function returns a null.

An exception is returned if the time series (*ts*) is null.

Example

Return the timestamp and opening price for 26-Nov-1996 for stock ACME:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
dummyval INTEGER;
tstDate date;

BEGIN

 -- Get the cell for 26-NOV-1996 from ACME's open and display it
 tstDate := TO_DATE('11/26/1996');

 SELECT ORDSYS.TimeSeries.Display(
    ORDSYS.TimeSeries.GetDatedElement(ts.open, tstDate),
                                    'GetDatedElement Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts
 WHERE ts.ticker='ACME';

END;
/
```

This example might produce the following output:

```
GetDatedElement Results :

   Timestamp : 11/26/1996 00:00:00
       Value : 76
```

GetNthElement

Format

```
ORDSYS.TimeSeries.GetNthElement
    (ts ORDSYS.ORDTNumSeriesIOTRef,
    target_index INTEGER
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumCell;
```

Description

Given a time series, a number (*target_index*), and optionally a date range, returns the Nth element (element whose position corresponds to *target_index*) in the specified time series, or within the date range if one is specified.

Parameters

ts
The input time series.

target_index
Positive integer specifying the position of the element to be returned.

startDate
Starting date within the time series to which *target_index* is to be applied. If *target_index* = 1, the function returns the element for *startDate*. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series to which *target_index* is to be applied. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *target_index* is not an integer, or is zero (0) or a negative number.

- *endDate* is earlier than *startDate*.

Example

Return the tenth opening price for stock ACME:

```
SELECT ORDSYS.TimeSeries.ExtractValue(
         ORDSYS.TimeSeries.GetNthElement(open, 10))
  FROM TSDEV.stockdemo_ts
  WHERE ticker = 'ACME';
```

This example might produce the following output:

```
ORDSYS.TIM
----------
        68
1 row selected.
```

GetSeries

Format

```
ORDSYS.TimeSeries.GetSeries(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef
    ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.GetSeries(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTVarchar2SeriesIOTRef
    ) RETURN ORDSYS.ORDTVarchar2Series;
```

Description

Given a reference to a time series of references (ORDTNumSeriesIOTRef or ORDTVarchar2SeriesIOTRef), returns a time series instance (ORDTNumSeries or ORDTVarchar2Series).

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

Usage

The function materializes the input time series.

An exception is returned if the time series (*ts*) is null.

Example

Return an instance of a specified time series (opening prices for stock ACME):

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
dummyval INTEGER;

BEGIN

 SELECT ORDSYS.TimeSeries.Display(
     ORDSYS.TimeSeries.GetSeries(ts.open), 'GetSeries Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts
 WHERE ts.ticker='ACME';

END;
/
```

This example might produce the following output:

```
GetSeries Results :

Name = OPEN ACME
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 11/14/1996 00:00:00      68
```

```
11/15/1996 00:00:00      69
11/18/1996 00:00:00      70
11/19/1996 00:00:00      71
11/20/1996 00:00:00      72
11/21/1996 00:00:00      73
11/22/1996 00:00:00      74
11/25/1996 00:00:00      75
11/26/1996 00:00:00      76
11/27/1996 00:00:00      77
11/29/1996 00:00:00      78
12/02/1996 00:00:00      79
12/03/1996 00:00:00      80
12/04/1996 00:00:00      81
12/05/1996 00:00:00      82
12/06/1996 00:00:00      83
12/09/1996 00:00:00      84
12/10/1996 00:00:00      85
12/11/1996 00:00:00      86
12/12/1996 00:00:00      87
12/13/1996 00:00:00      88
12/16/1996 00:00:00      89
12/17/1996 00:00:00      90
12/18/1996 00:00:00      91
12/19/1996 00:00:00      92
12/20/1996 00:00:00      93
12/23/1996 00:00:00      94
12/24/1996 00:00:00      95
12/26/1996 00:00:00      96
12/27/1996 00:00:00      97
12/30/1996 00:00:00      98
12/31/1996 00:00:00      99
-----------------------------
```

IsValidTS

Format

```
ORDSYS.TimeSeries.IsValidTS(
    ts  ORDSYS.ORDTNumSeriesIOTRef
    ) RETURN INTEGER;
```

or

```
ORDSYS.TimeSeries.IsValidTS(
    ts  ORDSYS.ORDTVarchar2SeriesIOTRef
    ) RETURN INTEGER;
```

Description

Returns 1 if the time series is valid and 0 if the time series is invalid.

Parameters

ts
The input time series.

Usage

A time series is invalid if one or more of the following conditions are true:

- The time series (*ts*) is null.
- The time series (*ts*) does not have an associated calendar.
- The calendar associated with the time series is invalid.
- The timestamps are not sorted.
- One or more timestamps are null, imprecise, or outside the date range of the calendar.
- One or more timestamps are included in the time series but should be excluded based on the calendar definition (for example, a Saturday timestamp that is in a Monday-Friday calendar and that is not an on-exception).

- One or more timestamps are excluded from the time series but should be included based on the calendar definition (for example, a Wednesday date that is not a holiday in a Monday-Friday calendar and for which there is no data). Such dates can be considered as "holes" in the time series.

Contrast this function with ValidateTS, which checks whether a time series is valid, and if the time series is not valid, generates a diagnostic message and tables with timestamps that are causing the time series to be invalid.

Example

Use the IsValidTS and ValidateTS functions and the DisplayValTS procedure with an invalid time series:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = ’MM/DD/YYYY HH24:MI:SS’;

DECLARE
 numTS  ORDSYS.ORDTNumSeries;
 tempVal integer;
 retIsValid  integer;
 retValTS    integer;
 loDateTab  ORDSYS.ORDTDateTab := NULL;
 hiDateTab  ORDSYS.ORDTDateTab := NULL;
 impDateTab ORDSYS.ORDTDateTab := NULL;
 dupDateTab ORDSYS.ORDTDateTab := NULL;
 extraDateTab ORDSYS.ORDTDateTab := NULL;
 missingDateTab ORDSYS.ORDTDateTab := NULL;
 outMesg varchar2(2000);

BEGIN

   --  Set the buffer size
   DBMS_OUTPUT.ENABLE(100000);

   --
   -- NOTE: Here, an instance of the time series is materialized
   -- so that it could be modified to generate an invalid time series.
   --
   SELECT ORDSYS.TIMESERIES.GetSeries(ts.open) INTO numTS
   FROM tsdev.stockdemo_ts ts
   WHERE ts.ticker = ’ACME’;

   -- Example of validating a valid time series.
   SELECT ordsys.timeseries.display(numTS, ’A VALID TIME SERIES’) INTO tempVal
```

```
FROM dual;
retIsValid := ORDSYS.TIMESERIES.IsValidTS(numTS);
retValTS := ORDSYS.TIMESERIES.ValidateTS(numTS, outMesg, loDateTab,
                          hiDateTab, impDateTab, dupDateTab,
                          extraDateTab, missingDateTab);
DBMS_OUTPUT.PUT_LINE('Value returned by IsValid  = ' || retIsValid);
DBMS_OUTPUT.PUT_LINE('Value returned by ValidateTS  = ' || retValTS);
ORDSYS.TIMESERIES.DisplayValTS(retValTS, outMesg, loDateTab, hiDateTab,
                       impDateTab, dupDateTab, extraDateTab, missingDateTab,
                       'Testing DisplayValTS');
DBMS_OUTPUT.NEW_LINE;

-- For illustration let us first create an invalid timeseries.
--
-- Here we are adjusting the calendar's minDate and maxDate to avoid
-- getting a huge list of missing dates.
--
numTS.cal.minDate := TO_DATE('10/28/1996');
numTS.cal.maxDate := TO_DATE('01/05/1997');

-- Add Dates Before numTS.cal.minDate
numTS.series(10).tstamp := numTS.cal.minDate - 1;
numTS.series(11).tstamp := numTS.cal.minDate - 2;

-- Add Dates Beyond numTS.cal.maxDate
numTS.series(12).tstamp := numTS.cal.maxDate + 1;
numTS.series(13).tstamp := numTS.cal.maxDate + 2;

-- Add some null timestamps
numTS.series(14).tstamp := NULL;
numTS.series(15).tstamp := NULL;

-- Add some imprecise dates (some are duplicated)
numTS.series(17).tstamp := numTS.series(16).tstamp + 1/24;
numTS.series(18).tstamp := numTS.series(16).tstamp + 15/24;

-- Add some duplicate timestamps
numTS.series(19).tstamp := numTS.series(18).tstamp;
numTS.series(21).tstamp := numTS.series(20).tstamp;

-- Add some extra dates in the middle
numTS.series(37).tstamp := TO_DATE('12/28/1996');
numTS.series(36).tstamp := TO_DATE('12/29/1996');

-- Add some holes at the end
```

```
   numTS.series(numTS.series.count).tstamp := TO_DATE('01/04/1997');

   -- Example of validating an invalid time series.
   SELECT ordsys.timeseries.display(numTS, 'AN INVALID TIME SERIES')
   INTO tempVal FROM dual;
   retIsValid := ORDSYS.TIMESERIES.IsValidTS(numTS);
   retValTS := ORDSYS.TIMESERIES.ValidateTS(numTS, outMesg,
                      loDateTab, hiDateTab, impDateTab,
                      dupDateTab, extraDateTab, missingDateTab);
   DBMS_OUTPUT.PUT_LINE('Value returned by IsValid  = ' || retIsValid);
   DBMS_OUTPUT.PUT_LINE('Value returned by ValidateTS  = ' || retValTS);
   ORDSYS.TIMESERIES.DisplayValTS(retValTS, outMesg, loDateTab, hiDateTab,
                   impDateTab, dupDateTab, extraDateTab, missingDateTab,
                   'Testing DisplayValTS');
END;
/
```

This example might produce the following output:

```
A VALID TIME SERIES :

Name = OPEN ACME
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 11/14/1996 00:00:00      68
 11/15/1996 00:00:00      69
 11/18/1996 00:00:00      70
 11/19/1996 00:00:00      71
 11/20/1996 00:00:00      72
 11/21/1996 00:00:00      73
 11/22/1996 00:00:00      74
 11/25/1996 00:00:00      75
 11/26/1996 00:00:00      76
 11/27/1996 00:00:00      77
 11/29/1996 00:00:00      78
 12/02/1996 00:00:00      79
 12/03/1996 00:00:00      80
 12/04/1996 00:00:00      81
 12/05/1996 00:00:00      82
 12/06/1996 00:00:00      83
 12/09/1996 00:00:00      84
 12/10/1996 00:00:00      85
 12/11/1996 00:00:00      86
 12/12/1996 00:00:00      87
 12/13/1996 00:00:00      88
 12/16/1996 00:00:00      89
 12/17/1996 00:00:00      90
 12/18/1996 00:00:00      91
 12/19/1996 00:00:00      92
 12/20/1996 00:00:00      93
 12/23/1996 00:00:00      94
 12/24/1996 00:00:00      95
 12/26/1996 00:00:00      96
 12/27/1996 00:00:00      97
 12/30/1996 00:00:00      98
 12/31/1996 00:00:00      99
 -----------------------------

Value returned by IsValid  = 1
Value returned by ValidateTS  = 1

DisplayValTS: Testing DisplayValTS:

TS-SUC: the input time series is a valid time series



AN INVALID TIME SERIES :
```

```
Name = OPEN ACME
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 10/28/1996 00:00:00
 MaxDate = 01/05/1997 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                     Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 10/27/1996 00:00:00      68
 10/26/1996 00:00:00      69
 01/06/1997 00:00:00      70
 01/07/1997 00:00:00      71
       72
       73
 11/22/1996 00:00:00      74
 11/22/1996 01:00:00      75
 11/22/1996 15:00:00      76
 11/22/1996 15:00:00      77
 11/29/1996 00:00:00      78
 11/29/1996 00:00:00      79
 12/03/1996 00:00:00      80
 12/04/1996 00:00:00      81
 12/05/1996 00:00:00      82
 12/06/1996 00:00:00      83
 12/09/1996 00:00:00      84
 12/10/1996 00:00:00      85
 12/11/1996 00:00:00      86
 12/12/1996 00:00:00      87
 12/13/1996 00:00:00      88
 12/16/1996 00:00:00      89
 12/17/1996 00:00:00      90
 12/18/1996 00:00:00      91
 12/19/1996 00:00:00      92
 12/20/1996 00:00:00      93
 12/29/1996 00:00:00      94
 12/28/1996 00:00:00      95
 12/26/1996 00:00:00      96
 12/27/1996 00:00:00      97
 12/30/1996 00:00:00      98
 01/04/1997 00:00:00      99
 -----------------------------

Value returned by IsValid  = 0
Value returned by ValidateTS  = 0

DisplayValTS: Testing DisplayValTS:

TS-WRN: the input time series has errors. See the message for details

message output by validateTS:

TS-ERR: the input time series is unsorted
TS-ERR: the time series has null timestamps
TS-ERR: the time series has timestamps < calendar minDate (refer LoDateTab)
TS-ERR: the time series has timestamps > calendar maxDate (refer HiDateTab)
TS-ERR: the time series has imprecise timestamps (refer impreciseDateTab)
TS-ERR: the time series has duplicate timestamps (refer DuplicateDateTab)

list of dates < calendar minDate - lowDateTab :

     10/26/1996 00:00:00     10/27/1996 00:00:00

list of dates > calendar maxDate - hiDateTab :

     01/06/1997 00:00:00     01/07/1997 00:00:00

list of imprecise dates - impreciseDateTab :

     11/22/1996 01:00:00     11/22/1996 15:00:00

list of duplicate dates - duplicateDateTab :

     11/22/1996 15:00:00     11/29/1996 00:00:00
```

```
ExtraDateTab :

     12/28/1996 00:00:00     12/29/1996 00:00:00     01/04/1997 00:00:00

MissingDateTab :

     10/28/1996 00:00:00     10/29/1996 00:00:00     10/30/1996 00:00:00
     10/31/1996 00:00:00     11/14/1996 00:00:00     11/15/1996 00:00:00
     11/18/1996 00:00:00     11/19/1996 00:00:00     11/20/1996 00:00:00
     11/21/1996 00:00:00     11/25/1996 00:00:00     11/26/1996 00:00:00
     11/27/1996 00:00:00     12/02/1996 00:00:00     12/23/1996 00:00:00
     12/24/1996 00:00:00     12/31/1996 00:00:00     01/01/1997 00:00:00
     01/02/1997 00:00:00     01/03/1997 00:00:00
```

Lag

Format

```
ORDSYS.TimeSeries.Lag (
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    units INTEGER
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.Lag (
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    lead_date DATE
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a positive or negative number (*units*) or a date (*lead_date*), and optionally a starting and ending timestamp within the time series, returns a time series that lags or (for negative numeric values) leads the input time series by the appropriate number of timestamps.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

units
Integer specifying the number of timestamps by which the output time series is to be adjusted. If *units* is positive, each element in the output time series is the same as the element in the input time series for that relative position minus the *units*. If *units* is negative, each element in the output time series is the same as the element in the input time series for that relative position plus the *units*.

lead_date
The date relative to the starting date reflecting the number of timestamps by which the output time series is to be adjusted. The function calculates the number of timestamps between *lead_date* and *startDate*, and then uses that number as if it were a *units* parameter value. (If *lead_date* is later than *startDate*, the effective units value is positive; if *lead_date* is before the starting date, the effective units value is negative.)

startDate
Starting date to be used in calculating the lead or lag value; also the starting date in the input time series for which the output time series is to be created. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date in the input time series for which the output time series is to be created. If *endDate* is specified, *startDate* must also be specified.

Usage

The function creates a time series whose elements reflect an input time series adjusted by a number of timestamps. For example, using the United States stock trading calendar for 1997, if the first timestamp in the input time series is 06-Jan-1997 (Monday) and the units value is 2, the first timestamp in the output time series is 02-Jan-1997 (Thursday) and its associated value (such as closing price) is the same as that for 06-Jan-1997 in the input time series. Subsequent elements of the output time series reflect the timestamp adjustment.

For example, assuming the United States stock trading calendar for 1997, Table 5–1 shows some time series data with a two-day lag period.

Table 5–1 Lagging a Time Series by Two Days

Input Time Series:		Output Time Series:	
Timestamp	**Closing Price**	**Timestamp**	**Closing Price**
06-Jan-1997	49.50	02-Jan-1997	49.50
07-Jan-1997	49.25	03-Jan-1997	49.25
08-Jan-1997	50.00	06-Jan-1997	50.00
...	...	...	...

For convenience, both the Lead and Lag functions are provided.The functions operate identically, except that they interpret the sign of the *units* value in opposite ways. For example, Lead with -10 for *units* is equivalent to Lag with 10 for *units*. Moreover, because of the way the *lead_date* parameter is interpreted, Lead and Lag with a *lead_date* operate identically.

The Lead and Lag functions do not operate on irregular time series. For an explanation of irregular time series, see Section 2.1.1.

Example

Return a time series starting with 03-Mar-1997 using closing prices from the time series from 01-Nov-1996 through 30-Nov-1996 for stock ACME. The returned time series has the same number of timestamps as are in the specified date range (*startDate* through *endDate*).

```
SELECT to_char(tstamp) tstamp, value
FROM stockdemo_ts ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Lag(ts.close,
               to_date('03-MAR-97','DD-MON-YY'),
               to_date('01-NOV-96','DD-MON-YY'),
               to_date('30-NOV-96','DD-MON-YY'))
           ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
03-MAR-97         59
```

```
04-MAR-97         60
05-MAR-97         61
06-MAR-97         62
07-MAR-97         63
10-MAR-97         64
  ...            ...
27-MAR-97         77
28-MAR-97         78
20 rows selected.
```

Last

Format

ORDSYS.TimeSeries.Last(

    ts ORDSYS.ORDTNumSeriesIOTRef

) RETURN ORDSYS.ORDTNumCell;

Description

Given a time series, returns the last element in it.

Parameters

ts
The input time series.

Usage

A null is returned if the time series (*ts*) is empty.

An exception is returned if the time series (*ts*) is null.

Example

Return the last timestamp and opening price for stock ACME in the *stockdemo_ts* time series:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
dummyval INTEGER;

BEGIN

 SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.Last(ts.open), 'Last Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts
 WHERE ts.ticker='ACME';

END;
```

```
/
```

This example might produce the following output:

```
Last Results :

   Timestamp : 12/31/1996 00:00:00
       Value : 99
```

LastN

Format

```
ORDSYS.TimeSeries.LastN(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    NumValues NUMBER
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series and a number of elements (*NumValues*) to return, returns the last *NumValues* elements in the time series.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

NumValues
Number of elements from the end of the time series to be returned.

startDate
Starting date within the time series for which *NumValues* elements are to be returned. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which *NumValues* elements are to be returned. If *endDate* is specified, *startDate* must also be specified.

Usage

The function returns a time series populated with the last *NumValues* cells from the input time series (*ts*). The calendar of the output time series is the same as that of the input time series.

An exception is returned if the time series (*ts*) is null, if *NumValues* is zero (0) or negative, or if *endDate* is earlier than *startDate*.

If *startDate* and *endDate* are specified, the time series is trimmed to the date range before the last *NumValues* cells are returned.

Example

Return the last 10 timestamps and opening prices in the time series for stock ACME.:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
dummyval INTEGER;

BEGIN

 SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.LastN(ts.open, 10), 'LastN Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts
 WHERE ts.ticker='ACME';

END;
/
```

This example might produce the following output:

```
LastN Results :

Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 12/17/1996 00:00:00      90
 12/18/1996 00:00:00      91
 12/19/1996 00:00:00      92
 12/20/1996 00:00:00      93
 12/23/1996 00:00:00      94
 12/24/1996 00:00:00      95
 12/26/1996 00:00:00      96
 12/27/1996 00:00:00      97
 12/30/1996 00:00:00      98
 12/31/1996 00:00:00      99
 -----------------------------
```

Lead

Format

```
ORDSYS.TimeSeries.Lead (
      [tsname VARCHAR2,]
      ts ORDSYS.ORDTNumSeriesIOTRef,
      units INTEGER
      [, startDate DATE
      , endDate DATE]
      ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.Lead (
      [tsname VARCHAR2,]
      ts ORDSYS.ORDTNumSeriesIOTRef,
      lead_date DATE
      [, startDate DATE
      , endDate DATE]
      ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a positive or negative number (*units*) or a date (*lead_date*), and optionally a starting and ending timestamp within the time series, returns a time series that leads or (for negative numeric values) lags the input time series by the appropriate number of timestamps.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

units
Integer specifying the number of timestamps by which the output time series is to be adjusted. If *units* is positive, each element in the output time series is the same as the element in the input time series for that relative position plus the *units*. If *units* is negative, each element in the output time series is the same as the element in the input time series for that relative position minus the *units*.

lead_date
The date relative to the starting date reflecting the number of timestamps by which the output time series is to be adjusted. The function calculates the number of timestamps between *lead_date* and *startDate*, and then uses that number as if it were a *units* parameter value. (If *lead_date* is later than *startDate*, the effective *units* value is positive; if *lead_date* is before *startDate*, the effective *units* value is negative.)

startDate
Starting date to be used in calculating the lead or lag value; also the starting date in the input time series for which the output time series is to be created. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date in the input time series for which the output time series is to be created. If *endDate* is specified, *startDate* must also be specified.

Usage

The function creates a time series whose elements reflect an input time series adjusted by a number of timestamps. For example, using the United States stock trading calendar for 1997, if the first timestamp in the input time series is 02-Jan-1997 (Thursday) and the units value is 2, the first timestamp in the output time series is 06-Jan-1997 (Monday) and its associated value (such as closing price) is the same as that for 02-Jan-1997 in the input time series. Subsequent elements of the output time series reflect the timestamp adjustment.

For example, assuming the United States stock trading calendar for 1997, Table 5–2 shows some time series data with a two-day lead period.

Table 5–2 Leading a Time Series by Two Days

Input Time Series:		Output Time Series:	
Timestamp	**Closing Price**	**Timestamp**	**Closing Price**
02-Jan-1997	49.00	06-Jan-1997	49.00
03-Jan-1997	50.00	07-Jan-1997	50.00
06-Jan-1997	49.50	08-Jan-1997	49.50
...	...	...	...

For convenience, both the Lead and Lag functions are provided. The functions operate identically, except that they interpret the sign of the *units* value in opposite ways. For example, Lead with -10 for *units* is equivalent to Lag with 10 for *units*. Moreover, because of the way the *lead_date* parameter is interpreted, Lead and Lag with a *lead_date* operate identically.

The Lead and Lag functions do not operate on irregular time series. For an explanation of irregular time series, see Section 2.1.1.

Example

Return a time series starting with 03-Mar-1997 using closing prices from the time series from 01-Nov-1996 through 30-Nov-1996 for stock ACME. The returned time series has the same number of timestamps as are in the specified date range (*startDate* through *endDate*).

```
SELECT to_char(tstamp) tstamp, value
FROM stockdemo_ts ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Lead(ts.close,
               to_date('03-MAR-97','DD-MON-YY'),
               to_date('01-NOV-96','DD-MON-YY'),
               to_date('30-NOV-96','DD-MON-YY'))
           ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
03-MAR-97         59
```

```
04-MAR-97         60
05-MAR-97         61
06-MAR-97         62
07-MAR-97         63
10-MAR-97         64
  ...            ...
27-MAR-97         77
28-MAR-97         78
20 rows selected.
```

Mavg

Format

```
ORDSYS.TimeSeries.Mavg(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    k INTEGER
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given an input ORDTNumSeries, returns a moving average for the time series, or for the date range if one is specified. Each value in the returned time series is the average of the value for the current timestamp plus the value for each of the previous specified number of timestamps minus one.

For example, a 30-day moving average of closing prices for a stock on any given date is the average of that day's closing price and the 29 preceding closing prices.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

k
Positive integer specifying the lookback window (number of timestamps, including the current one, over which to compute the moving average).

startDate
Starting date within the time series for which to return moving averages. If *startDate* is specified, *endDate* must also be specified.

endDate

Ending date within the time series for which to return moving averages. If *endDate* is specified, *startDate* must also be specified.

Usage

The returned time series has nulls for any entry where there are not *k-1* timestamps preceding it in the calendar. For example, if a stock trading calendar for 1997 starts on 02-Jan-1997, the series of 5-day moving averages of the closing price for a stock for the year has nulls for the closing price for the first four timestamps (02-Jan, 03-Jan, 06-Jan, and 07-Jan), because there are insufficient timestamps for computing the average.

Any nulls in the entries for the *k* timestamps are ignored, as explained in Section 2.4.1.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return a table of 10-day moving average values of the closing price for stock ACME for the month of December 1996:

```
SELECT to_char(tstamp) tstamp, value
FROM tsquick ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
             ORDSYS.TimeSeries.Mavg(ts.close,to_date('01-DEC-96','DD-MON-YY'),
             to_date('31-DEC-96','DD-MON-YY'),10)
       ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
02-DEC-96       74.5
03-DEC-96       75.5
04-DEC-96       76.5
05-DEC-96       77.5
06-DEC-96       78.5
09-DEC-96       79.5
10-DEC-96       80.5
```

```
11-DEC-96       81.5
12-DEC-96       82.5
13-DEC-96       83.5
16-DEC-96       84.5
17-DEC-96       85.5
18-DEC-96       86.5
19-DEC-96       87.5
20-DEC-96       88.5
23-DEC-96       89.5
24-DEC-96       90.5
26-DEC-96       91.5
27-DEC-96       92.5
30-DEC-96       93.5
31-DEC-96       94.5
21 rows selected.
```

Msum

Format

```
ORDSYS.TimeSeries.Msum(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    k INTEGER
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given an input ORDTNumSeries, returns a moving sum for the time series, or for the date range if one is specified. Each value in the returned time series is the sum of the value for the current timestamp plus the value for each of the previous specified number of timestamps minus one.

For example, a 30-day moving sum for a stock's daily trading volume on any given date is the sum of that day's volume and the 29 preceding daily volumes.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

k
Positive integer specifying the lookback window (number of timestamps, including the current one, over which to compute the moving sum).

startDate
Starting date within the time series for which to return moving sums. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which to return moving sums. If *endDate* is specified, *startDate* must also be specified.

Usage

The returned time series has nulls for any entry where there are not *k-1* timestamps preceding it in the calendar. For example, if a stock trading calendar for 1997 starts on 02-Jan-1997, the series of 5-day moving sums of the trading volume for a stock for the year has nulls for the volume for the first four timestamps (02-Jan, 03-Jan, 06-Jan, and 07-Jan), because there are insufficient timestamps for computing the sum.

Any nulls in the entries for the *k* timestamps are ignored, as explained in Section 2.4.1.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return a table of 30-day moving sum values of trading volume for stock ACME for December 1996:

```
SELECT to_char(tstamp) tstamp, value
FROM tsquick ts,
    TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
            ORDSYS.TimeSeries.Msum(ts.volume,to_date('01-DEC-96','DD-MON-YY'),
            to_date('31-DEC-96','DD-MON-YY'),30)
      ) AS ORDSYS.ORDTNumTab)) t
WHERE ts.ticker='ACME';
```

With the simplified data in the demo database (where all ACME daily volumes are 1000 and there are no ACME timestamps before November 1996), this example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
02-DEC-96
03-DEC-96
04-DEC-96
05-DEC-96
06-DEC-96
09-DEC-96
```

```
10-DEC-96
11-DEC-96
12-DEC-96
13-DEC-96      30000
16-DEC-96      30000
17-DEC-96      30000
18-DEC-96      30000
19-DEC-96      30000
20-DEC-96      30000
23-DEC-96      30000
24-DEC-96      30000
26-DEC-96      30000
27-DEC-96      30000
30-DEC-96      30000
31-DEC-96      30000
21 rows selected.
```

TrimSeries

Format

```
ORDSYS.TimeSeries.TrimSeries(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    startDate DATE,
    endDate DATE
    ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.TrimSeries(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTVarchar2SeriesIOTRef,
    startDate DATE,
    endDate DATE
    ) RETURN ORDSYS.ORDTVarchar2Series;
```

Description

Given an input ORDT series, returns an ORDT series of the same type with all data outside of the given date range removed. The calendar of the returned series will be the same as that of the original series.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

startDate
Starting date within the time series. You must specify a value, either null or not null. If you specify a null value, the starting date (*minDate*) of the calendar (if any) associated with *ts* is used.

endDate
Ending date within the time series. You must specify a value, either null or not null. If you specify a null value, the ending date (*maxDate*) of the calendar (if any) associated with *ts* is used.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the opening prices for stock ACME for dates in the calendar from 01-Dec-1996 through 31-Dec-1996:

```
SET SERVEROUTPUT ON
DECLARE
 tmp  INTEGER;
 tstDate1  DATE;
 tstDate2  DATE;
BEGIN
-- Set tstDate values
    tstDate1 := TO_DATE('12/01/1996 00:00:00','MM/DD/YYYY HH24:MI:SS');
    tstDate2 := TO_DATE('12/31/1996 00:00:00','MM/DD/YYYY HH24:MI:SS');
    SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.TrimSeries(open, tstDate1, tstDate2))
      INTO tmp
    FROM TSDEV.stockdemo_ts
    WHERE ticker = 'ACME';
END;
/
```

This statement might produce the following output:

```
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 01-JAN-90 00:00:00
 MaxDate = 01-JAN-01 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 07-JAN-96 00:00:00
 onExceptions  :
 offExceptions :
     28-NOV-96 00:00:00     25-DEC-96 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 02-DEC-96 00:00:00      79
 03-DEC-96 00:00:00      80
 04-DEC-96 00:00:00      81
 05-DEC-96 00:00:00      82
 06-DEC-96 00:00:00      83
 09-DEC-96 00:00:00      84
 10-DEC-96 00:00:00      85
 11-DEC-96 00:00:00      86
 12-DEC-96 00:00:00      87
 13-DEC-96 00:00:00      88
 16-DEC-96 00:00:00      89
 17-DEC-96 00:00:00      90
 18-DEC-96 00:00:00      91
 19-DEC-96 00:00:00      92
 20-DEC-96 00:00:00      93
 23-DEC-96 00:00:00      94
 24-DEC-96 00:00:00      95
 26-DEC-96 00:00:00      96
 27-DEC-96 00:00:00      97
 30-DEC-96 00:00:00      98
 31-DEC-96 00:00:00      99
 -----------------------------
```

TSAdd

Format

```
ORDSYS.TimeSeries.TSAdd (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    ts2 ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.TSAdd (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    k NUMBER
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the addition of the first two parameters.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts1
The time series (or first time series) whose elements are to be added either to corresponding elements in the second time series or to a constant.

ts2
The time series whose elements are to be added to corresponding elements in the first time series.

k
A constant to be added to corresponding elements in the first time series.

startDate
Starting date within the time series for which the addition is to be performed. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the addition is to be performed. If *endDate* is specified, *startDate* must also be specified.

Usage

The function performs a pairwise addition operation on each element of the time series. This operation determines the value of each element of the returned time series. For example:

- If two time series contain daily trade volumes for two stocks, each element of the returned time series contains the sum of the trade volumes for the two stocks for that day.
- If a time series (*ts1*) contains closing prices for a stock and if a constant (*k*) of 1 is specified, each element of the returned time series contains the closing price of *ts1* incremented by 1.

If *ts1* and *ts2* are specified, the function returns a time series whose calendar is the result of using the CombineCals function on the calendars associated with these two time series.

An exception is returned if one or more of the following conditions are true:

- An input time series is null.
- The calendars associated with *ts1* and *ts2* do not have the same frequency and aligned pattern.
- *endDate* is earlier than *startDate*.

Example

Add the high price for stock ACME and the low price for stock FUNCO for each trading day from 14-Nov-1996 through 14-Dec-1996:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
startDate date;
endDate   date;
dummyval INTEGER;

BEGIN

 startDate := TO_DATE('11/14/1996');
 endDate   := TO_DATE('12/14/1996');
 SELECT ORDSYS.TimeSeries.Display(
           ORDSYS.TimeSeries.TSAdd(ts1.high, ts2.low, startDate, endDate),
           'TSAdd Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts1, TSDEV.stockdemo_ts ts2
 WHERE ts1.ticker='ACME' and ts2.ticker='FUNCO';

END;
/
```

This example might produce the following output:

```
TSAdd Results :

Calendar Data:
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/14/1996 00:00:00      92.87
 11/15/1996 00:00:00      93.84
 11/18/1996 00:00:00      94.87
 11/19/1996 00:00:00      95.85
 11/20/1996 00:00:00      96.82
```

```
11/21/1996 00:00:00      97.84
11/22/1996 00:00:00      98.85
11/25/1996 00:00:00      99.81
11/26/1996 00:00:00      100.78
11/27/1996 00:00:00      101.71
11/29/1996 00:00:00      102.75
12/02/1996 00:00:00      103.88
12/03/1996 00:00:00      105.03
12/04/1996 00:00:00      106.02
12/05/1996 00:00:00      107.13
12/06/1996 00:00:00      107.75
12/09/1996 00:00:00      108.77
12/10/1996 00:00:00      109.8
12/11/1996 00:00:00      110.5
12/12/1996 00:00:00      111.41
12/13/1996 00:00:00      112.4
-----------------------------
```

TSAvg

Format

```
ORDSYS.TimeSeries.TSAvg (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the average of all non-null time series entries.

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the average is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the average is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the average, variance, and standard deviation of the closing price of stock ACME:

```
--
```

```
-- Compute various aggregate statistics.
--
SELECT ORDSYS.TimeSeries.TSAvg(close), ORDSYS.TimeSeries.TSVariance(close),
ORDSYS.TimeSeries.TSStdDev(close)
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
ORDSYS.TIM ORDSYS.TIM ORDSYS.TIM
---------- ---------- ----------
        79      143.5 11.9791486
1 row selected.
```

TSCount

Format

```
ORDSYS.TimeSeries.TSCount (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the count of all non-null time series entries.

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the count is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the count is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

Nulls are ignored in computing the count.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the total number of daily closing prices for stock ACME for the month of November 1996:

```
SELECT ORDSYS.TimeSeries.TSCount(close,
                        to_date('11/01/1996 00:00:00',
                                'MM/DD/YYYY HH24:MI:SS'),
                        to_date('11/30/1996 23:59:59',
                                'MM/DD/YYYY HH24:MI:SS')) TSCount
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
TSCOUNT
----------
        20
1 row selected.
```

TSDivide

Format

```
ORDSYS.TimeSeries.TSDivide (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    ts2 ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.TSDivide (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    k NUMBER
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the division of the first parameter by the second parameter.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts1
The time series (or first time series) whose elements are to be divided by either the corresponding elements in the second time series or a constant.

ts2
The time series whose elements are to be divided into corresponding elements in the first time series.

k
A constant to be divided into corresponding elements in the first time series.

startDate
Starting date within the time series for which the division is to be performed. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the division is to be performed. If *endDate* is specified, *startDate* must also be specified.

Usage

The function performs a pairwise division operation on each element of the time series (or first time series) by the corresponding element in the second time series or by a constant. This operation determines the value of each element of the returned time series. For example:

- If two time series contain daily trade volumes for two stocks, each element of the returned time series contains the result of dividing the volume in the first time series by the volume in the second time series for that day.
- If a time series (*ts1*) contains closing prices for a stock and if a constant (*k*) of 2 is specified, each element of the returned time series contains the closing price of *ts1* divided by 2.

If *ts1* and *ts2* are specified, the function returns a time series whose calendar is the result of using the CombineCals function on the calendars associated with these two time series.

An exception is returned if one or more of the following conditions are true:

- An input time series is null.
- The calendars associated with *ts1* and *ts2* do not have the same frequency and aligned pattern.
- *endDate* is earlier than *startDate*.

Example

Divide the high price for stock ACME by the low price for stock FUNCO for each trading day from 14-Nov-1996 through 14-Dec-1996:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
startDate date;
endDate   date;
dummyval INTEGER;

BEGIN

 startDate := TO_DATE('11/14/1996');
 endDate   := TO_DATE('12/14/1996');
 SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.TSDivide(ts1.high, ts2.low, startDate, endDate),
     'TSDivide Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts1, TSDEV.stockdemo_ts ts2
 WHERE ts1.ticker='ACME' and ts2.ticker='FUNCO';

END;
/
```

This example might produce the following output:

```
TSDivide Results :

Calendar Data:
 Frequency = 4 (day)
 MinDate = 11/01/1996 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/14/1996 00:00:00      2.890657729367406786761625471302890657 73
```

```
11/15/1996 00:00:00     2.9362416107382550335570469798657718l208
11/18/1996 00:00:00     2.9744449099287808965228320067029744449l
11/19/1996 00:00:00     3.0188679245283018867924528301886792452 8
11/20/1996 00:00:00     3.06465155331654072208228379513014273 72
11/21/1996 00:00:00     3.1040268456375838926174496644295302013 4
11/22/1996 00:00:00     3.144654088050314465408805031446540880 5
11/25/1996 00:00:00     3.1919361612767744645107097858042839143 2
11/26/1996 00:00:00     3.2380151387720773759461732548359966358 3
11/27/1996 00:00:00     3.2897511598481653310839308308730493462 7
11/29/1996 00:00:00     3.3263157894736842105263157894736842105 3
12/02/1996 00:00:00     3.3500837520938023450586264656616415410 4
12/03/1996 00:00:00     3.3707865168539325842696629213483146067 4
12/04/1996 00:00:00     3.4138218151540383014154870940882597835 1
12/05/1996 00:00:00     3.4397016162453377538334024036469125569 8
12/06/1996 00:00:00     3.5368421052631578947368421052631578947 4
12/09/1996 00:00:00     3.5759360538493899873790492217080353386 6
12/10/1996 00:00:00     3.6134453781512605042016806722689075630 3
12/11/1996 00:00:00     3.7021276595744680851063829787234042553 2
12/12/1996 00:00:00     3.7590773173857325929090132422041862451 9
12/13/1996 00:00:00     3.803418803418803418803418803418803418 8
-----------------------------
```

TSMax

Format

```
ORDSYS.TimeSeries.TSMax (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the highest (maximum) of all non-null time series entries.

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the maximum is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the maximum is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the highest closing price for stock ACME for the month of November 1996:

```
SELECT ORDSYS.TimeSeries.TSMax(close,
                      to_date('11/01/1996 00:00:00',
                              'MM/DD/YYYY HH24:MI:SS'),
                      to_date('11/30/1996 23:59:59',
                              'MM/DD/YYYY HH24:MI:SS')) TSMax
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
TSMAX
----------
        78
1 row selected.
```

TSMaxN

Format

```
ORDSYS.TimeSeries.TSMaxN (
    ts ORDSYS.ORDTNumSeriesIOTRef,
    NumValues INTEGER,
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumTab;
```

Description

Given an input ORDTNumSeries, a number of values to return, and optionally starting and ending dates, returns an ORDTNumTab with the specified number (*NumValues*) of the top (highest) values.

Parameters

ts
The input time series.

NumValues
Number of values to return.

startDate
Starting date within the time series for which the top values are to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the top values are to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

- *NumValues* is zero (0) or negative.

Example

Return the 10 highest closing prices for stock ACME for the month of November 1996:

```
SELECT * FROM THE( SELECT CAST(
                  ORDSYS.TimeSeries.TSMaxN(close, 10,
                         to_date('11011996','MMDDYYYY'),
                         to_date('11301996','MMDDYYYY'))
                  as ORDSYS.ORDTNumTab)
            FROM TSDEV.stockdemo_ts
            WHERE ticker ='ACME');
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
29-NOV-96         78
27-NOV-96         77
26-NOV-96         76
25-NOV-96         75
22-NOV-96         74
21-NOV-96         73
20-NOV-96         72
19-NOV-96         71
18-NOV-96         70
15-NOV-96         69
10 rows selected.
```

TSMedian

Format

```
ORDSYS.TimeSeries.TSMedian (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the median of all non-null time series entries.

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the median is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the median is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the median closing price for stock ACME for the month of November 1996:

```
SELECT ORDSYS.TimeSeries.TSMedian(close,
                        to_date('11/01/1996 00:00:00',
                                'MM/DD/YYYY HH24:MI:SS'),
                        to_date('11/30/1996 23:59:59',
                                'MM/DD/YYYY HH24:MI:SS')) TSMedian
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
TSMEDIAN
----------
      68.5
1 row selected.
```

TSMin

Format

ORDSYS.TimeSeries.TSMin (

ts ORDSYS.ORDTNumSeriesIOTRef

[,startDate DATE, endDate DATE]

) RETURN NUMBER;

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the lowest (minimum) of all non-null time series entries.

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the minimum is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the minimum is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the lowest closing price for stock ACME for the month of November 1996:

```
SELECT ORDSYS.TimeSeries.TSMin(close,
                        to_date('11/01/1996 00:00:00',
                                'MM/DD/YYYY HH24:MI:SS'),
                        to_date('11/30/1996 23:59:59',
                                'MM/DD/YYYY HH24:MI:SS')) TSMin
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
TSMIN
----------
        59
1 row selected.
```

TSMinN

Format

```
ORDSYS.TimeSeries.TSMinN (
    ts ORDSYS.ORDTNumSeriesIOTRef,
    NumValues INTEGER,
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumTab;
```

Description

Given an input ORDTNumSeries, a number of values to return, and optionally starting and ending dates, returns an ORDTNumTab with the specified number (*NumValues*) of the bottom (lowest) values.

Parameters

ts
The input time series.

NumValues
Number of values to return.

startDate
Starting date within the time series for which the bottom values are to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the bottom values are to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

- *NumValues* is zero (0) or negative.

Example

Return the 10 lowest closing prices for stock ACME for the month of November 1996:

```
SELECT * FROM THE( SELECT CAST(
                  ORDSYS.TimeSeries.TSMinN(close, 10,
                         to_date('11011996','MMDDYYYY'),
                         to_date('11301996','MMDDYYYY'))
                  as ORDSYS.ORDTNumTab)
            FROM TSDEV.stockdemo_ts
            WHERE ticker ='ACME');
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96         59
04-NOV-96         60
05-NOV-96         61
06-NOV-96         62
07-NOV-96         63
08-NOV-96         64
11-NOV-96         65
12-NOV-96         66
13-NOV-96         67
14-NOV-96         68
10 rows selected.
```

TSMultiply

Format

```
ORDSYS.TimeSeries.TSMultiply (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    ts2 ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.TSMultiply (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    k NUMBER
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the multiplication of the first parameter by the second parameter.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts1
The time series (or first time series) whose elements are to be multiplied by either the corresponding elements in the second time series or a constant.

ts2
The time series whose elements are to be multiplied by corresponding elements in the first time series.

k
A constant to be multiplied by corresponding elements in the first time series.

startDate
Starting date within the time series for which the multiplication is to be performed. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the multiplication is to be performed. If *endDate* is specified, *startDate* must also be specified.

Usage

The function performs a pairwise multiplication operation on each element of the time series (or first time series) by the corresponding element in the second time series or by a constant. This operation determines the value of each element of the returned time series. For example:

- If two time series contain daily trade volumes for two stocks, each element of the returned time series contains the result of multiplying the volume in the first time series by the volume in the second time series for that day.
- If a time series (*ts1*) contains closing prices for a stock and if a constant (*k*) of 2 is specified, each element of the returned time series contains the closing price of *ts1* multiplied by 2.

If *ts1* and *ts2* are specified, the function returns a time series whose calendar is the result of using the CombineCals function on the calendars associated with these two time series.

An exception is returned if one or more of the following conditions are true:

- An input time series is null.
- The calendars associated with *ts1* and *ts2* do not have the same frequency and aligned pattern.
- *endDate* is earlier than *startDate*.

Example

Multiply the high price for stock ACME by the low price for stock FUNCO for each trading day from 14-Nov-1996 through 14-Dec-1996:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
startDate date;
endDate   date;
dummyval INTEGER;

BEGIN

 startDate := TO_DATE('11/14/1996');
 endDate   := TO_DATE('12/14/1996');
 SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.TSMultiply(ts1.high, ts2.low, startDate, endDate),
     'TSMultiply Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts1, TSDEV.stockdemo_ts ts2
 WHERE ts1.ticker='ACME' and ts2.ticker='FUNCO';

END;
/
```

This example might produce the following output:

```
TSMultiply Results :

Calendar Data:
 Frequency = 4 (day)
 MinDate = 01/01/1990 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                   Value
 11/14/1996 00:00:00      1647.03
```

```
11/15/1996 00:00:00      1668.8
11/18/1996 00:00:00      1694.77
11/19/1996 00:00:00      1717.2
11/20/1996 00:00:00      1738.86
11/21/1996 00:00:00      1764.16
11/22/1996 00:00:00      1788.75
11/25/1996 00:00:00      1809.56
11/26/1996 00:00:00      1831.06
11/27/1996 00:00:00      1849.38
11/29/1996 00:00:00      1876.25
12/02/1996 00:00:00      1910.4
12/03/1996 00:00:00      1946.43
12/04/1996 00:00:00      1969.64
12/05/1996 00:00:00      2002.79
12/06/1996 00:00:00      1995
12/09/1996 00:00:00      2020.45
12/10/1996 00:00:00      2046.8
12/11/1996 00:00:00      2044.5
12/12/1996 00:00:00      2060.08
12/13/1996 00:00:00      2082.6
-----------------------------
```

TSProd

Format

```
ORDSYS.TimeSeries.TSProd (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the product (result of multiplication) of all non-null time series entries.

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the product is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the product is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the product resulting from multiplying the daily closing prices for stock ACME for the month of November 1996. (This example is not very plausible, but is presented merely to illustrate the syntax.)

```
SELECT ORDSYS.TimeSeries.TSProd(close,
                        to_date('11/01/1996 00:00:00',
                                'MM/DD/YYYY HH24:MI:SS'),
                        to_date('11/30/1996 23:59:59',
                                'MM/DD/YYYY HH24:MI:SS')) TSProd
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
TSPROD
----------
4.8177E+36
1 row selected.
```

TSStdDev

Format

```
ORDSYS.TimeSeries.TSStdDev (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the standard deviation of all non-null time series entries. (This function returns a value that is the square root of the value returned by the TSVar function.)

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the standard deviation is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the standard deviation is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

If the date range refers to a time series with fewer than two timestamps, a null is returned.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the average, variance, and standard deviation of the closing price of stock ACME:

```
--
-- Compute various aggregate statistics.
--
SELECT ORDSYS.TimeSeries.TSAvg(close), ORDSYS.TimeSeries.TSVariance(close),
ORDSYS.TimeSeries.TSStdDev(close)
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
ORDSYS.TIM ORDSYS.TIM ORDSYS.TIM
---------- ---------- ----------
        79      143.5 11.9791486
1 row selected.
```

TSSubtract

Format

```
ORDSYS.TimeSeries.TSSubtract (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    ts2 ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

or

```
ORDSYS.TimeSeries.TSSubtract (
    [tsname VARCHAR2,]
    ts1 ORDSYS.ORDTNumSeriesIOTRef,
    k NUMBER
    [,startDate DATE, endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the subtraction of the second parameter from the first parameter.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts1
The time series (or first time series) whose elements are to be decreased either by corresponding elements in the second time series or by a constant.

ts2
The time series whose elements are to be subtracted from corresponding elements in the first time series.

k
A constant to be subtracted from corresponding elements in the first time series.

startDate
Starting date within the time series for which the subtraction is to be performed. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the subtraction is to be performed. If *endDate* is specified, *startDate* must also be specified.

Usage

The function performs a pairwise subtraction operation on each element of *ts1*, decreasing it by either the corresponding element in *ts2* or by *k*. This operation determines the value of each element of the returned time series. For example:

- If two time series contain daily trade volumes for two stocks, each element of the returned time series contains the result of subtracting the *ts2* volume from the *ts1* volume for that day.
- If a time series (*ts1*) contains closing prices for a stock and if a constant (*k*) of 1 is specified, each element of the returned time series contains the closing price of *ts1* decreased by 1.

If *ts1* and *ts2* are specified, the function returns a time series whose calendar is the result of using the CombineCals function on the calendars associated with these two time series.

An exception is returned if one or more of the following conditions are true:

- An input time series is null.
- The calendars associated with *ts1* and *ts2* do not have the same frequency and aligned pattern.
- *endDate* is earlier than *startDate*.

Example

Subtract the low price for stock FUNCO from the high price for stock ACME for each trading day from 14-Nov-1996 through 14-Dec-1996:

```
CONNECT TSUSER/TSUSER
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';

DECLARE
tstCal ORDSYS.ORDTCalendar;
startDate date;
endDate   date;
dummyval INTEGER;

BEGIN

 startDate := TO_DATE('11/14/1996');
 endDate   := TO_DATE('12/14/1996');
 SELECT ORDSYS.TimeSeries.Display(
      ORDSYS.TimeSeries.TSSubtract(ts1.high, ts2.low, startDate, endDate),
     'TSSubtract Results') INTO dummyval
 FROM TSDEV.stockdemo_ts ts1, TSDEV.stockdemo_ts ts2
 WHERE ts1.ticker='ACME' and ts2.ticker='FUNCO';

END;
/
```

This example might produce the following output:

```
TSSubtract Results :

Calendar Data:
 Frequency = 4 (day)
 MinDate = 01/01/1990 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/14/1996 00:00:00      45.13
 11/15/1996 00:00:00      46.16
 11/18/1996 00:00:00      47.13
 11/19/1996 00:00:00      48.15
 11/20/1996 00:00:00      49.18
```

```
11/21/1996 00:00:00      50.16
11/22/1996 00:00:00      51.15
11/25/1996 00:00:00      52.19
11/26/1996 00:00:00      53.22
11/27/1996 00:00:00      54.29
11/29/1996 00:00:00      55.25
12/02/1996 00:00:00      56.12
12/03/1996 00:00:00      56.97
12/04/1996 00:00:00      57.98
12/05/1996 00:00:00      58.87
12/06/1996 00:00:00      60.25
12/09/1996 00:00:00      61.23
12/10/1996 00:00:00      62.2
12/11/1996 00:00:00      63.5
12/12/1996 00:00:00      64.59
12/13/1996 00:00:00      65.6
-----------------------------
```

TSSum

Format

```
ORDSYS.TimeSeries.TSSum (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the sum of all non-null time series entries.

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the sum is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the sum is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the sum of the daily trading volumes for stock ACME for the month of November 1996 (that is, the total ACME volume for the month):

```
SELECT ORDSYS.TimeSeries.TSSum(volume,
                      to_date('11/01/1996 00:00:00',
                              'MM/DD/YYYY HH24:MI:SS'),
                      to_date('11/30/1996 23:59:59',
                              'MM/DD/YYYY HH24:MI:SS')) TSSum
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
TSSUM
----------
     20000
1 row selected.
```

TSVariance

Format

```
ORDSYS.TimeSeries.TSVariance (
    ts ORDSYS.ORDTNumSeriesIOTRef
    [,startDate DATE, endDate DATE]
    ) RETURN NUMBER;
```

Description

Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the variance of all non-null time series entries. (This function is analogous to the SQL group function VAR.)

Parameters

ts
The input time series.

startDate
Starting date within the time series for which the variance is to be calculated. If *startDate* is specified, *endDate* must also be specified.

endDate
Ending date within the time series for which the variance is to be calculated. If *endDate* is specified, *startDate* must also be specified.

Usage

If the date range refers to a time series with fewer than two timestamps, a null is returned.

An exception is returned if one or more of the following conditions are true:

- The time series (*ts*) is null.
- *endDate* is earlier than *startDate*.

Example

Return the average, variance, and standard deviation of the closing price of stock ACME:

```
--
-- Compute various aggregate statistics.
--
SELECT ORDSYS.TimeSeries.TSAvg(close), ORDSYS.TimeSeries.TSVariance(close),
ORDSYS.TimeSeries.TSStdDev(close)
  FROM TSDEV.stockdemo_ts
  WHERE ticker='ACME';
```

This example might produce the following output:

```
ORDSYS.TIM ORDSYS.TIM ORDSYS.TIM
---------- ---------- ----------
        79      143.5 11.9791486
1 row selected.
```

ValidateTS

Format

```
ORDSYS.TimeSeries.ValidateTS(
    ts IN ORDSYS.ORDTNumSeriesIOTRef,
    outMesg OUT VARCHAR2,
    loDateTab OUT ORDSYS.ORDTDateTab,
    hiDateTab  OUT ORDSYS.ORDTDateTab,
    impreciseDateTab OUT ORDSYS.ORDTDateTab,
    duplicateDateTab OUT ORDSYS.ORDTDateTab,
    extraDateTab  OUT ORDSYS.ORDTDateTab,
    missingDateTab  OUT ORDSYS.ORDTDateTab
    ) RETURN INTEGER;
```

or

```
ORDSYS.TimeSeries.ValidateTS(
    ts IN ORDSYS.ORDTVarchar2SeriesIOTRef,
    outMesg OUT VARCHAR2,
    loDateTab OUT ORDSYS.ORDTDateTab,
    hiDateTab  OUT ORDSYS.ORDTDateTab,
    impreciseDateTab OUT ORDSYS.ORDTDateTab,
    duplicateDateTab OUT ORDSYS.ORDTDateTab,
    extraDateTab OUT ORDSYS.ORDTDateTab,
    missingDateTab OUT ORDSYS.ORDTDateTab
    ) RETURN INTEGER;
```

Description

Checks whether a time series is valid, and if the time series is not valid, generates a diagnostic message and tables with timestamps that are causing the time series to be invalid.

Parameters

ts
The time series to be checked for validity.

outMesg
If the time series is invalid (if the return value = 0), contains a diagnostic message describing any problems.

loDateTab
A table of dates before the starting date of the calendar associated with the time series.

hiDateTab
A table of dates after the ending date of the calendar associated with the calendar.

impreciseDateTab
A table of the imprecise timestamps found in the time series.

duplicateDateTab
A table of the duplicate timestamps found in the time series.

extraDateTab
A table of dates that are included in the time series but that should be excluded based on the calendar definition (for example, a Saturday timestamp that is in a Monday-Friday calendar and that is not an on-exception).

missingDateTab
A table of dates that are excluded from the time series but that should be included based on the calendar definition (for example, a Wednesday date that is not a holiday in a Monday-Friday calendar and for which there is no data). Such dates can be considered as "holes" in the time series.

Usage

The function returns one of the following values:

Value	Meaning
1	The time series is valid. No errors were found.
0	The time series in invalid.

A time series is invalid if one or more of the following conditions are true:

- The time series (*ts*) is null.
- The time series (*ts*) does not have an associated calendar.
- The calendar associated with the time series is invalid.
- The timestamps are not sorted.
- One or more timestamps are null, imprecise, or outside the date range of the calendar.
- One or more timestamps are included in the time series but should be excluded based on the calendar definition (for example, a Saturday timestamp that is in a Monday-Friday calendar and that is not an on-exception).
- One or more timestamps are excluded from the time series but should be included based on the calendar definition (for example, a Wednesday date that is not a holiday in a Monday-Friday calendar and for which there is no data). Such dates can be considered as "holes" in the time series.

Contrast this function with IsValidTS, which simply checks to determine if a time series is valid.

You can use the DisplayValTS procedure (documented in this chapter) to display the information returned by the ValidateTS function.

The ValidateTS function cannot be called from SQL. It must be called from PL/SQL because of the OUT parameters.

Example

Use the IsValidTS and ValidateTS functions and the DisplayValTS procedure with an invalid time series:

```
SET SERVEROUTPUT ON
ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI:SS';
```

```
DECLARE
 numTS  ORDSYS.ORDTNumSeries;
 tempVal integer;
 retIsValid  integer;
 retValTS    integer;
 loDateTab  ORDSYS.ORDTDateTab := NULL;
 hiDateTab  ORDSYS.ORDTDateTab := NULL;
 impDateTab ORDSYS.ORDTDateTab := NULL;
 dupDateTab ORDSYS.ORDTDateTab := NULL;
 extraDateTab ORDSYS.ORDTDateTab := NULL;
 missingDateTab ORDSYS.ORDTDateTab := NULL;
 outMesg varchar2(2000);

BEGIN

   --  Set the buffer size.
   DBMS_OUTPUT.ENABLE(100000);

   --
   -- NOTE: Here, an instance of the time series is materialized
   -- so that it could be modified to generate an invalid time series.
   --
   SELECT ORDSYS.TIMESERIES.GetSeries(ts.open) INTO numTS
   FROM tsdev.stockdemo_ts ts
   WHERE ts.ticker = 'ACME';

   -- Example of validating a valid time series.
   SELECT ordsys.timeseries.display(numTS, 'A VALID TIME SERIES') INTO tempVal
   FROM dual;
   retIsValid := ORDSYS.TIMESERIES.IsValidTS(numTS);
   retValTS := ORDSYS.TIMESERIES.ValidateTS(numTS, outMesg, loDateTab,
                              hiDateTab, impDateTab, dupDateTab,
                              extraDateTab, missingDateTab);
   DBMS_OUTPUT.PUT_LINE('Value returned by IsValid  = ' || retIsValid);
   DBMS_OUTPUT.PUT_LINE('Value returned by ValidateTS  = ' || retValTS);
   ORDSYS.TIMESERIES.DisplayValTS(retValTS, outMesg, loDateTab, hiDateTab,
                           impDateTab, dupDateTab, extraDateTab, missingDateTab,
                           'Testing DisplayValTS');
   DBMS_OUTPUT.NEW_LINE;

   -- For illustration let us first create an invalid timeseries.
   --
   -- Here we are adjusting the calendar's minDate and maxDate to avoid
   -- getting a huge list of missing dates.
   --
   numTS.cal.minDate := TO_DATE('10/28/1996');
   numTS.cal.maxDate := TO_DATE('01/05/1997');

   -- Add Dates Before numTS.cal.minDate
   numTS.series(10).tstamp := numTS.cal.minDate - 1;
   numTS.series(11).tstamp := numTS.cal.minDate - 2;

   -- Add Dates Beyond numTS.cal.maxDate
   numTS.series(12).tstamp := numTS.cal.maxDate + 1;
   numTS.series(13).tstamp := numTS.cal.maxDate + 2;

   -- Add some null timestamps
   numTS.series(14).tstamp := NULL;
   numTS.series(15).tstamp := NULL;

   -- Add some imprecise dates (some are duplicated)
   numTS.series(17).tstamp := numTS.series(16).tstamp + 1/24;
   numTS.series(18).tstamp := numTS.series(16).tstamp + 15/24;

   -- Add some duplicate timestamps
   numTS.series(19).tstamp := numTS.series(18).tstamp;
   numTS.series(21).tstamp := numTS.series(20).tstamp;

   -- Add some extra dates in the middle
   numTS.series(37).tstamp := TO_DATE('12/28/1996');
   numTS.series(36).tstamp := TO_DATE('12/29/1996');

   -- Add some holes at the end
   numTS.series(numTS.series.count).tstamp := TO_DATE('01/04/1997');

   -- Example of validating an invalid time series.
   SELECT ordsys.timeseries.display(numTS, 'AN INVALID TIME SERIES')
   INTO tempVal FROM dual;
   retIsValid := ORDSYS.TIMESERIES.IsValidTS(numTS);
   retValTS := ORDSYS.TIMESERIES.ValidateTS(numTS, outMesg,
                         loDateTab, hiDateTab, impDateTab,
                         dupDateTab, extraDateTab, missingDateTab);
   DBMS_OUTPUT.PUT_LINE('Value returned by IsValid  = ' || retIsValid);
   DBMS_OUTPUT.PUT_LINE('Value returned by ValidateTS  = ' || retValTS);
   ORDSYS.TIMESERIES.DisplayValTS(retValTS, outMesg, loDateTab, hiDateTab,
                    impDateTab, dupDateTab, extraDateTab, missingDateTab,
                    'Testing DisplayValTS');
END;
/
```

This example might produce the following output:

```
A VALID TIME SERIES :

Name = OPEN ACME
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 01/01/1990 00:00:00
 MaxDate = 01/01/2001 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                    Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 11/14/1996 00:00:00      68
 11/15/1996 00:00:00      69
 11/18/1996 00:00:00      70
 11/19/1996 00:00:00      71
 11/20/1996 00:00:00      72
 11/21/1996 00:00:00      73
 11/22/1996 00:00:00      74
 11/25/1996 00:00:00      75
 11/26/1996 00:00:00      76
 11/27/1996 00:00:00      77
 11/29/1996 00:00:00      78
 12/02/1996 00:00:00      79
 12/03/1996 00:00:00      80
 12/04/1996 00:00:00      81
 12/05/1996 00:00:00      82
 12/06/1996 00:00:00      83
 12/09/1996 00:00:00      84
 12/10/1996 00:00:00      85
 12/11/1996 00:00:00      86
 12/12/1996 00:00:00      87
 12/13/1996 00:00:00      88
 12/16/1996 00:00:00      89
 12/17/1996 00:00:00      90
 12/18/1996 00:00:00      91
 12/19/1996 00:00:00      92
 12/20/1996 00:00:00      93
 12/23/1996 00:00:00      94
 12/24/1996 00:00:00      95
 12/26/1996 00:00:00      96
 12/27/1996 00:00:00      97
 12/30/1996 00:00:00      98
 12/31/1996 00:00:00      99
 -----------------------------

Value returned by IsValid  = 1
Value returned by ValidateTS  = 1

DisplayValTS: Testing DisplayValTS:

TS-SUC: the input time series is a valid time series


AN INVALID TIME SERIES :

Name = OPEN ACME
Calendar Data:
Calendar Name = BUSINESS-96
 Frequency = 4 (day)
 MinDate = 10/28/1996 00:00:00
 MaxDate = 01/05/1997 00:00:00
 patBits:
          0,1,1,1,1,1,0
 patAnchor = 01/07/1996 00:00:00
 onExceptions  :
 offExceptions :
     11/28/1996 00:00:00     12/25/1996 00:00:00
Series Data:
 -----------------------------
 Date                   Value
 11/01/1996 00:00:00      59
 11/04/1996 00:00:00      60
 11/05/1996 00:00:00      61
 11/06/1996 00:00:00      62
 11/07/1996 00:00:00      63
 11/08/1996 00:00:00      64
 11/11/1996 00:00:00      65
 11/12/1996 00:00:00      66
 11/13/1996 00:00:00      67
 10/27/1996 00:00:00      68
 10/26/1996 00:00:00      69
 01/06/1997 00:00:00      70
 01/07/1997 00:00:00      71
       72
       73
 11/22/1996 00:00:00      74
 11/22/1996 01:00:00      75
 11/22/1996 15:00:00      76
 11/22/1996 15:00:00      77
 11/29/1996 00:00:00      78
 11/29/1996 00:00:00      79
 12/03/1996 00:00:00      80
 12/04/1996 00:00:00      81
 12/05/1996 00:00:00      82
 12/06/1996 00:00:00      83
 12/09/1996 00:00:00      84
 12/10/1996 00:00:00      85
 12/11/1996 00:00:00      86
 12/12/1996 00:00:00      87
 12/13/1996 00:00:00      88
 12/16/1996 00:00:00      89
 12/17/1996 00:00:00      90
 12/18/1996 00:00:00      91
 12/19/1996 00:00:00      92
 12/20/1996 00:00:00      93
 12/29/1996 00:00:00      94
 12/28/1996 00:00:00      95
 12/26/1996 00:00:00      96
 12/27/1996 00:00:00      97
 12/30/1996 00:00:00      98
 01/04/1997 00:00:00      99
 -----------------------------

Value returned by IsValid  = 0
Value returned by ValidateTS  = 0

DisplayValTS: Testing DisplayValTS:
```

```
TS-WRN: the input time series has errors. See the message for details

message output by validateTS:

TS-ERR: the input time series is unsorted
TS-ERR: the time series has null timestamps
TS-ERR: the time series has timestamps < calendar minDate (refer LoDateTab)
TS-ERR: the time series has timestamps > calendar maxDate (refer HiDateTab)
TS-ERR: the time series has imprecise timestamps (refer impreciseDateTab)
TS-ERR: the time series has duplicate timestamps (refer DuplicateDateTab)

list of dates < calendar minDate - lowDateTab :

     10/26/1996 00:00:00     10/27/1996 00:00:00

list of dates > calendar maxDate - hiDateTab :

     01/06/1997 00:00:00     01/07/1997 00:00:00

list of imprecise dates - impreciseDateTab :

     11/22/1996 01:00:00     11/22/1996 15:00:00

list of duplicate dates - duplicateDateTab :

     11/22/1996 15:00:00     11/29/1996 00:00:00

ExtraDateTab :

     12/28/1996 00:00:00     12/29/1996 00:00:00     01/04/1997 00:00:00

MissingDateTab :

     10/28/1996 00:00:00     10/29/1996 00:00:00     10/30/1996 00:00:00
     10/31/1996 00:00:00     11/14/1996 00:00:00     11/15/1996 00:00:00
     11/18/1996 00:00:00     11/19/1996 00:00:00     11/20/1996 00:00:00
     11/21/1996 00:00:00     11/25/1996 00:00:00     11/26/1996 00:00:00
     11/27/1996 00:00:00     12/02/1996 00:00:00     12/23/1996 00:00:00
     12/24/1996 00:00:00     12/31/1996 00:00:00     01/01/1997 00:00:00
     01/02/1997 00:00:00     01/03/1997 00:00:00
```

6 Time Scaling Functions: Reference

The Oracle8*i* Time Series library consists of:

- Data types (described in Section 2.3)
- Calendar functions (described in Chapter 4)
- Time series functions (described in Chapter 5)
- Time scaling functions (described in this chapter)
- Administrative tools procedures for creating time series schema objects (described in Chapter 7)

Calendar functions are mainly used by product developers, such as ISVs, to develop new time series functions and to administer and modify calendars.

Time series and time scaling functions and the administrative tools procedures are used mainly by application developers.

Syntax notes:

- The ORDSYS schema name and the package name must be used with the function name, although public synonyms can be created to eliminate the need for specifying the schema name (see Section 1.5). Each function is included in a PL/SQL package, such as Calendar, TimeSeries, or TimeScale. The ORDSYS schema name and the package name are included in the Format and in any examples.
- Function calls are not case sensitive, except for any quoted literal values. For example, the following code line excerpts are valid and semantically identical:

```
select CAST(TimeSeries.ExtractTable(close) AS ORDTNumTab)
select cast(TIMESERIES.extracttable(close) as ordtnumtab)
select cast(TiMeSeRiEs.eXtRaCtTaBlE(ClosE) As ordtNUMtab)
```

- The syntax and examples show the reference-based interface (types ORDTNumSeriesIOTRef and ORDTVarchar2SeriesIOTRef).

All time series and time scaling functions accept both references and instances as parameters. (For example, an ORDTNumSeriesIOTRef parameter could also be ORDTNumSeries.) All time series functions return instances. Thus, if you nest functions, such as *Cmax(Cmax(...), ...)*, the innermost nesting accepts a reference and returns an instance, and any other functions in the nesting accept an instance and return an instance.

For an explanation of the reference-based interface, see Section 2.7.2.

ScaledownInterpolate

Format

```
ORDSYS.TimeScale.ScaledownInterpolate(
    [tsname VARCHAR2,]
    inputTS ORDSYS.ORDTNumSeriesIOTRef,
    targetCal ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series in which data values are interpolated between values in the input time series. For example, in a *semi-annua*l (January and July) to *month* scaledown, if the data value for a January input timestamp is 100 and the data value for the next (July) input timestamp is 160, the data values for the monthly timestamps for January through June will be 100, 110, 120, 130, 140, and 150.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

inputTS
The input time series.

targetCal
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate

The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

Usage

inputTS cannot be an irregular time series (a time series with no associated calendar).

An exception is returned for any of the following conditions:

- The input time series (*inputTS*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is shorter than the frequency of the specified calendar (for example, the time series calendar's frequency is *day* and the specified calendar's frequency is *month*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *month* and the frequency of *targetCal* is *week*.)
- Any data in the input time series has no corresponding interval in the target time series.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Scale quarterly unemployment rate values down to monthly values, using interpolation:

```
SELECT to_char(tstamp) tstamp, value
  FROM myts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaledownInterpolate(ts.unemployment_rate,
                                              VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.region='1' AND cal.name  ='Monthly';
```

Assume the following timestamps and values for *unemployment_rate*:

Timestamp	Value
01-Jan-1998	4.0
01-Apr-1998	3.2
01-Jul-1998	5.1
01-Oct-1998	3.9

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-JAN-98          4
01-FEB-98 3.72444444
01-MAR-98 3.47555556
01-APR-98        3.2
01-MAY-98 4.15604396
01-JUN-98 5.14395604
01-JUL-98        6.1
01-AUG-98 5.35869565
01-SEP-98  4.6173913
01-OCT-98        3.9
10 rows selected.
```

Note that only 10 rows are returned here, as opposed to 12 rows in the ScaledownRepeat example. Interpolation cannot be performed for the months of November and December in this example because the input time series does not contain a timestamp for the following January.

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaledownRepeat

Format

```
ORDSYS.TimeScale.ScaledownRepeat(
    [tsname VARCHAR2,]
    inputTS ORDSYS.ORDTNumSeriesIOTRef,
    targetCal ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series in which data values in the input time series are repeated. For example, in a *semi-annua*l (January and July) to *month* scaledown, if the data value for a January input timestamp is 100 and the data value for the next (July) input timestamp is 160 (or any other value), the data values for the monthly timestamps for January through June will all be 100.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

inputTS
The input time series.

targetCal
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

Usage

inputTS cannot be an irregular time series (a time series with no associated calendar).

An exception is returned for any of the following conditions:

- The input time series (*inputTS*) or the specified calendar (*targetCal*) is null.
- The input time series (*inputTS*) does not have an associated calendar.
- The frequency of the calendar on which the time series is based is shorter than the frequency of the specified calendar (for example, the time series calendar's frequency is *day* and the specified calendar's frequency is *month*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *month* and the frequency of *targetCal* is *week*.)
- Any data in the input time series has no corresponding interval in the target time series.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Scale quarterly unemployment rate values down to monthly values, using repetition:

```
SELECT to_char(tstamp) tstamp, value
  FROM myts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaledownRepeat(ts.unemployment_rate,
                                              VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.region='1' AND cal.name  ='Monthly';
```

Assume the following timestamps and values for *unemployment_rate*:

Timestamp	Value
01-Jan-1998	4.0
01-Apr-1998	3.2
01-Jul-1998	5.1
01-Oct-1998	3.9

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-JAN-98          4
01-FEB-98          4
01-MAR-98          4
01-APR-98        3.2
01-MAY-98        3.2
01-JUN-98        3.2
01-JUL-98        6.1
01-AUG-98        6.1
01-SEP-98        6.1
01-OCT-98        3.9
01-NOV-98        3.9
01-DEC-98        3.9
12 rows selected.
```

Note that 12 rows are returned here, as opposed to only 10 rows in the ScaledownInterpolate example. Repetition is performed for the months of November and December based on the October value, and is not dependent on the value for the following January.

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaledownSplit

Format

```
ORDSYS.TimeScale.ScaledownSplit(
    [tsname VARCHAR2,]
    inputTS ORDSYS.ORDTNumSeriesIOTRef,
    targetCal ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series in which data values reflect the division of the data value in the input time series by the number of associated timestamps in the resulting time series. For example, in a *semi-annua*l (January and July) to *month* scaledown, if the data value for a January input timestamp is 100 and the data value for the next (July) input timestamp is 160 (or any other value), the data values for the monthly timestamps for January through June will all be 16.667 (1/6 of 100).

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

inputTS
The input time series.

targetCal
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate

The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

Usage

inputTS cannot be an irregular time series (a time series with no associated calendar).

An exception is returned for any of the following conditions:

- The input time series (*inputTS*) or the specified calendar (*targetCal*) is null.
- The input time series (*inputTS*) does not have an associated calendar.
- The frequency of the calendar on which the time series is based is shorter than the frequency of the specified calendar (for example, the time series calendar's frequency is *day* and the specified calendar's frequency is *month*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *month* and the frequency of *targetCal* is *week*.)
- Any data in the input time series has no corresponding interval in the target time series.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Scale quarterly widget production values down to monthly values, dividing each quarter's value evenly among the three months in that quarter:

```
SELECT to_char(tstamp) tstamp, value
  FROM myts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaledownSplit(ts.widget_production,
                                             VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.region='1' AND cal.name  ='Monthly';
```

With quarterly *widget_production* values of 1000, 1500, 900, and 1200, this example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-JAN-98 333.333333
01-FEB-98 333.333333
01-MAR-98 333.333333
01-APR-98        500
01-MAY-98        500
01-JUN-98        500
01-JUL-98        300
01-AUG-98        300
01-SEP-98        300
01-OCT-98        400
01-NOV-98        400
01-DEC-98        400
12 rows selected.
```

For example, one-third (333.33...) of the quarterly value of 1000 for 01-Jan is returned as the monthly value for 01-Jan, 01-Feb, and 01-Mar.

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupAvg

Format

```
ORDSYS.TimeScale.ScaleupAvg(
    [tsname VARCHAR2,]
    inputTS ORDSYS.ORDTNumSeriesIOTRef,
    targetCal ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the average value of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

inputTS
The input time series.

targetCal
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the average closing prices for stock SAMCO for each month for the entire time series:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupAvg(ts.close,
                             VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
```

```
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96   39.83125
01-DEC-96 38.2738095
2 rows selected.
```

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupAvgX

Format

```
ORDSYS.TimeScale.ScaleupAvgX(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the average value of each scaled group of values plus the immediately preceding source period.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

ScaleupAvgX is like ScaleupAvg, except that ScaleupAvgX also considers the last timestamp before the current scaling period. For example:

- The monthly average closing price for January for a stock is the average of the closing price on trading days in January and the last trading day in December.
- The quarterly average unemployment rate for the first calendar quarter (scaling up from monthly data) is the average of the rates for December, January, February, and March.

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the average closing prices for stock SAMCO for each month plus the last trading day of the preceding month for the entire time series:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupAvgX(ts.close,
                             VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96   39.83125
01-DEC-96 38.2727273
2 rows selected.
```

Note that the value for 01-Dec-1996 in this example is different from the value in the ScaleupAvg example, because this ScaleupAvgX value for 01-Dec considers the closing price for the last timestamp in November. (There is no October data in the *stockdemo_ts* table, and thus the 01-Nov values are the same in the ScaleupAvg and ScaleupAvgX examples.)

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupCount

Format

```
ORDSYS.TimeScale.ScaleupCount(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the count of non-null timestamps in each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the monthly count of daily closing prices for stock SAMCO for the period 01-Nov-1996 through 31-December 1996:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupCount(ts.close,
                   VALUE(cal),
                   to_date('01-NOV-1996','DD-MON-YYYY'),
                   to_date('31-DEC-1996','DD-MON-YYYY'))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96         20
01-DEC-96         21
2 rows selected.
```

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupFirst

Format

```
ORDSYS.TimeScale.ScaleupFirst(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the first non-null value of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.

  If IgnoreNulls (the default) is enabled, the first non-null value of the group is returned (unless all values of the group are null, in which case a null is returned). If IgnoreNullsOFF is enabled, the first value of the group is returned.

- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the first closing prices for stock SAMCO for the months of November and December of 1996:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupFirst(ts.close,
                   VALUE(cal),
                   to_date('01-NOV-1996','DD-MON-YYYY'),
                   to_date('31-DEC-1996','DD-MON-YYYY'))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96     41.875
01-DEC-96     38.125
2 rows selected.
```

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupGMean

Format

```
ORDSYS.TimeScale.ScaleupGMean(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the geometric mean of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

The geometric mean of each scaled group is computed by taking the sum of the logarithms (base 10) of the values for the corresponding source period, and then raising 10 to the power of the logarithm sum divided by the number of elements in the corresponding source period. That is: POWER(10, log_sum / number_elements).

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the geometric mean of closing prices for stock SAMCO for each month for the entire time series:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupGMean(ts.close,
                                      VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96 39.7833842
01-DEC-96 38.2719057
2 rows selected.
```

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupLast

Format

```
ORDSYS.TimeScale.ScaleupLast(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the last non-null value of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.

  If IgnoreNulls (the default) is enabled, the last non-null value of the group is returned (unless all values of the group are null, in which case a null is returned). If IgnoreNullsOFF is enabled, the last value of the group is returned.

- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the last closing prices for stock SAMCO for the months of November and December of 1996:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupLast(ts.close,
                   VALUE(cal),
                   to_date('01-NOV-1996','DD-MON-YYYY'),
                   to_date('31-DEC-1996','DD-MON-YYYY'))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96      38.25
01-DEC-96      39.75
2 rows selected.
```

Note that each timestamp reflects the first date of the month in the calendar (following the convention illustrated in Table 2–3 in Section 2.2.2), and each value in this case reflects the closing price on the last date for that month in the calendar.

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupMax

Format

```
ORDSYS.TimeScale.ScaleupMax(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the maximum value of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the highest (maximum) closing prices for stock SAMCO for each month in the entire time series:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupMax(ts.close,
                                      VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
```

```
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96      43.75
01-DEC-96      39.75
2 rows selected.
```

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupMin

Format

```
ORDSYS.TimeScale.ScaleupMin(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the minimum value of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options

Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the lowest (minimum) closing prices for stock SAMCO for each month in the entire time series:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupMin(ts.close,
                                     VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96     37.375
01-DEC-96     37.875
2 rows selected.
```

See also the Month function in Chapter 4 for an example of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

ScaleupSum

Format

```
ORDSYS.TimeScale.ScaleupSum(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the sum of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options
Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the sum of the daily trade volume for stock SAMCO for each month in the time series:

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupSum(ts.volume,
                   VALUE(cal))
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output:

```
TSTAMP    VALUE
--------- ----------
01-NOV-96   10207000
01-DEC-96    3719450
2 rows selected.
```

Note that the following example uses the Month function to produce the same output. Using the Month function eliminates the need to have and specify a stored calendar with a *month* frequency.

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupSum(ts.volume,
                   ORDSYS.Calendar.Month())
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO';
```

ScaleupSumAnnual

Format

```
ORDSYS.TimeScale.ScaleupSumAnnual(
    [tsname VARCHAR2,]
    ts ORDSYS.ORDTNumSeriesIOTRef,
    calendar ORDSYS.ORDTCalendar,
    annualfactor
    [, startDate DATE
    , endDate DATE]
    [,options]
    ) RETURN ORDSYS.ORDTNumSeries;
```

Description

Given a time series, a calendar to be used for scaling, and optionally starting and ending dates, returns a time series reflecting the sum, expressed as an annual rate, of each scaled group of values.

Parameters

tsname
Name of the returned time series. Specify this parameter if you need to assign a name to the resulting time series, for example, to insert it into an object table.

ts
The input time series.

calendar
The calendar to be used for the scaling.

annualfactor
The factor by which to multiply the sum of each scaled group in order to obtain the desired annualized value. You must specify a value, either null or not null. If you specify a null value, a default value is used depending on the frequency of *calendar*, as shown in Table 6–1.

Table 6–1 annualfactor Default Values for ScaleupSumAnnual

Frequency	annualfactor Default Value
second	31536000
minute	525600
hour	8760
day	365
week	52
month	12
quarter	4
year	1
10-day	36
semi-monthly	24
semi-annual	2

startDate
The starting date to be used. If *startDate* is specified, *endDate* must also be specified.

endDate
The ending date to be used. If *endDate* is specified, *startDate* must also be specified.

options
Either or both of the following options:

- ORDSYS.TimeScale.IgnoreNulls (default) | IgnoreNullsOFF. The default is to ignore null input values in performing the scaleup.
- ORDSYS.TimeScale.DiscardError (default) | DiscardErrorOFF. The default is to raise an exception if any data in the input time series has no corresponding interval in the target time series.

See Section 2.11.2 for detailed information about these options and examples of their use.

Usage

ScaleupSumAnnual is like ScaleupSum, except that ScaleupSumAnnual converts each scaled group to an annual rate by multiplying the scaled group’s value by the *annualfactor* value.

The pattern and exceptions lists of *calendar* are not considered.

An exception is returned for any of the following conditions:

- The input time series (*ts*) or the specified calendar (*targetCal*) is null.
- The frequency of the calendar on which the time series is based is greater than the frequency of the specified calendar (for example, the time series calendar's frequency is *month* and the specified calendar's frequency is *day*).
- The frequency of the calendar on which the time series is based is incompatible for scaling to the frequency of the specified calendar because the smaller interval does not divide evenly into the larger interval. (For example, the time series calendar's frequency is *week* and the frequency of *targetCal* is *month*.)
- Any data in the input time series has no corresponding interval in the target time series and DiscardErrorOFF is not specified.
- An interval of the input time series straddles two or more intervals of the target scaling calendar.

For an explanation of concepts related to time scaling, see Section 2.11.

Example

Return the sum of the daily trade volume for stock SAMCO for each month in the entire time series, with each month's value expressed as if it were an annual value. In this case, each monthly value is computed and then multiplied by 12, the default *annualfactor* for monthly data.

```
SELECT to_char(tstamp) tstamp, value
  FROM tsdev.stockdemo_ts ts, tsdev.stockdemo_calendars cal,
     TABLE (CAST(ORDSYS.TimeSeries.ExtractTable(
         ORDSYS.TimeScale.ScaleupSumAnnual(ts.volume,
                   VALUE(cal),
                   NULL)
       ) AS ORDSYS.ORDTNumTab)) t
   WHERE ts.ticker='SAMCO' and cal.name='Monthly';
```

This example might produce the following output. (Note that each value is 12 times the corresponding value in the ScaleupAvg example.)

```
TSTAMP    VALUE
--------- ----------
01-NOV-96  122484000
01-DEC-96   44633400
2 rows selected.
```

See also the ScaleupSum function in this chapter and the Month function in Chapter 4 for examples of using a calendar-creation function (in this case, Month) to perform scaling, as opposed to specifying a stored calendar that has the desired frequency.

7

Administrative Tools Procedures: Reference

The Oracle8*i* Time Series library consists of:

- Data types (described in Section 2.3)
- Calendar functions (described in Chapter 4)
- Time series functions (described in Chapter 5)
- Time scaling functions (described in Chapter 6)
- Administrative tools procedures for creating time series schema objects (described in this chapter)

Note: This chapter documents procedures, not functions. Procedures do not return values

The procedures described in this chapter simplify the task of creating the schema objects (tables, views, triggers, and so forth) required for using Oracle8*i* Time Series.

For an overview of these procedures and requirements for using them, see Section 2.12. For an example showing the use of several procedures to create a time series group, see Section 3.1. Many of these procedures are used in the quick-start demo, described in Section 1.6.1, and in the retrofit.sql file included with the retrofit demo.

Syntax notes:

- The ORDSYS schema name and the package name must be used with the procedure name, although public synonyms can be created to eliminate the need for specifying the schema name (see Section 1.5). Each procedure is included in a PL/SQL package named TSTools. The ORDSYS schema name and the package name are included in the Format and in any examples.

- Procedure and function calls are not case sensitive, except for any quoted literal values. For example, the following code line excerpts are valid and semantically identical:

  ```
  select CAST(TimeSeries.ExtractTable(close) AS ORDTNumTab)
  select cast(TIMESERIES.extracttable(close) as ordtnumtab)
  select cast(TiMeSeRiEs.eXtRaCtTaBlE(ClosE) As ordtNUMtab)
  ```

Add_Existing_Column

Format

```
ORDSYS.TSTools.Add_Existing_Column(
    colname IN VARCHAR2
    );
```

Description

Adds a column attribute from an existing flat table to a time series.

Parameters

colname
The name of the column attribute to be added to the time series.

Usage

Use this procedure when you are creating a time series from an existing flat table. To use this procedure, you must first call the Set_Flat_Attributes procedure and set *detail_table_exists* to 1.

An exception is raised if Begin_Create_TS_Group has not been called to initialize the context. Standard Oracle exceptions are raised if the number attributes are invalid.

If an exception is raised, call Get_Status to determine if the exception canceled an ongoing Begin_Create_TS_Group sequence.

Example

Create a time series group, specify the appropriate existing tables, and add existing columns to the time series group. (This example is taken in slightly modified form from the retrofit.sql file in the retrofit demo directory.)

```
DECLARE

BEGIN

--
-- Establish 'stockdemo_ts' as the time series group name for purposes of the
-- administrative tools procedures.
--
  ORDSYS.TSTools.Begin_Create_TS_Group('stockdemo_ts','flat');

  --
  -- Assert that the detail, map, and calendar tables exist,
  -- and define the names for these tables.
  -- Note that these tables are defined in a separate file.
  -- Explicitly set the name of the relational view.
  -- Explicitly set the names of the timestamp and time series name
  -- columns.
  --

  ordsys.tstools.set_flat_attributes(
        detail_table_name    => 'stockdemo',
        detail_table_exists  => 1,
        map_table_name       => 'stockdemo_metadata',
        map_table_exists     => 1,
        cal_table_name       => 'stockdemo_calendars',
        cal_table_exists     => 1,
        tstamp_colname       => 'tstamp',
        tsname_colname       => 'ticker',
        rel_view_name        => 'stockdemo_sv');

  --
  -- Tell TSTools the names of existing time series columns
  -- (as defined for the table stockdemo).
  --

  ORDSYS.TSTools.Add_Existing_Column('open');
  ORDSYS.TSTools.Add_Existing_Column('high');
  ORDSYS.TSTools.Add_Existing_Column('low');
  ORDSYS.TSTools.Add_Existing_Column('close');
  ORDSYS.TSTools.Add_Existing_Column('volume');

-- End the specification of schema objects and create the objects.

  ORDSYS.TSTools.End_Create_TS_Group;

  exception
    when others then
       begin
          ORDSYS.TSTools.Cancel_Create_TS_Group;
          raise;
       end;
```

```
END;
/
```

Add_Integer_Column

Format

```
ORDSYS.TSTools.Add_Integer_Column(
    colname IN VARCHAR2
    );
```

Description

Adds an integer column attribute to an ongoing flat time series creation specification.

Parameters

colname
The name of the column attribute to be added to the time series.

Usage

An exception is raised if Begin_Create_TS_Group has not been called to initialize the context. Standard Oracle exceptions are raised if the number attributes are invalid.

An exception returned by this procedure might clear the package state. If the package state is cleared, the ongoing Begin_Create_TS_Group sequence is canceled, and you must reissue the complete sequence of administrative tools procedure calls. If the package state is not cleared, the ongoing Begin_Create_TS_Group sequence is not canceled, and you can reissue just the most recent procedure call. You can call Get_Status to determine if an exception cleared the package state.

Example

The following example specifies a flat-model time series named *MYTS* and adds one VARCHAR2 column (*ticker*), four NUMBER columns (*open*, *close*, *low*, and *high*), and one INTEGER column (*volume*). The End_Create_TS_Group call ends the specification of the time series and creates the schema objects.

```
DECLARE

BEGIN
```

```
  ORDSYS.TSTools.Begin_Create_TS_Group('MYTS','flat');

  ORDSYS.TSTools.Add_Varchar2_Column('ticker',10);
  ORDSYS.TSTools.Add_Number_Column('open');
  ORDSYS.TSTools.Add_Number_Column('close');
  ORDSYS.TSTools.Add_Number_Column('low');
  ORDSYS.TSTools.Add_Number_Column('high');
  ORDSYS.TSTools.Add_Integer_Column('volume');

  ORDSYS.TSTools.End_Create_TS_Group;
END;
/
```

Add_Number_Column

Format

```
ORDSYS.TSTools.Add_Number_Column(
    colname IN VARCHAR2
    [,colprecision IN NUMBER,
    colscale IN NUMBER]
    );
```

Description

Adds a number column attribute to an ongoing flat time series creation specification.

Parameters

colname
The name of the column attribute to be added to the time series.

colprecision
The precision of the column attribute, that is, the maximum number of digits permitted to the left of the decimal point. Must be between 1 and 38. If *colprecision* is specified, *colscale* must also be specified.

colscale
The scale of the column attribute, that is, the number of digits to the right of the decimal point. Must be between -84 and 127. If *colscale* is specified, *colprecision* must also be specified.

Usage

If you specify *colprecision*, you must also specify *colscale*. If you specify either colprecision or colscale, you cannot omit the other parameter or specify a null for it. For example, to specify that *close* (closing price) can have up to 4 digits to the left of the decimal point and 3 digits to the right of the decimal point, specify the following:

```
ORDSYS.TSTools.Add_Number_Column('close',4,3);
```

For this definition, the following *close* values would be valid: 127.25, 9.875, 53, and 27.5.

For this definition, the following *close* values would be invalid: 12345.6 (exceeds *colprecision*) and 6.1234 (exceeds *colscale*).

An exception is raised if Begin_Create_TS_Group has not been called to initialize the context. Standard Oracle exceptions are raised if the number attributes are invalid.

An exception returned by this procedure might clear the package state. If the package state is cleared, the ongoing Begin_Create_TS_Group sequence is canceled, and you must reissue the complete sequence of administrative tools procedure calls. If the package state is not cleared, the ongoing Begin_Create_TS_Group sequence is not canceled, and you can reissue just the most recent procedure call. You can call Get_Status to determine if an exception cleared the package state.

Example

The following example specifies a flat-model time series named *MYTS* and adds one VARCHAR2 column (*ticker*), four NUMBER columns (*open*, *close*, *low*, and *high*), and one INTEGER column (*volume*). The End_Create_TS_Group call ends the specification of the time series and creates the schema objects.

```
DECLARE

BEGIN

  ORDSYS.TSTools.Begin_Create_TS_Group('MYTS','flat');

  ORDSYS.TSTools.Add_Varchar2_Column('ticker',10);
  ORDSYS.TSTools.Add_Number_Column('open');
  ORDSYS.TSTools.Add_Number_Column('close');
  ORDSYS.TSTools.Add_Number_Column('low');
  ORDSYS.TSTools.Add_Number_Column('high');
  ORDSYS.TSTools.Add_Integer_Column('volume');

  ORDSYS.TSTools.End_Create_TS_Group;
END;
/
```

Add_Varchar2_Column

Format

```
ORDSYS.TSTools.Add_Varchar2_Column(
    colname IN VARCHAR2,
    length IN INTEGER
    );
```

Description

Adds a VARCHAR2 column attribute to an ongoing flat time series creation specification.

Parameters

colname
The name of the column attribute to be added to the time series.

length
The name of the column attribute to be added to the time series. Must be between 1 and 4000.

Usage

An exception is raised if Begin_Create_TS_Group has not been called to initialize the context. Standard Oracle exceptions are raised if the number attributes are invalid.

An exception returned by this procedure might clear the package state. If the package state is cleared, the ongoing Begin_Create_TS_Group sequence is canceled, and you must reissue the complete sequence of administrative tools procedure calls. If the package state is not cleared, the ongoing Begin_Create_TS_Group sequence is not canceled, and you can reissue just the most recent procedure call. You can call Get_Status to determine if an exception cleared the package state.

Example

The following example specifies a flat-model time series named *MYTS* and adds one VARCHAR2 column (*ticker*), four NUMBER columns (*open*, *close*, *low*, and *high*), and one INTEGER column (*volume*). The End_Create_TS_Group call ends the specification of the time series and creates the schema objects.

```
DECLARE

BEGIN

  ORDSYS.TSTools.Begin_Create_TS_Group('MYTS','flat');

  ORDSYS.TSTools.Add_Varchar2_Column('ticker',10);
  ORDSYS.TSTools.Add_Number_Column('open');
  ORDSYS.TSTools.Add_Number_Column('close');
  ORDSYS.TSTools.Add_Number_Column('low');
  ORDSYS.TSTools.Add_Number_Column('high');
  ORDSYS.TSTools.Add_Integer_Column('volume');

  ORDSYS.TSTools.End_Create_TS_Group;
END;
/
```

Begin_Create_TS_Group

Format

```
ORDSYS.TSTools.Begin_Create_TS_Group(
    name IN VARCHAR2,
    storage_model IN VARCHAR2
    );
```

Description

Initiates the context for creating a time series group (the schema objects for a time series).

Parameters

name
Name of the time series group to be created.

storage_model
Storage model for the time series. Must be 'FLAT' or 'OBJECT' (not case sensitive).

Usage

To avoid possible naming conflicts, *name* should be different from any other object names under the current schema. (For example, user SCOTT should not create a time series group named *EMP* because there is already a table with that name.)

This procedure returns an error if the context for creating time series schema objects is active, that is, has been initiated and not canceled or closed.

Example

The following example specifies a flat-model time series named *MYTS* and adds one VARCHAR2 column (*ticker*), four NUMBER columns (*open*, *close*, *low*, and *high*), and one INTEGER column (*volume*). The End_Create_TS_Group call ends the specification of the time series and creates the schema objects.

```
DECLARE

BEGIN
```

```
  ORDSYS.TSTools.Begin_Create_TS_Group('MYTS','flat');

  ORDSYS.TSTools.Add_Varchar2_Column('ticker',10);
  ORDSYS.TSTools.Add_Number_Column('open');
  ORDSYS.TSTools.Add_Number_Column('close');
  ORDSYS.TSTools.Add_Number_Column('low');
  ORDSYS.TSTools.Add_Number_Column('high');
  ORDSYS.TSTools.Add_Integer_Column('volume');

  ORDSYS.TSTools.End_Create_TS_Group;
END;
/
```

Cancel_Create_TS_Group

Format

ORDSYS.TSTools.Cancel_Create_TS_Group;

Description

Cancels the creation of a time series group, that is, cancels the context initiated by the Begin_Create_TS_Group procedure.

Parameters

None.

Usage

This procedure clears all package state information that was created by Begin_Create_TS_Group and other Oracle8*i* Time Series administrative tools procedures. To create a time series group, you must reissue the complete sequence of administrative tools procedure calls.

Example

The following example cancels the creation of the current time series group if an exception occurs:

```
...
  ORDSYS.TSTools.End_Create_TS_Group;
  exception
    when others then
       begin
          ORDSYS.TSTools.Cancel_Create_TS_Group;
          raise;
       end;
...
```

Close_Log

Format

ORDSYS.TSTools.Close_Log;

Description

Closes the log file that had been opened by the Open_Log procedure.

Parameters

None.

Usage

This procedure is equivalent to calling UTL_FILE.FCLOSE. For information on the PL/SQL file I/O procedure UTL_FILE, see the *Oracle8i Application Developer's Reference - Packages* manual.

The log file (Open_Log...Close_Log) and the debug display (Trace_On...Trace_Off) contain the same information.

Example

The following example opens a log file named ts1.log in the logdir directory, creates time series schema objects, and closes the log file:

```
...
ORDSYS.TSTools.Open_Log('logdir','ts1.log');
ORDSYS.TSTools.Begin_Create_TS_Group('myts','flat');
...
ORDSYS.TSTools.End_Create_TS_Group;
ORDSYS.TSTools.Close_Log;
...
```

Display_Attributes

Format

ORDSYS.TSTools.Display_Attributes:

Description

Displays information about the time series group being created.

Parameters

None.

Usage

This procedure displays the current values of all attributes that can be set using the Set_xxx function (Set_Flat_Attributes or Set_Object_Attributes) appropriate for the current type of time series group.

The output is displayed to SERVEROUTPUT.

Example

The following example displays the attributes for the time series being created:

```
ORDSYS.TSTools.Display_Attributes;
```

This example might produce the following output:

```
current settings for begin_create_ts_group
NAME                         = MYTS
STORAGE_MODEL                = FLAT
SCHEMA                       = TSDEV
REL_VIEW_NAME                = MYTS_RVW
DETAIL_TABLE_NAME            = MYTS_TAB
DETAIL_TABLE_ATTR            = ORGANIZATION INDEX
DETAIL_TABLE_PK              = MYTS_TPK
DETAIL_TABLE_EXISTS          = 0
TSTAMP_COLNAME               = TSTAMP
TSNAME_COLNAME               = TSNAME
TSNAME_LENGTH                = 25
MAP_TABLE_NAME               = MYTS_MAP
MAP_TABLE_ATTR               =
MAP_TABLE_PK                 = MYTS_MPK
```

```
MAP_TABLE_EXISTS         = 0
CAL_TABLE_NAME           = MYTS_CAL
CAL_TABLE_ATTR           =
CAL_TABLE_PK             = MYTS_CPK
CAL_TABLE_EXISTS         = 0
REL_VIEW_TRIGGER_NAME    = MYTS_TR
-----------------------------------------
COLUMN NAME      = TICKER
       TYPE      = VARCHAR2
       LENGTH    = 10
       PRECISION =
       SCALE     =
COLUMN NAME      = OPEN
       TYPE      = NUMBER
       LENGTH    = 22
       PRECISION =
       SCALE     =
COLUMN NAME      = CLOSE
       TYPE      = NUMBER
       LENGTH    = 22
       PRECISION =
       SCALE     =
COLUMN NAME      = LOW
       TYPE      = NUMBER
       LENGTH    = 22
       PRECISION =
       SCALE     =
COLUMN NAME      = HIGH
       TYPE      = NUMBER
       LENGTH    = 22
       PRECISION =
       SCALE     =
COLUMN NAME      = VOLUME
       TYPE      = NUMBER
       LENGTH    = 22
       PRECISION =
       SCALE     =
```

Drop_TS_Group

Format

```
ORDSYS.TSTools.Drop_TS_Group(
    name IN VARCHAR2
    [, schema IN VARCHAR2]
    );
```

Description

Deletes the time series group definition and views associated with it. However, the underlying tables (calendar tables, detail data tables, and so on) are not deleted.

Parameters

name
Name of the time series group to be deleted.

schema
The schema (user) where the *name* objects are located. The default is the current schema.

Usage

Contrast this procedure with Drop_TS_Group_All, which deletes all the underlying tables. For example, if you have an existing time series table filled with data and want to add a column, you could use Drop_TS_Group as follows:

1. Use Drop_TS_Group.
2. Add the desired column to the underlying table (ALTER TABLE...ADD...).
3. Add data for the new column (INSERT...).
4. Create the time series schema objects again, including the new column.

If an attempt to delete a specific object fails, an exception is raised and the procedure attempts to delete any remaining appropriate objects.

To delete time series schema objects that were not created by the current user, you must have been granted the DBA or TIMESERIES_DBA role.

Example

The following example deletes the schema objects, but not the underlying tables, for the time series group *MYTS*:

```
DECLARE

BEGIN

  ORDSYS.TSTools.Drop_TS_Group('MYTS');
  exception
    when others then
        raise;

END;
/
```

Drop_TS_Group_All

Format

```
ORDSYS.TSTools.Drop_TS_Group_All(
    name IN VARCHAR2
    [, schema IN VARCHAR2]
    );
```

Description

Deletes the time series group definition and all tables, views, indexes, constraints, and triggers associated with it.

Parameters

name
Name of the time series group to be deleted.

schema
The schema (user) where the *name* objects are located. The default is the current schema.

Usage

Contrast this procedure with Drop_TS_Group, which does not delete the underlying tables.

If an attempt to delete a specific object fails, an exception is raised and the procedure attempts to delete any remaining appropriate objects.

To delete time series schema objects that were not created by the current user, you must have been granted the DBA or TIMESERIES_DBA role.

Example

The following example deletes all schema objects, including underlying tables, for the time series group *MYTS*:

```
DECLARE

BEGIN

  ORDSYS.TSTools.Drop_TS_Group_All('MYTS');
  exception
    when others then
        raise;

END;
/
```

End_Create_TS_Group

Format

ORDSYS.TSTools.End_Create_TS_Group(

[in_description IN VARCHAR2]

);

Description

Closes the context established by the Begin_Create_TS_Group procedure and creates all appropriate schema objects.

Parameters

in_description
Optional comment or other information; will be included in the log if logging is in effect.

Usage

An exception is raised if the time series being created is missing any required elements. For example, at least one column must be specified.

Example

The following example specifies a flat-model time series named *MYTS* and adds one VARCHAR2 column (*ticker*), four NUMBER columns (*open*, *close*, *low*, and *high*), and one INTEGER column (*volume*). The End_Create_TS_Group call ends the specification of the time series and creates the schema objects.

```
DECLARE

BEGIN

  ORDSYS.TSTools.Begin_Create_TS_Group('MYTS','flat');

  ORDSYS.TSTools.Add_Varchar2_Column('ticker',10);
  ORDSYS.TSTools.Add_Number_Column('open');
  ORDSYS.TSTools.Add_Number_Column('close');
  ORDSYS.TSTools.Add_Number_Column('low');
  ORDSYS.TSTools.Add_Number_Column('high');
  ORDSYS.TSTools.Add_Integer_Column('volume');

  ORDSYS.TSTools.End_Create_TS_Group;
END;
/
```

Get_Flat_Attributes

Format

```
ORDSYS.TSTools.Get_Flat_attributes(
    tstamp_colname OUT VARCHAR2,
    tsname_colname OUT VARCHAR2,
    tsname_length OUT NUMBER,
    rel_view_name OUT VARCHAR2,
    detail_table_name OUT VARCHAR2,
    detail_table_attr OUT VARCHAR2,
    detail_table_pk OUT VARCHAR2,
    detail_table_exists OUT INTEGER,
    map_table_name OUT VARCHAR2,
    map_table_attr OUT VARCHAR2,
    map_table_pk OUT VARCHAR2,
    map_table_exists OUT VARCHAR2,
    cal_table_name OUT VARCHAR2,
    cal_table_attr OUT VARCHAR2,
    cal_table_pk OUT VARCHAR2,
    cal_table_exists OUT INTEGER
    rv_trigger_name OUT VARCHAR2);
```

Description

Retrieves the attributes of a flat time series.

Parameters

tstamp_colname
Name of the timestamp column.

tsname_colname
Name of the column that identifies a time series instance.

tsname_length
Length of *tsname_colname*.

rel_view_name
Name of the relational view created on the underlying (detail) table identified by *detail_table_name*.

detail_table_name
Name of the table containing the composite data.

detail_table_attr
Attributes of the table identified by *detail_table_name*.

detail_table_pk
Primary key for the table identified by *detail_table_name*.

detail_table_exists
Contains 1 if the table identified by *detail_table_name* exists; contains 0 if this table does not exist.

map_table_name
Name of the table that maps time series to calendars.

map_table_attr
Attributes of the table identified by *map_table_name*.

map_table_pk
Primary key for the table identified by *map_table_name*.

map_table_exists
Contains 1 if the table identified by *map_table_name* exists; contains 0 if this table does not exist.

cal_table_name
Name of the table containing the calendar definitions.

cal_table_attr
Attributes of the table identified by *cal_table_name*.

cal_table_pk
Primary key for the table identified by *cal_table_name*.

cal_table_exists
Contains 1 if the table identified by *cal_table_name* exists; contains 0 if this table does not exist.

rv_trigger_name
Name of the INSTEAD OF trigger for insert, update, or delete operations on the relational view.

Usage

This procedure returns the attributes into variables that you specify. If you simply want to display the attributes of the time series being created, you can use the Display_Attributes procedure.

To return the attributes of an object-model time series, use the Get_Object_Attributes procedure.

Example

The following example gets the attributes of the flat time series being created:

```
DECLARE

  tstamp_colname       VARCHAR2(30);
  tsname_colname       VARCHAR2(30);
  tsname_length        NUMBER;
  rel_view_name        VARCHAR2(30);
  detail_table_name    VARCHAR2(30);
  detail_table_attr    VARCHAR2(30);
  detail_table_pk      VARCHAR2(30);
  detail_table_exists  INTEGER;
  map_table_name       VARCHAR2(30);
  map_table_attr       VARCHAR2(30);
  map_table_pk         VARCHAR2(30);
  map_table_exists     INTEGER;
  cal_table_name       VARCHAR2(30);
  cal_table_attr       VARCHAR2(30);
  cal_table_pk         VARCHAR2(30);
  cal_table_exists     INTEGER;
  rv_trigger_name      VARCHAR2(30);

BEGIN
```

```
   ORDSYS.TSTools.Get_Flat_Attributes(
          tstamp_colname,
          tsname_colname,
          tsname_length,
          rel_view_name,
          detail_table_name,
          detail_table_attr,
          detail_table_pk,
          detail_table_exists,
          map_table_name,
          map_table_attr,
          map_table_pk,
          map_table_exists,
          cal_table_name,
          cal_table_attr,
          cal_table_pk,
          cal_table_exists,
          rv_trigger_name);

   DBMS_OUTPUT.PUT_LINE('tstamp_colname       = '||tstamp_colname);
   DBMS_OUTPUT.PUT_LINE('tsname_colname       = '||tsname_colname);
   DBMS_OUTPUT.PUT_LINE('tsname_length        = '||tsname_length);
   DBMS_OUTPUT.PUT_LINE('rel_view_name        = '||rel_view_name);
   DBMS_OUTPUT.PUT_LINE('detail_table_name    = '||detail_table_name);
   DBMS_OUTPUT.PUT_LINE('detail_table_attr    = '||detail_table_attr);
   DBMS_OUTPUT.PUT_LINE('detail_table_pk      = '||detail_table_pk);
   DBMS_OUTPUT.PUT_LINE('detail_table_exists  = '||detail_table_exists);
   DBMS_OUTPUT.PUT_LINE('map_table_name       = '||map_table_name);
   DBMS_OUTPUT.PUT_LINE('map_table_attr       = '||map_table_attr);
   DBMS_OUTPUT.PUT_LINE('map_table_pk         = '||map_table_pk);
   DBMS_OUTPUT.PUT_LINE('map_table_exists     = '||map_table_exists);
   DBMS_OUTPUT.PUT_LINE('cal_table_name       = '||cal_table_name);
   DBMS_OUTPUT.PUT_LINE('cal_table_attr       = '||cal_table_attr);
   DBMS_OUTPUT.PUT_LINE('cal_table_pk         = '||cal_table_pk);
   DBMS_OUTPUT.PUT_LINE('cal_table_exists     = '||cal_table_exists);
   DBMS_OUTPUT.PUT_LINE('rv_trigger_name      = '||rv_trigger_name);

END;
/
```

This example might produce the following output:

```
tstamp_colname       = TSTAMP
tsname_colname       = TSNAME
```

```
tsname_length       = 25
rel_view_name       = MYTS_RVW
detail_table_name   = MYTS_TAB
detail_table_attr   = ORGANIZATION INDEX
detail_table_pk     = MYTS_TPK
detail_table_exists = 0
map_table_name      = MYTS_MAP
map_table_attr      =
map_table_pk        = MYTS_MPK
map_table_exists    = 0
cal_table_name      = MYTS_CAL
cal_table_attr      =
cal_table_pk        = MYTS_CPK
cal_table_exists    = 0
rv_trigger_name     = MYTS_TR
```

Get_Object_Attributes

Format

```
ORDSYS.TSTools.Get_Object_Attributes(
    object_table_name OUT VARCHAR2,
    object_table_type OUT VARCHAR2,
    object_table_exists OUT INTEGER,
    storage_table_name OUT VARCHAR2,
    rel_view_name OUT VARCHAR2,
    ov_trigger_name OUT VARCHAR2,
    nt_trigger_name OUT VARCHAR2,
    rv_trigger_name OUT VARCHAR2,
    object_table_attributes OUT VARCHAR2,
    storage_table_attributes OUT VARCHAR2,
    object_table_pk OUT VARCHAR2,
    );
```

Description

Retrieves the attributes of an object-model time series.

Parameters

object_table_name
Name of the table containing the composite data.

object_table_attr
Attributes of the table identified by *object_table_name*.

object_table_exists
Contains 1 if the table identified by *object_table_name* exists; contains 0 if this table does not exist.

storage_table_name
Name of the nested storage table.

rel_view_name
Name of the relational view created on the object table identified by *object_table_name*.

ov_trigger_name
Name of the INSTEAD OF trigger for insert and update operations on the object view.

nt_trigger_name
Name of the INSTEAD OF trigger for insert, update, and delete operations on the nested table.

rv_trigger_name
Name of the INSTEAD OF trigger for insert, update, and delete operations on the relational view.

object_table_attributes
Attributes of the table identified by *object_table_name*.

storage_table_attributes
Attributes of the nested storage table.

object_table_pk
Primary key of the table identified by *object_table_name*.

Usage

This procedure returns the attributes into variables that you specify. If you simply want to display the attributes of the time series being created, you can use the Display_Attributes procedure.

To return the attributes of a flat time series, use the Get_Flat_Attributes procedure.

Example

The following example gets the attributes of an object-model time series being created.

```
DECLARE

   object_table_name   VARCHAR2(30);
   object_table_type   VARCHAR2(30);
   object_table_exists INTEGER;
   storage_table_name  VARCHAR2(30);
   rel_view_name       VARCHAR2(30);
   ov_trigger_name     VARCHAR2(30);
   nt_trigger_name     VARCHAR2(30);
   rv_trigger_name     VARCHAR2(30);
   rv_utrigger_name    VARCHAR2(30);
   rv_dtrigger_name    VARCHAR2(30);
   object_table_attributes  VARCHAR2(30);
   storage_table_attributes VARCHAR2(30);
   object_table_pk     VARCHAR2(30);

BEGIN

   ORDSYS.TSTools.Get_Object_Attributes(
          object_table_name,
          object_table_type,
          object_table_exists,
          storage_table_name,
          rel_view_name,
          ov_trigger_name,
          nt_trigger_name,
          rv_trigger_name,
          object_table_attributes,
          storage_table_attributes,
          object_table_pk);

   DBMS_OUTPUT.PUT_LINE('object_table_name = '||object_table_name);
   DBMS_OUTPUT.PUT_LINE('object_table_type = '||object_table_type);
   DBMS_OUTPUT.PUT_LINE('object_table_exists = '||object_table_exists);
   DBMS_OUTPUT.PUT_LINE('storage_table_name = '||storage_table_name);
   DBMS_OUTPUT.PUT_LINE('rel_view_name = '||rel_view_name);
   DBMS_OUTPUT.PUT_LINE('ov_trigger_name = '||ov_trigger_name);
   DBMS_OUTPUT.PUT_LINE('nt_trigger_name = '||nt_trigger_name);
   DBMS_OUTPUT.PUT_LINE('rv_trigger_name = '||rv_trigger_name);
   DBMS_OUTPUT.PUT_LINE('object_table_attributes = '||object_table_attributes);
   DBMS_OUTPUT.PUT_LINE('storage_table_attributes = '
                                 ||storage_table_attributes);
   DBMS_OUTPUT.PUT_LINE('object_table_pk = '||object_table_pk);

END;
/
```

This example might produce the following output:

```
object_table_name = AUTO_PROD
object_table_type = ORDTNUMSERIES
object_table_exists = 0
storage_table_name = MYTS_STAB
rel_view_name = MYTS_RVW
ov_trigger_name = MYTS_TO
nt_trigger_name = MYTS_TNT
rv_trigger_name = MYTS_TR
object_table_attributes =
storage_table_attributes = ORGANIZATION INDEX
object_table_pk = MYTS_OTPK
```

Get_Status

Format

ORDSYS.TSTools.Get_Status(

out_status OUT INTEGER

);

Description

Checks to see if a time series creation sequence is in progress.

Parameters

out_status
Contains 1 if a time series creation sequence is in progress; contains 0 if a time series creation sequence is not in progress.

Usage

This call can be made after a previous TSTools procedure raises an exception, to determine if you need to reissue only the last administrative tools procedure call or the complete sequence of administrative tools procedure calls.

If the exception caused the package state to be cleared, *out_status* contains 0 and you must reissue the complete sequence of administrative tools procedure calls. If the exception did not cause the package state to be cleared, *out_status* contains 1 and you can reissue just the most recent administrative tools procedure call.

Example

The following example gets the status, stores it in a variable named *status*, and displays the value:

```
DECLARE
  status INTEGER;
BEGIN
  ORDSYS.TSTools.Get_Status(status);
  DBMS_OUTPUT.PUT_LINE('Status = '||status);
END;
/
```

This example might produce the following output:

```
Status = 0
```

Open_Log

Format

```
ORDSYS.TSTools.Open_Log(
    location IN VARCHAR2,
    filename IN VARCHAR2
    );
```

Description

Opens a log file that will contain the data definition language (DDL) statements generated by the administrative tools procedures.

Parameters

location
Directory location in which to create the log file on the server system. Must be a valid specification for the server system operating system.

filename
Name of the log file, including any extension.

Usage

This procedure is equivalent to calling UTL_FILE.FOPEN. For information on the PL/SQL file I/O procedure UTL_FILE, see the *Oracle8i Application Developer's Reference - Packages* manual.

To use this procedure, one or more directories for UTL_FILE output must be defined using the UTL_FILE_DIR parameter in the Oracle initialization file. For information about the UTL_FILE_DIR parameter, see the *Oracle8i Reference* manual.

The log file (Open_Log...Close_Log) and the debug display (Trace_On...Trace_Off) contain the same information.

Example

The following example opens a log file named ts1.log in the logdir directory, creates time series schema objects, and closes the log file:

```
...
ORDSYS.TSTools.Open_Log('logdir','ts1.log');
ORDSYS.TSTools.Begin_Create_TS_Group('myts','flat');
...
ORDSYS.TSTools.End_Create_TS_Group;
ORDSYS.TSTools.Close_Log;
...
```

Set_Flat_Attributes

Format

```
ORDSYS.TSTools.Set_Flat_Attributes(
    tstamp_colname IN VARCHAR2 DEFAULT NULL,
    tsname_colname IN VARCHAR2 DEFAULT NULL,
    tsname_length IN NUMBER DEFAULT NULL,
    rel_view_name IN VARCHAR2 DEFAULT NULL,
    detail_table_name IN VARCHAR2 DEFAULT NULL,
    detail_table_attr IN VARCHAR2 DEFAULT NULL,
    detail_table_pk IN VARCHAR2 DEFAULT NULL,
    detail_table_exists IN INTEGER DEFAULT NULL,
    map_table_name IN VARCHAR2 DEFAULT NULL,
    map_table_attr IN VARCHAR2 DEFAULT NULL,
    map_table_pk IN VARCHAR2 DEFAULT NULL,
    map_table_exists IN VARCHAR2 DEFAULT NULL,
    cal_table_name IN VARCHAR2 DEFAULT NULL,
    cal_table_attr IN VARCHAR2 DEFAULT NULL,
    cal_table_pk OUT VARCHAR2 DEFAULT NULL,
    cal_table_exists IN INTEGER DEFAULT NULL,
    rv_trigger_name IN VARCHAR2 DEFAULT NULL
    );
```

Description

Sets the attributes of a flat time series.

Parameters

tstamp_colname
Name of the timestamp column in a composite.

tsname_colname
Name of the column that identifies a time series instance in a composite.

tsname_length
Length of *tsname_colname*.

rel_view_name
Name of the relational view created on the underlying (detail) table identified by *detail_table_name*.

detail_table_name
Name of the table containing the composite data.

detail_table_attr
Attributes of the table identified by *detail_table_name*.

detail_table_pk
Primary key for the table identified by *detail_table_name*.

detail_table_exists
1 if the table identified by *detail_table_name* exists; 0 if this table does not exist.

map_table_name
Name of the table that maps time series to calendars.

map_table_attr
Attributes of the table identified by *map_table_name*.

map_table_pk
Primary key for the table identified by *map_table_name*.

map_table_exists
1 if the table identified by *map_table_name* exists; 0 if this table does not exist.

cal_table_name
Name of the table containing the calendar definitions.

cal_table_attr
Attributes of the table identified by *cal_table_name*.

cal_table_pk
Primary key for the table identified by *cal_table_name*.

cal_table_exists
1 if the table identified by *cal_table_name* exists; 0 if this table does not exist.

rv_trigger_name
Name of the INSTEAD OF trigger for insert, update, and delete operations on the relational view.

Usage

This procedure can be used to override some or all of the attributes of a flat-model time series. To leave an attribute unchanged, pass a null value for that attribute. To display the current attributes, use the Display_Attributes procedure; to retrieve the current attributes, use the Get_Flat_Attributes procedure.

If *detail_table_exists* is 1 (TRUE), the following attributes must be null: *detail_table_attr*, *tsname_length*, and *detail_table_pk*.

If *map_table_exists* is 1 (TRUE), the following attributes must be null: *map_table_attr* and *map_table_pk*.

If *cal_table_exists* is 1 (TRUE), the following attributes must be null: *cal_table_attr* and *cal_table_pk*.

An exception is raised if one or more of the following conditions are true: a time series is not being created, the time series being created is not of the flat model, or a calendar in the table identified by *cal_table_name* has an invalid frequency.

An exception is also raised if the procedure is called after a successful call to the same procedure during the creation of a time series group (that is, before the call to End_Create_TS_Group). For example, the following sequence of calls is *not* valid:

```
ORDSYS.TSTools.Set_Flat_Attributes(detail_table_name => 'mytable');
ORDSYS.TSTools.Set_Flat_Attributes(map_table_name => 'mymap');
```

However, the following call *is* valid:

```
ORDSYS.TSTools.Set_Flat_Attributes(detail_table_name => 'mytable',
                                   map_table_name => 'mymap');
```

For convenience in PL/SQL coding, because of the number of parameters for this procedure, you may want to use the association operator (=>) instead of positional notation. For example, to specify a maximum length of 25 for the timestamp column name, use the following:

```
ORDSYS.TSTools.Set_Flat_Attributes(tsname_length => 25);
```

Example

The following example begins the creation of schema objects for a flat time series named *MYTS*, and sets the *tsname_length* attribute to 25 (that is, maximum of 25 characters for the name of the time series):

```
ORDSYS.TSTools.Begin_Create_TS_Group('MYTS','flat');
ORDSYS.TSTools.Set_Flat_Attributes(tsname_length => 25);
....
```

Set_Object_Attributes

Format

```
ORDSYS.TSTools.Set_Object_Attributes(
    object_table_name IN VARCHAR2 DEFAULT NULL,
    object_table_type IN VARCHAR2 DEFAULT NULL,
    object_table_exists IN INTEGER DEFAULT NULL,
    storage_table_name IN VARCHAR2 DEFAULT NULL,
    rel_view_name IN VARCHAR2 DEFAULT NULL,
    ov_trigger_name IN VARCHAR2 DEFAULT NULL,
    nt_trigger_name IN VARCHAR2 DEFAULT NULL,
    rv_trigger_name IN VARCHAR2 DEFAULT NULL,
    object_table_attributes IN VARCHAR2 DEFAULT NULL,
    storage_table_attributes IN VARCHAR2 DEFAULT NULL,
    object_table_pk IN VARCHAR2 DEFAULT NULL
    );
```

Description

Sets the attributes of an object-model time series.

Parameters

object_table_name
Name of the object table.

object_table_type
Type associated with the object table: *numseries* or *varchar2series*.

object_table_exists
1 if the table identified by *object_table_name* exists; 0 if this table does not exist.

storage_table_name
Name of the nested storage table.

rel_view_name
Name of the relational view created on the object table identified by *object_table_name*.

ov_trigger_name
Name of the INSTEAD OF trigger for insert and update operations on the object view.

nt_trigger_name
Name of the INSTEAD OF trigger for insert, update, and delete operations on the nested table.

rv_trigger_name
Name of the INSTEAD OF trigger for insert, update, and delete operations on the relational view.

object_table_attributes
Attributes of the table identified by *object_table_name*. Must include an OVERFLOW clause if *object_table_type* is *varchar2series.*

storage_table_attributes
Attributes of the nested storage table.

object_table_pk
Primary key of the table identified by *object_table_name*.

Usage

This procedure can be used to override some or all of the attributes of an object-model time series. To leave an attribute unchanged, pass a null value for that attribute. To display the current attributes, use the Display_Attributes procedure; to retrieve the current attributes, use the Get_Object_Attributes procedure.

If *object_table_exists* is 1 (TRUE), the following attributes must be null: *object_table_attributes*, *storage_table_name*, *storage_table_attributes*, and *object_table_pk*.

An exception is raised if a time series is not being created or if the time series being created is not of the object model.

An exception is also raised if the procedure is called after a successful call to the same procedure during the creation of a time series group (that is, before the call to End_Create_TS_Group). For example, the following sequence of calls is *not* valid:

```
ORDSYS.TSTools.Set_Object_Attributes(object_table_name => 'mytable');
ORDSYS.TSTools.Set_Object_Attributes(storage_table_name => 'mystore');
```

However, the following call *is* valid:

```
ORDSYS.TSTools.Set_Object_Attributes(object_table_name => ’mytable’,
                                     storage_table_name => ’mystore’);
```

For convenience in PL/SQL coding, because of the number of parameters for this procedure, you may want to use the association operator (=>) instead of positional notation. For example, to specify *mytable* as the object table name, use the following:

```
ORDSYS.TSTools.Set_Object_Attributes(object_table_name => ’mytable’);
```

Example

The following example starts the creation of schema objects for an object-model time series, sets the object table name to *auto-prod* (because this time series will contain the number of automobiles produced each calendar frequency interval), and accepts default attributes for the other object-model time series group attributes. The example also displays the attributes.

```
DECLARE

BEGIN

  ORDSYS.TSTools.Begin_Create_TS_Group('myts','object');

  ORDSYS.TSTools.Set_Object_Attributes(
        object_table_name => 'auto_prod'
        );

  ORDSYS.TSTools.Display_Attributes;

END;
/
```

This example might produce the following output:

```
current settings for begin_create_ts_group
NAME                     = MYTS
STORAGE_MODEL            = OBJECT
SCHEMA                   = TSDEV
OBJECT_TABLE_NAME        = AUTO_PROD
OBJECT_TABLE_TYPE        = ORDTNUMSERIES
OBJECT_TABLE_EXISTS      = 0
STORAGE_TABLE_NAME       = MYTS_STAB
OBJECT_TABLE_ATTRIBUTES  =
STORAGE_TABLE_ATTRIBUTES = ORGANIZATION INDEX
OBJECT_TABLE_PK          = MYTS_OTPK
REL_VIEW_NAME            = MYTS_RVW
OBJECT_VIEW_TRIGGER_NAME = MYTS_TO
NESTED_TABLE_TRIGGER_NAME= MYTS_TNT
REL_VIEW_TRIGGER_NAME    = MYTS_TR
```

Trace_Off

Format

ORDSYS.TSTools.Trace_Off;

Description

Disables debugging for Oracle8*i* Time Series administrative tools procedures. Any data definition language (DDL) statements and errors encountered when generating DDL statements will not be logged to SERVEROUTPUT.

Parameters

None.

Usage

The log file (Open_Log...Close_Log) and the debug display (Trace_On...Trace_Off) contain the same information.

Example

The following example enables debugging for Oracle8*i* Time Series administrative tools procedures, creates time series schema objects, and disables debugging for Oracle8*i* Time Series administrative tools procedures:

```
...
ORDSYS.TSTools.Trace_On;
ORDSYS.TSTools.Begin_Create_TS_Group('myts','flat');
...
ORDSYS.TSTools.End_Create_TS_Group;
ORDSYS.TSTools.Trace_Off;
...
```

Trace_On

Format

ORDSYS.TSTools.Trace_On;

Description

Enables debugging for Oracle8*i* Time Series administrative tools procedures. Any data definition language (DDL) statements and errors encountered when generating DDL statements will be logged to SERVEROUTPUT.

Parameters

None.

Usage

The log file (Open_Log...Close_Log) and the debug display (Trace_On...Trace_Off) contain the same information.

Example

The following example enables debugging for Oracle8*i* Time Series administrative tools procedures, creates time series schema objects, and disables debugging for Oracle8*i* Time Series administrative tools procedures:

```
...
ORDSYS.TSTools.Trace_On;
ORDSYS.TSTools.Begin_Create_TS_Group('myts','flat');
...
ORDSYS.TSTools.End_Create_TS_Group;
ORDSYS.TSTools.Trace_Off;
...
```

A
Error Messages

This appendix lists the Oracle8*i* Time Series error messages, including the cause and recommended user action for each.

TS-00500, "internal error"

Cause: This is the generic internal error number for Time Series exceptions. This indicates that a process has encountered an exception.

Action: Report as a bug.

TS-00501, "the input patterns are not of the same length"

Cause: The input calendars have patterns of different lengths. For example, ’0,1,1,1,1,1,0’ and ’0,1,1,1,1,0’ were specified.

Action: Use calendars with patterns of the same length.

TS-00502, "patanchor cannot be on the 29th or 30th day of the month"

Cause: Oracle8*i* Time Series encountered a calendar having a pattern anchor on the 29th or 30th day of the month.

Action: Ensure that all calendar pattern anchors are not on the 29th or 30th day of the month.

TS-00503, "patanchor can be null only for all-zero or all-one patterns"

Cause: Pattern anchor was null, and pattern was not acceptable for a null *patanchor*. The anchor can be null only when using all-zero or all-one pattern bits.

Action: Supply a pattern anchor date, or adjust the pattern bits.

TS-00504, "illegal validflag parameter was passed to DisplayValCal/ DisplayValTS"

Cause: DisplayValCal or DisplayValTS was called with invalid parameters.

Action: Only call DisplayValCal and DisplayValTS with the output of ValidateCal or ValidateTS, respectively.

TS-00505, "illegal outmessage parameter was passed to DisplayValCal/ DisplayValTS"

Cause: DisplayValCal or DisplayValTS was called with invalid parameters.

Action: Only call DisplayValCal and DisplayValTS with the output of ValidateCal or ValidateTS, respectively.

TS-00506, "the calendar pattern is null"

Cause: Oracle8*i* Time Series encountered a calendar having a null pattern.

Action: Ensure that all calendars have a non-null pattern.

TS-00507, "the calendar has an imprecise mindate or maxdate"

Cause: Oracle8*i* Time Series encountered a calendar having an imprecise *mindate* or *maxdate*.

Action: Ensure that all calendar *mindate*s and *maxdate*s are precise.

TS-00508, "a NULL patanchor is illegal for calendars with frequencies - 5,7,10,16,18"

Cause: Oracle8*i* Time Series encountered a calendar having a null pattern anchor date with one of the following frequencies: *week* (5), *quarter* (7), *10-day* (10), *semi-monthly* (16), *semi-annual* (18).

Action: Ensure that all calendars with the frequency value in (5,10,16,18) have non-null pattern anchor dates.

TS-00509, "the input calendars have unequal pattern bits greater than 1"

Cause: Oracle8*i* Time Series encountered calendars having patterns with pattern bits greater than 1 and the corresponding pattern bits being unequal. For Union and Intersection operations, the two input calendars need to have matching pattern bits if the bits are greater than 1.

Action: Ensure that the two input calendars passed in have patterns with matching pattern bits.

TS-00510, "datetab has dates outside the bounds of the calendar"

Cause: DeriveExceptions encountered dates outside of the input calendar's *mindate* / *maxdate*.

Action: Adjust *mindate* / *maxdate* or remove extraneous dates from the input DateTab.

TS-00511, "calendar pattern bits array is either empty or null"

Cause: Oracle8*i* Time Series encountered a calendar with an empty or null array of pattern bits.

Action: Update the calendar to include a valid array of pattern bits.

TS-00512, "invalid frequency value"

Cause: Oracle8*i* Time Series encountered a calendar with an unsupported frequency.

Action: Restrict all calendars to frequencies: 1, 2, 3, 4, 5, 6, 7, 8, 10, 16, 18.

TS-00513, "the input dates are in the wrong order"

Cause: The date range provided was in reverse order.

Action: When specifying a date range, always list the earlier date first.

TS-00514, "calendar pattern has an imprecise anchor date"

Cause: Oracle8*i* Time Series encountered a calendar with an anchor having the wrong precision.

Action: Adjust the precision of the anchor to match the calendar's frequency.

TS-00515, "input date is beyond the calendar mindate/maxdate"

Cause: Oracle8*i* Time Series encountered a date less than the *mindate* or greater than the *maxdate*.

Action: Ensure that all input dates are within the *mindate*-*maxdate* range of the calendar.

TS-00516, "input date is greater than calendar maxdate"

Cause: Oracle8*i* Time Series encountered a date greater than *maxdate*.

Action: Ensure that all input dates are within the *mindate*-*maxdate* range of the calendar.

TS-00517, "unable to set precision of calendar pattern anchor"

Cause: Oracle8*i* Time Series encountered a calendar with a pattern anchor whose precision cannot be set. Setting the precision of the anchor takes it beyond the allowed Oracle dates.

Action: Ensure that the calendar pattern anchor is at least *frequency* units from the minimum Oracle date (Julian 1). Pattern anchors have to be within the following range: [Oracle Mindate + frequency, Oracle Maxdate]

TS-00519, "the series attribute of the time series type is null"

Cause: Oracle8*i* Time Series encountered a null series within a time series.

Action: Ensure that all time series have a non-null series component.

TS-00520, "the input calendar is null"

Cause: Oracle8*i* Time Series encountered a null calendar.

Action: Ensure that all calendars are non-null.

TS-00522, "error scaling date to calendar"

Cause: Input date cannot be scaled to given calendar.

Action: Ensure that the given calendar is valid and that the calendar's *mindate* and *maxdate* encompass all potential timestamp values.

TS-00523, "the input date is null"

Cause: Scaleup has encountered a null date. No scaling semantics are defined for a null date.

Action: Ensure that all input to Scaleup is non-null.

TS-00525, "the input time series is null"

Cause: Oracle8*i* Time Series encountered a null time series.

Action: Ensure that all time series are not atomically null.

TS-00526, "the input time series has a null calendar"

Cause: Oracle8*i* Time Series encountered a null calendar within a time series.

Action: Ensure that all time series include valid (non-null) calendars.

TS-00527, "error scaling up to the target calendar frequency"

Cause: Scaleup encountered a target calendar of finer frequency than that of the input time series' calendar.

Action: Scaleup requires a target calendar of equal or coarser (timestamps at less frequent intervals) frequency.

TS-00528, "calendar has a null mindate or a null maxdate"

Cause: Oracle8*i* Time Series encountered a calendar with a null *mindate* or *maxdate*.

Action: Ensure that all calendars have a valid *mindate* and *maxdate.*

TS-00529, "calendar mindate is greater than its maxdate"

Cause: Oracle8*i* Time Series encountered a calendar with *mindate* > *maxdate*.

Action: Ensure that all calendars have a valid *mindate* <= *maxdate*.

TS-00530, "series indexes must be greater than 0"

Cause: GetNthElement encountered an *index* less than 1.

Action: Use indexes greater than 0.

TS-00531, "the input time series has a null calendar reference"

Cause: Oracle8*i* Time Series encountered a time series with a null calendar reference.

Action: Ensure that all calendar references are valid.

TS-00532, "unable to DEREF calendar referenced by time series"

Cause: Oracle8*i* Time Series was unable to dereference a calendar reference.

Action: Verify that the user executing the query has select privileges for the calendar table storing the object, and that the correct calendar has been referenced by the time series reference.

TS-00533, "the time series has data beyond its calendar mindate/maxdate"

Cause: Oracle8*i* Time Series encountered a time series with data beyond *mindate* / *maxdate*.

Action: Ensure that all timestamps in a time series are within the calendar's *mindate* / *maxdate*.

TS-00534, "the number of rows requested must be a positive integer"

Cause: The requested number of rows was less than 0.

Action: Use a positive number to specify the number of rows requested.

TS-00535, "the time series ref has a null table_name parameter"

Cause: Oracle8*i* Time Series encountered a time series reference having a null *table_name*.

Action: Ensure that all time series references include a valid table name.

TS-00536, "the time series ref has a null tstamp_colname parameter"

Cause: Oracle8*i* Time Series encountered a time series reference having a null *tstamp_colname*.

Action: Ensure that all time series references include a valid timestamp column name.

TS-00537, "the time series ref has a null value_colname parameter"

Cause: Oracle8*i* Time Series encountered a time series reference having a null *value_colname*.

Action: Ensure that all time series references include a valid value column name.

TS-00538, "the time series ref has a null qualifier_colname parameter"

Cause: Oracle8*i* Time Series encountered a time series reference having a null *qualifier_colname*.

Action: Ensure that all time series references include a valid qualifier column name.

TS-00539, "the time series ref has a null qualifier_value parameter"

Cause: Oracle8*i* Time Series encountered a time series reference having a null *qualifier_value*.

Action: Ensure that all time series references include a valid qualifier value.

TS-00540, "the projected lead timestamp is beyond the calendar mindate/ maxdate"

Cause: The given parameters result in timestamps outside of *mindate* / *maxdate*.

Action: Adjust the lead timestamp or lead units to remain within calendar *min-date* / *maxdate*, or extend the *mindate* / *maxdate*.

TS-00541, "the projected lag timestamp is beyond the calendar mindate/maxdate"

Cause: The given parameters result in timestamps outside of *mindate* / *maxdate*.

Action: Adjust the lag timestamp or lag units to remain within calendar *mind-ate* / *maxdate*, or extend the *mindate* / *maxdate*.

TS-00542, "the window size for mavg/msum must be >= 1"

Cause: Window size parameter passed to moving average / sum was not greater than 0.

Action: Use a window size parameter greater than or equal to 1.

TS-00547, "the input fill type is invalid"

Cause: Fill has been called with a *filltype* less than 0 or greater than 2.

Action: Use a valid *filltype*: 0, 1, or 2.

TS-00548, "the target timestamp for leading is invalid"

Cause: The target timestamp input to the Lead function was invalid with respect to the input time series calendar.

Action: Ensure that the target timestamp input to the Lead function is a valid timestamp with respect to the input time series calendar.

TS-00551, "error parsing the SQL statement with the time series ref"

Cause: The SQL statement constructed from the time series reference was invalid.

Action: Verify the validity of the time series reference:

- Verify the validity of all components of the time series reference.
- No spaces or invalid punctuation may appear in table or column names.
- The user must have select privileges on the table referenced.
- The table name must be qualified with its schema name.

TS-00552, "error executing the SQL statement with the time series ref"

Cause: The SQL statement constructed from the time series reference was invalid.

Action: Verify the validity of the time series reference:

- Verify the validity of all components of the time series reference.
- No spaces or invalid punctuation may appear in table or column names.
- The user must have select privileges on the table referenced.
- The table name must be qualified with its schema name.

TS-00553, "divide by zero error"

Cause: An attempt was made to divide by zero with TSDivide.

Action: When dividing by a constant, ensure that the constant is nonzero.

TS-00554, "the input calendar patterns are not equal"

Cause: DeriveExceptions requires the calendar of the reference time series to have the same pattern as the calendar of the time series being processed.

Action: Ensure that DeriveExceptions is called only with time series having calendars with the same pattern.

TS-00555, "the input calendar frequencies are not equal"

Cause: DeriveExceptions requires the calendar of the reference time series to have the same frequency as the calendar of the time series being processed.

Action: Ensure that DeriveExceptions is called only with time series having calendars with the same frequency.

TS-00556, "mindate of the ref calendar exceeds the mindate of the target calendar"

Cause: DeriveExceptions encountered a reference time series whose calendar has a *mindate* greater than that of the calendar of the target time series.

Action: Ensure that DeriveExceptions is called only with appropriate time series.

TS-00557, "maxdate of the target calendar exceeds the maxdate of the ref calendar"

Cause: DeriveExceptions encountered a reference time series whose calendar has a *maxdate* less than that of the calendar of the target time series.

Action: Ensure that DeriveExceptions is called only with appropriate time series.

TS-00558, "the target calendar should have empty on/off exception lists"

Cause: DeriveExceptions encountered a target time series whose calendar has non-empty exception lists.

Action: Ensure that DeriveExceptions is called only with target time series whose calendars have empty exception lists.

TS-00559, "the caltype field in the calendar has an illegal value"

Cause: Oracle8*i* Time Series encountered a calendar with an invalid calendar type.

Action: Ensure that all calendars have valid calendar type value. Valid calendar types are: (Exception-driven calendars = 0)

TS-00560, "the input data includes imprecise timestamps"

Cause: DeriveExceptions function encountered an imprecise date in the time series (or datetab) input.

Action: Ensure that all the timestamps in the time series (datetab) are precise with respect to the target calendar before calling DeriveExceptions.

TS-00561, "begin_create_ts_group has not been called"

Cause: BEGIN_CREATE_TS_GROUP must be called before calling this procedure.

Action: Call BEGIN_CREATE_TS_GROUP before calling this procedure.

TS-00562, "the column name is a duplicate"

Cause: Two column names given for a time series were the same.

Action: Provide a unique column name for each time series column. Ensure that this column name does not conflict with any other column name including the explicit or default column name of the *tstamp* column or the column name of the *group_name* column. Use GET_ATTRIBUTES to determine default values.

TS-00563, "missing column attributes"

Cause: A time series was defined without defining columns.

Action: Define at least one column for the time series using ADD_VARCHAR2, ADD_NUMBER, or ADD_INTEGER.

TS-00564, "unknown storage model"

Cause: The time series storage model specified does not correspond to a valid storage model.

Action: Ensure that the procedure is called with a valid storage model descriptor: 'OBJECT' or 'FLAT'.

TS-00565, "wrong storage model"

Cause: The time series procedure cannot be called for the storage model currently being defined.

Action: Ensure that the procedure called is appropriate for the time series being created.

TS-00566, "unknown time series group"

Cause: The time series definition specified is not known.

Action: Ensure that the call references a known time series definition.

TS-00567, "unsupported datatype"

Cause: The column data type specified for a time series is not supported.

Action: Ensure that the column data type for a time series is NUMBER, INTEGER, or VARCHAR2.

TS-00568, "illegal call sequence"

Cause: The function is not being called in the correct sequence.

Action: Ensure that the function is called in the correct sequence.

TS-00569, "not all attributes dropped"

Cause: Not all objects belonging to a time series group could be dropped (deleted). This was either because underlying objects no longer exist or because another time series definition references them.

Action: Get privileges to drop (delete) the object directly.

TS-00570, "too many columns declared"

Cause: Too many columns were declared for the time series. Please consult the documentation for a limit on the maximum number of columns allowed.

Action: Declare another time series to accommodate the extra columns.

TS-00571, "detail table must exist"

Cause: ADD_EXISTING_COLUMN is invalid if the detail table does not exist.

Action: Call ADD_VARCHAR2_COLUMN or ADD_NUMBER_COLUMN procedure.

TS-00572, "column not found"

Cause: ADD_EXISTING_COLUMN was called for a column that does not exist in the detail table.

Action: Specify a NUMBER or VARCHAR2 column table in the existing detail table.

TS-00573, "detail table must not exist"

Cause: ADD_VARCHAR2_COLUMN, ADD_NUMBER_COLUMN is invalid if the detail table does not exist.

Action: Call ADD_COLUMN function when detail table exists.

TS-00574, "log file is already open"

Cause: OPEN_LOG was called when a log file is already open.

Action: Call CLOSE_LOG to close the current log file before calling OPEN_LOG.

TS-00575, "parameters conflict with detail_table_exists"

Cause: The *detail_table_attr*, *detail_table_pk*, or *tsname_length* field was set to non-null values when *detail_table_exists* was called with value of 1.

Action: When calling SET_FLAT_ATTRIBUTES with *detail_table_exists*=1, the *detail_table_attr, detail_table_pk*, and *detail_tsname_length* parameters must be null.

TS-00576, "parameters conflict with map_table_exists"

Cause: The *map_table_attr* or *map_table_pk* field was set to non-null values when *map_table_exists* was called with value of 1.

Action: When calling SET_FLAT_ATTRIBUTES with *map_table_exists*=1, the *map_table_attr* and *map_table_pk* parameters must be null.

TS-00577, "parameters conflict with cal_table_exists"

Cause: The *cal_table_attr* or *cal_table_pk* field was set to non-null values when *cal_table_exists* was called with value of 1.

Action: When calling SET_FLAT_ATTRIBUTES with *cal_table_exists*=1, the *cal_table_attr* and *cal_table_pk* parameters must be null.

TS-00578, "detail table not found"

Cause: The detail table specified in SET_FLAT_ATTRIBUTES could not be found.

Action: Ensure that the detail table specified in SET_FLAT_ATTRIBUTES exists.

TS-00579, "the tstamp field specified in SET_FLAT_ATTRIBUTES does not exist"

Cause: The *tstamp* field is not found in the existing detail table.

Action: Ensure that the *tstamp* column name specified in SET_FLAT_ATTRIBUTES is in the detail table.

TS-00580, "the tstamp field specified is not a DATE column"

Cause: The call specified a *tstamp* field that is not a DATE data type.

Action: Specify a *tstamp* column that is a DATE data type.

TS-00581, "the tsname field specified in SET_FLAT_ATTRIBUTES does not exist"

Cause: The *tsname* field is not found in the existing detail table.

Action: Ensure that the *tsname* field column name specified in SET_FLAT_ATTRIBUTES is in the detail table.

TS-00582, "the tsname field specified is not a VARCHAR2 column"

Cause: The call specified a *tsname* field that is not a VARCHAR2 data type.

Action: Specify a *tsname* column that is a VARCHAR2 data type.

TS-00583, "existing detail table missing primary key constraint"

Cause: An attempt was made to build a time series on a detail table that is missing a required primary key constraint.

Action: Ensure that the detail table has a primary key constraint on the *tsname* and *tstamp* columns.

TS-00584, "existing detail table missing index with tsname as first column"

Cause: An attempt was made to build a time series on a detail table that does not specify the *tsname* field as the first column of a primary key index.

Action: Ensure that the detail table has a primary key constraint on the *tsname* and *tstamp* columns.

TS-00585, "existing detail table missing index with tstamp as second column"

Cause: An attempt was made to build a time series on a detail table that does not specify the *tstamp* field as the second column of a primary key index.

Action: Ensure that the detail table has a primary key constraint on the *tsname* and *tstamp* columns.

TS-00586, "calendar table not found"

Cause: The calendar table specified in SET_FLAT_ATTRIBUTES could not be found.

Action: Ensure that the calendar table specified in SET_FLAT_ATTRIBUTES exists.

TS-00587, "calendar table not correct type"

Cause: The calendar table specified in SET_FLAT_ATTRIBUTES was not an object table of type ORDSYS.ORDTCALENDAR.

Action: Ensure that the calendar table specified in SET_FLAT_ATTRIBUTES is an object table of type ORDSYS.ORDTCALENDAR.

TS-00588, "calendar table missing primary key constraint"

Cause: An attempt was made to build a time series on a calendar table that is missing a required primary key constraint.

Action: Ensure that the calendar table has a primary key constraint on the *name* field.

TS-00589, "existing calendar table missing index with NAME as first column"

Cause: An attempt was made to build a time series on a calendar table that does not specify the *name* field as the first column of a primary key index.

Action: Ensure that the calendar table has a primary key constraint on the *name* field.

TS-00590, "map table not found"

Cause: The map table specified in SET_FLAT_ATTRIBUTES could not be found.

Action: Ensure that the map table specified in SET_FLAT_ATTRIBUTES exists.

TS-00591, "existing map table missing CALNAME field"

Cause: The map table specified should have field called *calname* of type VARCHAR2(256).

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00592, "the CALNAME field in the existing map table is not a VARCHAR2 field"

Cause: The existing map table specified should have a field called *calname* of type VARCHAR2.

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00593, "the CALNAME VARCHAR2 field is not of length 256"

Cause: The existing map table specified should have a VARCHAR2 field called *calname* of a length of 256.

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00594, "the existing map table is missing the tsname column"

Cause: The existing map table specified should have a field of the same name as the *tsname* column in the detail table.

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00595, "the tsname field in the map table is not a VARCHAR2 column"

Cause: The *tsname* field in the existing map table must be a VARCHAR2 field.

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00596, "the length of the tsname field in the existing map table is incorrect"

Cause: The length of the *tsname* field in the existing map table must be the same length as the *tsname* field in the detail table.

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00597, "the map table is missing a primary key constraint"

Cause: The map table specified must have a primary key constraint on the tsname field.

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00598, "the map table is missing an index on the tsname field"

Cause: An attempt was made to build a time series on a map table that does not specify the *tsname* field as the first column of a primary key index.

Action: Check to see that the existing map table specified is correct and has the required fields.

TS-00599, "illegal input param values"

Cause: The combination or the values of the input parameters are invalid.

Action: Check to see that the values and combination of input parameters to the call are correct.

TS-00600, "update of tstamp value is illegal"

Cause: An INSTEAD OF trigger detected an attempt to update the *tstamp* field.

Action: Updates of *tstamp* fields in a time series are not permitted.

TS-00601, "update of tsname value is illegal"

Cause: An update trigger detected an attempt to update the *tsname* field.

Action: Updates of *tsname* fields in a time series are not permitted.

TS-00602, "no calendar found"

Cause: An insert or delete trigger failed to retrieve a calendar for a regular time series.

Action: Check to see that the time series being updated has a calendar associated with it.

TS-00603, "tstamp date not valid"

Cause: An insert was done using a timestamp date value that was not valid for the calendar.

Action: Check to see that the timestamp date is valid for the calendar of the time series.

TS-00604, "time stamp must be next valid date before startdate"

Cause: An attempt was made to insert a timestamp that was not the first valid date before the starting date.

Action: Check to see that the timestamp date is valid for the calendar of the time series.

TS-00605, "time stamp must be next valid date after enddate"

Cause: An attempt was made to insert a timestamp that was not the first valid date after the ending date.

Action: Check to see that the timestamp date is valid for the calendar of the time series.

TS-00606, "cannot delete a legal date in the middle of a time series"

Cause: An attempt was made to delete a timestamp in the middle of a time series.

Cause: Delete timestamps from the ends of the time series.

TS-00607, "time series group exists"

Cause: The time series group specified already exists.

Action: Ensure that BEGIN_CREATE_TS_GROUP specifies a time series group that does not already exist.

TS-00608, "no time series instance found"

Cause: An insert or delete trigger failed to retrieve the time series instance.

Action: Check to see that the time series instance specified exists.

TS-00609, "begin_create_ts_group already called"

Cause: An attempt was made to call BEGIN_CREATE_TS_group while currently defining a time series group.

Action: Call CANCEL_CREATE_TS_group or complete a time series group definition that has been started.

TS-00610, "nothing to cancel"

Cause: Tried to call CANCEL_CREATE_TS_GROUP when no time series group definition has been started.

Action: Avoid making this call if a time series definition has not been started.

TS-00611, "the frequency is not valid"

Cause: The frequency passed into the function does not correspond to a valid calendar frequency value.

Action: Ensure that the call is passed a valid calendar frequency value.

TS-00612, "the time series type specified is not supported"

Cause: A wrong type was specified for the time series object table to be created.

Action: The only supported types for the time series object table are ORDSYS.ORDTNumSeries and ORDSYS.ORDTVarchar2Series.

TS-00613, "time series object table not found"

Cause: The time series object table specified in SET_OBJECT_ATTRIBUTES could not be found.

Action: Ensure that the time series object table specified in SET_OBJECT_ATTRIBUTES exists.

TS-00614, "existing object table is of different type"

Cause: The type of the (existing) object table does not match the type of the time series specified.

Action: Ensure that while trying to build a time series group on an existing object table, the type of the time series matches the type of the object table. Note that the only supported types for the time series are ORDSYS.ORDTNumSeries and ORDSYS.ORDTVarchar2Series.

TS-00615, "time series object table missing primary key constraint"

Cause: Tried to build a time series on an object table that is missing a required primary key constraint.

Action: Ensure that the time series object table has a primary key constraint on the *name* field.

TS-00616, "existing time series object table missing index on the NAME attribute"

Cause: An attempt was made to build a time series on an object table that does not specify the name field as the first column of a primary key index.

Action: Ensure that the time series object table has a primary key constraint on the *name* field.

TS-00617, "parameters conflict with object_table_exists"

Cause: The *object_table_attributes*, *storage_table_name*, *storage_table_attributes*, or *object_table_pk* field was set to non-null value when SET_OBJECT_ATTRIBUTES was called with *object_table_exists* set to 1.

Action: Ensure that when SET_OBJECT_ATTRIBUTES is called with *object_table_exists* set to 1, *object_table_attributes*, *storage_table_name*, *storage_table_attributes*, and *object_table_pk* are all set to null.

TS-00620, "time series is invalid"

Cause: An object view insert or update trigger failed because the new time series instance was not a valid time series.

Action: Check to see that the new time series being inserted or updated is a valid time series.

TS-00630, "an irregular time series is not a valid input"

Cause: DeriveExceptions function requires that the input time series be a regular time series. (An irregular time series does not have an associated calendar and therefore is not valid as input to the DeriveExceptions function.)

Action: Ensure that the time series input to the DeriveExceptions function is a regular time series.

TS-00631, "lead and lag operations not supported for irregular time series"

Cause: Lead and Lag operations require calendars to compute the timestamps of the resulting time series.

Action: Ensure that Lead and Lag are only used with time series that have calendars.

TS-00632, "fill is not supported for irregular time series"

Cause: Fill requires a calendar to compute the timestamps of the resulting time series.

Action: Ensure that Fill is only used with time series that have calendars.

TS-00633, "table attribute value is too large"

Cause: A table attribute value passed in is too large.

Action: Ensure that the table attribute VARCHAR2 value is less than 1023.

TS-00640, "time series cannot be scaled to target calendar - frequencies incompatible"

Cause: The frequencies of the time series and the calendar are not compatible.

Action: Ensure that the calendar associated with the scaled time series is compatible with the target calendar.

TS-00641, "time series cannot be scaled to target calendar - calendar anchors incompatible"

Cause: The calendar anchors associated with the time series and the calendar are not compatible.

Action: Ensure that the calendar associated with the scaled time series is compatible with the target calendar.

TS-00642, "time scaling error: input interval straddles two or more output intervals"

Cause: An interval of the source time series straddles two or more intervals of the target scaling calendar.

Action: Ensure that the time series to be scaled is compatible with the target calendar.

TS-00643, "time scaling error: input interval maps to non-existing output interval"

Cause: One or more cells of the input time series have no associated interval of the target calendar.

Action: Ensure that the time series to be scaled is compatible with the target calendar.

TS-00644, "time scaling error: permitDropData parameter out of bounds"

Cause: An invalid value of *DiscardError* was supplied as a parameter to Scaleup

Action: Ensure that the *DiscardError* option is either 0 or 1.

TS-00645, "scaledownrepeat is not supported for irregular time series"

Cause: A time series with a null calendar was passed to ScaleDownRepeat.

Action: Ensure that all time series used with ScaleDownRepeat have calendars.

TS-00646, "scaledownsplit is not supported for irregular time series"

Cause: A time series with a null calendar was passed to ScaleDownSplit.

Action: Ensure that all time series used with ScaleDownSplit have calendars.

TS-00647, "invalid scaleup option"

Cause: An unrecognized option has been used with a ScaleUp function.

Action: Consult the documentation for a list of valid options.

TS-00648, "invalid combination of scaleup options"

Cause: Multiple numeric options or a combination of numeric and named options has been used with ScaleUp.

Action: Consult the documentation for a list of valid options, and ensure that named options are not used with a numeric option, and that no more than one numeric option is specified.

TS-00649, "invalid scaleup option"

Cause: An unrecognized numeric option has been used with a ScaleUp function. Valid numeric options include 0, 1, 10, and 11.

Action: Be sure to only use valid numeric options, or consult the documentation for information about using named options.

TS-00650, "duplicate scaleup option"

Cause: A ScaleUp option has been specified twice, or two conflicting options have been specified.

Action: When using multiple named options, be sure not to duplicate options and not to use conflicting options. Consult the documentation for a list of conflicting options.

B

Oracle8*i* Time Series Metadata Views

This appendix describes the views that Oracle8*i* Time Series uses to store information about time series schema objects:

- ALL_TIMESERIES_GROUPS
- ALL_TIMESERIES_OBJS
- ALL_TIMESERIES_COLS
- DBA_TIMESERIES_GROUPS
- DBA_TIMESERIES_OBJS
- DBA_TIMESERIES_COLS
- USER_TIMESERIES_GROUPS
- USER_TIMESERIES_OBJS
- USER_TIMESERIES_COLS

These views are created when Oracle8*i* Time Series is installed, and they are updated when time series schema objects are created, deleted, or altered.

Access to these views is determined as follows:

- *ALL_TIMESERIES_xxx* views are accessible if you can use the time series from your schema. To use the time series, you must have been granted select privilege on the object relational view, the detail table, and the calendar table. Moreover, for *ALL_TIMESERIES_OBJS*, the corresponding *ALL_xxx* views must be accessible. For example, to access the map table as an object in *ALL_TIMESERIES_OBJS*, you must be able to access the time series and to access the map table in *ALL_TABLES*.
- *DBA_TIMESERIES_xxx* views are accessible if you have been granted either or both of the following roles: DBA or TIMESERIES_DBA.

- *USER_TIMESERIES_xxx* views are accessible if the objects have been created under your schema.

You can query these views to get information about time series schema objects. For example, to display the available information about all time series schemas, enter the following query:

```
SELECT * from ALL_TIMESERIES_GROUPS;
```

In addition to examining these views, you can examine certain standard Oracle dictionary views for metadata relating to specific schema names. For example, the following queries return names of objects associated with any time series schemas with names containing *MYTS*:

```
SELECT table_name   from USER_TABLES   where table_name like 'MYTS%';
SELECT trigger_name from USER_TRIGGERS where trigger_name like 'MYTS%';
SELECT view_name    from USER_VIEWS    where view_name like 'MYTS%';
SELECT table_name   from USER_OBJECT_TABLES where table_name like 'MYTS%';
```

For information about standard Oracle dictionary views, see the *Oracle8i Reference* manual.

B.1 View Definitions

This section shows the definitions of the Oracle8*i* Time Series metadata views.

For explanations of the columns in these views, see Section B.2.

B.1.1 ALL_TIMESERIES_xxx View Definitions

The following code example shows the definitions of the *ALL_TIMESERIES_GROUPS*, *ALL_TIMESERIES_OBJS*, and *ALL_TIMESERIES_COLS* views:

```
SVRMGR> DESCRIBE ALL_TIMESERIES_GROUPS;
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER                                   VARCHAR2(30)
GROUP_NAME                              VARCHAR2(30)
STORAGE_MODEL                           VARCHAR2(30)
DESCRIPTION                             VARCHAR2(4000)

SVRMGR> DESCRIBE ALL_TIMESERIES_OBJS;
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER                                   VARCHAR2(30)
```

```
GROUP_NAME                                VARCHAR2(30)
OBJ_NAME                                  VARCHAR2(30)
OBJ_TYPE                                  VARCHAR2(30)
TS_OBJ_TYPE                               VARCHAR2(30)
OWNED                                     CHAR(1)
STORAGE_MODEL                             VARCHAR2(30)
DESCRIPTION                               VARCHAR2(4000)

SVRMGR> DESCRIBE ALL_TIMESERIES_COLS;
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER                          NOT NULL VARCHAR2(30)
GROUP_NAME                     NOT NULL VARCHAR2(30)
TS_OBJ_TYPE                    NOT NULL VARCHAR2(30)
VIEW_NAME                      NOT NULL VARCHAR2(30)
COLUMN_NAME                    NOT NULL VARCHAR2(30)
DATA_TYPE                      NOT NULL VARCHAR2(106)
DATA_LENGTH                             NUMBER
DATA_PRECISION                          NUMBER
DATA_SCALE                              NUMBER
IS_TSNAME                      NOT NULL CHAR(1)
IS_TSTAMP                      NOT NULL CHAR(1)
IS_TSVALUE                     NOT NULL CHAR(1)
COLUMN_ID                      NOT NULL NUMBER
```

B.1.2 DBA_TIMESERIES_xxx View Definitions

The following code example shows the definitions of the *DBA_TIMESERIES_GROUPS*, *DBA_TIMESERIES_OBJS*, and *DBA_TIMESERIES_COLS* views:

```
SVRMGR> DESCRIBE DBA_TIMESERIES_GROUPS;
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER                          NOT NULL VARCHAR2(30)
GROUP_NAME                     NOT NULL VARCHAR2(30)
STORAGE_MODEL                  NOT NULL VARCHAR2(30)
DESCRIPTION                             VARCHAR2(4000)

SVRMGR> DESCRIBE DBA_TIMESERIES_OBJS;
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER                          NOT NULL VARCHAR2(30)
GROUP_NAME                     NOT NULL VARCHAR2(30)
OBJ_NAME                       NOT NULL VARCHAR2(30)
OBJ_TYPE                       NOT NULL VARCHAR2(30)
```

```
TS_OBJ_TYPE                    NOT NULL VARCHAR2(30)
OWNED                          NOT NULL CHAR(1)
STORAGE_MODEL                  NOT NULL VARCHAR2(30)
DESCRIPTION                             VARCHAR2(4000)

SVRMGR> DESCRIBE DBA_TIMESERIES_COLS;
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER                          NOT NULL VARCHAR2(30)
GROUP_NAME                     NOT NULL VARCHAR2(30)
TS_OBJ_TYPE                    NOT NULL VARCHAR2(30)
VIEW_NAME                      NOT NULL VARCHAR2(30)
COLUMN_NAME                    NOT NULL VARCHAR2(30)
DATA_TYPE                      NOT NULL VARCHAR2(106)
DATA_LENGTH                             NUMBER
DATA_PRECISION                          NUMBER
DATA_SCALE                              NUMBER
IS_TSNAME                      NOT NULL CHAR(1)
IS_TSTAMP                      NOT NULL CHAR(1)
IS_TSVALUE                     NOT NULL CHAR(1)
COLUMN_ID                      NOT NULL NUMBER
```

B.1.3 USER_TIMESERIES_xxx View Definitions

The following code example shows the definitions of the *USER_TIMESERIES_GROUPS*, *USER_TIMESERIES_OBJS*, and *USER_TIMESERIES_COLS* views:

```
SVRMGR> DESCRIBE USER_TIMESERIES_GROUPS;
Column Name                    Null?    Type
------------------------------ -------- ----
GROUP_NAME                     NOT NULL VARCHAR2(30)
STORAGE_MODEL                  NOT NULL VARCHAR2(30)
DESCRIPTION                             VARCHAR2(4000)

SVRMGR> DESCRIBE USER_TIMESERIES_OBJS;
Column Name                    Null?    Type
------------------------------ -------- ----
GROUP_NAME                     NOT NULL VARCHAR2(30)
OBJ_NAME                       NOT NULL VARCHAR2(30)
OBJ_TYPE                       NOT NULL VARCHAR2(30)
TS_OBJ_TYPE                    NOT NULL VARCHAR2(30)
OWNED                          NOT NULL CHAR(1)
STORAGE_MODEL                  NOT NULL VARCHAR2(30)
DESCRIPTION                             VARCHAR2(4000)
```

```
SVRMGR> DESCRIBE USER_TIMESERIES_COLS;
Column Name                    Null?    Type
------------------------------ -------- ----
GROUP_NAME                     NOT NULL VARCHAR2(30)
TS_OBJ_TYPE                    NOT NULL VARCHAR2(30)
VIEW_NAME                      NOT NULL VARCHAR2(30)
COLUMN_NAME                    NOT NULL VARCHAR2(30)
DATA_TYPE                      NOT NULL VARCHAR2(106)
DATA_LENGTH                             NUMBER
DATA_PRECISION                          NUMBER
DATA_SCALE                              NUMBER
IS_TSNAME                      NOT NULL CHAR(1)
IS_TSTAMP                      NOT NULL CHAR(1)
IS_TSVALUE                     NOT NULL CHAR(1)
COLUMN_ID                      NOT NULL NUMBER
```

B.2 Column Descriptions

This section describes the columns in the Oracle8*i* Time Series metadata views.

The corresponding *ALL_TIMESERIES_xxx*, *DBA_TIMESERIES_xxx*, and *USER_TIMESERIES_xxx* views have the same columns, except that the *ALL_TIMESERIES_xxx* and *DBA_TIMESERIES_xxx* views also include *owner* as the first column. The *USER_TIMESERIES_xxx* views do not contain an *owner* column.

B.2.1 xxx_TIMESERIES_GROUPS Columns

Table B–1 describes the columns in the *ALL_TIMESERIES_GROUPS*, *DBA_TIMESERIES_GROUPS*, and *USER_TIMESERIES_GROUPS*, views. Note that *owner* is not included in the *USER_TIMESERIES_GROUPS* view.

Table B–1 xxx_TIMESERIES_GROUPS Columns

Column Name	Data Type	Explanation
owner	VARCHAR2(30)	Identifies the schema under which the time series schema objects are defined.
group_name	VARCHAR2(30)	Contains the name of the time series schema, which is also the name of the time series object relational view (flat storage model) or object view (object storage model).

Table B–1 xxx_TIMESERIES_GROUPS Columns (Cont.)

Column Name	Data Type	Explanation
storage_model	VARCHAR2(30)	Contains *FLAT* for flat storage model or *OBJECT* for nested table storage.
description	VARCHAR2(4000)	Optional descriptive comment.

B.2.2 xxx_TIMESERIES_COLS Columns

Table B–2 describes the columns in the *ALL_TIMESERIES_COLS*, *DBA_TIMESERIES_COLS*, and *USER_TIMESERIES_COLS*, views. Note that *owner* is not included in the *USER_TIMESERIES*_COLS view.

Table B–2 xxx_TIMESERIES_COLS Columns

Column Name	Data Type	Explanation
owner	VARCHAR2(30)	Identifies the schema under which the time series schema objects are defined.
group_name	VARCHAR2(30)	Contains the name of the time series schema, which is also the name of the time series object relational view (flat storage model) or object view (object storage model).
ts_obj_type	VARCHAR2(30)	Contains *RELATIONAL VIEW* for a relational view or *OBJECT RELATIONAL VIEW* for an object view.
view_name	VARCHAR2(30)	Name of the view.
column_name	VARCHAR2(30)	Name of the column in the view (for example, *open*, *close*, *tstamp*, *ticker*, *volume* using the flat storage model).
data_type	VARCHAR2(106)	Data type of *column_name*.
data_length	NUMBER	Maximum length in bytes of *column_name* data.
data_precision	NUMBER	The precision of *column_name* (if numeric), that is, the maximum number of digits permitted to the left of the decimal point.
data_scale	NUMBER	The scale of *column_name* (if numeric), that is, the number of digits to the right of the decimal point.
is_tsname	CHAR(1)	Contains Y if *column_name* is the time series name; contains N if *column_name* is not the time series name.

Table B–2 xxx_TIMESERIES_COLS Columns (Cont.)

Column Name	Data Type	Explanation
is_tstamp	CHAR(1)	Contains *Y* if *column_name* is the timestamp column; contains *N* if *column_name* is not the timestamp column.
is_tsvalue	CHAR(1)	Contains *Y* if *column_name* is a data value column; contains *N* if *column_name* is not a data value column.
column_id	NUMBER	Internally assigned ID number.

B.2.3 xxx_TIMESERIES_OBJS Columns

Table B–3 describes the columns in the *ALL_TIMESERIES_OBJS*, *DBA_TIMESERIES_OBJS*, and *USER_TIMESERIES_OBJS*, views. Note that *owner* is not included in the *USER_TIMESERIES_*COLS view.

Table B–3 xxx_TIMESERIES_OBJS Columns

Column Name	Data Type	Explanation
owner	VARCHAR2(30)	Identifies the schema under which the time series schema objects are defined.
group_name	VARCHAR2(30)	Contains the name of the time series schema, which is also the name of the time series object relational view (flat storage model) or object view (object storage model).
obj_name	VARCHAR2(30)	Name of the Oracle DDL object.
obj_type	VARCHAR2(30)	Type of Oracle DDL object (for example, *TABLE* or *VIEW*).
ts_obj_type	VARCHAR2(30)	Type of Oracle time series object See Table B–4 for a list of *ts_obj_type* values.
owned	VARCHAR2(1)	Contains *Y* if the object was created by a TSTools procedure; contains *N* if was not originally created by a TSTools procedure.
storage_model	VARCHAR2(30)	Indicates the storage model: *FLAT* or *OBJECT*.
description	VARCHAR2(4000)	Optional descriptive comment.

Table B–4 lists the values that the *ts_obj_type* column in Table B–3 can contain.

Table B–4 ts_obj_type Column Values

Value	Explanation
CALENDAR TABLE	Calendar table in the flat storage model.
DETAIL TABLE	Detail table in the flat storage model.
MAP TABLE	Map table in the flat storage model.
OBJECT RELATIONAL VIEW	Time series view for time series and time scaling functions in the flat storage model.
OBJECT TABLE	Time series table in the object storage model.
OBJECT VIEW	Time series view for time series and time scaling functions in the object storage model.
OBJECT VIEW NT TRIGGER	Trigger on the object view (object storage model) for nested table insert, update, and delete operations.
OBJECT VIEW TRIGGER	Trigger on the object view (object storage model) for object insert and update operations.
RELATIONAL VIEW	Relational view for the flat storage model or object storage model.
RELATIONAL VIEW TRIGGER	Trigger on the relational view (flat or object storage model) for relational insert, update, and delete operations.
STORAGE TABLE	Nested table storage for the nested table (object storage model).

C

Deprecated Features

This appendix describes the new and old (deprecated) behavior or certain Oracle8*i* Time Series features. The old behaviors are *deprecated features* for the Oracle8*i* release. These deprecated features were documented in the previous edition of this guide, but are not in this edition. The features continue to work for this release, but they may not work in subsequent releases, and you are encouraged not to use them. If you are currently using any of these features, you are encouraged to choose a documented alternative before the next release of Oracle8*i* Time Series.

C.1 SetPrecision Function

The SetPrecision function takes a calendar rather than a frequency as one of its input parameters. The release 8.0.4 SetPrecision syntax specifying a timestamp and a frequency (timestamp IN INTEGER, frequency IN INTEGER) is still supported, but will not be supported in a future release. See the SetPrecision description in Chapter 4 for more information.

C.2 Lookback Window (k) Parameter for Mavg and Msum

The lookback window (*k*) parameter for the Mavg and Msum functions now comes before any optional start-end date range. The old format, as documented in the previous version of this guide, is a deprecated feature.

C.3 Scaleup Function (GROUP BY Interface)

The Scaleup function and explanations of the GROUP BY interface for scaleup operations are removed. The Scaleup function and the GROUP BY interface are still supported, but their use is discouraged and they may not be supported in a future release.The use of the collection-based interface for scaleup, as documented in Section 2.11.1, is encouraged.

C.4 Package for Scaleup Functions

All scaling functions (scaleup and scaledown) are included in the TimeScale package. Scaleup functions that were available in release 8.0.4 can still be called specifying the TimeSeries package; however, this usage is discouraged and may not be supported in a future release.

Glossary

anchor date

The date to be used for establishing the start of a pattern and (based on the pattern) which timestamps are to be included in and excluded from the calendar. For example, consider a pattern of ‘0,1,1,1,1,1,0’ over a *day* frequency defines a calendar over all weekdays. If an anchor date of 01-Jun-1997 (or any Sunday) is specified, then the 7-day pattern begins each Sunday; and Sunday and Saturday (0) are excluded from the calendar, while Monday through Friday (1) are included in the calendar.

calendar

A data structure that maps human-meaningful time values to underlying machine representations of time. The calendar definition includes a frequency and a pattern, and optionally exceptions and date boundaries (upper and lower).

exception

A timestamp that does not conform to the calendar pattern but that is significant for the calendar definition. There are two kinds of exceptions: off-exceptions and on-exceptions. (See the glossary definitions for each.)

frequency

The granularity of the calendar representation. The supported frequencies are *second, minute, hour, day, week, 10-day, semi-monthly, month, quarter, semi-annual,* and *year.*

group

See *time series group*.

irregular time series

A time series that does not have an associated calendar. Often, irregular time series are data-driven, where unpredictable bursts of data arrive at unspecified points in time or most timestamps cannot be characterized by a repeating pattern. However, an irregular time series can be used with predictable data where it is simply not necessary to deal with a calendar.

off-exception

An exception to the non-zero bits in the pattern, and thus is a timestamp to be excluded from the calendar. For example, to ensure that Wednesday, 25-Dec-1996, is excluded from the calendar when Wednesdays normally are included, define that date as an off-exception.

on-exception

An exception to the zero (0) bits in the pattern, and thus is a timestamp to be included in the calendar. For example, to ensure that Saturday, 28-Jun-1997, is included in the calendar when Saturdays are excluded, define that date as an on-exception.

pattern

The repeating pattern of frequencies and an anchor date that identifies a valid timestamp for the first element in the pattern. For example, if the frequency is set to *day*, the pattern can define which days of the week are included in the calendar.

pattern anchor date

See *anchor date*.

precision

The degree of exactness to which a timestamp needs to be specified. Each frequency has an associated precision. Oracle8*i* Time Series functions require that input timestamps be of the precision of the frequency associated with the calendar. A timestamp that is not consistent with the frequency is said to be *imprecise*.

regular time series

A time series that has an associated calendar. In a regular time series, data arrives predictably at predefined intervals. For example, daily summaries of stock market data form regular time series, and such time series might include the set of trade volumes and opening, high, low, and closing prices for stock XYZ for the year 1997.

time series

A set of timestamped data entries. Each time series consists of an identifier (such as stock ticker *ACME*), and multiple timestamp-value pairs (such as all trading days and the closing price for *ACME* on each trading day).

time series group

The schema objects for a time series. The time series group is created by administrative tools procedures, starting with a call to Begin_Create_TS_Group and ending with a call to End_Create_TS_Group.

Index