IBM Cognos Real-time Monitoring Workbench
Version 10.2.1

Modeling Reference
Note

Before using this information and the product it supports, read the information in "Notices" on page 441.

Product Information

This document applies to IBM Cognos Business Intelligence Version 10.2.1 and may also apply to subsequent releases.

Licensed Materials - Property of IBM

© Copyright IBM Corporation 2007, 2013.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Introduction .................................................. xv

What’s new .................................................. xvii
New features in version 10.2.1 ................................ xvii
New features in version 10.2.0 ................................ xvii
New features in version 10.1.1 ................................ xvii

Chapter 1. Getting started .................................. 1
Launching Real-time Monitoring Workbench .................. 1
Working with the product .................................... 2

Chapter 2. Access filters .................................... 3
Access filter conditions .................................... 3
  User names in access filters ............................... 4
  Lookup table in access filters ............................. 4
  User lookup tables in access filters ....................... 4
Access filter behavior and restrictions ...................... 5
Creating an access filter on a view ......................... 6
Creating access filters based on lookup tables .......... 7
Creating a simple access filter ............................ 7
Assigning an access filter to users and roles ............ 8

Chapter 3. Adapter Framework .............................. 9
Creating an external adapter framework ................. 9
External adapter plug-in ................................ 10
  Configuration file setup ................................ 12
  The ConfigXML file .................................. 12
  JAR files folder .................................... 13
Compile and start an external application ............... 14
External agents .......................................... 15
External data streams ................................... 15
External lookup tables .................................. 17

Chapter 4. Agents .......................................... 21
External sources ........................................ 21
Creating agents ......................................... 23
Editing agents .......................................... 24
Flat file agents .......................................... 25
  Creating a flat file agent .............................. 26
  The file agent program ............................... 26
File system agents ...................................... 29
  Creating a file system agent ......................... 29
Java Messaging Service queue agents ................ 29
  JNDI properties for connecting to a remote namespace ........................................ 31
Creating a JMS queue agent ............................ 32
JMS topic agents ......................................... 32
Creating a JMS topic agent .............................. 33
TIBCO Rendezvous agents ................................. 34
  Attributes .......................................... 34
Creating a TIBCO rendezvous agent ...................... 35
Web service agents ..................................... 35
  Web service agent attributes ......................... 36
Creating a web service agent ........................... 36

© Copyright IBM Corp. 2007, 2013
# IBM Cognos Real-time Monitoring Workbench Version 10.2.1: Modeling Reference

## Chapter 5. Alerts
- Creating a stand-alone alert .......................................................... 37
- Creating an alert associated with a rule ........................................... 38
- Cloning an existing alert ................................................................. 38
- Alert attributes ............................................................................... 38
- Message subject and body text ......................................................... 39
- Alert subscribers .............................................................................. 40
- Alert notification message management ......................................... 41
- Alert states ..................................................................................... 41
  - Alert escalation ........................................................................... 42
- Consolidating multiple messages ...................................................... 42
- Setting an alert to invoke an external web service ............................ 42
  - The web service user-defined function ......................................... 43
- Adding reportlets to an alert notification ....................................... 44
- Reportlet filtering .......................................................................... 44
  - Reportlet data based on option ..................................................... 46

## Chapter 6. Business rules
- About business rules ....................................................................... 49
- Creating a business rule .................................................................. 49

## Chapter 7. Common Event Infrastructure
- How Common Event Infrastructure works ...................................... 51
- Common Event Infrastructure data stream ...................................... 51
  - Prerequisites for creating a Common Event Infrastructure data stream......................................................... 52
  - Creating a data stream table from a Common Event Infrastructure connection ........................................... 53
- Common Event Infrastructure column information ....................... 54
- Clear state interval ......................................................................... 54

## Chapter 8. Cubes
- Measures ....................................................................................... 57
- Dimensions in cubes ...................................................................... 58
- Creating cubes ............................................................................... 59
  - Measure columns ....................................................................... 60
- Creating a cube with dimension columns ..................................... 60

## Chapter 9. Data types
- Data type conversion ................................................................. 63
  - Explicit cast ............................................................................... 63
  - Implicit cast ............................................................................... 63
  - Order of precedence .................................................................... 64
- C-SQL data types for numeric values ........................................... 65
  - Third-party data types that map to C-SQL numeric data types .... 65
  - Combining numeric types ......................................................... 66
  - Casting numeric types .............................................................. 66
  - Numeric to string guidelines ..................................................... 67
  - Decimal precision results ......................................................... 67
  - Casting .................................................................................... 67
  - Multiplication ........................................................................... 67
  - Division ................................................................................... 67
  - Addition and subtraction ........................................................... 68
  - All other functions .................................................................... 68
- String ............................................................................................ 68
  - String width ............................................................................. 68
- Third-party data types that map to VARCHAR ................................. 68
- String concatenation ..................................................................... 69
- String literal ................................................................................. 69
- Converting strings to other data types ....................................... 69
- Date-time ...................................................................................... 69
  - Converting between date-time and strings ............................... 70
Chapter 16. HTTP Post ........................................ 201

How HTTP Post works .................................. 201
Data stream tables for HTTP Post ....................... 201
   Creating a data stream table for HTTP Post ....... 202
   HTTP Post column information ..................... 203
Post to a data stream table from HTTP Post ........... 204
   Post to message fields .......................... 205
   Values posted in the URL ......................... 205
   Multiple events of input ........................... 206

Chapter 17. Java Messaging Service .................... 207

JMS data streams ......................................... 207
   Creating a data stream table from a Java Messaging Service connection .... 209
   Java Messaging Service column information .... 210
   Java Messaging Service data types ................. 210

Chapter 18. Java Database Connectivity ................. 211

Java Database Connectivity tables ...................... 211
   Lookup tables ....................................... 211
   Creating a Java Database Connectivity source for a data stream table .... 212
   Creating a Java Database Connectivity source for a lookup table ......... 213
   Data stream tables .................................. 213
   Query source ....................................... 215
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
<th>Start Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>306</td>
<td>Salesforce picklist function</td>
<td></td>
</tr>
<tr>
<td>307</td>
<td>Salesforce administration console</td>
<td></td>
</tr>
<tr>
<td>309</td>
<td><strong>Chapter 30. SAP connectivity</strong></td>
<td></td>
</tr>
<tr>
<td>309</td>
<td>Creating an SAP agent</td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>Importing operational data store objects</td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>Importing OLAP cubes</td>
<td></td>
</tr>
<tr>
<td>311</td>
<td><strong>Chapter 31. Scenarios</strong></td>
<td></td>
</tr>
<tr>
<td>312</td>
<td>Creating scenarios</td>
<td></td>
</tr>
<tr>
<td>313</td>
<td><strong>Chapter 32. SELECT statements</strong></td>
<td></td>
</tr>
<tr>
<td>313</td>
<td>Syntax</td>
<td></td>
</tr>
<tr>
<td>314</td>
<td>Select list</td>
<td></td>
</tr>
<tr>
<td>314</td>
<td>CASE expression</td>
<td></td>
</tr>
<tr>
<td>316</td>
<td>FROM clause</td>
<td></td>
</tr>
<tr>
<td>316</td>
<td>View constraints</td>
<td></td>
</tr>
<tr>
<td>316</td>
<td>Join operations</td>
<td></td>
</tr>
<tr>
<td>317</td>
<td>Inner joins</td>
<td></td>
</tr>
<tr>
<td>317</td>
<td>Outer joins</td>
<td></td>
</tr>
<tr>
<td>317</td>
<td>Nested joins</td>
<td></td>
</tr>
<tr>
<td>317</td>
<td>Table expressions</td>
<td></td>
</tr>
<tr>
<td>318</td>
<td>Syntax</td>
<td></td>
</tr>
<tr>
<td>318</td>
<td>Restrictions</td>
<td></td>
</tr>
<tr>
<td>318</td>
<td>HAVING clause example</td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>WHERE clause</td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>Predicates</td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>Aliases</td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>GROUP BY clause</td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>Aliases</td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>Derived views</td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>Stateful view semantics</td>
<td></td>
</tr>
<tr>
<td>322</td>
<td>ORDER BY clause</td>
<td></td>
</tr>
<tr>
<td>323</td>
<td><strong>Chapter 33. TIBCO Rendezvous</strong></td>
<td></td>
</tr>
<tr>
<td>323</td>
<td>How TIBCO Rendezvous works</td>
<td></td>
</tr>
<tr>
<td>323</td>
<td>TIBCO Rendezvous tables</td>
<td></td>
</tr>
<tr>
<td>323</td>
<td>Limitations</td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>Creating a data stream table for TIBCO Rendezvous</td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>TIBCO column information</td>
<td></td>
</tr>
<tr>
<td>325</td>
<td>TIBCO Rendezvous data types</td>
<td></td>
</tr>
<tr>
<td>329</td>
<td><strong>Chapter 34. Users</strong></td>
<td></td>
</tr>
<tr>
<td>329</td>
<td>Edit your own account information</td>
<td></td>
</tr>
<tr>
<td>329</td>
<td>Create, delete, or edit a user account</td>
<td></td>
</tr>
<tr>
<td>329</td>
<td>User details tab</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td>Delivery profiles tab</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td>Email</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td>Web service</td>
<td></td>
</tr>
<tr>
<td>331</td>
<td>Access permissions tab</td>
<td></td>
</tr>
<tr>
<td>333</td>
<td><strong>Chapter 35. User-defined functions</strong></td>
<td></td>
</tr>
<tr>
<td>333</td>
<td>User-defined function restrictions</td>
<td></td>
</tr>
<tr>
<td>333</td>
<td>Creating and using a user-designed function</td>
<td></td>
</tr>
<tr>
<td>334</td>
<td>Altering an existing user-designed function</td>
<td></td>
</tr>
<tr>
<td>334</td>
<td>Manifest files</td>
<td></td>
</tr>
<tr>
<td>337</td>
<td><strong>Chapter 36. Views</strong></td>
<td></td>
</tr>
<tr>
<td>337</td>
<td>Creating views</td>
<td></td>
</tr>
</tbody>
</table>
getSettings .................................. 434
onPropertyChange ............................... 434
Credential ................................... 434
getPassword.................................. 434
getUserName ................................. 435
setPassword .................................. 435
setUserName ................................. 435
Role ..................................... 435
getAttributes ................................. 435
getRoleDescription ............................... 435
getRoleName .................................. 436
getUsers ................................... 436
setAttributes.................................. 436
setRoleDescription................................ 436
setRoleName .................................. 437
setUsers ................................... 437
SessionInfo ................................... 437
getAttributes ................................. 437
getUser ................................... 437
setAttributes.................................. 437
setUser ................................... 438
User ..................................... 438
getAttributes ................................. 438
getUserDescription ............................... 438
getUserId ................................... 438
setAttributes.................................. 439
setUserDescription ............................... 439
setUserId ................................... 439
Exceptions ................................... 439
ApplicationNotAvailableException .......................... 440
InvalidCredentialException ............................. 440
SessionTimedOutException ............................. 440
Notices ................................... 441
Glossary .................................. 445
B ...................................... 445
C ...................................... 445
D ...................................... 445
E ....................................... 445
I ....................................... 445
M ...................................... 445
O ...................................... 445
Q ...................................... 446
R ...................................... 446
S ...................................... 446
T ...................................... 446
U ...................................... 446
V ...................................... 446
Index .................................... 447
Introduction

This document is intended for use with IBM® Cognos® Real-time Monitoring Workbench.

Audience

This document is intended for administrators and modelers.

Finding information

To find IBM Cognos product documentation on the web, including all translated documentation, access one of the IBM Cognos Information Centers (http://pic.dhe.ibm.com/infocenter/cogic/v1r0m0/index.jsp). Release Notes are published directly to Information Centers, and include links to the latest technotes and APARs.

You can also read PDF versions of the product release notes and installation guides directly from IBM Cognos product disks.

Accessibility features

IBM Cognos Real-time Monitoring Workbench does not currently support accessibility features that help users with a physical disability, such as restricted mobility or limited vision, to use this product.

Forward-looking statements

This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Samples disclaimer

The Sample Outdoors Company, Great Outdoors Company, GO Sales, any variation of the Sample Outdoors or Great Outdoors names, and Planning Sample depict fictitious business operations with sample data used to develop sample applications for IBM and IBM customers. These fictitious records include sample data for sales transactions, product distribution, finance, and human resources. Any resemblance to actual names, addresses, contact numbers, or transaction values is coincidental. Other sample files may contain fictional data manually or machine generated, factual data compiled from academic or public sources, or data used with permission of the copyright holder, for use as sample data to develop sample applications. Product names referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.
What's new

This section contains a list of new, changed, and removed features for this release. It will help you plan your upgrade and application deployment strategies and the training requirements for your users.

For information about upgrading, see the IBM Cognos Business Intelligence Installation and Configuration Guide for your product.

For information about other new features for this release, see IBM Cognos Business Intelligence New Features.

What's New information for past releases, including versions 8.3 and 8.4, is available by accessing documentation within the IBM Cognos Business Intelligence 10.2.1 information center (http://pic.dhe.ibm.com/infocenter/cbi/v10r2m1/index.jsp)

To review an up-to-date list of environments supported by IBM Cognos products, such as operating systems, patches, browsers, web servers, directory servers, database servers, and application servers, visit the IBM Cognos Customer Center (http://www.ibm.com/software/data/cognos/customercenter/).

New features in version 10.2.1

There are no new features in this release of IBM Cognos Real-time Monitoring Workbench.

New features in version 10.2.0

Listed below are new features since the last release.

Workbench Technical Reference is now the Workbench Modeling Reference

The document formerly named the IBM Cognos Real-time Monitoring Workbench Technical Reference is now named the IBM Cognos Real-time Monitoring Workbench Modeling Reference.

New features in version 10.1.1

Listed below are new features for version 10.1.1.

Closer integration between IBM Cognos Real-time Monitoring Dashboard and IBM Cognos Business Insight

When working with IBM Cognos Real-time Monitoring dashboard objects in IBM Cognos Business Insight, you can:

- convert one display type into another
- change the measure styles in a display
- change specific properties in a display
- use IBM Cognos Business Insight filter widgets
For more information, see the topics about changing display types, measure styles, and chart specific properties, and filtering widgets in the *IBM Cognos Real-time Monitoring Dashboard User Guide*.

**Streaming lookup tables support inline dimensions**

By using a streaming lookup table based on an event stream, the values in the lookup table are updated as quickly as the data is processed and can be sourced from non-JDBC sources such as JMS or WebSphere® MQ. You can now use inline dimensions in cubes and views without the need to create joins back to the dimension - you create the dimension in a lookup table using the same steps as a traditional dimension. For more information, see the topic about creating lookup tables on data streams and views in the *IBM Cognos Real-time Monitoring Workbench User Guide*.

**Support for upsert extended to flat-file and JDBC message service formats**

In version 10.1.0, the upsert functionality, which combines the actions of updating, inserting, and deleting data from data stream tables, was only available for Java™ Database Connectivity (JDBC) data streams. Version 10.1.1 extends support of this feature to flat file streams, JMS streams, and others. For more information, see the topics about enabling upsert for a data stream in the *IBM Cognos Real-time Monitoring Workbench User Guide*. See also the topic about upsert in the *IBM Cognos Real-time Monitoring Workbench Modeling Reference*.

**Search for objects**

In previous releases, finding objects of interest by using sorted lists could be time-consuming. Now, you can search for specific text contained in the XML definition. For more information, see the topic about searching the object library in the *IBM Cognos Real-time Monitoring Workbench User Guide*.

**View object dependencies and requirements**

You can use the Relationships tab to see all dependencies and requirements for an object. For more information, see the topic about object dependencies and requirements in the *IBM Cognos Real-time Monitoring Workbench User Guide*.

**Build an external adapter framework**

You can configure an external adapter to convert data in a previously unsupported format (such as JSON, and RSS Feed) into a format supported by RTM (such as tabular, flat file, XML). For more information, see the topic about adapter frameworks in the *IBM Cognos Real-time Monitoring Workbench Modeling Reference*.
Chapter 1. Getting started

Your system administrator must first install, configure, and start the IBM Cognos Real-time Monitoring server components. To receive events, or to retrieve lookup table data, the servers must locate and collect the data. For example, to access a DBMS to retrieve lookup table data, the servers must be configured with the access name and password for the database.

For more information, see the IBM Cognos Real-time Monitoring Installation and Configuration Guide.

Launching Real-time Monitoring Workbench

If you are a system administrator, you access IBM Cognos Real-time Monitoring Workbench first so that you can configure the system settings before other users access the product.

Before you begin

Before launching IBM Cognos Real-time Monitoring Workbench, ensure that the following conditions are met:

- Your web browser is Microsoft Internet Explorer 6.0 or newer.
- Cognos Real-time Monitoring servers are running.
- JavaScript is enabled.
- Adobe Flash version 9 or later is installed.

About this task

You must use the rtmadmin user account with the password manager to log in. For more information, see the section about configuring Cognos Real-time Monitoring after installation in the IBM Cognos Real-time Monitoring Installation and Configuration Guide.

When safe mode is enabled, Cognos Real-time Monitoring Workbench displays a banner indicating that the system is in safe mode and that all events are paused. Data streaming stops when the system is in safe mode. For more information, see the IBM Cognos Real-time Monitoring Installation and Configuration Guide.

Procedure

1. In your Web browser, type the following URL, where servername represents the host computer name and port number:
   http://servername/cognos/realtime/landing/landingpage.htm
   The Cognos Real-time Monitoring Welcome page opens.

2. Depending on your role as an administrator, modeler, or developer, click one of the following links:
   - Administer real-time content.
   - Model my real-time data.
   - Create my real-time dashboard.

3. Log in using the user name and password assigned by your administrator.
Note: If single sign-on between IBM Cognos Business Intelligence and Cognos Real-time Monitoring is enabled, and you are already logged in to IBM Cognos Business Intelligence, the login page does not appear. You are automatically logged in to Cognos Real-time Monitoring Workbench.

Working with the product

You can perform the following tasks using IBM Cognos Real-time Monitoring Workbench.

• Setting up user accounts.
  Everyone who uses Cognos Real-time Monitoring Workbench or Cognos Real-time Monitoring Dashboard needs a user account. Users can view their account information by clicking Account Settings in the application. System administrators can add and modify user accounts using the Administration Console.

• Creating and modifying agents.
  Agents collect events and lookup table data to make it available to Real-time Monitoring. System administrators create and modify agents using the Administration Console.

• Creating data streams and lookup tables.
  Agents insert data into data stream tables and lookup tables. These tables are the source of the business views that present and aggregate the information. You can create data streams and lookup tables using Workbench.

• Using business views.
  Business views contain, aggregate, and provide data stream and lookup table information. You can create complex business views using Scenario Modeler.

• Creating scenarios, rules, and alerts.
  You can use Cognos Real-time Monitoring to identify exceptional business events and notify users about the activity. You do this by creating scenarios, rules, and alerts in Scenario Modeler. You can also use Scenario Modeler to define reportlets that provide metrics associated with the events.
Chapter 2. Access filters

Different users can use access filters to see different rows of the same view or cube, depending on the criteria specified in the filter, eliminating the need to define a new view or cube for each user.

For example, consider this view of total sales by region:

<table>
<thead>
<tr>
<th>Total Sales Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>763000.00 West</td>
</tr>
<tr>
<td>489500.00 Central</td>
</tr>
<tr>
<td>522950.00 South</td>
</tr>
<tr>
<td>650740.00 East</td>
</tr>
</tbody>
</table>

By defining an access filter that specifies, for example, `OrderTotals.Region=Employees.Region`, you can limit users to seeing only the rows that apply to their business region. In this example, a user from the Central region looking at the view would see the following result:

<table>
<thead>
<tr>
<th>Total Sales Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>489500.00 Central</td>
</tr>
</tbody>
</table>

The filters are logical expressions similar to the WHERE clause of a view definition. For a complete description of filters, see "Access filter conditions." For more information about WHERE clauses, see "WHERE clause" on page 319.

Access filters are defined on a view-by-view and cube-by-cube basis, and are applied to users and roles having Filtered/Read-Only permission on the view or cube. For details about how access filters work, see "Access filter behavior and restrictions" on page 5.

Applying access filters to a view or cube requires that you first create the filter, and then assign it to users or roles, as described in the following sections:

- "Creating an access filter on a view" on page 6
- "Assigning an access filter to users and roles" on page 8

Access filter conditions

Access filter conditions are logical expressions that are applied to each row in the view, or dimension level in a cube.

A user looking at the view or cube sees only those rows where the expression evaluates to true.

At a minimum, each filter must contain some condition that evaluates data found in the view or cube. For example, the following simple condition shows only the rows in OrderTotals that are in the East business region:

`OrderTotals.Region='East'`
Access filters can include Boolean operators (AND, OR, and NOT), and they can use parentheses for grouping. For example, OrderTotals.Region='East' AND (CURRENT_USER()='Skyler' OR CURRENT_USER()='Nina')

**User names in access filters**

An access filter can include the names of specific users.

An access filter such as OrderTotal.Region='East' must be assigned to each user or role in the East region to limit the access for that user. A more powerful expression is one that names the user. The CURRENT_USER (see “CURRENT_USER” on page 154) function returns the login name of the user looking at the view. You can include that function in the filter condition to apply the filter to specific users.

For example, the following condition also identifies two users, so only these two users see the results for the East region:

```
OrderTotals.Region='East' AND (CURRENT_USER()='Skyler' OR CURRENT_USER()='Nina')
```

A limitation of the previous two examples is that they have literal values hard-coded into the expressions: the region name and the user names. This is problematic because you must edit the filters whenever the names change, and you must create an expression for each region.

A more powerful expression is one that you can apply to all users by dynamically retrieving information about the user and applying it to the view.

**Lookup table in access filters**

In addition to retrieving data from the current view, access filters can retrieve data from a lookup table.

If you define a lookup table that contains information about the users, you can compare that information to the data in the view to create a dynamic lookup table filter.

Consider the following filter that uses an Employees lookup table:

```
OrderTotals.Region=Employees.Region AND CURRENT_USER()=Employees.User_Name
```

Now you can apply the filter to many users and roles, and only those users assigned to the same business region as the data see the data. Examples are shown in the following sections.

To use dynamic lookups you must provide the information in an external lookup table.

**User lookup tables in access filters**

Lookup tables usually support data stream tables by providing additional contextual information.

When used in an access filter, a lookup table provides information that supports the filter; for example, information about the current user.
A lookup table for a user must have at least one column that contains the user name that matches the name that the user uses to log in to IBM Cognos Real-time Monitoring.

The CURRENT_USER function returns the user's login name as defined in IBM Cognos Real-time Monitoring, in the same character case, and as it appears in Cognos Real-time Monitoring Workbench. It is important that the character case is an exact match. Some DBMSs provide compares that are not case-sensitive, so this might not be an issue. To avoid mismatches, enter the names in the lookup table in a single case, and then use UPPER() or LOWER() in the filter expression, for example:

```
UPPER(CURRENT_USER())=Employees.User_Name
```

You cannot use UPPER() or LOWER() on the reference to the Employees lookup table. See “Column limitations in queries for lookup tables” on page 249 for details.

Similarly, all text columns referenced in a filter need to be aware of case issues.

To use a lookup table in an access filter, add the table to the filter's workset when defining the filter. See "Creating an access filter on a view" on page 6 for details. For more information about lookup tables, see Chapter 20, “Lookup tables,” on page 247.

A lookup table for a user must include the following conditions:

- One row for each user that is assigned a Filtered/Read-Only access permission. If the user is not found in the lookup table, the filter fails to find any rows for that user.
  For more information about permissions, see "Access permissions" on page 275.
- At least one text column that contains the user login name. If the DBMS provides case-sensitive matches, enter the names exactly as defined in Cognos Real-time Monitoring. If you plan to use UPPER() or LOWER() in your filters, enter the user name with the same characters in one text-case.
- One column for each reference in the filter, and the data types must match. For character values, the strings in the view must exactly match the strings in the lookup table.

If the lookup table data is cached, the filters can fail if the user data is not in the cache. If you add a user to the database, invalidate the lookup table cache before the user attempts to look at filtered views or cubes. For more information about caching, see “Caching lookup table queries” on page 250.

---

**Access filter behavior and restrictions**

Access filters are applied only when a user with Filtered/Read-only permission on a view looks at or requests data from the view, or defines a new view on top of the view.

The filters do not affect users or roles with Read-only or Read andWrite permission on the view, nor do they apply to users receiving reportlets sent as attachments to alert notifications.

When using access filters, remember:
The default access permission to the classes of view and cube objects is "No Access for all new users". Before any user can see the results, they must be assigned (directly or as a member of a role) Read-only or read/write permission on the types of View and Cube objects (all views and cubes) at least Filtered/Read-only permission on the specific view.

When a user is assigned multiple access filters to the same view or cube (perhaps as the result of being a member of multiple roles each with assigned filters) the user sees those rows where any of the filters is true for the row. For example, one filter might restrict a user to see West region data, but another might allow the user to see all results for a specific family of products. The result is that the user sees all results for the family, regardless of region.

Reportlets always include all data from the view that they reference, regardless of any access filters associated with the view. Users that receive reportlets as part of alert notifications always see the entire view referenced by the reportlet.

When a user with filtered access to a view creates a new view on top of the filtered view, the new view inherits that user’s filtered results, but not the filter definition. Subsequently, anyone else looking at the derived view sees the results as filtered for the creating user. For example, if Skyler can see only Total Sales from the West region, and he creates a new view called WrapUp derived from the Total Sales view, anyone else with permission to look at WrapUp sees the data for the West region, regardless of their own access permission to Total Sales.

When a user with filtered access to a view creates a rule based on the view, the rule inherits that user’s filtered results. That rule only applies to events that match the user’s access filter condition, and any subscribers to the alert associated with the view receive alerts only for the filtered events.

For more information about permissions, see “Access permissions” on page 275.

Creating an access filter on a view

To create an access filter, you must have read/write permission on the view.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab and select the view that will have the filter.
2. Select the Access Filters tab to see the list of filters currently associated with this view.
3. Click Create Access Filter to create a new filter.
4. Add a lookup table to the workset by clicking Add Lookup Tables.
5. In the Create Access Filter box, assign the filter's name and optionally provide a description.
6. Define the filter condition following the instructions in “Access filter conditions” on page 3. You can now assign the filter to users that have Read-Filtered access to the view.
Creating access filters based on lookup tables

You can create access filters based on lookup tables. When used in an access filter, a lookup table provides information that supports the filter; for example, information about the current user.

Before you begin

You must have read/write permission on the cube.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Select the cube on which you want to create an access filter.
3. Click the Access Filters tab to see the list of filters currently associated with this cube.
4. Click Create Access Filter.
5. In the Configure Cube Filter window, assign the filter’s name and optionally provide a description.
6. Click Add Filter. If you create more than one filter, an AND operation is performed on the filters to produce the result of the access filter.
7. Choose a function.
8. Click Browse and select a lookup table and columns.
9. Click Browse and select a dimension and member.
10. Assign the filter to users that have Filtered/Read-only access to the cube. For information about how your definition affects filter conditions, see “Access filter conditions” on page 3.

Creating a simple access filter

Simple filters restrict the view results to include only those rows that meet the specified condition.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Select the cube on which you want to create an access filter.
3. Click the Access Filters tab to see the list of filters currently associated with this cube.
4. Click Create Access Filter.
5. Select Use simple filters.
6. Click Browse and select the dimension to use for the simple filter.
7. Select a member. For example, if you have a location dimension, one of the members might be a region or state.
8. Select a value for the filter. For example, if you selected a state as the member in a location dimension, the value might be California.
9. Click Add.
10. Click Save.
11. Assign the filter to users that have Filtered / Read-only access to the cube.
Assigning an access filter to users and roles

You can assign an access filter to a user or role.

When multiple filters are assigned, the user sees the rows that meet any of the conditions. For example, when one filter shows only rows that are in the user’s department classification, and another shows rows applicable to their business region, the user sees the row if either condition is true.

**Before you begin**

To perform this procedure you must have read/write permission on the view or cube, and the filters must already be defined.

**Procedure**

1. In IBM Cognos Real-time Monitoring Workbench, click the **Workbench** tab.
2. Select the view to which you want to assign the filtered read permission.
3. Click **Activities**, and select **Permissions**.
4. Select one or more users or roles to have the filtered access.
5. Click **Change Permissions**.
6. Choose **Filtered/Read only**, then select one or more access filters to use.
7. Click **OK**.
8. Click **Done**.
Chapter 3. Adapter Framework

The adapter framework is a set of components that you can use to add external agents to IBM Cognos Real-time Monitoring.

Some agents are tightly coupled to the data source that they use; it is not possible for Real-time Monitoring to communicate with the data source to exchange data.

The adapter framework makes it possible to connect IBM Cognos Real-time Monitoring to external applications and data sources using your own external agent, rather than one of the agents available in IBM Cognos Real-time Monitoring. You can create external adapters that are configurable and can support general operations such as start, stop, and poll.

When you implement an external adapter agent, you can also create data stream and lookup table objects that use your agent.

The adapter framework consists of several components that you must implement to connect your external application to IBM Cognos Real-time Monitoring. These components are:

- External agent configuration file
  The external agent configuration file is an XML file that specifies the configuration of the external agent. The configuration file specifies the type of data that the agent receives, which also determines the types of data streams and lookup tables available to you after you configure an external agent. For more information, see "Configuration file setup" on page 12.

- External agent
  You define the external agent in IBM Cognos Real-time Monitoring to connect to your external application. For more information, see "Creating agents" on page 23.

- External client
  The external client is the external application that adapts your data source to the requirements of Cognos Real-time Monitoring. The external client has two components: the external client framework and the external client plug-in.
  - External client framework
    You implement the external client framework to handle the receiving of commands and the sending of data to Cognos Real-time Monitoring. The external client framework is provided to you in the Java archive (JAR) file that contains the APIs for the external client plug-in.
  - External client plug-in
    You create the external client plug-in using the APIs of the external client framework. The plug-in retrieves data from your data source.

Creating an external adapter framework

You can create a framework to add external agents to IBM Cognos Real-time Monitoring.
**Procedure**

1. Implement an external client plug-in.
   For more information, see "External adapter plug-in."

2. Create an AgentConfig XML file based on your plug-in.
   For more information, see "Configuration file setup" on page 12.

3. Package the ExternalAgentConfig XML into an ExternalAgentConfig JAR file.
   For more information, see "The ConfigXML file" on page 12.

4. Upload the JAR file to Cognos Real-time Monitoring Workbench as a JAR file.
   For more information, see "JAR files folder" on page 13.

5. Place your compiled plug-in together with the external adapter JAR file onto
the machine from which your plug-in can access the data for which it was
created. Include the classes of the plug-in in the class path when starting the
external adapter.
   For more information, see "Compile and start an external application" on page 14.

6. In Cognos Real-time Monitoring Workbench, create an agent of the type
defined in your ExternalAgentConfig XML file.

7. Create a data stream or lookup table in the Workbench, using the agent you
created.

8. Use the data in Cognos Real-time Monitoring to build views and cubes.

---

**External adapter plug-in**

You can use an existing plug-in, such as the one that comes with IBM Cognos
Real-time Monitoring, or you can implement a new plug-in.

The plug-in implements a class that extends from the class

This class extracts data from the source system and translates it into one of the
data formats that the external adapter framework supports.

The following methods must be implemented in the plug-in class. You can see an
example here: install_location/webcontent/samples/sdk/adapters/
jdbc_sample/src/com/externalApplication/jdbc, where install_location specifies
the IBM Cognos Business Intelligence installation location.

**Init()**

This method is called when the external adapter data stream is created. This
method has two parameters, one for agent details and one for data stream details.
If a procedure needs to be executed during initialization of the data stream, this
method can add it.

**Attention:** No method is called when the agent is created in the Workbench.
Validation of the agent input data can be done using the testAgentCredentials()
method described in a following section.

**Destroy()**

This method is called during the deletion of the data stream. It flushes all the data
created during the init() method.
testAgentCredentials()

This method is used for the Test Connection functionality. This method can be used to validate the data provided for the agent. If the agent does not require validation, then this method must be implemented empty, which returns a default of true.

validateDataStream()

This method validates the data stream data that the plug-in receives. The plug-in provides the default implementation for this method, which returns true. For example, you have a JDBC application and you provide a table name as input. Validation for the data stream checks that the table name is correct or that the table name exists in the database. This method is optional.

getTabularData()

This method gets the data from the tabular data stream and puts the data in the response object in a format that Cognos Real-time Monitoring recognizes. This method has two parameters, one for the data stream details and one for the response, which is sent back to the Cognos Real-time Monitoring application.

The IExternalAdapterTabularDataStreams getInputData() call returns the input specified in the Input field of the data source in Cognos Real-time Monitoring Workbench. The isIncrementalField() method of the IExternalAdapterTabularDataStreamColumn objects indicate whether the plug-in treats the column as an incremental field. The plug-in maintains the largest value encountered by this column. When the plug-in returns data, it takes account of this largest value and the value returned by getInitialValue().

Each cell object represents the separate columns specified during the creation of the data stream. The class object is ExternalAdapterTabularDataStreamCellData. It contains two different objects: the data type of the column and the actual data.

getFlatFileData() and getXMLData()

These methods are used for the flat file and XML data streams.

These files also accept the two parameters as in getTabularData(). However, the response object format is different from the tabular one. For both of these functions, instead of sending a list of rows, a FileInputStream is populated.

Exception.properties

The exception.properties file displays a detailed localized message of the exception in the user interface.

Exception classes, which are created in the API, take a message key as a parameter. When a key is passed to the exception class, it first extracts the message from exception.properties file for the appropriate locale and displays it on the user interface.

You can see an existing exception properties file, exception_en.properties. By default, this file is in install_location/realtime/webcontent/samples/sdk/adapters/template, where install_location specifies the IBM Cognos BI installation location.
Configuration file setup

The configuration file determines what transport types and what options are available in the Create Agent dialog box.

You can use an existing sample configuration file, such as sampleAgentConfig1.xml. By default, this file is in install_location/realtime/webcontent/samples/sdk/adapters/jdbc_sample/config, where install_location specifies the IBM Cognos BI installation location.

The configuration file is an XML file that contains:
- name – the identifier for this agent.
- nameKey – the type name of agent. This is a localizable key for the string that is displayed in the Create Agent dialog box.
- supportsIncrementingField – a Boolean flag that indicates whether the adapter supports an incrementing field.
- transportType – a comma-separated list that indicates the format of the data. This determines what the user interface renders.
  - TABULAR
  - DELIMITED
  - FIXEDWIDTH
  - XMLFILE
- ExternalAgentVersionSupported – version of the external adapter plug-in.

The ConfigXML file

You compress the ConfigXML in a Java archive (JAR) file.

The JAR file uses a folder structure similar to the following illustration:

![Folder structure of JAR file](image)

<table>
<thead>
<tr>
<th>JAR file component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration files</td>
<td>When you upload the JAR file, IBM Cognos Real-time Monitoring converts these XML files into ExternalAgentConfig objects. You see these objects as the dependencies of the uploaded JAR file.</td>
</tr>
<tr>
<td>Manifest file</td>
<td>This file provides the representation of the JAR to Cognos Real-time Monitoring. This file contains the information for configuration files and the resource bundle.</td>
</tr>
</tbody>
</table>
Table 1. JAR file components (continued)

<table>
<thead>
<tr>
<th>JAR file component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource files</td>
<td>These files provide the localized strings. This string is the display name of the ExternalAgentConfig object. The localized string is provided to Cognos Real-time Monitoring according to the locale of the application. The keys in the resource file are given in the configuration file as name key. When the agent config object is created, using the name key and locale, the actual string from the resource bundle is fetched and displayed on the screen.</td>
</tr>
</tbody>
</table>

You can use a Java build tool, such as Ant, to create the JAR file. You use Ant to compile and build the adapter, compress the files and create a JAR folder.

If you use Ant to automate this process, you can use the included sample build.xml, located here: install_location\realtime\webcontent\samples\sdk\fileagent\com\ibm\cognos\externaladapter, where install_location specifies the IBM Cognos BI installation location.

You run the build-config-jar target to build the ConfigXML JAR file.

If you want to do this manually, you must create a folder and inside that folder, recreate the same directory structure shown in the preceding diagram. Then you change directories to the created folder and execute the command that creates the JAR.

When you create the JAR, do not add the root folder in the JAR file.
- Use the DOS command CD to navigate to the root folder.
  - For example, if the name of the root folder is AgentConfig:
    
    ```
    CD <path of the root directory> AgentConfig
    ```
  - Then you run the command:
    ```
    jar cvf * <jar file name>
    ```
    The 'v' is optional.

**JAR files folder**

JAR files contain one or more Java programs.

The JAR files folder provides access to the JAR files available to IBM Cognos Real-time Monitoring. Select the folder to see a list of the JAR files already installed, and to upload new JAR files. Select a JAR file in the folder to see the functions that JAR contains and the objects that depend on the functions in that JAR.

**Procedure**
1. From the Workbench, click the JAR Files folder.
2. Click New JAR.
3. Type a name for your JAR in the Name field, and optionally, a description.
4. Click Browse to locate the file in the Path field.
5. Click Save.
   - The JAR file is now available to the system.
Compile and start an external application

You use a build.xml file to compile the external adapter plug-in.

The main class value in the build.xml file is always com.ibm.cognos.externaladapter.ExternalAdapterServer because this class starts the external server. The file ExternalAdapter.jar contains the core external adapter
classes required for the compilation of the plug-in class.

Starting the external adapter process

To run the external agent, you must invoke the Java process with the
com.ibm.cognos.externaladapter.ExternalAdapterServer class and a
properly configured classpath. The classpath must include the ExternalAdapter.jar,
the JAR file containing the classes for your plug-in implementation, any JAR files
that your implementation depends on, log4j-1.2.7.jar, and the directory where
your config.properties and exception*.properties files reside. For example:
java
-classpath lib\ExternalAdapter.jar;lib\your plug-in JAR;lib\log4j-1.2.7.jar;lib\SampleApplication.jar;
com.ibm.cognos.externaladapter.ExternalAdapterServer

Attention: If using the supplied example build.xml, then the run target sets up
the classpath and starts the external adapter.

Values of the parameters in the build.xml file

• Property for the directory that contains all the build files:
  <property name="build.dir" value="ExternalAdapterSampleApplication"/>

• Property for the name and destination of the executable sample application file:
  <property name="externalapplicationjar" value="SampleApplication.jar"/>

• Property for the source directory that contains all the Java files created for
  plug-in implementation, such as plug-in class:
  <property name="src" value="."/>

• Property for the directory that contains all the compiled classes:
  <property name="classes" value="${build.dir}"/>

config.properties

A configuration file is necessary to run external applications. The server class uses
this file to load specific configuration properties, such as the PORT property.

PORT specifies the port number on which the external application runs and
communicates with IBM Cognos Real-time monitoring.

For an example of a config.properties file, see here: install_location\realtime\webcontent\samples\sdk\fileagent\com\ibm\cognos\externaladapter, where
install_location specifies the IBM Cognos BI installation location.

Additional JAR Files

The external application requires additional JAR files while booting.

The external application requires the files log4j-1.2.7.jar, ExternalAdapter.jar,
and SampleApplication.jar. If starting the application requires more JAR files,
such as JDBC driver JAR files, you must add them to the classpath when you run the external adapter.

**External agents**

You can create an external agent based on the ExternalAgentConfig object of an uploaded JAR file.

Create an agent. For more information, see “Creating agents” on page 23.

In the **Agent Type** dropdown list, look for the localized string of the ExternalAgentTypeNameKey, as defined in the ExternalAgentConfig object. For more information, see “Configuration file setup” on page 12.

After you select the agent, under **Agent Details**, specify the following attributes:

*Table 2. External agent details*

<table>
<thead>
<tr>
<th>Agent detail</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>External Host Name</strong></td>
<td>The IP address of the machine where external application is running.</td>
</tr>
<tr>
<td><strong>External Port</strong></td>
<td>The port number on which the external application is running, specified in the config.properties file. For more information, see “Compile and start an external application” on page 14</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>The connection properties required by the external application. This same value is given to the function testAgentCredentials() and init() in the plug-in class. For more information, see “External adapter plug-in” on page 10</td>
</tr>
</tbody>
</table>

**External data streams**

After implementing your external agent in IBM Cognos Real-time Monitoring, you can configure data streams that your external agent can use.

The types of data streams you can create depend on the transportType elements you specify in the configuration file for your external agent. For more information, see “Configuration file setup” on page 12.

For more information about the data streams, see Chapter 10, “Data streams,” on page 77.

You can create four types of data streams, depending on the transport type that your external agent supports. The types of data streams are:

- Tabular
- Flat file - delimited
- Flat file - fixed width
- XML file

You select the type from the window that displays after selecting the external agent.
**Tabular data streams**

A tabular data stream is similar to an HTTP Post data stream. A tabular data stream has the attributes described in the following table. You can configure a tabular data stream in the Cognos Real-time Monitoring Workbench if the configuration file for your external agent has Tabular specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the data stream table. There is one field for every column in the data stream table, each with the attributes described in the following table:

Each field in the message can be a simple field that maps directly into a column of the data stream table, or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table. Complex fields are treated as flat files in either delimited (CSV) or fixed-width format.

**Delimited flat file data streams**

A data stream for delimited flat files receives data where each field (column) is separated by a character (typically a comma). You can configure a data stream for delimited flat files in the Cognos Real-time Monitoring Workbench if the configuration file for your external agent has Delimited specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the data stream table. There is one field for every column in the data stream table, each with the attributes described in the following table:

**Fixed-width flat files data streams**

A data stream for fixed-width flat files receives data where each field (column) has the same predefined width in each file row, similar to a spreadsheet table. You can configure a data stream for fixed-width flat files in the Cognos Real-time Monitoring Workbench if the configuration file for your external agent has FixedWidth specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the data stream table. There is one field for every column in the data stream table, each with the attributes described in the following table:

**XML data streams**

An XML data stream receives data formatted in XML. You can configure an external XML data stream in the IBM Cognos Real-time Monitoring Workbench if the configuration file for your external agent has XMLFile specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the data stream table. There is one field for every column in the data stream table, each with the attributes described in the following table:
Clear State Interval and Polling

All external data streams have **Clear State Interval** and **Polling** tabs.

For more information about the **Clear State Interval** tab, see "Clear state interval" on page 54.

The **Polling** tab is where you specify how frequently the agent queries the external source for new data.

<table>
<thead>
<tr>
<th>Table 3. Tabular data stream attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>Polling on interval</td>
</tr>
<tr>
<td>Disable data stream after this number of consecutive errors</td>
</tr>
<tr>
<td>Incrementing Field</td>
</tr>
<tr>
<td>Initial Value</td>
</tr>
</tbody>
</table>

External lookup tables

After implementing your external agent in IBM Cognos Real-time Monitoring, you can configure lookup tables that your external agent can use.

The types of lookup table you can create depend on the transportType elements you specify in the configuration file for your external agent. For more information about the configuration file, see "Configuration file setup" on page 12.

For more information about the lookup tables, see Chapter 20, “Lookup tables,” on page 247

You can create four types of lookup tables, depending on the transport type that your external agent supports. The types of lookup tables are:

- Tabular
- Flat file - delimited
- Flat file - fixed width
- XML file
You select the type from the window that displays after selecting the external agent.

**Tabular lookup tables**

A tabular lookup table is similar to file system lookup table. A tabular lookup table has the attributes described in the following table. You can configure a tabular lookup table in the IBM Cognos Real-time Monitoring Workbench if the configuration file for your external agent has Tabular specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the lookup table. There is one field for every column in the lookup table, each with the attributes described in the following table:

Each field in the message can be a simple field that maps directly into a column of the lookup table, or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table. Complex fields are treated as flat files in either delimited (CSV) or fixed-width format.

**Delimited flat file lookup tables**

A lookup table for delimited flat files receives data where each field (column) is separated by a character (typically a comma). You can configure a lookup table for delimited flat files in the IBM Cognos Real-time Monitoring Workbench if the configuration file for your external agent has Delimited specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the lookup table. There is one field for every column in the lookup table, each with the attributes described in the following table:

**Fixed-width flat files lookup tables**

A lookup table for fixed-width flat files receives data where each field (column) has the same predefined width in each file row, similar to a spreadsheet table. You can configure a lookup table for fixed-width flat files in the IBM Cognos Real-time Monitoring Workbench if the configuration file for your external agent has FixedWidth specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the lookup table. There is one field for every column in the lookup table, each with the attributes described in the following table:

**XML data lookup tables**

An XML lookup table receives data formatted in XML. You can configure an external XML lookup table in the IBM Cognos Real-time Monitoring Workbench if the configuration file for your external agent has XMLFile specified for the transport type.

The fields on the **Column Information** tab define how to map the fields in the data source into columns in the lookup table. There is one field for every column in the lookup table, each with the attributes described in the following table:
Data Caching and Invalidation Schedule

Each type of external lookup table has a Data Caching tab and an Invalidation Schedule tab.

If you plan to use a JDBC store for your data cache, you will need a JDBC agent.

The Data Caching tab contains the options for data caching:

- **Cache Type**
  You can use either in-memory or JDBC store for data caching.
  - **In Memory**
    When you enable in-memory caching, the entire external lookup table is cached into memory from the external data source and becomes a replacement for the external query source. This is the same as selecting prefetch caching for lookup table sources from JDBC and Web service agents as described under Prefetch Caching.
    If you select this option, JDBC Store is not available, and the JDBC Agent field is grayed out.
  - **JDBC Store**
    When you select JDBC Store, the lookup table data is pushed out to a local database as specified by the JDBC Agent.
    The JDBC Agent drop-down list shows the available agents. Select the agent to use for the local database.

**Fetch and store on restart**

This option specifies that when you restart IBM Cognos Real-time Monitoring, the system always retrieves data from the external query source, then stores it in the local database. Otherwise, if there is a local store that has been previously populated, that store is used until the next invalidation schedule. If there is no local store that has been previously populated, the external source is queried immediately upon restart.

- **Fallback to external source**
  This option specifies that if the system cannot access the local database, it uses the data from the external database. This option is not available for lookup tables based on a file system agent.

**Number of result sets to cache**

This option specifies the count of result sets to cache in memory. Each set of results can contain one or more rows of lookup table data related to the data stream. For example, if three queries are made and each result set contains five rows, 15 rows are stored in the cache.

The invalidation schedule identifies when to invalidate the cache and discard all information currently in the cache. To create an invalidation schedule, click the Add Schedule button. You can

- clear the cache at a specific time or a time interval
- clear the cache every day, on a specific day or days of the week, or a specific day or days of the month
- clear the cache during a specific month or months

For example, to schedule the clearing of the cache every 3 months on the 30th day of the month at 10:00 PM, you can perform the following actions:

- Select **Add a specific time** and set **Schedule at** to the time 10:00 PM.
- Select **On the day(s) of the months selected below** and check 30.
• For the month, select March, June, September, and December.
Chapter 4. Agents

Agents are processes that retrieve information from external sources when an update to that information occurs.

When an agent detects this new information, it passes that data to data streams and lookup tables for use by views.

External sources

There are agents to handle different external information sources.

Information is either pushed into the system as updates to the information occur, or pulled into the system as a result of a request from a database or from a text file. Information for lookup tables is always pulled from the source.

The following sections summarize the available sources and identify the source agents they support. The table in each section indicates whether the agent pushes or pulls update information to a data stream or lookup table.

JDBC database connectivity

When Java database connectivity (JDBC) uses as an external source, it is usually from a relational database (RDBMS).
Table 4. JDBC push or pull

<table>
<thead>
<tr>
<th>Agent</th>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC (See Chapter 18, “Java Database Connectivity,” on page 211.)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Java Messaging Service from a Java application

When JMS is an external source, it is JMS from a Java application.

Table 5. JMS push or pull

<table>
<thead>
<tr>
<th>Agent</th>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS Queue (See Chapter 17, “Java Messaging Service,” on page 207.)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>JMS Topic (See Chapter 17, “Java Messaging Service,” on page 207.)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Text files

Text files are flat files. Text files can use either a flat file agent or a file system agent.

Table 6. Text files push or pull

<table>
<thead>
<tr>
<th>Agent</th>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat files (See Chapter 13, “Flat files,” on page 93.)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>File system (See Chapter 12, “Cognos Real-time Monitoring file system,” on page 87.)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

TIBCO Rendezvous (RV)

When TIBCO RV is an external source, it is from a business application using TIBCO message streams.

Table 7. TIBCO Rendezvous push or pull

<table>
<thead>
<tr>
<th>Agent</th>
<th>Data Stream Push</th>
<th>Data Stream Pull</th>
<th>Lookup Table Pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCO Rendezvous (See Chapter 33, “TIBCO Rendezvous,” on page 323)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
**Web service**

When a web service is an external source, the source is from a web application over an HTTP connection.

*Table 8. Web service push or pull*

<table>
<thead>
<tr>
<th>Agent</th>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services (See &quot;Web service external processes&quot; on page 337.)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**SAP connectivity**

When the external source is a database for the SAP system, the ERP agent is used to access the SAP system as a source type through an SAP metadata JDBC agent.

*Table 9. SAP push or pull*

<table>
<thead>
<tr>
<th>Agent</th>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERP (See Chapter 30, “SAP connectivity,” on page 309.)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**Salesforce system**

When the external source is a Salesforce system, the source is Salesforce tables.

*Table 10. Salesforce push or pull*

<table>
<thead>
<tr>
<th>Agent</th>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salesforce (See Chapter 29, “Salesforce,” on page 303.)</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Creating agents**

You create agents to retrieve information from external sources when an update to that information occurs.

**Tip:**

Use data source-based JDBC agents instead of URL-based agents. Data source-based (or JNDI) agents allow connection pooling, but URL-based agents do not.

Ensure that the number of connections in the JNDI connection pool is sufficient. The default is typically 5 or 20, which is too low when all lookup tables and data streams share the same agent. Increase the number of connections to between 50 and 100.

Use the **Max Rows Per Query** setting for a JDBC agent to spread the load. If this setting is not set, a query using a JDBC agent fetches the entire set of results at once. If the set of results is a large amount of data, the system does not process...
any data until the full result-set is fetched. If this number is specified, only a specific number of rows are fetched from the database and processed. The remaining rows are fetched in the next fetch cycle. The fetch cycles are determined by polling. Polling tells the object how frequently to query the DBMS for new events. A good technique is to poll frequently, taking 5000 rows per poll. Do not use this technique for lookup table agents because not all rows are loaded.

Use different agents for lookup tables and data streams. Ensure that the lookup table connection pool size is large enough to handle each of the lookup tables that are built on it. If you do not have prefetch on your lookup table, the lookup table must have a dedicated agent to prevent locking out other lookup tables or data streams from accessing the database.

For JBoss implementations, you must configure a corresponding JNDI definition in a celequest_context-ds.xml file. For more information, see the JBoss deployment instructions in the installation guide.

For more information about agent types, see "External sources" on page 21.

If you are using an external agent, the name of your agent is displayed in the list of available agent types.

For information about creating an ERP agent, see "Creating an SAP agent" on page 309. For information about creating a JDBC agent, see Chapter 18, "Java Database Connectivity," on page 211. For information about creating a Salesforce agent, see Chapter 29, “Salesforce,” on page 303.

For more information about source types, see the following topics:
- “Flat file agents” on page 25
- “Creating a file system agent” on page 29
- “Java Database Connectivity agents” on page 221
- “Java Messaging Service queue agents” on page 29
- “JMS topic agents” on page 32
- “TIBCO Rendezvous agents” on page 34
- “Web service agents” on page 35

Before you begin

Before you create an agent, you must have create permissions for agents and the connection specifications for the specific agent type. For more information about permissions, see “Granting permission to create objects” on page 278.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > Agent.
4. Choose the agent type for the agent.
5. Enter the details for the specific source type.
6. Save the object as enabled.

Editing agents

You might need to modify an existing agent.
Before you begin

Before you edit an agent, you must have read and write permissions for the agent. For more information about permissions, see “Access permissions” on page 275.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Select the agent by performing one of the following tasks:
   - In the Tables and Views pane, find the agent that you want to edit and double-click its name.
   - Select the agent from the right pane, click Activities, and then click Edit.
3. If you double-clicked to select the agent, click Edit at the top of the Dependencies screen. Otherwise, proceed to step 5.
4. Enter the details for the specific agent type.
5. Save the object as enabled.

Flat file agents

A flat file agent retrieves data from a text file.

<table>
<thead>
<tr>
<th>Table 11. Flat file push or pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data stream push</td>
</tr>
<tr>
<td>No</td>
</tr>
</tbody>
</table>

File processing

The flat file agent searches for files in a specified location at a defined interval. The name of the files to search for can include asterisk (*) and question mark (?) wildcard characters. When the agent locates a file, it retrieves the event or lookup table data and then either deletes, moves, or renames the source file. When multiple files are located in the named location, the agent processes them in file name order.

Prerequisites

Before creating a flat file agent, you need:
- Create permissions for agents. For more information, see “Granting permission to create objects” on page 278.
- A running File agent program. For more information, see “The file agent program” on page 26.

A file agent has the following attributes:

<table>
<thead>
<tr>
<th>Table 12. File agent attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>Name</td>
</tr>
</tbody>
</table>
## Creating a flat file agent

This section describes how to create a flat file agent.

### Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the **Workbench** tab and click the **Activities** button.
2. Select **Create New**, then **Agent** from the drop-down list.
3. Choose **Flat File** as the agent type.
4. Complete the fields that define the attributes for the agent.
5. Save the agent as enabled. It immediately begins monitoring for events.

## The file agent program

The file agent program is a stand-alone Java program that runs on a host gathering data from a text file. The host might be different from the IBM Cognos Real-time Monitoring server host. When the file agent program finds data, it passes the data to the Cognos Real-time Monitoring servers for processing.

The file agent program does not apply to lookup table data from a text file.

The agent has two XML configuration files:

- **TestAgent.xml** (based on **VCAgent.xsd**) defines the connection information, such as how to locate the Cognos Real-time Monitoring servers and how those servers can locate the file agent.
- **FileAgent.xml** (based on **FileAgent.xsd**) identifies the text file and what to do with the file when finished uploading its data.

To start the agent, run the `cqagent.jar` file in Java and pass the **TestAgent.xml** configuration file as an argument as follows:

```java
java -jar ...\cqagent.jar TestAgent.xml
```

Optionally, you can identify the logging configuration file directory and logging level by including logging properties. The following example sets the logging level to all messages:

```java
java 
  -Dcom.cognos.obi.property.Logging Directory=C:\logs\agents
  -Dcom.cognos.obi.property.Detailed Log File Level=All
  -jar ...\cqagent.jar TestAgent.xml
```

### TestAgent.xml

This section describes the configuration attributes and elements of the **TestAgent.xml** file.
Table 13. TestAgent.xml attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverPort (attribute)</td>
<td>(Optional: default 80) HTTP port on the application server that is running Cognos Real-time Monitoring servers and that the agent uses to communicate to the server. This is the same port that users use to connect to Cognos Real-time Monitoring Workbench.</td>
</tr>
<tr>
<td>pingInterval (attribute)</td>
<td>(Optional: default 20 seconds) How often the agent tests to see whether Cognos Real-time Monitoring server is running. When the server is not running, the agent does not gather events.</td>
</tr>
<tr>
<td>agentName</td>
<td>(Required) Identifies this agent and is the same Name to use when creating the agent in the Administration Console. This name must be unique among agents. See “Object namespace” on page 261 for details.</td>
</tr>
<tr>
<td>serverHost</td>
<td>(Required) Name of the host machine running the Cognos Real-time Monitoring servers. If they are running on the same machine as the File Agent, specify localhost as the name.</td>
</tr>
<tr>
<td>agentImplClass</td>
<td>(Required) Agent implementation class. Do not change this value. Currently com.cognos.obi.agent.FileAgent is required.</td>
</tr>
<tr>
<td>agentImplConfigFile</td>
<td>(Required) Identifies the configuration file for the implementation (the text file component), usually FileAgent.xml. See “FileAgent.xml” on page 28 for more information.</td>
</tr>
<tr>
<td>agentPort</td>
<td>(Required) Port used to communicate to the agent on the host. Used for communication by the server to the agent for disable and enable status changes. Use any valid port number, such as 5050.</td>
</tr>
<tr>
<td>pollingInterval</td>
<td>(Required) How frequently (in seconds) to look for new events.</td>
</tr>
<tr>
<td>loggingDirectory</td>
<td>(Optional: default is configuration file directory). Directory in which to log file information. The log file name is <code>agentName.log</code>.</td>
</tr>
<tr>
<td>serverPath</td>
<td>Specify this attribute to point to the URL of the ffs servlet. This can point to the application server or the web server. If cqagent can access the application web server, change to /realtime/ffs. If using the web server interface, change to /cognos/realtime/ffs.</td>
</tr>
</tbody>
</table>

Example

The following example names the data stream agent as "orderStatusEvent", identifies the implementation configuration file as FileAgent.xml, and sets the server port to 8080:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<VCagent xmlns="http://cognos.obi.com/5"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://cognos.obi.com/5 VCAgent.xsd"
serverPort="80"
>
<agentName>flatFileEvent</agentName>
<serverHost>localhost</serverHost>
<agentImplClass>com.cognos.obi.agent.FileAgent</agentImplClass>
<agentImplConfigFile>FileAgent.xml</agentImplConfigFile>
```
FileAgent.xml

This file configures the text file component (the implementation) of the file agent.

The name and location of the file are identified in the TestAgent.xml file (see "TestAgent.xml" on page 26), and it is usually located in the same directory as that file. This configuration file has four configuration elements, though most configurations use the <filename> and <fileDisposal> elements only.

Elements

The file has the following configuration elements.

Table 14. FileAgent.xml elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name and location of the source text file that contains the events. The file is assumed to be in the same directory as the configuration file unless you identify another location in the filename. You can use relative or complete file path specifications. And the filename can include * and ? wildcard characters. On UNIX systems, use a slash to separate directory path names, such as data_streams/file*.txt. On Microsoft Windows systems, use two back slashes to separate directory path names, such as data_streams\file*.txt.</td>
</tr>
<tr>
<td>type</td>
<td>Identifies the source as a STREAM or FILE. Use FILE when the entire text file must be uploaded atomically (all or nothing), such as for an XML file. Otherwise, use STREAM to upload lines in batches defined by the buffersize element.</td>
</tr>
<tr>
<td>buffersize</td>
<td>(Optional: default is 4,000+EOL) Count of characters to buffer or send in batch to the server. Use this setting to avoid uploading excessively large amounts of text at one time.</td>
</tr>
<tr>
<td>fileDisposal</td>
<td>(Optional) Specifies what to do with the source file after uploading its data. The choices are the default Delete, which deletes the file after upload; Move, which moves the file to a directory specified by the target attribute, and Rename; which renames the file by adding the extension attribute to the file name. Both move and rename overwrite any existing files of the same name in the target location without warning.</td>
</tr>
</tbody>
</table>

Example

This example identifies the source text file as orderStatusData.txt in the data_stream\ subdirectory on a Windows host and moves the finished file into the ..\done\ sibling directory:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<FileAgent xmlns="http://cognos.obi.com/5"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:schemaLocation="http://cognos.obi.com/5 FileAgent.xsd">
   <fileName>data_stream\orderStatusData.txt</fileName>
   <fileDisposal/>
</FileAgent>
```
File system agents

A file system agent retrieves lookup table data from a text file by searching for files in a specified location at a defined interval. When the agent locates a file, it retrieves the lookup table data from the source file.

<table>
<thead>
<tr>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

A file system agent has the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Yes</td>
<td>Identifies the agent. This name must be unique among agents.</td>
</tr>
<tr>
<td>Status</td>
<td>Yes</td>
<td>Whether the agent is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Save in</td>
<td>Yes</td>
<td>The folder in which to store the agent. The default folder is Public Folders. Click the Choose Folder button to select a folder.</td>
</tr>
<tr>
<td>Description</td>
<td>No</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Root File Path</td>
<td>Yes</td>
<td>Specifies the file path to the text file to be used by the file system agent.</td>
</tr>
</tbody>
</table>

Creating a file system agent

You can create a file system agent to retrieve lookup table data from a text file.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > Agent.
4. Choose File System as the agent type.
5. Complete the required fields that define the attributes for the agent.
6. Save the agent as enabled.

Java Messaging Service queue agents

A Java Messaging Service (JMS) queue agent communicates with a JMS message producer through a JMS queue running in the application server environment.

The agent tells the producer which JMS messages to send. The producer then sends those messages to the data stream through the agent.
JMS queue agents are asynchronous, that is, they receive event messages as the events occur. You cannot retrieve lookup table data from a JMS queue agent.

Table 17. JMS queue agents push or pull

<table>
<thead>
<tr>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

A JMS queue agent has the following attributes:

Table 18. JMS queue agents attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Yes</td>
<td>Identifies the agent. This name must be unique among agents. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Yes</td>
<td>Whether the agent is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Save in</td>
<td>Yes</td>
<td>The folder in which to store the agent. The default folder is Public Folders. Click the Choose Folder button to select a folder.</td>
</tr>
<tr>
<td>Description</td>
<td>No</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Queue Connection Factory</td>
<td>Yes</td>
<td>Identifies the J2EE connection factory that maintains the queue that you want. This string identifies the factory by its JNDI lookup name. For example, jms.ManufacturingQueue. Enter CNowBusQueueConnectionFactoryJNDI in this field. In an Oracle WebLogic environment, the factory JNDI name is identified on the Oracle WebLogic Console, Services &gt; JMS &gt; Connection Factories &gt; factoryName &gt; General tab. In JBoss, the default is ConnectionFactory.</td>
</tr>
<tr>
<td>Acknowledge mode</td>
<td>No</td>
<td>Protocol to use when acknowledging receipt of the message. AUTO - provider acknowledges message when it is delivered (Default). CLIENT - acknowledges the message when the agent receives it. DUPS OK - tells the publisher that it is OK to send a message more than once. The subsequent receipts of the same message are treated as new and unique events.</td>
</tr>
<tr>
<td>User name</td>
<td>No</td>
<td>User name to use to connect to the JMS factory.</td>
</tr>
<tr>
<td>Password</td>
<td>No</td>
<td>Password for the user name.</td>
</tr>
</tbody>
</table>
Table 18. JMS queue agents attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI properties</td>
<td>No</td>
<td>(Optional) Java Naming and Directory Interface (JNDI) properties necessary to create or maintain the agent in the JMS table. These name-value pairs allow you to specify JMS properties recognized by the JNDI. When the JMS Queue is running in a different namespace than the Cognos Real-time Monitoring servers, define the properties described in &quot;JNDI properties for connecting to a remote namespace.&quot;</td>
</tr>
</tbody>
</table>

JNDI properties for connecting to a remote namespace

When the JMS queue is running in a different namespace than the IBM Cognos Real-time Monitoring servers, you need to define JNDI properties to make the connection.

The names of these JNDI properties are:

- java.naming.factory.initial
- java.naming.provider.url

Further, if you are using security, also define these properties:

- java.naming.security.authentication
- java.naming.security.principal
- java.naming.security.credentials

Example for Oracle WebLogic JNDI

- java.naming.factory.initial=weblogic.jndi.WLInitiallookup
- java.naming.provider.url=t3://localhost:9180

Example for IBM Websphere

- java.naming.factory.initial=com.ibm.websphere.naming.WsnInitiallookup
- java.naming.provider.url=iiop://localhost:9180

Example for JBOSS

- java.naming.factory.initial=org.jnp.interfaces.Naminglookup
- java.naming.provider.url=jnp://localhost:1099

Example for Sun One Directory Server LDAP

- java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
- java.naming.provider.url=ldap://ldap.mycompany.com:59226/dc=mycompany,dc=com

Also, prefix the Queue Connection Factory parameter value with: cn=
Creating a JMS queue agent

You can create a JMS queue agent to communicate with a JMS message producer through a JMS queue running in the application server environment.

Before you begin

Before creating a JMS queue agent, you need the following items:

- A custom message driven bean (MDB), which has the data stream name hard coded in the properties file and that receives messages from the queue, must be deployed in the host application server. Application servers do not allow dynamic subscriptions to JMS queues. A custom MDB must be created for this purpose. For assistance creating and configuring this MDB, refer to the README.txt files under the samples folder in your <c10_installdir>.
- Create permission for agents. For more information, see “Granting permission to create objects” on page 278.
- The JNDI location of the topic factory in the application server that is receiving the queue messages.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > Agent.
4. Choose JMS Queue as the source type.
5. Complete the required fields that define the needed attributes.
6. Save the agent as enabled.

JMS topic agents

A Java Messaging Service (JMS) topic agent communicates with a JMS message producer through a JMS topic running in the application server environment.

The agent tells the publisher which messages to send. The publisher then sends those messages to the data stream through the agent.

JMS topic agents are asynchronous, and they receive event messages as the events occur. As shown in the following table, you cannot retrieve lookup table data from a JMS topic agent.

<table>
<thead>
<tr>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

A JMS agent has the attributes described in the following table:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Yes</td>
<td>Identifies the agent. This name must be unique among agents. For more information about object names, see “Object namespace” on page 261.</td>
</tr>
</tbody>
</table>
### Table 20. JMS topic agent attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Yes</td>
<td>Whether the agent is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Save in</td>
<td>Yes</td>
<td>The folder in which to store the agent. The default folder is Public Folders. Click the <strong>Choose Folder</strong> button to select a folder.</td>
</tr>
<tr>
<td>Description</td>
<td>No</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Topic Connection Factory</td>
<td>Yes</td>
<td>Identifies the J2EE connection factory that maintains the topics that you want. This string identifies the factory by its JNDI lookup name; for example, jms.ManufacturingTopic. jms.ManufacturingQueue. Enter CNowBusTopicConnectionFactoryJNDI in this field. In an Oracle WebLogic environment, the factory JNDI name is identified on the Oracle WebLogic Console, Services &gt; JMS &gt; Connection Factories &gt; factoryName &gt; General tab. In JBoss, the default is ConnectionFactory.</td>
</tr>
<tr>
<td>Acknowledge mode</td>
<td>No</td>
<td>Protocol to use when acknowledging receipt of the message. The default is <strong>AUTO</strong>, where the provider acknowledges the message when it is delivered; <strong>CLIENT</strong>, which acknowledges the message when the agent receives it; and <strong>DUPS OK</strong>, which tells the publisher that it is OK to send a message more than once. The subsequent receipts of the same message are treated as new and unique events.</td>
</tr>
<tr>
<td>User name</td>
<td>No</td>
<td>User name to use to connect to the JMS factory.</td>
</tr>
<tr>
<td>Password</td>
<td>No</td>
<td>Password for the user name.</td>
</tr>
<tr>
<td>JNDI properties</td>
<td>No</td>
<td>Optional Java naming and directory interface (JNDI) properties necessary to make or maintain the agent to the JMS table. These name-value pairs allow you to specify JMS properties recognized by the JNDI. When the JMS topic is running in a different namespace from the Cognos Real-time Monitoring servers, define the properties described in JNDI properties for connecting to a remote namespace on page 31.</td>
</tr>
</tbody>
</table>

### Creating a JMS topic agent

You can create a JMS topic agent to communicate with a JMS message producer through a JMS topic running in the application server environment.

### Before you begin

Before creating an agent, you need the following items:

- A custom message driven bean (MDB), which has the data stream name hard coded and that subscribes to the topic, must be deployed to the host application server.

Application servers do not allow dynamic subscriptions to JMS topics. A custom MDB must be created for this purpose.
Create permission for agents. For more information, see "Granting permission to create objects" on page 278.

The JNDI location of the topic factory in the application server that is publishing the topics.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > Agent.
4. Choose JMS Topic as the source type.
5. Complete the required fields that define the attributes for the agent.
6. Save the agent as enabled.

TIBCO Rendezvous agents

A TIBCO Rendezvous agent communicates with a TIBCO Rendezvous daemon running in the application server environment. The daemon listens for messages on a TIBCO Rendezvous message stream.

When the daemon finds a message requested by one of the TIBCO Rendezvous tables (see "TIBCO Rendezvous tables" on page 323), it retrieves the message data and passes it to the table through the agent.

TIBCO Rendezvous agents are asynchronous, and they receive event messages as the events occur, as shown in the following table. You cannot retrieve lookup table data from a TIBCO Rendezvous agent.

<table>
<thead>
<tr>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Attributes

This section provides details on the attributes of a TIBCO Rendezvous agent.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Yes</td>
<td>Identifies the agent. This name must be unique among agents. For more information, see &quot;Object namespace&quot; on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Yes</td>
<td>Whether the agent is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Save in</td>
<td>Yes</td>
<td>The folder in which to store the agent. The default folder is Public Folders. Click Choose Folder to select a folder.</td>
</tr>
<tr>
<td>Description</td>
<td>No</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Service</td>
<td>No</td>
<td>TIBCO Rendezvous service port. Leave this blank to use the default port 7500. Change this value only if your TIBCO Rendezvous administrator gives you another port.</td>
</tr>
</tbody>
</table>
Table 22. TIBCO Rendezvous agent attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>Yes</td>
<td>Identifies the network interface to use when the host is connected to more than one network, or when the host supports multicasting (in which case the address looks similar to .222.1.2.3). Change this value only when the host machine is not on the default network, then use the IP address provided by your TIBCO Rendezvous administrator.</td>
</tr>
<tr>
<td>Daemon</td>
<td>No</td>
<td>Port of the routing daemon on the TIBCO Rendezvous host found on the network and identified by the Network attribute. Leave this blank to use the default port 7500. Change this value only if your TIBCO Rendezvous administrator gives you another port.</td>
</tr>
</tbody>
</table>

Creating a TIBCO rendezvous agent

This section describes how to create an agent to communicate with a TIBCO Rendezvous daemon running in the application server environment.

Before you begin

Before creating an agent, you must:

- Assign create permissions for agents. For more information, see “Granting permission to create objects” on page 278.
- Connect to the TIBCO Rendezvous listener daemon.

To connect, you need the service, network, and daemon names. For specific values, consult the IT specialist who maintains your TIBCO Rendezvous system.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > Agent.
4. Choose TIBCO Rendezvous as the source type
5. Complete the fields that define the attributes for the agent.
6. Save the agent.

Web service agents

A web service agent communicates with an application running on a web application server to retrieve lookup table data. The agent connects to the application through an HTTP connection using Simple Object Access Protocol (SOAP).

Web service agents are synchronous. They retrieve lookup table data as the result of a specific request.

Table 23. Web service agent push or pull

<table>
<thead>
<tr>
<th>Data stream push</th>
<th>Data stream pull</th>
<th>Lookup table pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Chapter 4. Agents  35
Web service agent attributes

This section provides details on the attributes of a web service agent.

Table 24. Web service agent attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Yes</td>
<td>Identifies the agent. This name must be unique among agents. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Yes</td>
<td>Whether or not the agent is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Save in</td>
<td>Yes</td>
<td>The folder in which to store the agent. The default folder is Public Folders. Click the Choose Folder button to select a folder.</td>
</tr>
<tr>
<td>Description</td>
<td>No</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>URL</td>
<td>Yes</td>
<td>HTTP location of the web service definition language file (WSDL) that describes the service, the data it provides, and how to exchange data with the service. The returned data must be a SOAP doc-style message; RPC binding is not supported.</td>
</tr>
<tr>
<td>User name</td>
<td>No</td>
<td>(Optional) User name to use when connecting to the service. This parameter is passed to the server when the server requires a user name.</td>
</tr>
<tr>
<td>Password</td>
<td>No</td>
<td>(Optional) User password to use when connecting to the service. This parameter is passed to the server when the server requires a password.</td>
</tr>
</tbody>
</table>

Creating a web service agent

You can create a web service agent to communicate with an application running on a web application server retrieve lookup table data.

Before you begin

Before creating a web service agent, you must:

- Assign create permissions for agents. See “Granting permission to create objects” on page 278 for details.
- Know the HTTP location of the web service definition language file (WSDL) that defines the service to use. The service must use SOAP binding to publish data. RPC binding is not supported.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > > Agent.
4. Choose Web Service as the agent type.
5. Complete the fields that define the attributes for the agent.
6. Save the agent as enabled.
Chapter 5. Alerts

Alerts are notifications of exceptional incidents sent to users or external systems.

An alert can be a simple message indicating that an incident occurred. An alert can also be more detailed, including information that indicates the cause of the incident and possible solutions.

An alert can be stand-alone or associated with a rule. It can also be a clone of an existing alert.

Each alert message contains text that describes the exceptional incident to the subscriber. This text appears on the Workbench tab and in any other device identified by the delivery profile of the subscriber. How the message is rendered depends on the device that displays it to the subscriber.

Creating a stand-alone alert

In IBM Cognos Real-time Monitoring Workbench, you can create a stand-alone alert that is not associated with a rule.

Before you begin

To create an alert, you need the following permissions:

- Create permissions for business activities. See “Granting permission to create objects” on page 278 for details.
- Read and write permissions for the business activity that contains the alert.
- Read-only permissions for the view or cube that provides data to the alert.

Procedure

1. In Scenario Modeler, select an existing business activity.
2. In the System Objects pane, select an existing scenario to contain the alert.
3. Select the Alerts tab.
4. Click New Alert.
5. Choose the data source that provides data to the alert.
   - If the scenario has a default view, that scenario appears as selected. Choose another source to monitor by clicking Select Data Source.
   - For a view, choose the view.
   - For a cube, choose the dimension level in a cube. Optionally, you can also apply a filter that further restricts the data that the cube provides to the alert.
   - If the source contains data, that data appears to provide a sample of what to expect. When the source is empty, the form displays just the column names and the message No Data Available.
6. Complete the fields on the Create Alert window:
   - Alert Name
   - Description (Optional)
   - Severity
   - Subscribers
Use the **Add/Remove** button to select the users and roles as the subscribers for this event. They can be either **Mandatory Recipients** or **Optional Recipients**.

- **Subject**
  Use the **Add Field** button to insert fields into the subject line for this alert. These are fields that insert actual values in the alert.

- **Body** (Optional)
  Use this field to create an alert message for the subscribers to this alert. You can use **Add Field**, **Add Reportlet**, and **Add Acknowledgement** buttons to construct a message with relevant data.

### Creating an alert associated with a rule

You can create an alert that is associated with a rule, as opposed to a stand-alone alert.

#### Before you begin

To create an alert, you need the following permissions:

- **Create permissions** for business activities. See “[Granting permission to create objects](#)” on page 278 for details.
- **Read and write permissions** for the business activity that contain the alert.
- **Read-only permissions** for the view or cube that provides data to the alert.

#### Procedure

1. Follow the instructions for “Creating rules” on page 297.
2. On the **Basic** tab, click **Choose**.
3. In the Select Alert window, select an existing alert or create a new alert by clicking **Create Alert**.

### Cloning an existing alert

You can clone an existing alert to save time and effort.

#### Procedure

1. Copy the definition of an existing alert to a new alert.
2. Edit the alert you want to clone.
3. Change the alert name and change other attributes that differ from the original alert.
4. Choose **Save as New Alert**.

### Alert attributes

You can modify the notification settings that control the maximum number of alerts that can be sent within a specified time interval.

For more information, see the topic on setting alert notifications limitations in the *IBM Cognos Real-time Monitoring Workbench User Guide*.

Every alert has the attributes that are in the following list:

- **Alert Name**
  Identifies the alert object. The name can contain letters and numerals only.
This name must be unique among alerts within the same scenario. See “Object namespace” on page 261 for details.

Status  Specifies whether the rule is enabled (receiving new data stream information) or disabled. When the containing scenario is disabled, you cannot make the alert enabled. The scenario must be enabled before you can enable the alert.

Severity  A hint about how important a message is. Values are HIGH, NORMAL (default), or LOW. Messages arriving in Real-time Monitoring Dashboard are sorted into folders corresponding to the importance level. Further, email messages are flagged accordingly with the ‘Importance’ mail header field per mail standards.

Description  Optional description that can contain any text characters.

Data source  View or cube dimension level that defines the columns in the alert. This should be the same source as the associated rule, or one derived from that source; otherwise, the generated alert might not contain valid information.

Subscribers  Users who receive the alert. See “Alert subscribers” on page 40 for details.

Subject  A text message that is the subject of the alert, similar to an email subject line. Can contain column references to the underlying business view.

Body  A text message that is the body of the alert. Can contain column references to the underlying business view, and can contain acknowledgments and reportlets. For details about acknowledgments, see the IBM Cognos Real-time Monitoring Workbench User Guide. For details about reportlets, see Chapter 26, “Reportlets,” on page 291.

**Message subject and body text**

Each alert message uses text to describe the exceptional incident to the subscriber. This is the text that appears in IBM Cognos Real-time Monitoring Workbench, IBM Cognos Real-time Monitoring Dashboard, and in any other device identified by the subscriber’s delivery profile. How the message is rendered depends on the device that displays it to the subscriber.

The subject and body alert attributes define the text of the message. Each attribute contains static text and fields. When the alert generates the message, it replaces the fields with the values from the columns of the same name in the business view row that caused the alert.

For example, consider this default message definition:

Subject: NOTICE -- A customer has opened a problem ticket. Body: CUST_NAME is a HIGH tier customer and has opened problem ticket number TICKET.

When the alert is activated, it generates a message similar to the following message:
Alert subscribers

When you define an alert, you can also declare one or more users to receive the alert notification.

By clicking Add/Remove next to the subscribers list in the Alert editor, you can designate individual users and roles to receive the alert. You can also identify columns in the view of the alert that provide lists of users, roles, or email addresses to receive the notification.

The Alert Subscribers window has two tabs; one for individual subscriptions and one for data-driven subscriptions.

The Individual Subscription tab is where you choose the users to receive the notification.
- Mandatory subscribers always receive the notification. These users cannot voluntarily unsubscribe to the alert in IBM Cognos Real-time Monitoring Dashboard. They must be removed from the Individual Subscription tab.
- Optional subscribers receive notifications, but they can unsubscribe using the Cognos Real-time Monitoring Dashboard.

The Data-Driven Subscription tab is where you identify columns in the view that contain the names or addresses of users or roles to receive the notification. A column in the view can contain either a list of users and roles, or a list of email addresses to receive the notifications. Each list of values in a column is separated by a comma or semicolon, and each item can optionally be enclosed in quotes (").
Select the column to use from the drop-down lists Data-driven Recipient (User or Role) and Data-driven Recipients (Email Addresses).

Data-driven subscriptions are mandatory: the users, roles, and email addresses that receive them cannot unsubscribe. Users who receive these subscriptions as a result of an email address list do not see them in the Real-time Monitoring Dashboard list of subscriptions. When the column contains multiple instances of the same email address, only one message is sent. However, slight differences in the entries will generate one message for each instance. For example, these two variations of the same address myname@obi.com and "Name" <myname@obi.com> generate two messages.
Alert notification message management

Alerts remain in the Alert Manager list until they are deleted manually by the user or automatically by the system.

The system removes messages after a number of days specified by the system administrator in the Systems Settings window, as described in the sections about system settings in the *IBM Cognos Real-time Monitoring Workbench User Guide*.

Also, you can control the maximum number of alerts that can be sent within a specified time interval. For more information, see the topic about setting alert notification limitations in the *IBM Cognos Real-time Monitoring Workbench User Guide*.

Users receive alert notifications on the devices identified by their delivery profiles in the delivery profiles tab (see “Delivery profiles tab” on page 330).

For details about using the Alert Manager in IBM Cognos Real-time Monitoring Dashboard to view, subscribe, and delete notifications, see Interacting with Alerts in the dashboard documentation.

Alert states

IBM Cognos Real-time Monitoring provides two kinds of alerts, stateless and stateful.

Stateless alerts are one-time notifications about the business condition when the condition happens.

Stateless alerts are sent when the rule condition is met. For example, a stateless alert might send a notification to a warehouse manager when a product inventory count falls below a specific threshold. Note that every subsequent change in inventory levels for that product also sends a notification as long as the inventory count remains below the threshold.

Stateful alerts have a status that is raised or acknowledged as long as the business condition exists, and is lowered when the condition does not exist.

With a stateful alert, warehouse managers receive the alert when the inventory falls below threshold, and do not receive another until the alert is lowered, presumably after inventory levels have been restored above the threshold. When multiple parties have interest in an alert, one can choose to handle the raised alert and acknowledge it. This is done by clicking the Acknowledge link in the message body. The creator of the alert places the link in the message body when creating the alert. For more information, see the topic about acknowledgements in the *IBM Cognos Real-time Monitoring Workbench User Guide*. 
By combining a stateful alert with a Holds For time period, you can delay the notification. For example, alert the warehouse manager only when the inventory count for a product has remained below a threshold for one day: the alert condition holds for 1 day. In this case, the manager does not receive the notification if the inventory drops within a day of being restocked.

Alert escalation

You can monitor the state of an alert and generate new alerts when conditions demand.

For example, if an alert has not been handled in a timely manner, a new alert can be sent to users at a higher level in the organization, in effect escalating the original alert. To test for these states, use the IS_RAISED (see “IS_RAISED” on page 153) function. See “Rules that monitor alerts” on page 301 for details.

Consolidating multiple messages

Events may contain multiple rows of information. When the event meets a rule condition, that rule generates one alert for each row of the event.

Often it is desirable to send only one message describing all of the alerts. This is called a consolidated alert.

For example, consider a new purchase order entering the system (an event). If the quantity of items in-stock is insufficient to fulfill the order, an alert might note that condition. When multiple line items on the order have insufficient inventory, each generates a new alert. To send just one notification instead, use a consolidated alert.

Procedure

In the Create Rule window, select the Consolidate multiple messages from same event option.

Setting an alert to invoke an external web service

To configure an alert message to invoke an external web service, you must create a packaged Java function that constructs the web service message based on the alert payload.

This function determines which data elements in the alert payload are mapped to the inputs expected by the web service.
It is uploaded to IBM Cognos Real-time Monitoring as a user-defined function (UDF). The web service is specified in a web service profile that calls the UDF. The web service profile can then be set as a subscriber to the alert that you want.

Creating this function requires programming expertise. For more information, see “The web service user-defined function.”

Procedure
1. Create a new alert.
   Do not subscribe users to the new alert. You will add subscribers later.
2. Create a function that constructs the web service message using parts of the alert data, and package as a JAR file.
   For more information, see “The web service user-defined function.”
3. Upload the JAR file as a user-defined function in the Workbench.
   For more information, see “Creating and using a user-designed function” on page 334.
4. Create a new user and save it.
   You must create and save the new user so that you can change the default dashboard profile properties.
5. Edit the new user as follows:
   • In the Edit User window, click the Delivery Profiles tab.
   • Open the default dashboard profile.
   • Deselect the Automatically add this profile option.
   • Click OK.
6. For the new user, create a delivery profile as follows:
   • In the Edit User window, select the Delivery Profile tab.
   • Click Create New Profile.
   • In the resulting window, for Profile Type, select Web Service.
   • For Profile Name, enter an appropriate value.
   • For Web Service URL, enter the endpoint for the WSDL.
   • For Method, specify the method (operation) that you want to be invoked in the web service.
   • Supply Username and Password, as required.
   • For UDF, select the JAR file you uploaded for this web service invocation.
   • Ensure that the Automatically add this profile option is not selected.
   • Click OK.
7. Return to the configuration for the alert you created in step 1.
8. Add the user you just created as a subscriber.
9. Click Save.
   When the alert fires, the web service will be invoked.

The web service user-defined function
This function receives as input a DOM element that describes the alert data generated by IBM Cognos Real-time Monitoring and that conforms to the alertMessage.xsd. It generates as output an AXIS message containing the SOAP that is sent to the web service.
The function you create must extend the class
com.cognos.obi.api.function.webservice.IAlertWSMessageConstructor.

com.cognos.obi.api.function.webservice.IAlertWSMessageConstructor interface:
package com.cognos.obi.api.function.webservice;
import com.cognos.obi.api.function.*;
import org.apache.axis.Message;
import org.w3c.dom.Element;
import java.lang.Exception;
public interface IAlertWSMessageConstructor extends IUDFunction
{
    public Message constructMessage(Element alertData, String
        wsdlURI,
        String soapOperation) throws Exception;
}

Adding reportlets to an alert notification
Reportlets describe the contents of a view and present that information in a report
that is attached to the alert message.

Reportlets often provide information about an incident that puts the incident into
context.

Alerts displayed in the Operational Dashboard embed the reportlet as an in-line
HTML table. The Send as option specifies the format of the reportlet to attach to
the alert notification sent to user subscription profiles. The reportlet can be
embedded in the body of the message, or included as an attachment in one of the
available formats.

Procedure
1. Open the Add Reportlet window.
2. Select the reportlet from the list, and click OK to add it to the alert.

Reportlet filtering
When you create the reportlet, you identify the view or cube from which the
reportlet draws data. However, the data that appears in the reportlet depends on
the type of the source, stateless or stateful, and how the rows of data are filtered.

How the rows of data are filtered is specified on the Add Reportlet box. In
general, the following options affect how data is filtered:
- The Reportlet data based on option specifies whether a reportlet's stateful
  source should include data based on all events, or just those that have met the
  rule condition.
- The Reportlet data is option limits whether the rows in the reportlet's source
  include only those related to the event that activated the alert, or all previous
  event data also in the source.

To better understand how these settings affect the reportlet data, consider these
two views that track and report on product orders. The OrderDetails stateless view
is a summary of each order event, while the OrderAggregates stateful view tracks
the average quantity for each product ordered.

Now consider these events:

<table>
<thead>
<tr>
<th>prod_name</th>
<th>ord_qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>nails</td>
<td>1000</td>
</tr>
<tr>
<td>plywood</td>
<td>1000</td>
</tr>
<tr>
<td>nails</td>
<td>4000</td>
</tr>
<tr>
<td>nails</td>
<td>4000</td>
</tr>
<tr>
<td>plywood</td>
<td>5000</td>
</tr>
</tbody>
</table>

After the event has entered the data stream, the OrderAggregates view has these values:

<table>
<thead>
<tr>
<th>prod_name</th>
<th>AVG(ord_qty)</th>
<th>Ct</th>
</tr>
</thead>
<tbody>
<tr>
<td>nails</td>
<td>3000</td>
<td>3</td>
</tr>
<tr>
<td>plywood</td>
<td>3000</td>
<td>2</td>
</tr>
</tbody>
</table>

By default, a reportlet using the OrderAggregates view shows the details for both products, regardless of which product event might have generated the alert. Even though only the last three events met the rule condition of ord_qty greater than 3,000, the reportlet shows the results from all events, which might not be what you intended.

The two filtering options on the Add Reportlet box alter the results by filtering the results that appear in the reportlet.

The Reportlet data is option causes the reportlet to show the following data:

- All of the data in its view or cube face
- Only those data related to the event found by the rule

To show only the event with the related data, you must define the relationship between the event and reportlet sources. For example, if you want the reportlet to show only the result for "products" in both views, define the relation by picking the prod_name column from both views. This tells the reportlet to show only those rows in the OrderAggregates view whose prod_name value matches the name in the OrderDetails view. Then, the reportlet shows the nails value only when the rule generates the alert.

```
prod_name  AVG(ord_qty)  Ct
-----------  -------------  ---
```
When working with a cube face, you pick columns that best identify the event to the reportlet.

**Reportlet data based on option**

This option specifies whether a stateful view should include data that is based on all events, or just those that have met the rule condition.

Following the example in [Reportlet filtering](#) on page 44, the OrderAggregates view AVG(ord_qty) column has a value of 3,000 for “nails” after both events are processed. This is what happens when the setting for this option is **Data Stream data**. However, the rule condition says to generate an alert only when the order quantity is greater than 3,000. To track only events that have met the rule condition, change the setting for this option to **Rule Filter of Data Stream Data**. Then the reportlet shows 4,000 as the average because 4,000 is the average of the two events greater than 3,000.

The following illustrations show the view results on the example data when you use the two reportlet filtering options. Notice that the first event does not pass the rule filter, and does not appear in those views.

![Example data](image)

**Figure 4. View results on the example data**

Similarly, the second event also does not pass the rule filter. Notice that the view that shows event-related data now only includes the plywood event.
The third event now passes the rule filter, and appears in the bottom views. Once again, "nails" is the product in the event related to the views.

The fourth "nails" event also passes the rule filter.

The final event again changes the event related to the views. Note that all events are reflected in one view, while only those that passed the rule filter are in the rule filter view.
<table>
<thead>
<tr>
<th>Events</th>
<th>Data Stream Data</th>
<th>Rule Filter of Data Stream Data (ord_qty&gt;3000)</th>
<th>Only data related to the event (prod_name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod_name</td>
<td>ord_qty</td>
<td>Ct</td>
<td>prod_name</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>nails</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>plywood</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nails</td>
<td>4000</td>
<td></td>
<td>nails</td>
</tr>
<tr>
<td>plywood</td>
<td>4000</td>
<td></td>
<td>plywood</td>
</tr>
<tr>
<td>plywood</td>
<td>5000</td>
<td></td>
<td>plywood</td>
</tr>
</tbody>
</table>
Chapter 6. Business rules

This chapter describes business rules for monitoring data streams and producing messages that describe alert conditions.

About business rules

Business rules monitor data streams looking for exceptional business conditions and produce alert messages that describe the conditions when they exist. Further, rules can monitor a found condition and identify when it no longer exists.

In IBM Cognos Real-time Monitoring Dashboard, you can create business rules from templates that predefine conditional logic. You pick the rule condition and identify the values to test. The system notifies you whenever the condition is met. For example, with the rule "When order total is greater than amount" you identify the value of amount, then receive notifications every time an order is larger than the amount you chose.

This section covers creating business rules from templates and managing the rules in Cognos Real-time Monitoring Dashboard. Rule templates allow an advanced user to create a complex rule condition and define parameters for certain values so that Cognos Real-time Monitoring Dashboard users can create customized instances of the template. For example, a user wants to instantiate a business rule where the sales amount is greater than $20,000, whereas another user wants to be notified for any sales order greater than $5,000. Business rule templates are created in Cognos Real-time Monitoring Workbench. For more information, see the IBM Cognos Real-time Monitoring Workbench User Guide.

Attention: IBM Cognos Real-time Monitoring supports another kind of business rule: rules defined and managed entirely by the Scenario Modeler in IBM Cognos Real-time Monitoring Workbench. For more information, see the technical documentation for the workbench.

The Business Rule Manager lists the rules available to you and is where you create, manage, and delete them.

Deleting a business rule does not affect alert messages in the Alert Manager. Any messages in the Alert Manager generated from a business rule remain in the Alert Manager until removed.

Creating a business rule

You create business rules in the Business Rule Manager.

You need at least Read permission to the business activity in the Scenario Modeler where the rule template is defined.

For more information, see the topic about permissions in the IBM Cognos Real-time Monitoring Workbench Modeling Reference and the topic about working with rule templates in the IBM Cognos Real-time Monitoring Workbench User Guide.
**Procedure**

1. Open the **Business Rule Manager** and click **Create Business Rule**.
2. Select the rule template to use.
3. Identify the parameter values for the rule.
4. Name the rule, and optionally provide a description.
5. Save the new rule.

   The new rule appears in the **Business Rule Manager**, and the new rule immediately starts looking for new exceptional conditions.

   **Attention:** Business rules are based on the definition of the rule template at the time of creation. Any subsequent changes to the template (including its alert message or reportlet) do not affect the existing business rules created from that template.
Chapter 7. Common Event Infrastructure

The Common Event Infrastructure (CEI) is a service that can create, store, and distribute events.

IBM WebSphere uses the Common Event Infrastructure. The CEI receives system or business information encapsulated in a Common Business Event (CBE). When a significant event occurs, a CBE is created and sent to the CEI server by using an emitter where it can be persisted or distributed or both.

IBM Cognos Real-time Monitoring provides the functionality to communicate with a server that is using the CEI.

How Common Event Infrastructure works

IBM Cognos Real-time Monitoring communicates with a servlet and a message-driven bean (MDB).

The MDB is deployed on the server that is running Common Event Infrastructure (CEI). The MDB subscribes to the JMS topic or queue that is referenced by the JMS transmission process. The JMS transmission process is in turn referenced by the CEI emitter. All messages are serialized and sent to Cognos Real-time Monitoring through HTTP to an HTTP Post servlet. The HTTP request from the MDB includes parameters that direct the message to the CEI data stream in Cognos Real-time Monitoring.

The diagram illustrates how a CEI message is delivered to Cognos Real-time Monitoring.

Common Event Infrastructure data stream

Common Business Events (CBEs) are XML documents, and the Common Event Infrastructure (CEI) data stream treats them as such.

You can retrieve rows and columns by using XPath expressions in the same way that you retrieve XML flat files. CEI data stream processes a CBE in the same manner as a flat file. Therefore, configure your events in the same way that you configure events for a flat file.
Prerequisites for creating a Common Event Infrastructure data stream

Before creating a Common Event Infrastructure (CEI) data stream, you must complete several tasks.

Do the following tasks:
- Create a CEI message driven bean (MDB).
- Deploy the CEI MDB.
- Find a sample flat file (optional).
- Configure CEI to connect to the MDB.

You must have IBM WebSphere Process Server installed, configured, and running for CEI.

A CEI MDB is available in the directory `webcontent/samples/sdk/cei`.

For the successful creation of the `cei_mdb.ear` file when performing the test execution for CEI, the following JAR files are needed:
- `emf.jar`
- `com.ibm.events.client_6.1.0.jar`

For the successful creation of the `eventsimulator.war` file when performing the test execution for CEI, the following JAR files are needed:
- `ws_runtime.jar`
- `com.ibm.events.client_6.1.0.jar`

For the file `ws_runtime.jar`, in the IBM WebSphere installation location go to `install location/usr/IBM/WebSphere/AppServer/deploytool/itp/plugins/com.ibm.websphere.v61_6.1.200`.

For the file `com.ibm.events.client_6.1.0.jar`, in the IBM WebSphere installation location go to `install location/usr/IBM/WebSphere/AppServer/plugins`.

The following table describes the attributes for a CEI data stream.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the table. The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see &quot;Object namespace&quot; on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which to save the agent. The default is <strong>Public Folders</strong>. Click the <strong>Choose Folder</strong> button to select a folder.</td>
</tr>
</tbody>
</table>
Table 25. CEI data stream attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Column Information</td>
<td>The Column Information fields define how to map the fields from the CEI message into columns in the data stream table. There is one column for every field in the data stream table. For more information, see “Common Event Infrastructure column information” on page 54.</td>
</tr>
<tr>
<td>Clear State Interval</td>
<td>This tab contains several options for clearing persisted event data. For more information, see “Clear state interval” on page 54.</td>
</tr>
</tbody>
</table>

Deploying a message driven bean
You can deploy a message driven bean.

Procedure
1. Login to the Process Server Integrated Solutions console.
2. Go to Applications, Enterprise, Install.
3. Browse for the cei_mdb.ear file in the deploy folder and upload it.

Configuring Common Event Infrastructure to connect to the message driven bean
You can configure Common Event Infrastructure to connect to the MDB.

Procedure
1. Make sure that the configuration of your CEI application enables authorization for emitters.
2. Go to the administrative console for IBM WebSphere.
3. Make sure the topic or queue that you defined in your MDB exists. The topic or queue must match QueueConnectionFactory.
4. Make sure that you have an Activation Specification that associates the topic or queue to the MDB. When you deploy the MDB, you use the Activation Specification to subscribe to the topic or queue.
5. Configure JMS Transmission to specify the topic or queue to which your MDB is subscribed, and assign a JNDI name to the JMS Transmission.
6. Configure the CEI emitter that the CEI application is using to refer to JMS Transmission by the JNDI name you assigned to it in step 5.

Creating a data stream table from a Common Event Infrastructure connection
You can create a data stream table from a Common Event Infrastructure (CEI) connection.
**Procedure**

1. In IBM Cognos Real-time Monitoring Workbench, click the **Workbench** tab.
2. Click **Activities**.
3. Select **Create New**.
4. Click **Data Stream**.
5. Select **Single Data Stream**, then click **Continue**
6. Select **Common Event Infrastructure**, then click **Continue**.
7. Under **Column Information**, define the columns of the data stream table.
8. Save the CEI data stream as enabled.

**Common Event Infrastructure column information**

The column information fields define how to map the fields from the JMS message into columns in the data stream table. There is one field for every column in the data stream table.

Each field in the message can be a simple field that maps directly to a data stream column. Or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table. Complex fields are treated as flat files in either delimited (CSV), fixed-width, or XML formats. For detailed descriptions of these file types, see Chapter 13, “Flat files,” on page 93.

Each column in the data stream table has the attributes described in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Name</td>
<td>Name of the column in the data stream table.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Data type of the data stream column. For more information about data types, see [Java Messaging Service data types](on page 210).</td>
</tr>
<tr>
<td>Format</td>
<td>(Optional) Format of the data stream column for VARCHAR (string) and DECIMAL values.</td>
</tr>
</tbody>
</table>

Add columns by clicking **Add Field** or **Add Flat File Field**.

**Clear state interval**

Several options are available for clearing in-memory data from all views and cubes that receive data from a data stream.

The **Clear State Interval** tab contains the following options:

- **Do Not Clear State**
  - Is the default. In-memory data is not cleared, and the data is propagated from the data stream to the views, objects, and dashboards.
- **Clear State on a Schedule**
  - Selecting this option activates the scheduling controls **Change Time Source, Add Schedule, Edit Schedule**, and **Remove Schedule**.
- **Always clear state for new data**
The in-memory data for each view and cube is erased before loading new rows of data into the views and cubes. Each row of data in a single poll can trigger a clearing of the in-memory state of data. Select Treat all rows as a single event to keep multiple rows in memory.

**Clear state on a schedule**
You can use this option to activate scheduling controls.

If you click Add Schedule, you can specify the following options:
- Clear the state at a specific time or a time interval.
- Clear the state every day, on a specific day or days of the week, or a specific day or days of the month.
- Clear the state during a specific month or months.

For example, to schedule the clear state interval to occur every three months on the 30th day of the month at 10 p.m.:
1. Select Add a specific time and set Schedule at to the time 10:00 p.m.
2. Select On the day(s) of the months selected below and select 30.
3. For the month, select March, June, September, and December.

Click Edit Schedule to modify a selected schedule.

Click Remove Schedule button to delete a selected schedule.

**Change time source**
You can select time options that specify when to clear data.

Change Time Source opens a window where you can select when to clear data, according to the following options:
- At the specified time
  Data is cleared only at the times specified in a schedule. Selecting this option might mean that no data is in the view for a certain period. To avoid having no data in a view, select the option When data arrives after the specified time.
- When data arrives after the specified time
  Existing data is cleared when new data arrives after the time specified in a schedule.
- When the value in the timestamp column is after the specified time
  The data is cleared based on the value in a specified timestamp column. If the timestamp column contains a time that is later than the time in a schedule, existing data is cleared. If the time in the timestamp column is earlier than the time in a schedule, existing data is not cleared. Use the Select column drop-down list to select the column to use for timestamps. If there are no timestamp columns, this option is not available.
Chapter 8. Cubes

A cube is a set of data organized by dimensions and measures for aggregating different subsets of the larger set of data.

When cubes are rendered as dashboard objects, they can be used to specify categories that filter data to show the results that meet your selection. For example, a cube of sales data might provide aggregations of the same data by product, time, or sales region dimensions. Looking at the cube, you might choose to view the total sales of a product (Nails) within a business region (West) during a fiscal quarter (Q1):

```
West
  Q1  January  Hardware
Nails  120,000
West
  Q1  March  Hardware  Nails
       98,000
Total          218,000
```

By quickly removing the product dimension specification, you switch the classification to see all sales for that region and quarter:

```
West
  Q1  January  Hardware  Nails
       120,000
West
  Q1  March  Hardware  Nails
       98,000
```

```
West
  Q1  March  Hardware  Screws
       97,000
```

```
West
  Q1  January  Lumber  Studs
       137,000
Total          452,000
```

Or for all sales during the month of March:

```
West  Q1  March  Hardware  Nails
       98,000
```

```
West  Q1  March  Hardware  Screws
       97,000
```

```
East  Q1  March  Lumber  Plywood
       92,000
```

```
South  Q1  March  Hardware  Nails
       98,000
Total          385,000
```
Or for all West region sales of the Lumber family of products:

<table>
<thead>
<tr>
<th>West</th>
<th>Q1</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>Lumber</td>
</tr>
</tbody>
</table>

Total: 137,000

For more information, see the examples of cube charts and cube tables in the IBM Cognos Real-time Monitoring Dashboard User Guide.

**Measures**

Measures are the central values that are aggregated and analyzed.

In the examples in the cubes topic, total sales is the aggregate value. In each example, the total is measuring the sum of all sales in the set. Measures are built with the C-SQL Set functions, including SUM, AVG, MIN, MAX, STD_DEVIATION, and VARIANCE.

For more information about measures, see "Measure columns" on page 60.

**Dimensions in cubes**

A dimension is a ranked order of classifications that describe smaller, more distinct sets of related data from the highest to lowest level.

In the examples in the cubes topic, the business region is one level of a geographical dimension, the quarter and month columns are each levels of a time dimension, and the product family and product name are part of an inventory dimension. In the time dimension, months are smaller sets of fiscal quarters, just as product name is a smaller set of the product family level. The following table shows examples of dimensions:

<table>
<thead>
<tr>
<th>Table 27. Some examples of dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>year</td>
</tr>
<tr>
<td>quarter</td>
</tr>
<tr>
<td>month</td>
</tr>
<tr>
<td>week</td>
</tr>
<tr>
<td>day</td>
</tr>
<tr>
<td>hour</td>
</tr>
<tr>
<td>minute</td>
</tr>
</tbody>
</table>

For more information, see Chapter 11, "Dimensions," on page 83.

You can limit user access to data in a cube with an access filter. For more information, see Chapter 2, "Access filters," on page 3.
Creating cubes

Cubes are similar to business views in that they aggregate event data, but they do so across different dimensions.

The view that a cube aggregates is a fact table: a view or table in an event that contains one or more columns to measure (aggregate), and also contains columns that identify the dimensional elements associated with the event. A fact table might contain an event similar to the following example:

<table>
<thead>
<tr>
<th>Cost</th>
<th>Quantity</th>
<th>Product</th>
<th>State</th>
<th>Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.00</td>
<td>1600</td>
<td>Nails</td>
<td>California</td>
<td>January</td>
</tr>
</tbody>
</table>

However, in practice the dimensional elements are stored in dimensions (special lookup tables) and referenced by IDs, for example:

<table>
<thead>
<tr>
<th>Cost</th>
<th>Quantity</th>
<th>prod_id</th>
<th>region_id</th>
<th>ddim_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.00</td>
<td>1600</td>
<td>100</td>
<td>7</td>
<td>39</td>
</tr>
</tbody>
</table>

The following illustration shows a cube built from the OrderDetails fact table and measures total sales across various business regions, products, and time.

---

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the **Workbench** tab.
2. Select **Cubes**.
3. From the **Activities** menu, click **Create New**.
4. Click **Browse** and choose a fact table or tables and columns to measure.
5. Specify how to measure them (aggregate formulas to use).
6. Choose one or more dimensions that classify the measurements.

---

Figure 8. A cube built from the OrderDetails fact table
Measure columns

Measure columns define the aggregations that the cube calculates. A cube must have at least one measure column. Each measure column defines an expression that contains a C-SQL set function that aggregates other columns from the fact table.

For example, to determine the total sales from the OrderDetails fact table, a measure column might be defined as:

```
SUM(OrderDetails.prod_cost*OrderDetails.order_qty) AS TotalOrderSales
```

Creating a cube with dimension columns

Dimension columns categorize the measurements. A cube must have at least one dimension, and can have more. Further, the data in the fact table must be able to identify a unique element in each associated dimension.

For more information, see Chapter 11, “Dimensions,” on page 83.

Before you begin

Before creating a cube, you must have Create permission for views, cubes, and dimensions. You must also have the following prerequisites:

- At least Read-Only access to the dimensions to include. See “Creating dimensions” on page 86 for details.
- A dimension with Geo Categories enabled, if the cube is used for geography charts.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > Cube.
4. Enter a name for the cube in the Name field, and optionally provide a description of the cube.
5. Choose the Fact Table that contains the data to measure, and that contains columns that identify the dimension elements. Click Browse to select the Fact Table.
6. From the Select Object box, select the object to use for the Fact Table and click the OK button.
7. Define one or more “Measure columns.”
   - Click Add Measure Column to define a column.
   - Name the column in the Measure Name field.
   - Define the measure formula with a C-SQL “Set functions” on page 116 in the Aggregate Expression field. The function must reference a column from the fact table. For example, the following SUM() expression totals the product of the cost and quantity columns:
     
     ```
     SUM(OrderDetails.prod_cost*OrderDetails.order_qty)
     ```

     For more information about C-SQL expressions, see Chapter 14, “Formulas,” on page 115.

8. Define one or more dimension columns.
   - Click Add Dimension to define a column.
• Choose the dimension to include from the **Dimension** column drop-down list. This list includes all dimensions that you have at least Read-Only access to.

• Identify the **key columns** in the dimension and in the fact table. See “**Key columns**” on page 85 for more details about the keys.

  The data type for the key in the fact table must be the same for the key in the dimension (lookup table). You cannot, for example, mix integer and decimal types; both must be either integer or decimal.

9. Save the cube.
Chapter 9. Data types

IBM Cognos Real-time Monitoring and C-SQL support various SQL-99 data types.

Table 28. C-SQL data types

<table>
<thead>
<tr>
<th>C-SQL data type</th>
<th>Data type category</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>“Boolean operators” on page 75</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>“C-SQL data types for numeric values” on page 65</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>“C-SQL data types for numeric values” on page 63</td>
</tr>
<tr>
<td>INTEGER</td>
<td>“C-SQL data types for numeric values” on page 65</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>“Date-time” on page 69</td>
</tr>
<tr>
<td>LONG</td>
<td>“C-SQL data types for numeric values” on page 65</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>“Date-time” on page 69</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>“String” on page 68</td>
</tr>
</tbody>
</table>

C-SQL provides means for converting data of one type to another type. For more information, see “Data type conversion.”

Data type conversion

In IBM Cognos Real-time Monitoring, there are two ways to convert values from one data type to another: explicit casting and implicit casting.

Explicit cast

Any C-SQL argument can contain CAST() to convert the data type of a value.

For more information, see “CAST” on page 131.

For example, you can cast a character string of numerals into a numeric value and use the result as an argument to FLOOR (see “FLOOR” on page 144):

FLOOR( CAST( ‘1234.56' AS DECIMAL) )

Implicit cast

C-SQL automatically attempts to convert a data type to the correct type for the argument where the value is encountered.

For example, if C-SQL encounters the VARCHAR '1234.56' in the FLOOR() argument, it automatically converts the value to a DOUBLE PRECISION numeric before truncating the decimal digits. For example:

FLOOR( '1234.56' ) << Implicit cast to DOUBLE PRECISION.
When a value of one data type is compared to a value of a different type, C-SQL first converts one of the values to match the other. In the following example, C-SQL converts the VARCHAR string to a BOOLEAN before evaluating the expression:

'\texttt{true}' = \texttt{TRUE} \quad \text{\textless{} Implicit cast to BOOLEAN.}

Context also affects casting. For example, because the following arithmetic add operator expects numeric arguments, and even though both values are characters, the values are first cast to numeric:

'\texttt{2}' + '\texttt{3}' \quad \text{\textless{} Both cast to numeric to match operator data type.}

The value must be convertible to the required type or the expression results in an incorrect data type error.

**Order of precedence**

Each possible data type conversion is assigned an order of precedence.

The following table shows which types are convertible, and the order of precedence assigned to each possible data type conversion, where zero (0) is the highest precedence and a million (1,000,000) is the lowest:

<table>
<thead>
<tr>
<th>To \ From</th>
<th>VARCHAR</th>
<th>BOOLEAN</th>
<th>TIMESTAMP</th>
<th>DOUBLE PRECISION</th>
<th>DECIMAL</th>
<th>INTEGER</th>
<th>LONG</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>0</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>10</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONG</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the comparison example in ["Implicit cast" on page 63](#), C-SQL converts the VARCHAR value to a BOOLEAN value because the precedence level for that conversion is 1, as opposed to 10 for converting the BOOLEAN value to a VARCHAR value. Similarly, in the FLOOR("1234.56") example C-SQL converts the string to a DOUBLE PRECISION because DOUBLE PRECISION has a higher precedence than DECIMAL, even though a decimal might seem to be more appropriate to the value.

See the descriptions of the individual C-SQL data types for specific details about converting those types.
C-SQL data types for numeric values

C-SQL has several data types for numeric values.

Table 30. Data types for numeric values

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>Signed integer 32 Bits -2,147,483,648 minimum value 2,147,483,647 maximum value</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Decimal 1-digit minimum 256 digits maximum The total count of DECIMAL digits, both before and after the decimal separator is 256.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>IEEE 754 floating point 64 Bits +/-4.94065645841246544E-324 minimum value +/-1.79769313486231570E-308 maximum value To express a DOUBLE PRECISION as a literal, use scientific notation, such as 1e24.</td>
</tr>
<tr>
<td>LONG</td>
<td>Signed long 64 bits -2^64 minimum value 2^64 - 1 maximum value</td>
</tr>
</tbody>
</table>

Attention:
- The total count of DECIMAL digits, both before and after the decimal separator, is 256.
- To express a DOUBLE PRECISION as a literal, use scientific notation, such as 1e24.

Third-party data types that map to C-SQL numeric data types

The C-SQL numeric data types map to data types in other support systems.

The following table describes the mapping of C-SQL numerics to data types in other support systems, where p means precision:
<table>
<thead>
<tr>
<th>Support System</th>
<th>Integer</th>
<th>Long</th>
<th>Decimal</th>
<th>Double</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-SQL/ JDBC</td>
<td>INTEGER</td>
<td>LONG</td>
<td>DECIMAL</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>Java</td>
<td>int</td>
<td>long</td>
<td>BigDecimal</td>
<td>double</td>
</tr>
<tr>
<td>Oracle</td>
<td>Number(p=38)</td>
<td>Number((19,0))</td>
<td>Number</td>
<td>Number(p=38)</td>
</tr>
<tr>
<td>SQL Server</td>
<td>Int(32 bit)</td>
<td>BigInt</td>
<td>Decimal</td>
<td>double real(4 bytes)</td>
</tr>
<tr>
<td>Sybase</td>
<td>Int(32 bit)</td>
<td>BigInt</td>
<td>Decimal(p=38)</td>
<td>double real(4 bytes)</td>
</tr>
<tr>
<td>MySQL</td>
<td>TINYINT SMALLINT MEDIUMINT INT, INTEGER BIGINT</td>
<td>BIGINT</td>
<td>DECIMAL NUMERIC</td>
<td>DOUBLE REAL</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>SMALLINT INT, INTEGER BITINT</td>
<td>BIGINT</td>
<td>DECIMAL NUMERIC</td>
<td>DOUBLE PRECISION REAL</td>
</tr>
</tbody>
</table>

**Combining numeric types**

When combining two different numeric types, the result is the type with higher precedence. For example, adding an INTEGER to a DECIMAL results in a DECIMAL sum.

Precedence is based on the "Order of precedence" on page 64.

Casting a fractional number to an integer truncates the fraction (rounds down) to fit the target. For example, forcing a DOUBLE PRECISION into an INTEGER truncates the fractional part of the value.

**Casting numeric types**

Casting numerics to types of different storage size is permissible, if the target is large enough to hold the result; otherwise the conversion fails with a number-out-of-range error. For example, attempting to put a floating-point numeric type of a larger storage size into a location of a smaller size results in an error.
Numeric to string guidelines

You must be aware of some guidelines when casting numerics to strings.

These guidelines include:

- For DECIMAL numbers, the result is zero-padded in the decimal values to match the precision and scale defined for the column. For example, if a column is defined as precision 5 and scale 4, a value of 1.1 in the column is cast as '1.1000'.
- For DOUBLE PRECISION numbers, the 'e' is cast to uppercase. For example, +1e11 is converted to '1.0E11'.

Decimal precision results

All decimal numbers have two components: precision and scale.

- Precision
  The count of digits, both to the left and right of the decimal point. The maximum is 256. The minimum is 1.
- Scale
  The count of digits of the fractional component. It is less than or equal to the precision. When no scale is specified, the default is 2.

In instances where a value has greater scale or precision than the target storage, such as a database field with a smaller precision, IBM Cognos Real-time Monitoring truncates decimals and rounds down the result to make it fit.

Casting

When casting a decimal value, you can declare the precision and scale.

You declare the precision and scale as follows:

DECIMAL( precision, scale )

For example,

CAST( '4.012345', DECIMAL(5,4) ) --> 4.0123

When casting from a decimal formatted column to a string, the result is zero-padded on the decimals to match the scale. For example, when the column is precision 5 and scale 4, implicitly casting a value of 1.1 in the column to a string results in '1.1000'.

Multiplication

In multiplication, the resulting precision is the sum of the precisions, and the scale is the sum of the scales.

The results are determined by using this calculation:

PrecisionResult = MIN( PrecisionLeft+PrecisionRight,256)  
ScaleResult = MIN( ScaleLeft+ScaleRight, 256)

For example, the result of (4.55*1.414) is precision 7 (3+4) and scale 5 (2+3).

Division

The results in division are determined by using this calculation.
For example, the result of (4.55/1.414) is scale 4 (2+4-1) and precision 10 (3+3+4).
The calculation is:
ScaleResult = MIN(MAX((ScaleLeft+PrecisionRight-ScaleRight+1),2),256)
PrecisionResult = MIN((PrecisionLeft + ScaleRight + ScaleResult),256)

**Addition and subtraction**
The results in addition and subtraction are determined by using this calculation.
PrecisionResult = MIN( (MAX( PrecisionLeft - ScaleLeft, 
                         PrecisionRight - ScaleRight) + 
                         MAX( ScaleLeft, ScaleRight ) + 1),256)
ScaleResult = MAX( ScaleLeft, ScaleRight )

**All other functions**
For other functions and operations, the result is determined by the value with the
largest precision and the value with the largest scale. The results can be
determined from the same value.
PrecisionResult = MIN( MAX( Precision[i] ), 256)
ScaleResult = MIN( MAX( Scale[i] ), 256)

**String**
The C-SQL VARCHAR data type maintains character string values.

*Table 32. The C-SQL VARCHAR data type*

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Padding</th>
<th>Minimum size</th>
<th>Maximum size</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>Variable length</td>
<td>No</td>
<td>1 character</td>
<td>Infinite characters. Note that an error occurs if you attempt to store a value in a DBMS that is larger that the size of the column defined in the table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(default), can be null.</td>
<td></td>
</tr>
</tbody>
</table>

**String width**
Though the maximum size limit for string values is infinite, try not to exceed 255 characters because that is the limit imposed on many DBM systems.

However, to improve performance, assist data storage, and aid in string compares, declare an appropriate maximum width when defining a VARCHAR column. The width must be large enough to hold the maximum length of any string result inserted into the field. Text that is longer than the maximum width is truncated when the string is stored.

**Third-party data types that map to VARCHAR**
The C-SQL/JDBC string type VARCHAR maps to data types in other support systems.

The following table describes the mapping of C-SQL/JDBC string type VARCHAR to data types in other support systems:
### String concatenation
You can concatenate two strings.

To concatenate two strings, use either CONCAT or the || operator. For more information, see “CONCAT” on page 155.

### String literal
To express a string as a literal, you enclose the text in single quotes ('). To include a single quote, include two.

For example:

'Couldn't' Returns: Couldn't

### Converting strings to other data types
When combining a string with another data type, or when expressing a string where another data type is expected, the string is automatically converted to the new type based on the order of precedence.

For more information, see “Order of precedence” on page 64.

During conversion, the following behaviors occur:

- All leading and trailing spaces are removed.
- If the string contains an invalid character or invalid formatting, an error occurs. An invalid character is one that is inappropriate for the target data type. For example, 'hello' cannot be converted to an INTEGER.
- Formatting that is not consistent with the definition of a literal data value of the target type is invalid. For example, for a string to implicitly convert successfully to a TIMESTAMP data type, the source string must be contained in the default C-SQL date format. See “Date-time” for details about formatting strings for date-time types.

### Date-time
Date-time data types store date and time-of-day of that date as a single value.
You cannot directly access a date-time data type as its own internal numeric representation. Instead, to access date-time values in a meaningful way, C-SQL provides several functions for manipulating the values, and provides literal constructs for representing the values in expressions.

Table 34. Date-time data types

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP literal (see &quot;TIMESTAMP literal” on page 71)</td>
<td>A character string representation of a date-time value. Can be any combination of year, month, day-of-month, hour, minute, second, and fractional seconds.</td>
</tr>
<tr>
<td>INTERVAL literal (see &quot;INTERVAL literal” on page 72)</td>
<td>A character string representation of an interval: a span of time comprised of years and months, or days, hours, minutes, and seconds.</td>
</tr>
</tbody>
</table>

Date-time values are in the time zone of the locale of the server that runs IBM Cognos Real-time Monitoring.

Converting between date-time and strings

You can convert a date-time value to a character string (VARCHAR) and convert a string to a date-time data value. For both types of conversions, you can specify the format of the string.

Convert a date-time to a character string (VARCHAR) with TO_CHAR (see “TO_CHAR” on page 138), and convert a string to a date-time with TO_DATE (see “TO_DATE” on page 139).

When a TIMESTAMP literal (see "TIMESTAMP literal” on page 71) is included in a string, an argument automatically converts the value to a string using the default date-time format, which is yyyy-MM-dd hh:mm:ss.SSS. For more information about converting between date-time and string values, see “Data type conversion” on page 63.

Comparing date-time values

A date-time value is stored internally as a number that represents the date-time in milliseconds. Therefore, you must be careful when comparing two date-time values.

For example, this comparison is only true when both dates have the same number of milliseconds:

```
first_date = second_date
```

If exact granularity is not important, consider first converting the date-time values to strings that represent just the date portion:

```
TO_CHAR(first_date,"yyyy-MM-dd") = TO_CHAR(second_date,"yyyy-MM-dd")
```

According to the “Order of precedence” on page 64, comparing a string to a date-time first casts the string to a date-time before the comparison occurs. Consider this example where birth_date is a date-time value. If birth_date has a time associated with it, the comparison never evaluates to true:

```
'2010-02-18' = birth_date
```

A more exact comparison is to first cast birth_date to a string without a time:
'2010-02-18' = TO_CHAR( birth_date, "yyyy-MM-dd" )

**Date-time arithmetic**

You can add and subtract intervals of years, months, days, hours, minutes, and seconds on date-time values.

See the descriptions of those functions for details. Some query clauses, however, require an INTERVAL literal.

For information about INTERVAL literal, see "INTERVAL literal" on page 72

**Third-party data types that map to the C-SQL date-time type**

The C-SQL date-time type maps to data types in other support systems.

The following table describes the mappings of the date-time type.

**Table 35. Mappings of the date-time type**

<table>
<thead>
<tr>
<th>Support system</th>
<th>Date-time type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-SQL/JDBC</td>
<td>Date-time</td>
</tr>
<tr>
<td>Java</td>
<td>Date</td>
</tr>
<tr>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>Timestamp</td>
</tr>
<tr>
<td>Oracle</td>
<td>Date(YMDHMS)</td>
</tr>
<tr>
<td>SQL-Server</td>
<td>Datetime(YMDHMS.xx)</td>
</tr>
<tr>
<td></td>
<td>SmallDateTime (YMDHMS)</td>
</tr>
<tr>
<td>Sybase</td>
<td>Datetime (YMDHMS.xx)</td>
</tr>
<tr>
<td></td>
<td>SmallDateTime (YMDHMS)</td>
</tr>
<tr>
<td>MySQL</td>
<td>DATE</td>
</tr>
<tr>
<td></td>
<td>DATETIME</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td></td>
<td>YEAR</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>TIMESTAMP</td>
</tr>
</tbody>
</table>

**TIMESTAMP literal**

The TIMESTAMP literal represents a date-time value as a character string. To express a date-time as a literal value, you prefix the data with the word 'TIMESTAMP', and enclose the entire data in single quotes (').

For example:

TIMESTAMP '2010-03-05 19:45:23.123'
The format of the string is yyyy-MM-dd hh:mm:ss.SSS, where S (the fractional seconds) are optional and can be from zero to nine digits of precision. See “Date-time formatting” on page 74 for details about the formatting characters.

**INTERVAL literal**

An INTERVAL literal identifies a span of time comprised of years and months (year-month intervals) or of days, hours, minutes, and seconds (day-time intervals).

You cannot combine year-month and day-time in one interval declaration. Intervals are applied to date-time values to calculate the span of time from that instance.

Typically, intervals are used in expressions to offset date-time columns and TIMESTAMP literals, such as when declaring the range from a date or time in query windows. For more information, see Chapter 38, “Query windows,” on page 361. For example, the following query window totals all events that arrive in the last hour (implicitly applied to the arrival time of the latest event to arrive):

```
SUM(Qty) AS Total_Of_Qty OVER (RANGE INTERVAL '1' HOUR PRECEDING)
```

When applying an interval to a date-time, the interval is added to or subtracted from the value. For example, if the current date-time is 5 March 2010 at 7:45 p.m., adding an interval of one year to that date results in 5 March 2011 at the exact same time. Calendar arithmetic follows Gregorian calendar rules. For more information, see “DATE_DIFF” on page 134.

**Year-month intervals**

A year-month INTERVAL uses either, or combines both, of the date-time fields YEAR or MONTH.

The possible definitions are:

```
INTERVAL 'yy' YEAR[(<precision>)]
INTERVAL 'mm' MONTH[(<precision>)]
INTERVAL 'ymm' YEAR[(<precision>)] TO MONTH[(<precision>)]
```

The following examples define intervals of three years and of 10 months:

```
INTERVAL '3' YEAR
INTERVAL '10' MONTH
```

You can define a fraction interval by expressing the result in total months, such as 46 months, or by combining the field. The following example identifies an interval of 3 years and 10 months:

```
INTERVAL '3-10' YEAR TO MONTH
```

You can specify a value of zero (0) for either field. The following intervals are each two years:

```
INTERVAL '2-0' YEAR TO MONTH
INTERVAL '1-12' YEAR TO MONTH
INTERVAL '0-24' YEAR TO MONTH
```

**Precision of year-month intervals**

The <precision> argument is an ANSI standard that declares the maximum count of digits in the integer. By default, the <precision> is 2.
Given this default, the following two declarations of 100 month intervals each fail:

```
INTERVAL '100' MONTH(2)  << ERROR, precision is less than value size.
INTERVAL '100' MONTH   << ERROR, default precision is 2.
```

To use more than two digits, declare a precision in the following manner:

```
INTERVAL '100' MONTH(3)
```

When using both fields, apply the precision on the YEAR field only; the MONTH field uses its default precision of 2. For example, the following is erroneous because the month is greater than the default.

```
INTERVAL '100-123' YEAR(3) TO MONTH  << ERROR, month is 3 digits
```

**Day-time intervals**

A day-time INTERVAL is composed of a combination of days, hours, minutes, and seconds.

The possible definitions are as follows, and where .nn is a fraction of a second:

```
INTERVAL 'dd'     DAY
INTERVAL 'ddhh'   DAY TO HOUR
INTERVAL 'ddhh:mm' DAY TO MINUTE
INTERVAL 'ddhh:mm:ss[.nn ]' DAY TO SECOND
INTERVAL 'hh'     HOUR
INTERVAL 'hh:mm'  HOUR TO MINUTE
INTERVAL 'hh:mm:ss[.nn ]' HOUR TO SECOND
INTERVAL 'mm'     MINUTE
INTERVAL 'mm:ss[.nn ]' MINUTE TO SECOND
INTERVAL 'ss[.nn ]' SECOND
```

Here are some examples of day-time intervals:

```
INTERVAL '27 23:59:59.999999999' DAY TO SECOND
INTERVAL '100 10:10' DAY(3) TO MINUTE
```

**Precision of day-time intervals**

Each of the day-time fields also has a precision argument.

For example:

```
MINUTE(<precision>)
```

The `<precision>` argument is an ANSI standard that declares the maximum count of digits in the integer. By default, the `<precision>` is 2 (except for fractional seconds whose default is 9, see the following example for details). Therefore, the following two declarations of 100 hour intervals each fail:

```
INTERVAL '100' HOUR(2)  << ERROR, precision is less than value size.
INTERVAL '100' HOUR   << ERROR, default precision is 2.
```

To use more than two digits, declare a precision in the following manner:

```
INTERVAL '100' HOUR(3)
```
When declaring precision for SECOND with a fractional component, specify two precision values separated by a comma. Consider the following examples:

```
INTERVAL '12.345' SECOND(2, 3)
INTERVAL '12.123456789' SECOND(2, 9)
INTERVAL '12.123456789' SECOND
```

Notice that the last two examples have the same effect because the default is (2, 9) for SECOND.

When using multiple fields, express the precision on the first field only. The remaining fields use their default. For example, the precision in the following example applies to the minutes only and does not affect the fractional seconds:

```
INTERVAL '100:23.123456789' MINUTE(3) TO SECOND
```

### Date-time formatting

You can define the format of the date-time string for some functions.

The `TO_CHAR` (see “TO_CHAR” on page 138) and `TO_DATE` (see “TO_DATE” on page 139) functions both have arguments that define the format of the date-time string. The format date pattern string shown in the following table is identical to the one used by the Java SimpleDateFormat class, which uses these letters in patterns:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Date-time component</th>
<th>Presentation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era designator</td>
<td>Text</td>
<td>AD</td>
</tr>
<tr>
<td>y</td>
<td>Year</td>
<td>Year</td>
<td>2010; 10</td>
</tr>
<tr>
<td>M</td>
<td>Month in year</td>
<td>Month</td>
<td>July; Jul; 10</td>
</tr>
<tr>
<td>w</td>
<td>Week in year</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>W</td>
<td>Week in month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>Day in year</td>
<td>Number</td>
<td>189</td>
</tr>
<tr>
<td>d</td>
<td>Day in month</td>
<td>Number</td>
<td>10</td>
</tr>
<tr>
<td>F</td>
<td>Day of week in month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>Day in week</td>
<td>Text</td>
<td>Tuesday; Tue</td>
</tr>
<tr>
<td>a</td>
<td>Am/pm marker</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>H</td>
<td>Hour in day (0-23)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>k</td>
<td>Hour in day (1-24)</td>
<td>Number</td>
<td>24</td>
</tr>
<tr>
<td>K</td>
<td>Hour in am/pm (0-11)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>Hour in am/pm (1-12)</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>m</td>
<td>Minute in hour</td>
<td>Number</td>
<td>30</td>
</tr>
<tr>
<td>s</td>
<td>Second in minute</td>
<td>Number</td>
<td>55</td>
</tr>
<tr>
<td>S</td>
<td>Fraction of a second (one S always returns an integer of 0 to 9 digits)</td>
<td>Number</td>
<td>978</td>
</tr>
<tr>
<td>'</td>
<td>escape for text</td>
<td>Delimiter</td>
<td></td>
</tr>
<tr>
<td>&quot;</td>
<td>single quote</td>
<td>Literal</td>
<td></td>
</tr>
</tbody>
</table>
Some letters have multiple results, depending on the number of consecutive letters in the format. The result is the value that best fits the pattern. For numbers, if the pattern is bigger than the value, the result is padded with leading zeros (0). See the examples in the following table for details.

The examples in the following table, modified from the Java SimpleDateFormat class documentation, show how date and time patterns are interpreted in the US locale. The date and time are 2010-08-04 12:08:56 local time in the US Pacific timezone.

### Table 37. Date and time patterns interpreted in the US locale

<table>
<thead>
<tr>
<th>Date and Time Pattern</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>'d M yy'</td>
<td>4 8 10</td>
</tr>
<tr>
<td>'dd MM yy'</td>
<td>04 08 10</td>
</tr>
<tr>
<td>'ddd MMMM yyy'</td>
<td>004 Aug 2010</td>
</tr>
<tr>
<td>'ddddd MMMMM yyyyy'</td>
<td>0004 August 2010</td>
</tr>
<tr>
<td>&quot;yyyyMMdd G 'at' HH:mm:ss z&quot;</td>
<td>2010.08.04 AD at 12:08:56 PDT</td>
</tr>
<tr>
<td>&quot;EEE, MMM d, &quot;yy&quot;</td>
<td>Sat, Aug 4, '10</td>
</tr>
<tr>
<td>&quot;h:mm a&quot;</td>
<td>12:08 PM</td>
</tr>
<tr>
<td>&quot;hh 'o'clock' a, zzzz&quot;</td>
<td>12 o’clock PM, Pacific Daylight Time</td>
</tr>
<tr>
<td>&quot;K:mm a, z&quot;</td>
<td>0:08 PM, PDT</td>
</tr>
<tr>
<td>&quot;yyyy:MMMMM:dd GGG hh:mm aaa&quot;</td>
<td>02010.August.04 AD 12:08 PM</td>
</tr>
<tr>
<td>&quot;EEE, d MMM yyyy HH:mm:ss Z&quot;</td>
<td>Sat, 4 Aug 2010 12:08:56 -0700</td>
</tr>
<tr>
<td>&quot;yyMMddHHmmssZ&quot;</td>
<td>100704120856-0700</td>
</tr>
</tbody>
</table>

### Boolean operators

C-SQL follows the SQL-99 use of three-valued logic (TRUE, FALSE, and UNKNOWN) to support NULL value semantics.

For example:

```sql
WHERE OnSale IS TRUE
WHERE (Age >= 21) IS UNKNOWN
```

### Truth table for NOT

When using Boolean operators to evaluate the truth of an expression, the values are evaluated as described in the following truth tables:

#### Table 38. Truth table for NOT

<table>
<thead>
<tr>
<th>NOT</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>
Truth table for AND

The following table is a truth table for AND.

<table>
<thead>
<tr>
<th>AND</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

Truth table for OR

The following table is a truth table for OR.

<table>
<thead>
<tr>
<th>OR</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

Truth table for IS

The following table is a truth table for IS.

<table>
<thead>
<tr>
<th>IS</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

TRUE is greater than FALSE in comparisons.
Chapter 10. Data streams

Data stream tables receive events. Events are data produced by external business applications that record transactions, identify changes in business state, and synthesize the details about business activities.

IBM Cognos Real-time Monitoring receives events in data stream tables. Events drive the internal processing of Cognos Real-time Monitoring.

Views are built on the data stream tables, then aggregate the event information and drive the rules that look for exceptional business conditions.

For more information, see Chapter 36, “Views,” on page 337 and Chapter 28, “Rules,” on page 297.

This section describes how data stream tables work, the external sources that they support, their properties, and how to create and edit data stream tables.

How data streams work

Events come from business applications, databases, and text files.

Usually agents automatically receive or retrieve the event data and load it into a data stream table. Alternatively, you can manually load data from text files with IBM Cognos Real-time Monitoring Workbench. As events arrive, they are processed and data from the events is passed to the business views. The views then aggregate the data and retrieve lookup table data relative to the event.

Events are processed in the order that they are received in the system. When one agent receives a large quantity of events, any new events received by other agents are queued behind the first set and are not processed until the first set is processed.
External event sources for data streams

Agents can access several external event sources for data streams.

For more information about how IBM Cognos Real-time Monitoring provides agents to retrieve and receive event data, see Chapter 4, “Agents,” on page 21.

Some events flow (are pushed) into the data stream tables as they happen. Other events are loaded (are pulled) into the data stream tables as the result of a request, such as from a database or a text file. The following table summarizes the available event sources and identifies the source agent they support, and identifies whether the data is pulled or pushed to the data stream table.

Table 42. Event sources

<table>
<thead>
<tr>
<th>External source</th>
<th>Agent</th>
<th>Data stream push or pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Event Infrastructure</td>
<td>None. (See Chapter 7, “Common Event Infrastructure,” on page 51)</td>
<td>Push</td>
</tr>
<tr>
<td>Java Messaging Service (JMS)</td>
<td>Java Messaging Service (JMS) (see Chapter 17, “Java Messaging Service,” on page 207)</td>
<td>Push</td>
</tr>
<tr>
<td>Text file (XML or flat)</td>
<td>Flat Files (see Chapter 13, “Flat files,” on page 93)</td>
<td>Pull</td>
</tr>
<tr>
<td>TIBCO Rendezvous (RV)</td>
<td>TIBCO Rendezvous (see Chapter 33, “TIBCO Rendezvous,” on page 323)</td>
<td>Push</td>
</tr>
<tr>
<td>HTTP Post action</td>
<td>HTTP Post (see Chapter 16, “HTTP Post,” on page 201)</td>
<td>Push</td>
</tr>
<tr>
<td>Salesforce</td>
<td>Salesforce (see Chapter 29, “Salesforce,” on page 303)</td>
<td>Pull</td>
</tr>
<tr>
<td>Web service</td>
<td>None. (See “Web service data streams” on page 349)</td>
<td>Push</td>
</tr>
</tbody>
</table>

Data stream attributes

Every data stream table has a name, description, and status attribute, and most have an agent.

The following table describes the attributes:

Table 43. Data stream attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the table and is the name accessed by the business views that depend on this table. The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see ”Object namespace” on page 261.</td>
</tr>
</tbody>
</table>
Table 43. Data stream attributes  (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save in</td>
<td>Specifies the folder in which the data stream table is to be stored. Use the Choose Folder button to select or create a folder. If you do not specify a folder, the default is Public Folder.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Agent</td>
<td>An agent that receives or retrieves the data stream information, and passes the data to the data stream table. For information about agent types, see Chapter 4, “Agents,” on page 21.</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Process data in the order of arrival</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn off this attribute. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
</tbody>
</table>

Creating data stream tables

You can create a data stream table to receive events.

Tip:

Use the Treat all rows as a single event option only when no other option is available. Although enabling this option can increase the rows processed per second, it can degrade user response, that is, the time it takes for the user interface to refresh. Also, enabling this option can use more transient memory, so that if the physical memory limit of the system is reached, the number of rows processed per second is also reduced.

The purpose of the Always clear state for new data option is to clear the state at a specific time every day or month. However, when used in combination with the Treat all rows as a single event option (which not take advantage of the streaming capabilities of Cognos Real-time Monitoring), many rows of data are read and processed multiple times. This has a negative effect on performance and should be avoided.

When specifying a source type for a data stream, note that each type has specific attributes. For more information, see the following topics:

- Chapter 13, “Flat files,” on page 93
- Chapter 16, “HTTP Post,” on page 201
- Chapter 17, “Java Messaging Service,” on page 207
- Chapter 18, “Java Database Connectivity,” on page 211
- Chapter 29, “Salesforce,” on page 303
Before you begin

Before creating a data stream table, you must have Create permission for tables, and Read-only access permission for the agent that supplies the table with data.

For information about Create permission, see “Granting permission to create objects” on page 278.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New.
4. Select Data Stream.
5. Choose the source type.
6. Complete the fields in the new data stream form.
7. Save the object as enabled.
   The object is immediately ready to receive data.

Editing data stream tables

You can edit the attributes of data stream tables.

Attention: Editing the attributes of a data stream table causes the object to lose state, and might invalidate dependent views. For example, if you remove a column, any view or rule that references that column becomes invalid. However, if you redefine the column in the table, the dependent views are automatically revalidated.

Each type of data stream has its own specific attributes. For details, see the following sections:

- Chapter 13, “Flat files,” on page 93
- Chapter 16, “HTTP Post,” on page 201
- Chapter 17, “Java Messaging Service,” on page 207
- Chapter 18, “Java Database Connectivity,” on page 211
- Chapter 29, “Salesforce,” on page 303
- Chapter 33, “TIBCO Rendezvous,” on page 323
- Chapter 37, “Web services,” on page 349

Before you begin

Before editing a data stream table, you must have Read and Write permissions for tables, and Read-only access permission on the agent that supplies the table with data. For more information, see “Access permissions” on page 275.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Select Data Streams.
3. Under Object Name, select the data stream you want to edit.
4. From the Activities menu, click Edit.
5. Change the definitions as needed.
6. Save the object as enabled.
   The object is immediately ready to receive data.
Chapter 11. Dimensions

Dimensions are ranked orders of related data and are used by cubes to categorize measurements.

A dimension is a ranked order of classifications that, from highest to lowest level, each describe decreasingly smaller sets of related data. The following table presents some examples of dimensions where the top level of each list contains the largest set of related items, and the bottom level contains the smallest set.

Table 44. Examples of dimensions

<table>
<thead>
<tr>
<th>Time</th>
<th>Geography</th>
<th>Inventory</th>
<th>Security</th>
<th>Taxonomy</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>continent</td>
<td>classification</td>
<td>type</td>
<td>kingdom</td>
</tr>
<tr>
<td>quarter</td>
<td>country or region</td>
<td>type</td>
<td>rating</td>
<td>phylum</td>
</tr>
<tr>
<td>month</td>
<td>region</td>
<td>manufacturer</td>
<td>company</td>
<td>class</td>
</tr>
<tr>
<td>week</td>
<td>state</td>
<td>model</td>
<td>CUSIP</td>
<td>order</td>
</tr>
<tr>
<td>day</td>
<td>county</td>
<td>configuration</td>
<td></td>
<td>family</td>
</tr>
<tr>
<td>hour</td>
<td>city</td>
<td></td>
<td></td>
<td>genus</td>
</tr>
<tr>
<td>minute</td>
<td>district</td>
<td></td>
<td></td>
<td>species</td>
</tr>
</tbody>
</table>

Cubes categorize measurements by dimensions, and within dimensions by levels and values. For example, a location dimension can filter the results to show the measurements for all cities in a state. The following illustration shows several locations filtered by state name, which limits the measurements to just three cities.

Figure 10. Locations filtered by state name

Dimensions draw their values from lookup tables, where each dimension level is one column in the table, and each row is a unique dimension element. In the
example shown in the previous illustration, the region names are in one column, states in another, and cities in a third. When you create a dimension, you identify the existing source lookup table and the columns to include, and you arrange the columns into the level hierarchy.

Level hierarchy

The level hierarchy is what enables rollup and drill-down in cubes. When a user is viewing data for one level, they can roll up to see a higher level of measurements, or drill-down to see the data categorized at the next smaller level.

For example, when looking at the results for a state, the user might choose to roll up to see the measurement for all states in the region. Alternatively, the user might click the results to see the results for each city in the state. The level hierarchy defines the levels in the dimension.

The following illustration shows four columns in the source file, but only three contain dimension level data. These three are selected and arranged in the containing hierarchy of largest to most-specific levels.

If you select Use this for Geography Charts, a Geo Categories column is displayed. For information about Geo categories, see Geo categories.

Alias names in cubes

After identifying the columns, you can optionally assign alias names that the users see when they work with the cube.

The Order By column field

In IBM Cognos Real-time Monitoring Dashboard, dimension values are presented in their sort order as provided by the server.

For example, a list of month names appears in alphabetical order, starting with April and ending with September, rather than in the order they occur in a year. To specify another order, use the Order By column field. This field identifies another column that contains the values to use for sorting. For example, instead of using the month name column, use the month number column.

Also, if your Order By column field has multiple values that correspond to a single value in the dimension, the minimum value of the Order By column is used for the sort order. In such cases, be careful when selecting the column to use for sorting values. For example, the use of month names as the sort order for quarters could produce unpredictable results because the months would be sorted by alphabetical rather than numerical order.

Geo categories

Geo categories enable geographic information to be used in cubes.

A geo category setting specifies the level of accuracy that IBM Cognos Real-time Monitoring Dashboard uses for the column. For example, if a column contains the names of states or provinces, you would assign it the geo category State/Province.

The available categories are:
• Not Used
• Country or region
• State/Province
• City
• Street
• Postal Code

Use the **Not Used** category for a column that you do not want to map on a Geography chart or does not match an available category. For example, a column of REGION_NAME that contains data such as North, South, East, and West cannot be mapped to a geographic location, and you would assign this column to the **Not Used** geo category.

**Key columns**

To use the lookup table as a dimension, your event data must identify the unique dimension element (row) that it belongs to.

In the location lookup table data, the unique value in each element is the city name; however, while that name could be used for identification purposes, for performance reasons it is better to use a number.

The data for the locations lookup table must have an integer ID as follows:

<table>
<thead>
<tr>
<th>region_id</th>
<th>region_name</th>
<th>region_state</th>
<th>region_city</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------</td>
<td>-----------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>West</td>
<td>Nevada</td>
<td>Reno</td>
</tr>
<tr>
<td>2</td>
<td>South</td>
<td>Texas</td>
<td>Austin</td>
</tr>
<tr>
<td>3</td>
<td>East</td>
<td>New York</td>
<td>Rochester</td>
</tr>
<tr>
<td>4</td>
<td>Central</td>
<td>Ohio</td>
<td>Toledo</td>
</tr>
<tr>
<td>5</td>
<td>West</td>
<td>California</td>
<td>Pasadena</td>
</tr>
</tbody>
</table>

The fact table that provides the value for the cube to measure also includes the key value to identify the associated dimension. For example, this order record is associated with Ojai, California, in the West region:

<table>
<thead>
<tr>
<th>order_id</th>
<th>region_id</th>
<th>total_sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>102341</td>
<td>7</td>
<td>120000...</td>
</tr>
</tbody>
</table>

The data type for the key in the fact table must be the same for the key in the dimension (lookup table). You cannot, for example, mix integer and decimal types; both must be either integer or decimal.

When you define the dimension, identify one or more key fields that can be used to identify the specific level. You can choose any column that is not already a dimension level because levels are automatically assumed to be potential keys. The following illustration shows one column, REGION_ID, because all the other columns in the dimension are assigned to levels. Later, when you define the cube, identify the key column in the dimension that maps to the key in the fact table.

For information about defining cubes, see “Creating cubes” on page 59.
Creating dimensions

You can create dimensions to use in cubes.

Tip: Turn on caching for the lookup table for optimum performance. When caching is off, performance for cubes might be slowed dramatically. For more information, see “Caching lookup table queries” on page 250.

Before you begin

Before creating a dimension, you must have Create permission for views, cubes, and dimensions, and have at least Read-only access to the lookup table that provides the dimension elements.

For more information, see Chapter 20, “Lookup tables,” on page 247.

After you create a dimension, you can immediately use it in cubes. For more information, see Chapter 8, “Cubes,” on page 57.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > Dimension.
4. Enter a name for the dimension in the Name field and optionally provide a description of the dimension.
5. Click Browse to choose the lookup table that contains the dimension elements.
6. In the Select Object window, select the object to use for the Fact Table.
7. Click OK.
8. Select Use this for Geography Charts if you plan to implement geography maps. This adds a Geo Categories column to the Selected Fields list.
9. Define the levels of the hierarchy:
   • Add fields to the hierarchy from the Available Fields list.
   • Order the levels from largest set (top) to smallest (bottom).
   • Identify one or more key columns to include.
   • Optionally assign alias names to the levels.
   • Optionally assign geography categories to each level, if you selected the Use this for Geography Charts check box. The possible categories are Not Used, country or region, State/Province, City, Street, and Postal Code.
10. Save the dimension.
Chapter 12. Cognos Real-time Monitoring file system

The file system that you use to load data from a flat file into a lookup table.

Using flat files for lookup table data, you can also fetch data from a file and store it on a local database through the JDBC Store option.

For more information, see "Lookup tables from flat files."

Lookup tables from flat files

The file system agent loads data from a flat file into a lookup table through batch mode processing.

Based on an invalidation schedule, the file system agent periodically reads the flat file. When the file is read, the agent retrieves the data and passes it to the lookup table for processing.

Lookup tables created from flat files receive data by uploading data from a flat file into IBM Cognos Real-time Monitoring Workbench. The two flat file formats for lookup table data are:

- Delimited files
- Fixed-width files

For detailed information about these types of files, see Chapter 13, “Flat files,” on page 93.

Before creating a lookup table from data in a flat file, you need Create permissions for tables, and Read-only access permission on the agent that provides data to the table.

For fixed-width and delimited files, you might want a sample file that contains data in the format of the actual data for the lookup table. Use the sample when you create the lookup table to ensure that the fields map correctly into the table. Using a sample file is optional. You can create the lookup table without a source, but having a source assists you with the creation of a lookup table.

The following table describes the basic attributes for lookup tables from flat files. These attributes are similar to attributes for data stream tables used with flat files. For more information about flat files, see Chapter 13, “Flat files,” on page 93.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the lookup table.                                                                                                         Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which the lookup table is to be stored. Use the Choose Folder button to select or create a folder. If you do not specify a folder, the default is Public Folder.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>File System Agent</td>
<td>An existing file system agent that retrieves data streams and lookup table data from a text file. Create an agent with Cognos Real-time Monitoring Workbench. For more information, see “File system agents” on page 29.</td>
</tr>
<tr>
<td>Root Path</td>
<td>The root path specified when you created the file system agent.</td>
</tr>
<tr>
<td>Relative Path</td>
<td>The path relative to the root path of the text file that is to be read for lookup table data.</td>
</tr>
<tr>
<td>Start import on row</td>
<td>For fixed-width and delimited files, identifies the row of the text in the source file that contains the first data to import. Default is 1, the first row. Use this option if the text contains unnecessary introduction or header information.</td>
</tr>
<tr>
<td>Disable lookup table after this number of consecutive errors</td>
<td>The number of consecutive errors that is received from the system before it disables the lookup table. Once disabled, a lookup table must be re-enabled manually. The default is 5.</td>
</tr>
<tr>
<td>Use this row for Column Names</td>
<td>When using a sample file for fixed-width and delimited files, this option identifies a row in the sample that includes the column names. These names identify each column on the Field Information tab. When this option is not selected, the default names are Field1, Field2, and so on.</td>
</tr>
<tr>
<td>Skip rows that use the following regular expression</td>
<td>For fixed-width and delimited files, this option identifies the rows to ignore in the source file before importing data stream data. For example, if the file contains a row that contains ???, the source might actually start on the third row of the file. In such a case, you would specify ??? and any rows with that regular expression would be skipped.</td>
</tr>
<tr>
<td>Allow short rows</td>
<td>For delimited files only. For information about delimited files, see “Delimited files” on page 99.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>For delimited files only. For information about delimited files, see “Delimited files” on page 99.</td>
</tr>
<tr>
<td>Escape character</td>
<td>For delimited files only. For information about delimited files, see “Delimited files” on page 99.</td>
</tr>
<tr>
<td>Text Qualifier</td>
<td>For delimited files only. For information about delimited files, see “Delimited files” on page 99.</td>
</tr>
<tr>
<td>Number formats</td>
<td>Number formatting specifications. Default is comma (,) thousands separator and dot (.) decimal separator.</td>
</tr>
</tbody>
</table>
The Information field tab

For fixed-width and delimited files, use this tab to specify details about each column in the table, including the name, data type, and formatting applicable to the type.

Table 46. Attributes for table columns

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Name</td>
<td>Use this field to change the column name from the default names (Field1, Field2, and so on) or when you want to correct or change the field name imported from a sample file.</td>
</tr>
<tr>
<td>Index</td>
<td>When selected, an index for the field is built. It is critical that you select the right index in order to have good performance when the JDBC store is enabled for data caching. Select Index for those columns you use in join conditions.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Specifies the data type for the column. The data type for the selected column must match the data type in the other column used for the join.</td>
</tr>
<tr>
<td>Formatting</td>
<td>Specifies the formatting for the data type. Formatting is only applicable to the DECIMAL, TIMESTAMP, and VARCHAR data types. To change the format, select &lt;Change Formatting&gt; from the drop-down list.</td>
</tr>
</tbody>
</table>

The Data Caching tab

You can use this tab to specify the options for data caching.

The settings for data caching include Cache type, Fetch and store on restart, Fallback to external source, Number of result sets to cache, and Invalidation Schedule.

Cache type

You can use either in-memory or JDBC store for data caching.

- In-Memory
  When you enable in-memory caching, the entire external lookup table is cached into memory from the external data source and becomes a replacement for the external query source. Enabling in-memory caching is the same as selecting prefetch caching for lookup table sources from JDBC and web service agents as described under “Prefetch caching” on page 251.
  If you select this option, JDBC Store is not available, and the JDBC Agent field is disabled.

- JDBC Store
  When you select JDBC Store, the lookup table data is pushed out to a local database as specified by the JDBC Agent.
  The JDBC Agent drop-down list shows the available agents. Select the agent to use for the local database.

Fetch and store on restart

This option specifies that when IBM Cognos Real-time Monitoring is restarted, the system always retrieves data from the external query source then store it in the local database. Otherwise, if there is a local store that
was previously populated, that store is used until the next invalidation schedule. If there was no local store that was previously populated, the external source is queried immediately upon restart.

**Fallback to external source**
This option specifies that if the system cannot access the local database, it uses the data from the external database. This option is not available while configuring a lookup table that is based on a file system agent.

**Number of result sets to cache**
This option specifies the count of result sets to cache in-memory. Each set of results can contain one or more rows of lookup table data related to the data stream. For example, if three queries are made and each result set contains five rows, 15 rows would be stored in the cache.

**Invalidation Schedule**
Identifies when to invalidate the cache and discard all information currently in the cache, for example 6:00 a.m. every Monday.

**Clearing the cache**
You can add a schedule to indicate when to clear the cache.

**Procedure**
1. On the Data Caching tab, click Add Schedule.
2. Select the time or times when you want the cache to be cleared.

**Creating a lookup table from a flat file**
You can use a flat file as a source type for a lookup table.

**About this task**
The creation of a lookup table from a flat file is similar to the procedure for creating data stream sources from flat files. For details about the source type, see:
- "Delimited files" on page 99
- "Fixed-width files" on page 100

**Procedure**
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab, and then click the Activities button.
2. Select Create New then Lookup Table.
3. Select File System as the lookup table source.
4. (Optional for fixed-width and delimited files.) Identify a sample file to assist in mapping the columns. This file is a sample of the real data file. Data from this file appears in the next step to assist you as you map the lookup table data into the table.
5. For fixed-width files, define the positions of the data columns with the Set Field Widths window.
6. Define the format-specific field information: delimited or fixed-width.
7. Click the Data Caching tab.
8. Select the Cache Type.
   - Enabling In Memory as the type of data caching to use is the same as selecting Enable prefetch for JDBC and Web service sources. See "Prefetch caching" on page 251 for information about prefetch caching.
• Enabling **JDBC store** as the type of data caching specifies that the fetch-and-store mechanism is to be used for data caching. For more information about fetch-and-store, see “Java Database Connectivity store caching” on page 252.

9. Save the file source as enabled.
Chapter 13. Flat files

A flat file is a text file that contains the information about one or more events or lookup table data.

With flat files for data streams, each line in the file is usually one event record (one row in the data stream table) and the data in the row maps into the columns in the data stream table. The rows can be formatted as fixed width, delimited, or XML files. Fixed-width and delimited files can contain multiple rows for the same events. For more information, see "Multi-row events" on page 97.

For flat file data for lookup tables, each line in the file is a row (or one record). The data in the row maps into the columns in the lookup table. The rows can be formatted as fixed width, delimited, or XML files.

Data from flat files

You can use data from flat files to load event data from a text file into a flat file data stream table. This section describes how data from flat files work and how to create a data stream table from a flat file source.

How data from flat files works

Events from flat files are loaded in batch mode into the data stream table for flat files, although events are processed individually as they are loaded into the table.

There are two ways to load flat files into data stream tables:

- Automatically
  The flat file agent (see "Flat file agents" on page 25) periodically looks to see whether the associated file exists. When the file is found, the agent retrieves it and passes it to the data stream object for data stream processing.

- Manually
  The Upload Data File button on the detail page for loading the flat file into the data stream table loads a manually selected file when chosen. For details, see the IBM Cognos Real-time Monitoring Workbench User Guide.
Data stream tables from flat files

Data stream tables for flat files receive data files from flat file agents or by uploading a data file in IBM Cognos Real-time Monitoring Workbench.

The three flat file formats are described as follows:

- “Delimited files” on page 99.
- “Fixed-width files” on page 100.
- “XML file support” on page 100.

Before creating data from a flat file for a data stream table, you need the following prerequisites:

- Permissions
  Create permissions (see Chapter 23, “Permissions,” on page 275) for tables (see “Granting permission to create objects” on page 278), and Read Only access permission on the agent that will feed the table.

- Sample Fixed-width and delimited files (Optional)
  A sample file that contains data in the format of the actual events. Use the sample when you create the data stream object to ensure that the fields map correctly into the data stream table. This is optional. You can create the data stream object without a source, but having it greatly assists you with the creation of a data stream object.

- The schema of the XML files to load
  Specifically, you need to know the names of the attributes that contain the column information for the event column, and the XML path to the element that contains the columns for each event. See “XPath location of columns and rows in XML files” on page 101 for details. The following table describes the attributes.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the data stream object.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which the data stream object is to be stored. Use the Choose Folder button to select or create a folder. If you do not specify a folder, the default is Public Folder.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Process data in the order of arrival</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn this off. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
<tr>
<td>File agent</td>
<td>An existing file agent that retrieves events and lookup table data from a text file. Create an agent with the Workbench tab. For more information, see “Flat file agents” on page 25.</td>
</tr>
<tr>
<td>Start import on row</td>
<td>For fixed-width and delimited files, identifies the row of the text in the source file that contains the first data to import. Default is 1, the first row. Use this option if the text contains unnecessary introduction or header information.</td>
</tr>
<tr>
<td>Disable data stream after this number of consecutive errors</td>
<td>Disables the data stream when a consecutive count of errors occur. For example, if set to 5, disables the data stream after 5 consecutive errors. However, if 4 errors occur, and then no errors occur followed by 2 errors, the data stream remains enabled. The default is off: Do not disable.</td>
</tr>
<tr>
<td>Use this row for column names</td>
<td>When using a sample for fixed-width and delimited files, this option identifies a row in the sample that includes the column names. These names identify each column in the Column Information details. When this option is not specified, the default names are Field1, Field2, and so on.</td>
</tr>
<tr>
<td>Skip rows</td>
<td>For fixed-width and delimited files identifies the rows to ignore in the source file before importing event data. For example, if the file contains some title and header information, the source might actually start on the third row of the file. In such a case, you would specify 2 as the count of rows to skip.</td>
</tr>
</tbody>
</table>
Table 47. Column information attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow short rows</td>
<td>For delimited files only. See “Delimited files” on page 99 for details.</td>
</tr>
<tr>
<td>Treat all rows in the result set as a single event</td>
<td>All rows returned in the result set are considered an event. Otherwise, every row returned from the table is considered a separate event.</td>
</tr>
<tr>
<td>Delimiter</td>
<td>For delimited files only. For more information about delimited files, see “Delimited files” on page 99.</td>
</tr>
<tr>
<td>Escape character</td>
<td>For delimited files only. For more information about delimited files, see “Delimited files” on page 99.</td>
</tr>
<tr>
<td>Text qualifier</td>
<td>For delimited files only. For more information about delimited files, see “Delimited files” on page 99.</td>
</tr>
<tr>
<td>Number formats</td>
<td>Number formatting specifications. Default is comma (,) thousands separator and dot (.) decimal separator.</td>
</tr>
<tr>
<td>Column Information</td>
<td>For delimited and fixed-width files, Column Information specifies details about each column in the table, including the name, data type, and formatting applicable to the type. For more information, see “Delimited files” on page 99 and “Fixed-width files” on page 100. For XML files, the Column Information identifies the source elements and how they map into the columns of the data stream table. For more information, see “XML field information” on page 101.</td>
</tr>
<tr>
<td>Event Key</td>
<td>For fixed-width and delimited files, identifies key field columns for multi-row events. For more information about multi-row events, see “Multi-row events” on page 97.</td>
</tr>
<tr>
<td>Clear State Interval</td>
<td>This tab contains several options for clearing persisted event data. For more information about the Clear State Interval tab, see “Clear state interval” on page 54.</td>
</tr>
</tbody>
</table>

Creating a data stream from a flat file

A data stream is an object that represents an event. You can use a flat file to create a data stream.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab, and click the Activities button.
2. Select Create New then Data Stream.
3. Select Flat File as the source type.
4. Select Single Data Stream.
5. Optionally for fixed-width and delimited files, identify a sample file to assist in mapping the columns. This file is a sample of the real data file. Data from this file appears in the next step to assist you as you map the event data into the table.
6. For fixed-width files, define the positions of the data columns with the Set Field Widths window.
7. Identify the data stream attributes.
8. On the **Column Information** tab, define the format-specific column information. For details about the source type, see:
   - “Delimited files” on page 99.
   - “Fixed-width files” on page 100.
   - “XML file support” on page 100.

9. Save the data stream as enabled if you want the data stream to accept data immediately.

**Multi-row events**

A fixed-width or delimited file can contain multiple rows for the same event.

For example, a purchase order event might contain one row for each line-item in the order. When this event is loaded into the data stream table, each row is treated as part of the same event. The system does not treat each row as a new event.

To identify the rows as containing data for the same event, each row must have some identifying data that is unique to the event. For example, the following sample data contains line items for three purchase orders where each order is identified by the POID column. The first order has three items, the second has one, and the third has two.

```
POID,ITEM_NO,ITEM_NAME,ITEM_QTY,ITEM_COST,ITEM_TOTAL
0697,1,Smoke Shifter,100,5.00,500.00
0697,2,Nano Webber,50,6.00,300.00
0697,3,Locking Rail Key,25,7.50,187.50
0698,1,Nano Webber,50,6.00,300.00
0699,1,Tall Bar Stool,100,60.00,6000.00
0699,2,Can of Levers,250,1.50,375.00
```

When defining the column information for this event, you identify the POID column as the key field by selecting **Event Key**. Each event can have one or more event key fields.

In the file, the rows for each event must appear together, and the data in the event key fields must be unique to the event. As soon as the data in one of the fields is not the same as the previous row, the row for that field is a new event. For example, the following sample is treated as three separate events, even though the last row has the same event key value as the first two rows.

```
POID,ITEM_NO,ITEM_NAME,ITEM_QTY,ITEM_COST,ITEM_TOTAL
0697,1,Smoke Shifter,100,5.00,500.00
0697,2,Nano Webber,50,6.00,300.00
0100,1,Tall Bar Stool,100,60.00,6000.00
0697,3,Locking Rail Key,25,7.50,187.50
```

If any row contains invalid data, that row is discarded and does not affect subsequent rows. For example, the third row in the following sample contains a character ('X') where an integer is expected. In this sample, the third row is discarded, and the fourth is included as the third row in the event.

```
POID,ITEM_NO,ITEM_NAME,ITEM_QTY,ITEM_COST,ITEM_TOTAL
0697,1,Smoke Shifter,100,5.00,500.00
0697,2,Nano Webber,50,6.00,300.00
0697,X,Tall Bar Stool,100,60.00,6000.00
0697,4,Locking Rail Key,25,7.50,187.50
```
**Time zones in data from flat files**

For data from delimited and fixed flat files, you can specify a date format for a time stamp field.

In addition to the date format, you can also specify a time zone to specify how the server interprets the time stamp value in the field. That is, the time stamp can be assumed to be in the same time zone as the application server or another specific time zone. This is useful when the application server is in a different time zone than the one in which events occur, and you want the time stamp to display in a view for that time zone. The default is the time zone in which the server is located.

The following table shows examples of a time stamp with different formats and time zones. When the time zone is set to GMT+5:30 and GMT+13:00, the View result for the time stamp shows the date as 2006-10-25 instead of the field value.

<table>
<thead>
<tr>
<th>Field value in flat file</th>
<th>Time stamp format</th>
<th>View result</th>
</tr>
</thead>
</table>

**Setting a custom date format and time zone**

Data from delimited and fixed flat files can be set with a custom date format and time zone to specify how the server interprets the time stamp value in the field.

**Procedure**

1. Click **Edit This Data Stream** for the data stream.
2. On the **Column Information** tab, find the field with a TIMESTAMP data type and select **Change Formatting** under **Formatting**.
3. Select a format from **Date/Time Format** or create a custom format.
4. Select a time zone from **Time Zone**.
5. Click **OK**.

The time stamp format and time zone displays in the **Formatting** column of your time stamp field.

**Creating a custom time format**

Data from delimited and fixed flat files can be set with a custom time format to specify how the server interprets the time stamp value in the field.

**Procedure**

1. Open the Custom Date Format window.
2. Select **Custom Format** from **Date/Time Format**.
3. Build your custom format in the **Custom Format** field by entering date-time component letters.
Use the list displayed in the date format list for the available definitions. The Sample field displays a sample of your format as you build it. For example, if you specify a custom format of \texttt{E MMM/dd/yyy G h:m:s.S a 'UTC'}, the sample field displays:

\texttt{Fri June/14/2010 AD 10:49:6.144 AM UTC}

4. Click \textbf{OK}.

\section*{Delimited files}

In a delimited file, each field (column) is separated by a character, such as a comma.

For example:

0703,00001,Assigned,13,2003-03-05 14:23:00, Sridar
0706,00004,Open,13,2003-03-05 19:50:00,
0706,00004,Resolved,13,2003-03-05 19:50:00, Niku

Delimited files are also called comma-separated value (CSV) files.

The field separator character, escape character, and text qualifier are each customizable.

- **Separator character**
  Separates each field in the row, usually a comma character (,).

- **Escape character**
  Precedes characters that are not to be used as a separator, usually a backslash character (\). For example, if the separator character is a comma, and the text contains a real comma, then the real comma is prevented from being used as a separator with a preceding backslash. For example, the comma after Altadena is not a field separator:

  123 Buena Loma Dr,Altadena\, CA,91001

- **Text qualifier**
  Text strings are further bounded by this character, usually a double quote ("). Use this option when text strings are qualified to be different from other data types. For example, this event has text fields that contain numerals, but one of the fields (data value 13) is numeric.

  "0706","00004","Open",13,2010-03-05 19:50:00,""

Source rows that do not contain enough data to fill the row generate an error. To permit the data without generating an error, turn on \textbf{Allow Short Rows}. For example, this text generates an error when the third row is imported unless short rows are allowed:

1,2,3,4,5,6
1, ....
1

When the \textbf{Use this row for Column Names} option is selected, the names in that row appear as the column names. Otherwise, assign the names manually. Additionally, for each column, assign a data type and optionally declare a data format.
Fixed-width files

In fixed-width files, each field (column) has the same predefined width in each file row, similar to a spreadsheet table.

For example:
070300001Assigned13 2010-03-05 14:23:00Sridar
070600004open 13 2010-03-05 19:50:00
070600004Resolved13 2010-03-05 19:50:00Niku

To import a fixed-width file, you need to use the Set Field Widths box to identify the column positions that begin each field of data. When you provide a sample, the sample data is shown, and you click the columns to indicate the start of a field.

If you do not have a sample, you need to identify the starting position of each column in the text. Remember that the first field starts at position zero (0).

Once the column positions are defined, you can assign names and declare their data types and formats.

When the Use this row for Column Names option is selected, the names in that row appear as the column names. Otherwise, assign the names manually. Additionally, for each column, assign a data type and optionally declare a data format.

XML file support

IBM Cognos Real-time Monitoring supports the generation of an event from data sources in XML format.

However, Cognos Real-time Monitoring views and data stream tables are in table format (that is, rows and columns) while XML data is hierarchical. This requires the flattening of the XML file in a data stream table, where the hierarchical data is mapped to tabular data. The process for flattening XML is described in the section “How XML files are flattened” on page 103.

To define an XML-based data stream object in Cognos Real-time Monitoring, an application designer must specify the columns that will be output to the data stream table. Each column in the data stream table has a name, a type, and a source. Each column has a source specified as an XPath expression, meaning that the column will be filled by finding elements that match the given XPath, and taking the value of the element in the XML input to use as the value of the column in an output row. Each column XPath should either specify an element or an attribute. Usually, a column XPath that specifies an element matches only atomic elements in the input document; that is, elements that contain only text and no other elements. For more information, see “How elements are evaluated” on page 111.

The mapping of the hierarchical XML data to a table is accomplished by having the source for each column in the table specified as an XPath expression. The columns in the table are filled by finding elements that match the XPath, taking the value of the elements in the XML input and using the value in the column of an output row. Each column XPath specifies an atomic element or an attribute. For more information about how XPaths are used to locate rows and columns, see, “XPath location of columns and rows in XML files” on page 103.
XPath location of columns and rows in XML files

You can use XPaths to locate the columns and rows in the XML file.

When defining the Field Information for a data stream object, XPaths locate the columns and rows in the XML file as follows:

- Schema XPath identifies the absolute XPath of the element of the document under which all other XML data defined by the relative XPaths reside. This can be a slash (/) referring to the root node of the XML document, or it can be a more complex XPath. If an XML document contains multiple elements that satisfy the schema XPath, a table is generated for each such element and the output of the document is defined as the union of the output of the elements that match the schema XPath. A schema XPath is an absolute path to the element in the XML structure, and always begins with a slash (/) followed by the root element and path to the column element; for example, "/problem_tickets/ticket/ticket_id" is such a schema XPath.

- Relative XPath identifies a column element or attribute relative to the row element. When the column is a child element of the row element, the XPath is either just the element name, or it begins with "child:". For example, these are valid Relative XPaths from the previous example:
  - ticket_id
  - child::status
  - customer/customer_name
  - child::customer/customer_name

To locate an attribute, put an at symbol (@) before the attribute name:

- customer/@cust_id

The XPath standard for locations defines additional XML node mappings not supported by IBM Cognos Real-time Monitoring data stream tables.

XML field information

Each column in the data stream table is defined as a field in the XML data stream editor.

Each field has the attributes shown in the following table:

Table 49. XML field attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema XPath</td>
<td>The schema XPath expression selects a node list where each node encapsulates a row in the resulting data set. For more information about schema XPaths, see XPath location of columns and rows in XML files.</td>
</tr>
<tr>
<td>Field Name</td>
<td>Name of the column in the data stream table.</td>
</tr>
<tr>
<td>Relative XPath</td>
<td>Element in the XML file that contains the data for this field. For more information about XPaths, see XPath location of columns and rows in XML files.</td>
</tr>
<tr>
<td>XML Data Type</td>
<td>Data type of the XML element. For information about XML data types, see &quot;XML data types&quot; on page 102.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Data Type</td>
<td>IBM Cognos Real-time Monitoring data type of the column in the data stream table. For more information, see Chapter 9, “Data types,” on page 63.</td>
</tr>
<tr>
<td>Formatting</td>
<td>Formatting of the decimal, string, or date-time value.</td>
</tr>
</tbody>
</table>

### XML data types

The XML data types map to IBM Cognos Real-time Monitoring data types.

The following table shows the mapping.

<table>
<thead>
<tr>
<th>XSD</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyURI</td>
<td>Varchar</td>
</tr>
<tr>
<td>base64Binary</td>
<td>Varchar</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>byte</td>
<td>Integer</td>
</tr>
<tr>
<td>date</td>
<td>Time stamp (time portion zeroed out)</td>
</tr>
<tr>
<td>dateTime</td>
<td>Time stamp (the fractional part of a second is supported up to 9 significant digits)</td>
</tr>
<tr>
<td>decimal</td>
<td>Decimal if scale is not equal to 0. If scale equals 0, the data type is determined by precision as follows: Integer if precision is from 1 to 9. Long if precision is from 10 to 18. Decimal if precision is greater than 18.</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>duration</td>
<td>Varchar (as a string)</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>Varchar</td>
</tr>
<tr>
<td>ENTITY</td>
<td>Varchar</td>
</tr>
<tr>
<td>float</td>
<td>Double</td>
</tr>
<tr>
<td>gDay</td>
<td>Varchar (as a string) defines the day (DD)</td>
</tr>
<tr>
<td>gMonth</td>
<td>Varchar (as a string) defines the month (MM) portion of the date</td>
</tr>
<tr>
<td>gMonthDay</td>
<td>Varchar (as a string) defines the month and day (MM-DD) portion of the date</td>
</tr>
<tr>
<td>gYear</td>
<td>Varchar (as a string) defines the year (CCYY) portion of the date</td>
</tr>
<tr>
<td>gYearMonth</td>
<td>Varchar (as a string) defines the year and month (CCYY-MM) portion of the date</td>
</tr>
<tr>
<td>hexBinary</td>
<td>Varchar</td>
</tr>
<tr>
<td>ID</td>
<td>Varchar</td>
</tr>
<tr>
<td>IDREF</td>
<td>Varchar</td>
</tr>
</tbody>
</table>
Table 50. Mapping table (continued)

<table>
<thead>
<tr>
<th>XSD</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDREFS</td>
<td>Varchar</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>integer</td>
<td>Integer</td>
</tr>
<tr>
<td>language</td>
<td>Varchar</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>Name</td>
<td>Varchar</td>
</tr>
<tr>
<td>NCName</td>
<td>Varchar</td>
</tr>
<tr>
<td>negativeInteger</td>
<td>Integer</td>
</tr>
<tr>
<td>NM_TOKEN</td>
<td>Varchar</td>
</tr>
<tr>
<td>NM_TOKENS</td>
<td>Varchar (as a single string)</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td>Integer</td>
</tr>
<tr>
<td>nonPositiveInteger</td>
<td>Integer</td>
</tr>
<tr>
<td>normalizedString</td>
<td>Varchar</td>
</tr>
<tr>
<td>NOTATION</td>
<td>Varchar</td>
</tr>
<tr>
<td>positiveInteger</td>
<td>Integer</td>
</tr>
<tr>
<td>QName</td>
<td>Varchar</td>
</tr>
<tr>
<td>QName</td>
<td>Varchar</td>
</tr>
<tr>
<td>short</td>
<td>Integer</td>
</tr>
<tr>
<td>string</td>
<td>Varchar</td>
</tr>
<tr>
<td>time</td>
<td>Varchar</td>
</tr>
<tr>
<td>token</td>
<td>Varchar</td>
</tr>
<tr>
<td>unsignedByte</td>
<td>Integer</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>Decimal</td>
</tr>
<tr>
<td>unsignedLong</td>
<td>Decimal</td>
</tr>
<tr>
<td>unsignedShort</td>
<td>Integer</td>
</tr>
</tbody>
</table>

How XML files are flattened

The mapping of the hierarchical XML data to a table is accomplished by having the source for each column in the table specified as an XPath expression.

The examples in this section always give the absolute XPath for a column. However, when configuring an XML data stream object, you use an absolute schema XPath and column events are XPaths that are relative to that schema XPath.

The columns in the table are filled by finding elements that match the given XPath, taking the value of the elements in the XML input, and using the value in the column of an output row. For example, the following XML document has the XPaths: a/b, a/c, a/d:

```xml
<a>1
  <b>0</b>
  <c>1</c>
  <d>2</d>
</a>
```
The resulting table has one row:

0, 1, 2

In this next example, the document is slightly more complicated and has the XPaths: a/b, a/c/d a/c/e/f.

The resulting table also has one row:

0,1,2

In these examples, both XML documents result in a table with one row because each XPath expression matches exactly one element. The resulting table has one row, where each column is filled by the value of the element that matches the XPath that defines the source for a column.

In some cases, more than one element can match an XPath expression for a column source. In the following example, the XPaths are a/b and a/c, which match more than one column.

It looks like the resulting table should be:

0,2
0,3

However, the resulting table is really:

0,2
0,3
1,2
1,3

All four elements are children of a single parent element <a> and the source for the columns in each row is specified by the XPaths a/b and a/c. This means that 0,2 is no more correct than 0,3 and both rows should be in the table. In this case, the resulting table is created from a cross product of the elements.
To ensure that when XML is flattened the correct result is produced for all cases, IBM Cognos Real-time Monitoring performs the following tasks:

- Strips the input XML of all extraneous elements so that values are not duplicated where such duplication is not necessary, which would result in duplicate rows in the output.
- Rationalizes the document that is stripped of extraneous elements.

To accomplish the last task, the document is distributed into a number of intermediate tables, where each table has a primary key (pk), parent foreign key (parent_fk), and a value. The final result table is created through a key join of all these intermediate tables. For example, the following XML document can be rationalized into eight intermediate tables a, a/b, a/b/c, a/b/d, a/b/d/e, a/b/d/f, a/b/d/f/g, and a/h.

```xml
<a>
  <b>
    <c>0</c>
    <d>
      <e>1</e>
      <e>2</e>
      <f>
        <g>3</g>
      </f>
    </d>
  </b>
  <b>
    <c>4</c>
    <d>
      <e>5</e>
      <f>
        <g>6</g>
        <g>7</g>
      </f>
    </d>
  </b>
  <h>8</h>
  <h>9</h>
</a>
```

Next, primary keys (pk) are assigned to each element in a depth-first traversal of the document.

```xml
<a pk='0'>
  <b pk='1'>
    <c pk='2'>0</c>
    <d pk='3'>
      <e pk='4'>1</e>
      <e pk='5'>2</e>
      <f pk='6'>
        <g pk='7'>3</g>
      </f>
    </d>
  </b>
</a>
```
Now the tables are key joined using the primary keys assigned to each element. The following SELECT statement defines the final output table for the data stream table:

```sql
SELECT a/b/c.value, a/b/d/e.value, a/b/d/f/g.value, a/h.value
FROM (((((((a/
  FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
  FULL OUTER JOIN /a/b/c on /a/b/c.parent_fk = /a/b.pk)
  FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk)
  FULL OUTER JOIN /a/b/d/e on /a/b/d/e.parent_fk = /a/b/d.pk)
  FULL OUTER JOIN /a/b/d/f on /a/b/d/f.parent_fk = /a/b/d.pk)
  FULL OUTER JOIN /a/b/d/f/g on /a/b/d/f/g.parent_fk = /a/b/d/f.pk)
  FULL OUTER JOIN /a/h on /a/h.parent_fk = a.pk
FROM (((((/a
  FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
  FULL OUTER JOIN /a/b/c on /a/b/c.parent_fk = /a/b.pk)
  FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk)
  FULL OUTER JOIN /a/b/d/e on /a/b/d/e.parent_fk = /a/b/d.pk)
  FULL OUTER JOIN /a/b/d/f on /a/b/d/f.parent_fk = /a/b/d.pk)
  FULL OUTER JOIN /a/b/d/f/g on /a/b/d/f/g.parent_fk = /a/b/d/f.pk)
  FULL OUTER JOIN /a/h on /a/h.parent_fk = a.pk
```

The resulting table is:

```
0,1,3,8
0,2,3,8
4,5,6,8
4,5,7,8
```
How missing elements are handled
In cases where elements do not have the same child elements, a full outer join is used to generate the table because defining the output with an inner join would not produce the results that you want.

For example, the following document does not have a <d> child element in the first <b> element, and the second <b> element has a <d> element but no <c> element:

```xml
<a>
  <b>
    <c>0</c>
  </b>
  <b>
    <d>1</d>
  </b>
</a>
```

The annotated document is:

```xml
<a pk='0'>
  <b pk='1'>
    <c pk='2'>0</c>
  </b>
  <b>
    <d pk='3'>1</d>
  </b>
</a>
```

The SELECT statement is:

```sql
SELECT /a/b/c.value, /a/b/d.value
FROM ((/a
    FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
    FULL OUTER JOIN /a/b/c on /a/b/c.parent_fk = /a/b.pk)
    FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk
```

The SELECT statement with the primary key projection is:

```sql
SELECT /a.pk, /a/b.pk, /a/b/c.pk, /a/b/d.pk
FROM ((/a
    FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
    FULL OUTER JOIN /a/b/c on /a/b/c.parent_fk = /a/b.pk)
    FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk
```

The final result is:

```
0,null
null,1
```
How attributes are handled
When elements in a document contain attributes, the attributes are represented as
columns in the intermediate table.

For example, in the following document, the column XPaths are: a/@b, a/c/@d,
and a/c.

```
<a b='0'>
  <c d='1'>2</c>
  <c d='3'>4</c>
</a>
```

The primary keys are assigned to the elements as follows:

```
<a pk='0' b='0'>
  <c pk='1' d='1'>2</c>
  <c pk='2' d='3'>4</c>
</a>
```

The select statement is:

```
SELECT /a/@b, /a/c.d, /a/c.value
FROM /a FULL OUTER JOIN /a/c on /a/c.parent_fk = /a.pk
```

The select statement with primary keys is:

```
SELECT /a/@b, /a/c.d, /a/c.pk
FROM /a FULL OUTER JOIN /a/c on /a/c.parent_fk = /a.pk
```

The resulting table is:

```
0,1,2
0,3,4
```

When an XML element has two children with the same tag name, a simple
unpredicated XPath expression returns a set of nodes. For example, in the
following document the XPath expression /a/b returns two rows:

```
<a>
  <b>0</b>
  <b>1</b>
</a>
```

The rows returned are:

```
0
1
```

In XPath expressions, you can use index predicates to find a node or a specific
value in a node. For example, the expression /a/b[1] returns 0, and the expression
/a/b[2] returns 1. That is, each XPath returns one row. The expression /a/b[3]
returns nothing because there is no third child in the element <a>. In the following
element, the XPath /a/b/c[1] returns two rows.

```
<a>
  <b>
    <c>0</c>
    <c>1</c>
  </b>
</a>
```
The rows returned are:
0
2

This is because the XPath /a/b/c[1] specifies the first child in the element <b>, and there are two <b> child elements in the parent element <a>.

**Ignored elements**
Elements in a document that do not match any column XPath or do not have any children that match a column XPath are ignored.

For example, given the following document and the XPath /a/b/c, only a single row is returned.

```
<a>
  <b>
    <c>0</c>
    <d>
      <e>1</e>
      <e>2</e>
    </d>
  </b>
</a>
```

Only one row is returned because the /a/b/d element does not match the column XPath /a/b/c, nor does it have any children that match. Therefore, after the first pass the document becomes:

```
<a>
  <b>
    <c>0</c>
  </b>
</a>
```

The following document also produces one row.

```
<a>
  <b>
    <c>0</c>
  </b>
  <b>
    <d>1</d>
  </b>
</a>
```

The previous document produces only one row because the second /ab element matches the column XPath. However it does not have any children that match. Therefore, this document also becomes the following after the first pass:

```
<a>
  <b>
    <c>0</c>
  </b>
</a>
```
The same process also occurs with attributes. Given the XPath /a/b/@id, the following document produces only one row after the first pass because the second <b> element does not have an id attribute.

Thus after the first pass, the document becomes

Index predicates

Index predicates make it possible to overcome a particular problem where one row needs to be returned but the XML code contains two children tags with the same name that contain different values.

For example, the following document contains two occurrences of the tag <keyword>:

For the schema XPath /sales, the column XPaths are:

This returns two rows:

However, there was only one sale, so only one row should have been returned: 0, $125, Salesforce. That is, you wanted to create a view on top of the event SELECT SUM(amount) FROM sales, and you only needed one keyword. In this case, you could provide the following XPaths:
This would return the row: 0, $124, Salesforce. However, if you need both keywords, you would need to use a different method than using index predicates.

Index predicates can occur at any level within an XPath expression, not just the last level. For example, the expression /a/b[2]/c returns two rows given the following document:

```xml
<a>
  <b>
    <c>0</c>
    <c>1</c>
  </b>
  <b>
    <c>2</c>
    <c>3</c>
  </b>
</a>
```

The two rows returned are:

2
3

**How elements are evaluated**

In certain cases, elements return a value of null when evaluated by IBM Cognos Real-time Monitoring.

These cases are as follows:

- The element is empty. For example, there is one /a/b element in the following document, which Cognos Real-time Monitoring defines as null:

  ```xml
  <a>
    <b/>
  </a>
  ```

- The element has children but no text. For example, the following document contains an element <c>, which has a child element <d> but does not contain any text.

  ```xml
  <a>
    <b>0</b>
    <c>
      <d>1</d>
      <d>2</d>
    </c>
  </a>
  ```

  The output from this document given the XPaths /a/b, a/b/c, and /a/b/c/d would be:

  0, null, 1
  0, 2, null

  If an element is not an atomic element and contains a child element, it returns the text from the element as well as the child. For example, the following document contains the element <b>, which is not atomic, and a child element <c>:

  ```xml
  <a>
    <b>hello
    <c>
  ```
The column XPaths /a/b and /a/b/c would return the following rows:

hello, com, world

**Issues when flattening XML files**

There are issues to be aware of when flattening XML files.

It is illegal to mix predication. For example, the set of XPaths /a/b[2], /a/b are illegal. This is because the semantics that IBM Cognos Real-time Monitoring uses to flatten XML state that as soon as index predicates are used with a given XPath prefix, all elements that match that prefix are implicitly rewritten by appending the index. Each element must be either written with the index predicate or without it. By writing a given XPath prefix in one column with an index predicate and in another column without an index predicate, all elements corresponding to that prefix become ambiguous.

With data from a flat file, every match to the schema XPath defines a new event, which can define any number of rows. For example, an event from a flat file that has the schema XPath /a/b and the column XPath c generates two events from the following document, both with one row.

```
<a>
  <b>
    <c>0</c>
  </b>
  <b>
    <c>1</c>
  </b>
</a>
```

However, for an XML document that comes through JMS, one document always generates one event. (This rarely makes a difference.)

White space is illegal in an XPath expression. You cannot save an event with white space in the XPaths. The following XPaths are illegal:

- b/ c
- b/c [1]

If you specify more than one column with the same XPath, you always get the same value in both columns. For example, the following document returns the result 0,1,1 given the schema /a and the columns b, c, c.

```
<a>
  <b>0</b>
  <c>1</c>
</a>
```

The semantic query is:

```sql
SELECT /a/b.value, /a/c.value, /a/c.value
FROM
  (/a FULL OUTER JOIN /a/b ON /a/b.parent_fk = /a.pk)
```
FULL OUTER JOIN /a/c ON /a/c.parent_fk = /a.pk
Chapter 14. Formulas

All formulas in IBM Cognos Real-time Monitoring are expressions in the C-SQL language, a derivative of ANSI SQL.

Some of the formulas are simple expressions, such as field expressions that define the values in business view columns. Other expressions are more complex and represent entire components of the C-SQL query statement (SELECT), such as the WHERE, WINDOW, and JOIN clauses.

All formulas in IBM Cognos Real-time Monitoring accept operators and constants (see Chapter 22, “Operators and constants,” on page 271) that can manipulate the values, and they can accept most C-SQL functions (see Chapter 15, “C-SQL functions,” on page 123) to further process results.

For detailed descriptions of the C-SQL SELECT statement, operators, and functions, see Chapter 32, “SELECT statements,” on page 313.

Functions

Functions return values that are system information, such as the current time; manipulations of data, such as converting a string of characters to uppercase; or are evaluations of sets of data, such as the total of all prices in a set of purchase orders.

C-SQL functions can be used in most formulas in IBM Cognos Real-time Monitoring. However, some are limited by the operations allowed in the context of the formula. The next section “Function types”, describes the types of functions and tells where they are allowed.

If you are looking for a function for a specific task, see “Function categories” on page 118 to see what tasks the functions can perform.

For a detailed description of each function, see Chapter 15, “C-SQL functions,” on page 123.

Function types

C-SQL has five types of functions: scalar, set, rank, moving set, and tumbling set. The type distinctions determine where you are allowed to include the function in a formula.

Scalar functions

Scalar functions operate on a single item and provide a single result.

For example, the ABS (see “ABS” on page 140) function returns the absolute value of a (single) number. Scalar functions can appear in any C-SQL expression.

Scalar functions are described in the following sections:
- “ABS” on page 140
- “CAST” on page 131
Set functions

Set functions perform aggregations on sets of business view rows and produce a single result for the set.

For example, SUM() provides the total of all the rows in a column in a view. A set function can be used only in the select list of a SELECT statement: the field definitions of a view.

A set function can reference another set function, but the results are the same as if only the referenced (inner) function was expressed alone. For example SUM(AVG(Order_Total)) has the same result as AVG(Order_Total).

Set functions are described in the following sections:

- "CEIL" on page 141
- "CHAR_LENGTH" on page 155
- "CONCAT" on page 155
- "CURRENT_TIMESTAMP" on page 132
- "CURRENT_USER" on page 154
- "DATE_ADD" on page 133
- "DATE_DIFF" on page 134
- "DISPLAY_MONEY" on page 141
- "EXP" on page 143
- "FLOOR" on page 144
- "GREATEST" on page 135
- "IS_RAISED" on page 153
- "LAST_DAY" on page 135
- "LEAST" on page 136
- "LOG" on page 145
- "LOWER" on page 157
- "LPAD" on page 158
- "LTRIM" on page 159
- "MOD" on page 146
- "POSITION" on page 159
- "POWER" on page 147
- "PRIOR_VALUE" on page 136
- "ROUND" on page 149
- "RPAD" on page 161
- "RTRIM" on page 162
- "SIGN" on page 150
- "SQRT" on page 151
- "SUBSTRING" on page 162
- "TIMESTAMPDIFF" on page 137
- "TO_CHAR" on page 138
- "TO_DATE" on page 139
- "TRUNC" on page 151
NULL is ignored when you compute set function, moving set function, and rank function values. For example, the average of (3, NULL, 3) is 3, not NULL, and it is not 2.

**Rank functions**

Rank functions compute the scalar result for a column in each row in a set, with respect to the entire set.

A rank function can be used only in the select list of a SELECT statement.

Rank functions are described in the following sections:
- “NTILE” on page 184
- “RANK” on page 185
- “RATIO_TO_REPORT” on page 187

**Moving set functions**

Moving set functions perform calculations on a set of the latest rows in a view.

The set of rows to include is determined only when a new event arrives. At that time, only the latest rows that meet the set criteria are included in the calculation.

Moving set functions are defined by applying "MOV_" to an existing set function. For example, to calculate a moving average, use MOV_AVG(). A moving set can be determined by a count of events or as a duration of time. The following example calculates the mean average of Order_Total for the last 12 hours. As new orders are inserted into the view they are included in the calculation; however, orders older than 12 hours are excluded.

```
SELECT MOV_AVG(Order_Total, HOUR, 12) FROM Purchase_Orders
```

Moving set functions are a shorthand way to express a simple query window. See "MOV_function" on page 126 for a complete description.

Moving set functions are described in the following sections:
- “MOV_AVG” on page 168
- “MOV_COUNT” on page 169
- “MOV_MAX” on page 170
- “MOV_MIN” on page 171
- “MOV_SUM” on page 172
- “MOV_STD_DEVIATION” on page 182
- “MOV_VARIANCE” on page 183
Tumbling set functions

Tumbling set functions perform calculations on a windowed set of the rows in a view.

The set of rows to include is determined when a new event arrives, and the set empties when full. Tumbling set functions are a shorthand way to express a tumbling window query.

For more information, see “Tumbling windows” on page 375.

Tumbling set functions are described in the following sections:

- “TUMBLE_AVG” on page 175
- “TUMBLE_COUNT” on page 176
- “TUMBLE_MAX” on page 177
- “TUMBLE_MIN” on page 179
- “TUMBLE_SUM” on page 180
- “TUMBLE_STD_DEVIATION” on page 189
- “TUMBLE_VARIANCE” on page 191

Function categories

This section shows the categories of C-SQL functions.

Alerts function

- “IS_RAISED” on page 153

Conversion functions

- “CAST” on page 131
- “DISPLAY_MONEY” on page 141
- “TO_CHAR” on page 138
- “TO_DATE” on page 139

Date and time functions

- “CURRENT_TIMESTAMP” on page 132
- “DATE_ADD” on page 133
- “DATE_DIFF” on page 134
- “GREATEST” on page 135
- “TIMESTAMP_DIFF” on page 137
- “LEAST” on page 136
- “LAST_DAY” on page 135
- “TO_CHAR” on page 138
- “TO_DATE” on page 139

Math functions

- “ABS” on page 140
- “CAST” on page 131
- “CEIL” on page 141
- “EXP” on page 143
- “FLOOR” on page 144
Math functions
This section shows the C-SQL math functions.
• “ABS” on page 140
• “CAST” on page 131
• “CEIL” on page 141
• “EXP” on page 143
• “FLOOR” on page 144
• “LOG” on page 145
• “MOD” on page 146
• “POWER” on page 147
• “ROUND” on page 149
• “SIGN” on page 150
• “SQRT” on page 151
• “SUM” on page 174
• “TRUNC” on page 151

Ranking functions
This section shows the C-SQL ranking functions.
• “NTILE” on page 184
• “RANK” on page 185
• “RATIO_TO_REPORT” on page 187

Rules functions
This section shows the C-SQL rules functions.
• “IS_RAISED” on page 153
• “CURRENT_USER” on page 154

Statistical functions
This section shows the C-SQL statistical functions.
• “AVG” on page 164
• “COUNT” on page 165
• “GREATEST” on page 135
• “LEAST” on page 136
• “MAX” on page 167
• “MIN” on page 167
• “MOV_AVG” on page 168
• “MOV_COUNT” on page 169
Text and String functions
This section shows the C-SQL text and string functions.
- "CAST" on page 131
- "CHAR_LENGTH" on page 155
- "CONCAT" on page 155
- "DISPLAY_MONEY" on page 141
- "GREATEST" on page 135
- "LEAST" on page 136
- "LOWER" on page 157
- "LTRIM" on page 159
- "LPAD" on page 158
- "POSITION" on page 159
- "RPAD" on page 161
- "RTRIM" on page 162
- "SUBSTRING" on page 162
- "TO_CHAR" on page 138
- "TO_DATE" on page 139
- "UPPER" on page 164

Time-series and aggregation functions
This section shows the C-SQL time-series and aggregation functions.
- "AVG" on page 164
- "COUNT" on page 165
- "MIN" on page 167
- "MAX" on page 167
- "MOV_function" on page 126
- "MOV_AVG" on page 168
Views functions

This section shows the C-SQL Views functions.

- “CURRENT” on page 166
- “IS_RAISED” on page 153
- “PREV” on page 173
- “PRIOR_VALUE” on page 136
Chapter 15. C-SQL functions

These tables provide a preview of C-SQL functions. You can use the links for a full description.

C-SQL functions can appear in commands and rule formulas where an expression is accepted.

For a general discussion of functions, and a list of the function categories, see “Function categories” on page 118 and Chapter 14, “Formulas,” on page 115.

The following tables provide a brief description for each C-SQL function and a link to the section that provides more information.

Table 51. C-SQL functions A to I

<table>
<thead>
<tr>
<th>C-SQL function and link</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ABS” on page 140</td>
<td>Returns the absolute value of a number</td>
</tr>
<tr>
<td>“AVG” on page 164</td>
<td>Returns the average value (arithmetic mean) of a set of numeric values</td>
</tr>
<tr>
<td>“CASE” on page 131</td>
<td>Returns the result of an expression that corresponds to a matching true condition</td>
</tr>
<tr>
<td>“CAST” on page 131</td>
<td>Converts a value from one Cognos Real-time Monitoring type to another Cognos Real-time Monitoring type</td>
</tr>
<tr>
<td>“CEIL” on page 141</td>
<td>Returns the smallest integer, rounded up from zero, greater than or equal to a number</td>
</tr>
<tr>
<td>“CHAR_LENGTH” on page 155</td>
<td>Returns the length of a string</td>
</tr>
<tr>
<td>“CONCAT” on page 155</td>
<td>Returns a string that is the concatenation of 2 characters or strings</td>
</tr>
<tr>
<td>“concatList” on page 193</td>
<td>Returns a string that is the concatenation of a list of characters or strings</td>
</tr>
<tr>
<td>“concatSet” on page 194</td>
<td>Returns an alphabetically ordered set of strings</td>
</tr>
<tr>
<td>“COUNT” on page 165</td>
<td>Returns the count of rows in a view or set</td>
</tr>
<tr>
<td>“CURRENT” on page 166</td>
<td>Returns a value from the latest or last row in a set</td>
</tr>
<tr>
<td>“CURRENT_TIMESTAMP” on page 132</td>
<td>Returns the current date and time in the server time zone</td>
</tr>
<tr>
<td>“CURRENT_USER” on page 154</td>
<td>Returns the login name of the current user</td>
</tr>
<tr>
<td>“DATE_ADD” on page 133</td>
<td>Adds a duration of time to a date-time value</td>
</tr>
<tr>
<td>“DATE_DIFF” on page 134</td>
<td>Subtracts a duration from a date-time value</td>
</tr>
<tr>
<td>“DISPLAY_MONEY” on page 141</td>
<td>Formats a number as a currency value</td>
</tr>
<tr>
<td>“EXP” on page 143</td>
<td>Returns e raised to a specific power</td>
</tr>
<tr>
<td>“FLOOR” on page 144</td>
<td>Returns the largest integer less than or equal to an expression</td>
</tr>
<tr>
<td>“gammaDist” on page 195</td>
<td>Returns the gamma distribution of a value</td>
</tr>
</tbody>
</table>
### Table 51. C-SQL functions A to I (continued)

<table>
<thead>
<tr>
<th>C-SQL function and link</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“GREATEST” on page 135</td>
<td>Returns the greatest of a list of expression results</td>
</tr>
<tr>
<td>“IS_ACKED” on page 152</td>
<td>Returns true when the specified alert is acknowledged</td>
</tr>
<tr>
<td>“IS_RAISED” on page 153</td>
<td>Returns true when the specified alert is in a raised state</td>
</tr>
</tbody>
</table>

### Table 52. C-SQL functions L to N

<table>
<thead>
<tr>
<th>C-SQL function and link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“LAST_DAY” on page 135</td>
<td>Returns the date of the last day of the month that contains a specified date</td>
</tr>
<tr>
<td>“LEAST” on page 136</td>
<td>Returns the least value of a list of expressions</td>
</tr>
<tr>
<td>“LOG” on page 145</td>
<td>Returns the logarithm of a number from a specific base</td>
</tr>
<tr>
<td>“logNormDist” on page 195</td>
<td>Returns the cumulative lognormal distribution of a value</td>
</tr>
<tr>
<td>“LOWER” on page 157</td>
<td>Converts all uppercase characters in a string to lowercase</td>
</tr>
<tr>
<td>“LPAD” on page 158</td>
<td>Inserts one or more instances of a string into the start of another string</td>
</tr>
<tr>
<td>“LTRIM” on page 159</td>
<td>Removes characters from the start of a string</td>
</tr>
<tr>
<td>“MAX” on page 167</td>
<td>Returns the maximum value from a set</td>
</tr>
<tr>
<td>“median” on page 196</td>
<td>Returns the median (middle) number in a set</td>
</tr>
<tr>
<td>“MIN” on page 167</td>
<td>Returns the minimum value from a set</td>
</tr>
<tr>
<td>“MOD” on page 146</td>
<td>Returns the modulus (remainder) of a division</td>
</tr>
<tr>
<td>“mode” on page 197</td>
<td>Returns the most frequently occurring number in a set</td>
</tr>
<tr>
<td>“MOV_function” on page 126</td>
<td>Limits the rows used in a set function calculation to a set of the latest rows in the view</td>
</tr>
<tr>
<td>“MOV_AVG” on page 168</td>
<td>Returns the moving average value (arithmetic mean) of a moving window set of numeric values</td>
</tr>
<tr>
<td>“MOV_COUNT” on page 169</td>
<td>Returns the count of rows in a moving window set</td>
</tr>
<tr>
<td>“MOV_MAX” on page 170</td>
<td>Returns the maximum value from a moving window set</td>
</tr>
<tr>
<td>“MOV_MIN” on page 171</td>
<td>Returns the minimum value from a moving window set</td>
</tr>
<tr>
<td>“MOV_SUM” on page 172</td>
<td>Returns the sum of a moving window set of numeric values</td>
</tr>
<tr>
<td>“MOV_STD_DEVIATION” on page 182</td>
<td>Returns sample standard deviation of a moving window set of numbers</td>
</tr>
</tbody>
</table>
### Table 52. C-SQL functions L to N (continued)

<table>
<thead>
<tr>
<th>C-SQL function and link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“MOV_VARIANCE” on page 183</td>
<td>Returns the square of the sample standard deviation of a moving window set of numbers</td>
</tr>
<tr>
<td>“NTILE” on page 184</td>
<td>Determines the tier rank of each value in a set with respect to the entire set</td>
</tr>
</tbody>
</table>

### Table 53. C-SQL functions P to Z

<table>
<thead>
<tr>
<th>C-SQL function and link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“POSITION” on page 159</td>
<td>Returns the position of a character or string within a string</td>
</tr>
<tr>
<td>“POWER” on page 147</td>
<td>Returns a value raised to a specific power</td>
</tr>
<tr>
<td>“PREV” on page 173</td>
<td>Returns a value from the next to last row in a set</td>
</tr>
<tr>
<td>“PRIOR_VALUE” on page 136</td>
<td>Returns the prior value of a column, alias, or expression.</td>
</tr>
<tr>
<td>“RANK” on page 185</td>
<td>Determines the rank of each value in a set with respect to the entire set</td>
</tr>
<tr>
<td>“RATIO_TO_REPORT” on page 187</td>
<td>Calculates the ratio of a value to the sum of the values for the entire set</td>
</tr>
<tr>
<td>“ROUND” on page 149</td>
<td>Returns a number rounded up to a specified count of decimal places</td>
</tr>
<tr>
<td>“RPAD” on page 161</td>
<td>Adds one or more instances of a string to the end of another string</td>
</tr>
<tr>
<td>“RTRIM” on page 162</td>
<td>Removes characters from the end of a string</td>
</tr>
<tr>
<td>“SAFE_DIVIDE” on page 149</td>
<td>Returns a quotient of two values, unless the quotient is 0, in which case it returns an alternate quotient value</td>
</tr>
<tr>
<td>“SIGN” on page 150</td>
<td>Identifies the arithmetic sign of a number</td>
</tr>
<tr>
<td>“SQRT” on page 151</td>
<td>Returns the square root of a number</td>
</tr>
<tr>
<td>“STD_DEVIATION” on page 188</td>
<td>Returns sample standard deviation of a set of numbers</td>
</tr>
<tr>
<td>“SUBSTRING” on page 162</td>
<td>Returns the portion of a string identified by position and length</td>
</tr>
<tr>
<td>“SUM” on page 174</td>
<td>Returns the sum of a set of numeric values</td>
</tr>
<tr>
<td>“SUM_OVER_GROUPS” on page 189</td>
<td>Returns a running sum of the numeric values ordered by the column specified in the arguments</td>
</tr>
<tr>
<td>“TIMESTAMP_DIFF” on page 137</td>
<td>Returns the interval of time between 2 timestamps</td>
</tr>
<tr>
<td>“TO_CHAR” on page 138</td>
<td>Converts a date-time to a character string</td>
</tr>
<tr>
<td>“TO_DATE” on page 139</td>
<td>Converts a character string to a date-time value</td>
</tr>
<tr>
<td>“TRUNC” on page 151</td>
<td>Truncates a number to a specific count of decimal places</td>
</tr>
<tr>
<td>“TUMBLE_AVG” on page 175</td>
<td>Returns the average value (arithmetic mean) of a tumbling window set</td>
</tr>
</tbody>
</table>
Table 53. C-SQL functions P to Z (continued)

<table>
<thead>
<tr>
<th>C-SQL function and link</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“TUMBLE_COUNT” on page 176</td>
<td>Returns the count of rows in a tumbling window set</td>
</tr>
<tr>
<td>“TUMBLE_MAX” on page 177</td>
<td>Returns the maximum value from a tumbling window set</td>
</tr>
<tr>
<td>“TUMBLE_MIN” on page 179</td>
<td>Returns the minimum value from a tumbling window set</td>
</tr>
<tr>
<td>“TUMBLE_SUM” on page 180</td>
<td>Returns the sum of a tumbling window set of numeric values</td>
</tr>
<tr>
<td>“TUMBLE_STD_DEVIATION” on page 189</td>
<td>Returns sample standard deviation of a tumbling window set of numbers</td>
</tr>
<tr>
<td>“TUMBLE_VARIANCE” on page 191</td>
<td>Returns the square of the sample standard deviation of a tumbling window set of numbers</td>
</tr>
<tr>
<td>“UPPER” on page 164</td>
<td>Converts all lowercase characters in a string to uppercase</td>
</tr>
<tr>
<td>“VARIANCE” on page 192</td>
<td>Returns the square of the sample standard deviation of a set of numbers</td>
</tr>
<tr>
<td>“yield” on page 198</td>
<td>Computes the yield given a set of test results and calculates the yield</td>
</tr>
</tbody>
</table>

MOV_function

The functions that have the form MOV_function are scalar functions that limit the rows used in a set function calculation to a set of the latest rows in a view.

The following table lists the sections that provide information about the moving window set functions.

Table 54. Moving window set functions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“MOV_AVG” on page 168</td>
<td>Returns the moving average value (arithmetic mean) of a moving window set of numeric values.</td>
</tr>
<tr>
<td>“MOV_COUNT” on page 169</td>
<td>Returns the count of rows in a moving window set.</td>
</tr>
<tr>
<td>“MOV_MAX” on page 170</td>
<td>Returns the maximum value from a moving window set.</td>
</tr>
<tr>
<td>“MOV_MIN” on page 171</td>
<td>Returns the minimum value from a moving window set.</td>
</tr>
<tr>
<td>“MOV_SUM” on page 172</td>
<td>Returns the sum of a moving window set of numeric values.</td>
</tr>
<tr>
<td>“MOV_STD_DEVIATION” on page 182</td>
<td>Returns sample standard deviation of a moving window set of numbers.</td>
</tr>
<tr>
<td>“MOV_VARIANCE” on page 183</td>
<td>Returns the square of the sample standard deviation of a moving window set of numbers.</td>
</tr>
</tbody>
</table>
Moving set functions are shorthand for simple query windows. For a complete discussion, see Chapter 38, “Query windows,” on page 361.

Syntax of MOV_function functions

All moving set functions have the same syntax:

\[
\text{mov\_function}(\text{numeric, window, size [\text{,timestampColumn}]} )
\]

In the example, mov\_function is an existing set function. The return type of the moving function is the same as the return type of the named function. The numeric is typically a column in the view, but can contain other functions and operators, though it cannot reference a rank function.

The window and size arguments specify which rows are included in the set. The window argument determines whether size is the count of rows in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR). For example, a set of the last six events limits the set to no more than six events per group when using the GROUP BY clause. For more information, see “GROUP BY clause” on page 320. Some events generate multiple rows.

The following illustration shows a moving set function that returns the sum of the last six prices, where the moving set function is defined as MOV\_SUM(Price, EVENT, 6). Initially, only five prices are in the window, so the function returns the sum of five prices. As the window grows, the sum changes because earlier prices are not included in the sum as shown by the last two windows in the illustration. The expression MOV\_SUM(Price, EVENT, 6) is equivalent to the complete window expression SUM (Price) OVER (EVENT, ‘5’ PRECEDING REFERENCE FRAME).

![Figure 13. A moving set function that returns the sum of the last six prices](image)

When using a time-series span (instead of an event span), the size of the set varies depending on when the events were recorded in the view. For example, when using a time-series of one hour, only those rows that entered the view in the last hour are used in the calculation.

The following illustration shows a moving set function that returns the sum of the prices recorded in the last hour, where the moving set function is defined as MOV\_SUM(PRICE, HOUR, 1). Initially, five prices are in the window, so the sum includes the sum of those five prices because they entered the view within the last hour. As the window grows, only those prices that entered the view within the last hour are included in the sum. As shown in the illustration, a later sum includes only two prices because the previous prices entered the view outside the last hour. The expression MOV\_SUM(PRICE, HOUR, 1) is equivalent to the complete window expression SUM(Price) OVER (RANGE ‘1’ PRECEDING REFERENCE OPERATOR).
Attention: It is possible for an event to arrive in time to be included in a view, but then to be discarded because by the time it reaches the view, it is no longer in the time span of the view. For example, if the event enters the data stream table a few milliseconds before it would be excluded from a derived view, it might be included or excluded, depending on how long it takes to process and propagate the event in the base views.

The optional timestampColumn argument instructs the system to use the value of a field in the view as the reference point for starting the time-series span. When you omit this option, the system calculates the time-series based on the system clock, such as the last hour from now. When you name a timestamp column instead, the calculation is based on the time span from the value that is the most recent value in the column from any row in the view.

In the following illustration, where the moving set function is defined as MOV_SUM(Price, HOUR, 1, Time), only two prices out of six are used in the sum because the prices selected are based on the value in the Time column, not the order in which they entered the view. The expression MOV_SUM(Price, HOUR, 1, Time) is equivalent to the complete window expression SUM (Price) OVER (ORDER BY Time RANGE '1' HOUR PRECEDING REFERENCE OPERATOR).

Time-series spans

Time span calculations use the Gregorian calendar and are calculated to the second that the event was recorded in the vc_timestamp column.
For example, if the span is one day, and an event arrives just before midnight, it excludes most events on the previous day. However, an event that arrives just after midnight includes almost all of the events on the previous day. Here are some additional considerations:

In locales where daylight saving time is observed, durations of days, months, and years are adjusted accordingly. While 1 day is typically 24 hours long, it can be 23 or 25 hours, depending on the time of year.

Month calculations are based on the day of the month: a one month span on 5 April includes all dates after 5 March. When the day of the month does not exist at the start of the window, the end of the month is used. For example, a one month span on 31 May starts after 30 April.

Similarly, year calculations are based on the day of the year, and adjust as necessary for leap years.

For the purposes of parallel execution, you can choose to not process events in the order in which they are timestamped. In this case, the order of the data within a group is arbitrary and produces only approximate Moving Set values that might not be reproducible for the same input events during a subsequent evaluation of the same set.

The set of events included in a moving window view is determined when a new event enters the view. Events that are filtered out of a view before they enter the view, such as when excluded by a WHERE clause, do not affect the view and do not cause the view to update.

**View warning**

Do not use a moving set function in a derived view to perform a calculation on a moving set function result in a base view, because the derived function always returns the current value in the base view, regardless of the span of the window.

If you need such an aggregation, place the functions in the same view. See the example in "AVG" on page 164 for details.

**GROUP BY interactions**

Using a moving set function on a view defined with a GROUP BY clause populates the groups.

For more information, see "GROUP BY clause" on page 320.

**Time-series spans**

Time-series spans apply to all events in all groups. Only events that fall within the time span are included in the groups. Events that do not meet the time span definition are excluded. When all data streams are removed from a group set, the group is empty. If no other columns retain the group, it is removed from the view. Consider this example where average prices are tracked in groups by product for the last hour. When a product no longer has data streams in the last hour, the group for that product is removed.

```sql
SELECT MOV_AVG(Price,HOUR,1) AS Av_pr
FROM ...
GROUP BY Product
```
However, in this variation the presence of the SUM() function causes the view to retain every data stream group, but the average price for a group of the last hour can be empty. Querying an empty group returns NULL.

```
SELECT MOV_AVG(Price, HOUR, 1) AS Av_pr, SUM(Price) AS Total
FROM ...
GROUP BY Product
```

### Data stream spans

Data stream spans apply their size to each group in the view. Each group tracks a count of data streams determined by the size of the span. Groups are never removed from the view, and data streams are removed from the sets only when they are pushed out by a newer data stream. Consider the view in the following example, where MOV_AVG() tracks up to three data streams for each group. When the fourth data stream, whose Ix value is 100, is inserted, the first data stream is dropped from the moving average calculation of the Ix=100 group. However, within this example, once the 200 group is created, the set remains constant with the one data stream.

```
SELECT IX,
  MOV_AVG(Price, EVENT, 3) AS Av_pr,
  COUNT(1) AS Ct
FROM ...
GROUP BY Ix
```

*Figure 16. A moving average tracks up to three data streams per group*

For data stream spans that have data streams with multiple rows in the view, the entire data stream is treated as one item in the set based on the timestamp (vc_timestamp) and data stream ID (vc_event_id).

All columnNames referred directly by a rank function or scalar function must appear in the set of columns listed in the GROUP BY list.

### Case statement

A case statement evaluates a list of conditions and returns the results from one of the possible expressions.

The case statement consists of the functions described in the following topics:

- "CASE" on page 131
- "ELSE" on page 131
- "END" on page 131
- "THEN" on page 131
- "WHEN" on page 131
CASE
A CASE expression returns the result of an expression that corresponds to a matching true condition.

Optionally, each condition can return NULL instead. If no condition is found to be true, the expression returns the result of the ELSE condition or NULL when ELSE is omitted.

For detailed information and examples, see “CASE expression” on page 314.

ELSE
ELSE is the expression returned if no comparison operation evaluates to TRUE.

If this argument is omitted and no comparison operation evaluates to TRUE, the case statement returns NULL.

For detailed information and examples, see “CASE expression” on page 314.

END
END finishes a case statement.

For detailed information and examples, see “CASE expression” on page 314.

THEN
THEN is the expression returned if an input expression equals a WHEN expression and evaluates to TRUE, or a Boolean expression evaluates to TRUE.

The result expression is any valid C-SQL expression.

For detailed information and examples, see “CASE expression” on page 314.

WHEN
WHEN is either a simple expression to which an input expression is compared when using the simple CASE format, or it is the Boolean expression evaluated when using the searched CASE format.

For detailed information and examples, see “CASE expression” on page 314.

Conversion
Conversion functions convert a data type to another data type.

CAST
This scalar function converts a value from one IBM Cognos Real-time Monitoring type to another Cognos Real-time Monitoring type.

Syntax
CAST( value AS vcDataType )

Parameters
- value
  Value to convert.
vcDataType

One of the C-SQL data types (see Chapter 9, “Data types,” on page 63) to convert to.
- INTEGER
- DECIMAL
- DOUBLE PRECISION
- VARCHAR
- TIMESTAMP
- BOOLEAN
- LONG

**Return type**

Same as vcDataType argument.

**Remarks**

Types are cast according to the Order of precedence table in “Data type conversion” on page 63. CAST() returns an error if a type cannot be cast as specified in an expression. For example, the following is an error because C-SQL attempts to cast ‘4.5’ to an INTEGER, but the decimal is an illegal character for INTEGER types:

3 < CAST( '4.5' AS INTEGER )

When casting from a decimal formatted column to a string, the result is zero-padded on the decimals to match the scale, just as when casting from a string to a decimal. For example,

CAST( '1.1' AS DECIMAL(5,4) ) --> 1.1000
CAST( CAST( '1.1' AS DECIMAL(5,4) ) AS VARCHAR ) -->'1.1000'

**Example**

Cast a date string into a timestamp:

```sql
SELECT CAST('1997-10-22' AS TIMESTAMP )
FROM Cambrian;
```

**See also**

“Data type conversion” on page 63 provides details about converting types.

“TO_CHAR” on page 138 converts the timestamp to a character string of specified format.

“TO_DATE” on page 139 converts a character string to a date.

---

**Date and time**

Date and time functions are used to handle data related to dates, time, and timestamps.

**CURRENT_TIMESTAMP**

This scalar function returns the current date and time in the server time zone.
Syntax
CURRENT_TIMESTAMP()

Return type
Date-Time.

Example
LAST_DAY(CURRENT_TIMESTAMP()) returns the date of the last day of the current month.

See also
“TO_CHAR” on page 138 converts a timestamp to a character string.
“LAST_DAY” on page 135 returns the date of the last day of a month.
“DATE_ADD” adds a duration to a date-time.
“DATE_DIFF” on page 134 subtracts a duration from a date-time.
“TIMESTAMP_DIFF” on page 137 returns a time interval between two timestamps.

DATE_ADD
This scalar function adds a duration of time to a date-time value.

Syntax
DATE_ADD( timestamp, [ durationType, ] duration )

Parameters
• timestamp
  The date-time to adjust.
• durationType
  Type of the duration value; one of these literals:
  – SECOND
  – MINUTE
  – HOUR
  – DAY (default)
  – MONTH
  – YEAR

Return type
Date-Time.

Remarks
Uses Gregorian calendar addition rules.
Example

DATE_ADD( CURRENT_TIMESTAMP(), 2) returns a date-time two days in the future from now.

DATE_ADD( timestamp, DAY, 14 ) returns a value 2 weeks after the date.

See also

“DATE_DIFF” subtracts a duration from a date-time.

“CURRENT_TIMESTAMP” on page 132 returns the current date and time.

“TIMESTAMP_DIFF” on page 137 returns a time interval between two timestamps.

DATE_DIFF

This scalar function subtracts a duration from a date-time value.

Syntax

DATE_DIFF( timestamp, [ durationType, ] duration )

Parameters

• timestamp
  The date-time from which to subtract some duration of time.

• durationType
  Type of the duration value; one of these literals:
  – SECOND
  – MINUTE
  – HOUR
  – DAY (default)
  – MONTH
  – YEAR

Return type

Date-Time.

Remarks

Uses Gregorian calendar subtraction rules.

Durations that span leap year days and seconds generally ignore the leap value. For example, subtracting one year from 3 March 1976 results in 3 March 1975 without being affected by the 29 February 1976 leap day. However, subtracting one year from 29 February results in a 28 February date.

Examples

DATE_DIFF( CURRENT_TIMESTAMP(), 2) returns a date-time two days ago from now.
See also

“DATE_ADD” on page 133 adds a duration to a date-time.

“CURRENT_TIMESTAMP” on page 132 returns the current date and time.

“TIMESTAMP_DIFF” on page 137 returns a time interval between two timestamps.

GREATEST

This scalar function returns the greatest of a list of expression results.

Syntax

```
GREATEST( value, value [, value ... ] )
```

Parameters

- `value`
  A value to be used for the comparison. All values after the first are converted to
  the data type of the first.

Return type

Same data type as argument.

Example

Selects the string with the greatest value:

```
SELECT Greatest(
    'SCHOLAR',
    'SKYLER',
    'SHUELLER')
FROM Students;
```

Greatest
--------
SKYLER

See also

“LEAST” on page 136 determines the least value from a list.

“MAX” on page 167 returns the maximum value from a set.

“Function types” on page 115 discusses moving sets.

LAST_DAY

This scalar function returns the date of the last day of the month that contains a
specified date.

Syntax

```
LAST_DAY( dateTime )
```
Parameters

- dateTime
  A valid date (TIMESTAMP).

Return type

Date-Time.

Example

LAST_DAY( CURRENT_TIMESTAMP() ) returns the date of the last day of the current month.

See also

“CURRENT_TIMESTAMP” on page 132 returns the current date and time.

LEAST

This scalar function returns the least value of a list of expressions.

Syntax

LEAST( value, value [, value ... ] )

Parameters

- value
  A value to be used for the comparison. All values after the first are converted to the data type of the first.

Return type

Same data type as the argument.

Example

SELECT Least
  ('SCHOLAR',
   'SKYLER',
   'SHUELLER')
FROM Students;

Least
-----
SCHOLAR

See also

“GREATEST” on page 135 determines the greatest value from a list.

“MIN” on page 167 returns the minimum value from a set.

“Function types” on page 115 discusses moving sets.

PRIOR_VALUE

This scalar function returns the prior value of a column, alias, or expression.
Syntax

PRIOR_VALUE( columnName )

Parameters

• columnName
  Column or alias of any data type to evaluate.

Return type

Same data type as argument.

Remarks

Returns a NULL if there is no prior value, the first time the function is called on the columnName.

Use PRIOR_VALUE() when the data (data streams) enter the system grouped and ordered.

This function is not permitted in the WHERE clause (see “WHERE clause” on page 319) of a view definition.

Example

Consider this query that identifies how long a task took to complete [as a percentage of an hour] based on minutes since the previous task completed:

SELECT Task, CAST((TIMESTAMP_DIFF(PRIOR_VALUE(Completed), Completed, MINUTE ) /60 , DECIMAL(5,4)) AS Hours
FROM Tasks_Completed

Task     Hours
--------- -----
Startup  0.0887
Initialize 0.1012
Begin job 4.3243
Finish job 0.2500
Clean up 0.1285
Shut down 0.6667
Have milk shake 0.6667

See also

“PREV” on page 173 returns a value from the next to last row in a set.

TIMESTAMP_DIFF

This scalar function returns, as an absolute value, the interval of time between two timestamps.

Syntax

TIMESTAMP_DIFF( startTime, endTime, unit )
**Parameters**
- startTime
  Start Date-Time.
- endTime
  End Date-Time.
- unit
  Type of the time interval to return; one of these literals:
  - SECOND
  - MINUTE
  - HOUR
  - DAY MONTH (30 days) YEAR (12 months or 360 days)

**Return type**

INTEGER.

**Remarks**

Uses absolute time difference rounded up to the nearest whole value; does not use
Gregorian calendar arithmetic.

Rounds the result to the nearest integer. For example, the difference between 10:00
and 10:29 in HOUR units is zero (0), but 10:00 and 10:30 return one (1).

Returns NULL if either timestamp is NULL.

**Example**

Return the count of days from now until the end of the month.

```
TIMESTAMP_DIFF( CURRENT_TIMESTAMP(),
    LAST_DAY( CURRENT_TIMESTAMP() ), DAY)
```

Return True when a problem ticket is open for more than 30 minutes:

```
TIMESTAMP_DIFF( ticket_opened, CURRENT_TIMESTAMP(), MINUTE) > 30
```

Return the number of days between two dates as a positive number, regardless of
which date is oldest:

```
ABS( TIMESTAMP_DIFF( father_birthdate, mother_birthdate, DAY) )
```

**See also**

- "DATE_ADD" on page 133 adds a duration to a date-time.
- "DATE_DIFF" on page 134 subtracts a duration from a date-time.
- "CURRENT_TIMESTAMP" on page 132 returns the current date and time.

**TO_CHAR**

This scalar function converts a date-time to a character string.
Syntax

\texttt{TO\_CHAR( date, [ format ] )}

\textbf{Parameters}

- \texttt{date}
  
  Date-time (see “Date-time” on page 69) value to convert.

- \texttt{format}
  
  Date pattern of string identical to the one used by the Java SimpleDateFormat class, and is described in “Date-time formatting” on page 74. Omit this option to convert by using the default format, which is “yyyy-MM-dd HH:mm:ss.SSSSSSSS”.

\textbf{Return type}

VARCHAR.

\textbf{Remarks}

See “Converting between date-time and strings” on page 70 for a complete discussion about the conversion.

\textbf{Examples}

\texttt{TO\_CHAR(CURRENT\_TIMESTAMP(), 'd MMMM yy')} returns ’5 March 03’.

\textbf{See also}

“CAST” on page 131 converts one data type to another.

“TO\_DATE” converts a character string to a date.

“CURRENT\_TIMESTAMP” on page 132 returns the current date and time.

\textbf{TO\_DATE}

This scalar function converts a character string to a date-time value.

\textbf{Syntax}

\texttt{TO\_DATE( string, [ format ] )}

\textbf{Parameters}

- \texttt{string}
  
  Date string (VARCHAR) or literal to convert.

- \texttt{format}
  
  Date pattern of string identical to the one used by the Java SimpleDateFormat class, and is described in “Date-time formatting” on page 74. Omit this option to convert by using the default format, which is “yyyy-MM-dd HH:mm:ss.SSSSSSSS”.

\textbf{Return type}

Date-time.
Remarks

If you omit the time values from the pattern, a zero (0) fills the portion of the TIMESTAMP which causes the time to be set to midnight.

Examples

TO_DATE('2003-02-18') is identical to TO_DATE('2003-02-18', "yyyy-MM-dd").

TO_DATE('2003-02-18 12:00:00', "yyyy-MM-dd HH:mm:ss") assigns noon as the time.

To strip the time portion off a TIMESTAMP, convert it to character and back to date:
TO_DATE( TO_CHAR( a_timestamp, 'yyyy-MM-dd' ) )

See also

"CAST" on page 131 converts one data type to another.
"TO_CHAR" on page 138 converts a date to a character string.
"CURRENT_TIMESTAMP" on page 132 returns the current date and time.

Numbers

Number functions are used for performing operations on numerical data.

ABS

This scalar function returns the absolute value of a number.

Syntax

ABS( numeric )

Parameters

- numeric
  An expression that evaluates to a numeric.

Return type

Numeric, same data type as numeric argument.

Example

Return the difference in ages between two persons, regardless of which is older.
SELECT ABS( father_age - mother_age ) AS "Difference of parents ages"
FROM Family

See also

"SIGN" on page 150 returns the arithmetic sign of a number.
CEIL

This scalar function returns the smallest integer, rounded up from zero, greater than or equal to a number.

Syntax

CEIL( numeric )

Parameters

• numeric
  Number to round.

Return type

Same data type is numeric result.

Example

CEIL( 1234.56 ) returns 1235.00.

CEIL( -2.75 ) returns -2.00.

See also

"FLOOR" on page 144 returns the largest value less than or equal to a number.

DISPLAY_MONEY

This scalar function formats a number as a currency value.

Syntax

DISPLAY_MONEY( number [, languageCode, countryCode ] )

Parameters

• number
  Number to format.

• languageCode
  A two-letter ISO 639 language code. Helps determine the currency symbol to display.

• countryCode
  A two-letter ISO 3166 country or region code. Specifies the thousands separator, decimal separator, and count of decimal digits to display based on what is appropriate for the country or region.

Return type

VARCHAR.

Remarks

Returns the number formatted as a currency string.

Omitting the languageCode and countryCode uses the symbol and format appropriate for country or region that your computer is configured to use by default.
Some currency symbols require that the browser is configured to the correct code page for the language.

**Common codes**

The following table lists some of the common ISO 639 two-letter language codes:

*Table 55. Common country or region codes*

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
<th>Language</th>
<th>Code</th>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afrikaans</td>
<td>af</td>
<td>Fiji</td>
<td>fj</td>
<td>Dutch</td>
<td>nl</td>
</tr>
<tr>
<td>Arabic</td>
<td>ar</td>
<td>Faroese</td>
<td>fo</td>
<td>Norwegian</td>
<td>no</td>
</tr>
<tr>
<td>Catalan</td>
<td>ca</td>
<td>French</td>
<td>fr</td>
<td>Punjabi</td>
<td>pa</td>
</tr>
<tr>
<td>Corsican</td>
<td>co</td>
<td>Hebrew</td>
<td>he</td>
<td>Polish</td>
<td>pl</td>
</tr>
<tr>
<td>Czech</td>
<td>cs</td>
<td>Hindi</td>
<td>hi</td>
<td>Portuguese</td>
<td>pt</td>
</tr>
<tr>
<td>Danish</td>
<td>da</td>
<td>Croatian</td>
<td>hr</td>
<td>Russian</td>
<td>ru</td>
</tr>
<tr>
<td>German</td>
<td>de</td>
<td>Italian</td>
<td>it</td>
<td>Serbo-Croatian</td>
<td>sh</td>
</tr>
<tr>
<td>Greek</td>
<td>el</td>
<td>Inuktitut</td>
<td>iu</td>
<td>Swedish</td>
<td>sv</td>
</tr>
<tr>
<td>English</td>
<td>en</td>
<td>Japanese</td>
<td>ja</td>
<td>Turkish</td>
<td>tr</td>
</tr>
<tr>
<td>Spanish</td>
<td>es</td>
<td>Korean</td>
<td>ko</td>
<td>Urdu</td>
<td>ur</td>
</tr>
<tr>
<td>Persian</td>
<td>fa</td>
<td>Mongolian</td>
<td>mn</td>
<td>Yoruba</td>
<td>yo</td>
</tr>
<tr>
<td>Finnish</td>
<td>fi</td>
<td>Nepali</td>
<td>ne</td>
<td>Chinese</td>
<td>zh</td>
</tr>
</tbody>
</table>

The following table lists some of the common ISO 3166 two-letter country or region codes.

*Table 56. Common ISO 3166 two-letter country or region codes*

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Code</th>
<th>Country or region</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUSTRALIA</td>
<td>AU</td>
<td>NEPAL</td>
<td>NP</td>
</tr>
<tr>
<td>AUSTRIA</td>
<td>AT</td>
<td>NETHERLANDS</td>
<td>NL</td>
</tr>
<tr>
<td>BRAZIL</td>
<td>BR</td>
<td>NEW ZEALAND</td>
<td>NZ</td>
</tr>
<tr>
<td>CANADA</td>
<td>CA</td>
<td>NORWAY</td>
<td>NO</td>
</tr>
<tr>
<td>CHINA</td>
<td>CN</td>
<td>OMAN</td>
<td>OM</td>
</tr>
<tr>
<td>CROATIA (local name: Hrvatska)</td>
<td>HR</td>
<td>PAKISTAN</td>
<td>PK</td>
</tr>
<tr>
<td>DENMARK</td>
<td>DK</td>
<td>PITCAIRN</td>
<td>PN</td>
</tr>
<tr>
<td>FIJI</td>
<td>FJ</td>
<td>POLAND</td>
<td>PL</td>
</tr>
<tr>
<td>FINLAND</td>
<td>FI</td>
<td>PORTUGAL</td>
<td>PT</td>
</tr>
<tr>
<td>FRANCE</td>
<td>FR</td>
<td>SAUDI ARABIA</td>
<td>SA</td>
</tr>
<tr>
<td>GERMANY</td>
<td>DE</td>
<td>SOUTH AFRICA</td>
<td>ZA</td>
</tr>
<tr>
<td>GREECE</td>
<td>GR</td>
<td>SPAIN</td>
<td>ES</td>
</tr>
<tr>
<td>INDIA</td>
<td>IN</td>
<td>SWITZERLAND</td>
<td>CH</td>
</tr>
<tr>
<td>IRAN (Islamic republic of)</td>
<td>IR</td>
<td>THAILAND</td>
<td>TH</td>
</tr>
<tr>
<td>ISRAEL</td>
<td>IL</td>
<td>TURKEY</td>
<td>TR</td>
</tr>
</tbody>
</table>
Table 56. Common ISO 3166 two-letter country or region codes (continued)

<table>
<thead>
<tr>
<th>Country or region</th>
<th>Code</th>
<th>Country or region</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITALY</td>
<td>IT</td>
<td>UNITED ARAB EMIRATES</td>
<td>AE</td>
</tr>
<tr>
<td>JAPAN</td>
<td>JP</td>
<td>UNITED KINGDOM</td>
<td>GB</td>
</tr>
<tr>
<td>KOREA (Demo. people's republic of)</td>
<td>KP</td>
<td>UNITED STATES</td>
<td>US</td>
</tr>
<tr>
<td>MEXICO</td>
<td>MX</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Examples

Here are some examples that format the number 12345.678:

Table 57. Examples that format the number 12345.678

<table>
<thead>
<tr>
<th>Language/country or region</th>
<th>Formula</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>English/USA</td>
<td>DISPLAY_MONEY(12345.678,'en','us')</td>
<td>$12,345.68</td>
</tr>
<tr>
<td>Swedish/Sweden</td>
<td>DISPLAY_MONEY(12345.678,'sv','se')</td>
<td>12 345,68 kr</td>
</tr>
<tr>
<td>German/Germany</td>
<td>DISPLAY_MONEY(12345.678,'de','de')</td>
<td>12.345,68 dm</td>
</tr>
</tbody>
</table>

EXP

This scalar function returns e raised to a specific power.

Syntax

EXP( power )

Parameters

• power

  The power (DOUBLE) to which to raise e.

Return type

DOUBLE PRECISION.

Remarks

Returns e raised to the nth power, where e = 2.71828183...

Example

EXP(4) raises e to the 4th power and returns 54.59815.

See also

"LOG" on page 145 returns the logarithm of a number from a specific base
"POWER" on page 147 raises a number to a specific power.
FLOOR
This scalar function returns the largest integer that is less than or equal to an expression.

Syntax
FLOOR( numeric )

Parameters
• numeric
  Number to floor.

Return type
Numeric, same data type as numeric argument.

Example
FLOOR('1234.56') returns the integer 1234.00, after first implicitly casting the string literal to a DECIMAL.

FLOOR(-2.75) returns -3.00.

This function behaves in the same way as the Microsoft Excel INT() function.

See also
"CEIL" on page 141 returns the smallest integer rounded up.

GREATEST
This scalar function returns the greatest of a list of expression results.

Syntax
GREATEST( value, value [, value ... ] )

Parameters
• value
  A value to be used for the comparison. All values after the first are converted to the data type of the first.

Return type
Same data type as argument.

Example
Selects the string with the greatest value:
SELECT Greatest(
  'SCHOLAR',
  'SKYLER',
  'SHUELLER')
FROM Students;
SKYLER

See also

“LEAST” on page 136 determines the least value from a list.
“MAX” on page 167 returns the maximum value from a set.
“Function types” on page 115 discusses moving sets.

LEAST

This scalar function returns the least value of a list of expressions.

Syntax

LEAST( value, value [, value ... ] )

Parameters

• value
  A value to be used for the comparison. All values after the first are converted to
  the data type of the first.

Return type

Same data type as the argument.

Example

SELECT Least
  ('SCHOLAR',
   'SKYLER',
   'SHUELLER')
FROM Students;

Least

--------

SCHOLAR

See also

“GREATEST” on page 135 determines the greatest value from a list.
“MIN” on page 167 returns the minimum value from a set.
“Function types” on page 115 discusses moving sets.

LOG

This scalar function returns the logarithm of a number from a specific base.

Syntax

LOG( numeric [, base ] )

Parameters

• numeric
Number (DOUBLE) from which to retrieve the logarithm; must be greater than 1.

- `base`
  Base (DOUBLE) of the logarithm; must be greater than zero (0). Omit this option to use the natural log of numeric.

**Return type**
DOUBLE PRECISION.

**Remarks**
This can be expressed mathematically as $\log_{\text{numeric}} \text{base}$.

**Example**
LOG(8,64) returns 2.0.
LOG(2) returns 0.301029...

**See also**
"EXP" on page 143 raise to a specific power.
"POWER" on page 147 raise a value to a specific power.

**MOD**
This scalar function returns the modulus (remainder) of a division.

**Syntax**
MOD(dividend, divisor)

**Parameters**
- `dividend`
  Numeric to divide.
- `divisor`
  Numeric to divide by.

**Return type**
INTEGER.

**Remarks**
When divisor is zero (0), returns dividend.

This function behaves differently from the classical mathematical modulus function when the dividend is negative. The classical modulus can be expressed with this formula:
$$\text{dividend} - \text{divisor} * \text{FLOOR} (\text{dividend} / \text{divisor})$$

The following function uses this updated formula:
$$\text{SIGN} (\text{dividend}) *$$
This table illustrates the difference between the MOD function and the classical modulus formula:

<table>
<thead>
<tr>
<th>Dividend</th>
<th>Divisor</th>
<th>MOD (dividend, divisor)</th>
<th>Classical modulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>-4</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>-3</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>-4</td>
<td>-3</td>
<td>-3</td>
</tr>
</tbody>
</table>

Most database management systems use the same formula as IBM Cognos Real-time Monitoring, while spreadsheet applications like Microsoft Excel use the classical modulus.

**Example**

Return the remainder of the dividend divided by the divisor:

```sql
SELECT MOD(11,4) "Modulus" FROM Census;
```

Modulus
--------
3

**POWER**

This scalar function returns a value raised to a specific power.

**Syntax**

```sql
POWER( numeric, power )
```

**Parameters**

- `numeric`
  - Number to raise.
- `power`
  - Power\(^n\) to raise numeric. Must be an integer when numeric is negative.

**Return type**

Same data type as the numeric argument.

**Remarks**

This can be expressed mathematically as number power.

**Example**

`POWER(3,5)` returns 243.
PRIOR_VALUE

This scalar function returns the prior value of a column, alias, or expression.

Syntax

PRIOR_VALUE( columnName )

Parameters

• columnName
  Column or alias of any data type to evaluate.

Return type

Same data type as argument.

Remarks

Returns a NULL if there is no prior value, the first time the function is called on
the columnName.

Use PRIOR_VALUE() when the data (data streams) enter the system grouped and
ordered.

This function is not permitted in the "WHERE clause” on page 319 of a view
definition.

Example

Consider this query that identifies how long a task took to complete [as a
percentage of an hour] based on minutes since the previous task completed:

```
SELECT Task, CAST(
    (TIMESTAMP_DIFF(
        PRIOR_VALUE(Completed), Completed, MINUTE ) /60
        , DECIMAL(5,4)
    ) AS Hours
AS Hours
FROM Tasks_Completed
```

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>0.0887</td>
</tr>
<tr>
<td>Initialize</td>
<td>0.1012</td>
</tr>
<tr>
<td>Begin job</td>
<td>4.3243</td>
</tr>
<tr>
<td>Finish job</td>
<td>0.2500</td>
</tr>
<tr>
<td>Clean up</td>
<td>0.1285</td>
</tr>
<tr>
<td>Shut down</td>
<td>0.6667</td>
</tr>
</tbody>
</table>
ROUND

This scalar function returns a number rounded up to a specified count of decimal places.

Syntax

ROUND( number, [ places ] )

Parameters

• number
  The numeric expression to round
• places
  Count of decimal places to round to. A negative integer rounds to whole number digits. Default is zero (0) to remove any fractional components.

Return type

Numeric, same data type as number argument.

Examples

ROUND(1294.5078) returns 1294.
ROUND(1294.5078, 0) returns 1294.
ROUND(1294.5078, 1) returns 1294.5.
ROUND(1294.5078, 2) returns 1294.51.
ROUND(1294.5078, -2) returns 1300.

See also

See also "TRUNC" on page 151 removes digits from a number.

SAFE_DIVIDE

This scalar function returns a quotient determined by a dividend, a divisor, and an alternate divide-by-zero quotient.

If the divisor is non-zero, this function returns the quotient of the dividend divided by the divisor. If the divisor is zero, or there is some other runtime data error while calculating the quotient, such as a DECIMAL out of range, the alternate quotient is returned.

The practice is to use decimals for both dividend and divisor values.

Syntax

SAFE_DIVIDE(numberDividend, numberDivisor, numberAlternateQuotient)
Parameters

- **numberDividend**
  The numerical value to be divided.
  The data type is determined by how the value is entered. If the value is entered without a decimal (for example, 19), the value is cast as an integer. If the value is entered with a decimal (for example, 19.00), the value is cast as decimal.

- **numberDivisor**
  The numerical value by which the dividend is divided.
  This value is automatically cast as the same data type as the numberDividend.

- **numberAlternateQuotient**
  The numerical value to be returned if the resulting quotient is 0 or if there is some other runtime data error while calculating the quotient.

Return type

Numeric, same data type as number argument.

Examples

SAFE_DIVIDE (100, 20, 4321) returns 5.

SAFE_DIVIDE (100.19.5, 4321) returns 5. The 19.5 is cast as an integer with a value of 19.

SAFE_DIVIDE (100.00, 19.5, 4321) returns 5.12. The quotient is carried to the same number of decimal points as the numberDividend, in this case two decimal points.

SAFE_DIVIDE (SAFE_DIVIDE(100.000, 19.5, 4321) returns 5.128. The quotient is carried to the same number of decimal points as the numberDividend, in this case three decimal points.

SAFE_DIVIDE (100.0, 4321) returns 4321. Because the quotient of the numberDividend and numberDivisor has a value of 0, the numberAlternateQuotient value is returned.

SIGN

This scalar function identifies the arithmetic sign of a number.

Syntax

SIGN( number )

Parameters

- **number**
  The numeric value to evaluate.

Return type

INTEGER.

Remarks

Returns 1 if the number is positive, 0 if the number is zero, and -1 if the number is negative.
These expressions return identical results:
(number * SIGN(number) )
ABS(number)

**See also**

"ABS" on page 140 returns the absolute value of a number.

**SQRT**

This scalar function returns the square root of a number.

**Syntax**

```
SQRT( number )
```

**Parameters**

- `number`

  The number (DOUBLE) to evaluate. Must be greater than zero (0).

**Return type**

DOUBLE PRECISION.

**Example**

```
SQRT(42) returns 6.480...
```

**TRUNC**

This scalar function truncates a number to a specific count of decimal places for a decimal number.

For a timestamp, this function returns zero in the fields below the specified precision.

**Syntax**

```
TRUNC( decimalNumber [, places ] )
TRUNC( timestamp [, 'D' | 'h' | 'm' | 's' ] )
```

**Parameters**

- `decimalNumber`
  Number to truncate.

- `timestamp`
  Timestamp to truncate.

- `places`
  Count of decimal places to truncate to. When omitted, truncates all decimals and returns an integer. When negative, converts digits to zero.

- `'D'`
  Return only day information in the timestamp. Hours, minutes, and seconds are returned as zero.

- `'h'`
  Return only day and hour information in the timestamp. Minutes and seconds are returned as zero.
• 'm'
  Return only day, hour, and minute information in the timestamp. Seconds are
  returned as zero.
• 's'
  Return only day, hour, and second information in the timestamp, but do not
  show milliseconds.

**Return type**

Numeric, same data type as decimalNumber argument.

**Examples**

TRUNC(1234.567) returns 1234.
TRUNC(1234.567, 1) returns 12345.6.
TRUNC(1234.567, -2) returns 1200.
TRUNC(2007-06-27 11:01:414.213, 'D') returns 2007-06-27 0:0:0.0
TRUNC(2007-06-27 11:01:414.213, 'D 'h' 'm') returns 2007-06-27 11:01:0.0

**See also**

"ROUND" on page 149 rounds the number up to a specified count of decimal places.

---

**IS_ACKED**

This scalar function returns true when the specified alert is in an acknowledged state.

**Syntax**

IS_ACKED('alertName')

**Parameters**

- alertName
  The fully qualified name of an alert (see Chapter 5, “Alerts,” on page 37). The name must include the containing business activity and scenario names, for example, 'activityName.scenarioName.alertName'.

**Return type**

Boolean.

**Remarks**

Returns True if the alert exists and is in an acknowledged state. Otherwise, if the alert is not acknowledged or does not exist, it returns False.
Because this function returns False when the alert does not exist, there is no test to ensure that the name you entered is a valid alert in the system. Therefore, misspelling the name causes the function to always return False.

Use this function in a rule condition to test the state of an alert and to generate a new alert when the tested alert is not acknowledged for a period of time.

When used in a view definition, the view definition cannot have a set function.

**IS_RAISED**

This scalar function returns true when the specified alert is in a raised state.

**Syntax**

\[
\text{IS\_RAISED('alertName')}
\]

**Parameters**

- alertName
  
  Fully qualified name of an alert (see Chapter 5, “Alerts,” on page 37). The name must include the containing business activity and scenario names, such as ‘activityName.scenarioName.alertName’.

**Return type**

Boolean.

**Remarks**

Returns True if the alert exists and is in a raised state. Otherwise, if the alert is in a lowered state or does not exist, it returns False.

Because this function returns False when the alert does not exist, there is no test to ensure that the name you entered is a valid alert in the system. Therefore, misspelling the name causes the function to always return False.

Use this function in a rule condition to test the state of an alert and to generate a new alert when the tested alert remains raised for a period of time.

When used in a view definition, the view definition cannot have a set function.

For more information, see “Rules that monitor alerts” on page 301.

**PRIOR\_VALUE**

This scalar function returns the prior value of a column, alias, or expression.

**Syntax**

\[
\text{PRIOR\_VALUE( columnName )}
\]

**Parameters**

- columnName
  
  Column or alias of any data type to evaluate.
**Return type**

Same data type as argument.

**Remarks**

Returns a NULL if there is no prior value, the first time the function is called on the columnName.

Use PRIOR_VALUE() when the data (data streams) enter the system grouped and ordered.

This function is not permitted in the “WHERE clause” on page 319 of a view definition.

**Example**

Consider this query that identifies how long a task took to complete [as a percentage of an hour] based on minutes since the previous task completed:

```sql
SELECT Task, CAST(
    (TIMESTAMP_DIFF(PRIOR_VALUE(Completed), Completed,
    MINUTE) /60,
    , DECIMAL(5,4)
    ) AS Hours
) AS Hours
FROM Tasks_Completed
```

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>0.0887</td>
</tr>
<tr>
<td>Initialize</td>
<td>0.1012</td>
</tr>
<tr>
<td>Begin job</td>
<td>4.3243</td>
</tr>
<tr>
<td>Finish job</td>
<td>0.2500</td>
</tr>
<tr>
<td>Clean up</td>
<td>0.1285</td>
</tr>
<tr>
<td>Shut down</td>
<td>0.6667</td>
</tr>
<tr>
<td>Have milk shake</td>
<td>0.6667</td>
</tr>
</tbody>
</table>

**See also**

“PREV” on page 173 returns a value from the next to last row in a set.

---

**System**

The function in this category is used to return a login name.

**CURRENT_USER**

This scalar function returns the login name of the current user.

**Syntax**

```sql
CURRENT_USER()
```

**Return type**

VARCHAR.
Remarks

Returns the login name of the user as defined in IBM Cognos Real-time Monitoring, in the same character case, and as it appears in Cognos Real-time Monitoring Workbench. When using in a comparison, be sure to match the character case exactly.

This function is primarily for use in access filters. See Chapter 2, “Access filters,” on page 3, especially the section “User lookup tables in access filters” on page 4, for examples and uses.

Other

These functions can be used with alerts.

Text

Text functions are used to perform operations on strings.

CHAR_LENGTH

This scalar function returns the length of a string.

Syntax

CHARACTER_LENGTH( string )

Parameters

- string
  
  String or VARCHAR expression result whose length to evaluate.

Return type

INTEGER.

Remarks

Alternate spelling is:

CHAR_LENGTH( string )

Returns an integer that is the length of the string. Returns NULL if the string is NULL.

The length of a string is determined by its displayable characters, and not necessarily the storage length of the string. For example, a Unicode character requires 16-bits of storage [which might be considered as two characters of storage on some systems] but the actual character length is 1.

CONCAT

This scalar function returns a string that is the concatenation of two characters or strings.

Syntax

CONCAT( string1, string2 )
Parameters
- string
  A character string value or VARCHAR expression result.

Return type
VARCHAR.

Remarks
Returns string2 appended to the end of string1. Returns NULL if either string is NULL.

The || operator ("String operators" on page 271) is identical to this function.

Examples
CONCAT('a', 'b') returns 'ab'.

'a'||'b' returns 'ab'.

See also
"concatList" on page 193 returns a string that is the concatenation of a list of characters or strings.
"concatSet" on page 194 returns an alphabetically ordered set of strings.
"String operators" on page 271 describes the || operator.

GREATEST
This scalar function returns the greatest of a list of expression results.

Syntax
GREATEST( value, value [, value ... ] )

Parameters
- value
  A value to be used for the comparison. All values after the first are converted to the data type of the first.

Return type
Same data type as argument.

Example
Selects the string with the greatest value:
SELECT Greatest(
  'SCHOLAR',
  'SKYLER',
  'SHUELLER')
FROM Students;
Greatest
--------
SKYLER

See also

“LEAST” on page 136 determines the least value from a list.
“MAX” on page 167 returns the maximum value from a set.
“Function types” on page 115 discusses moving sets.

LEAST

This scalar function returns the least value of a list of expressions.

Syntax

LEAST( value, value [, value ... ] )

Parameters

• value
  
  A value to be used for the comparison. All values after the first are converted to the data type of the first.

Return type

Same data type as the argument.

Example

SELECT Least(
  'SCHOLAR',
  'SKYLER',
  'SHUELLER')
FROM Students;

Least
------
SCHOLAR

See also

“GREATEST” on page 135 determines the greatest value from a list.
“MIN” on page 167 returns the minimum value from a set.
“Function types” on page 115 discusses moving sets.

LOWER

This scalar function converts all uppercase characters in a string to lowercase.

Syntax

LOWER( string )
Parameters
- string
  String to convert.

Return type
VARCHAR.

Example
LOWER('Stage Right') returns 'stage right'.

See also
"UPPER" on page 164 converts to all uppercase.

LPAD
This scalar function inserts one or more instances of a string into the start of another string.

Syntax
LPAD( string, length, [ padChar ] )

Parameters
- string
  Character or string to alter.
- length
  The display length of the returned string. Must be zero (0) or greater. When using multi-byte characters, the length is the count of characters that display or print, not the count of multi-bytes.
- padChar
  Character or string to insert. Default is a single space or blank character (' ').

Return type
VARCHAR.

Remarks
Returns a string in the same character type as the string parameter.

When the length is smaller than the length of the string, returns the string truncated to that length.

Examples
LPAD('ABC',6,'x') returns 'xxxABC'.
LPAD('ABC',6,'xo') returns 'xoxABC'.
LPAD('ABC',2,'x') returns 'AB'.
LPAD('ABC',2,'x') returns 'AB'.
See also

“RPAD” on page 161 adds characters to the end of a string.

LTRIM

This scalar function removes characters from the start of a string.

Syntax

LTRIM( wholeString [, setString ] )

Parameters

- wholeString
  
  String to trim.

- setString
  
  Characters to remove; default is a single blank space (" ").

Return type

VARCHAR.

Remarks

Recursively removes all instances of setString from the start of wholeString until wholeString no longer starts with setString, and returns the result.

Examples

LTRIM(' ZZZ') returns 'ZZZ'.

LTRIM('aaaZZZ','a') returns 'ZZZ'.

LTRIM('ababaZZZ','ab') returns 'aZZZ'.

LTRIM('abcabaZZZ','abc') returns 'abaZZZ'.

See also

“RTRIM” on page 162 removes characters from the end of a string.

POSITION

This scalar function returns the position of a character or string within a string.

Syntax

POSITION( sourceForString , searchInString )

Alternate form: POSITION( sourceForString IN searchInString )

Parameters

- sourceForString
  
  String to search for.

- searchInString
  
  String expression result in which to search.
**Return type**

INTEGER.

**Remarks**

Returns the position, starting from 1, of the first instance of sourceForString in the sourceInString result. When CHARACTER_LENGTH(sourceForString) is zero (0), returns 1. Returns NULL when either argument is NULL.

**Examples**

POSITION( 'a' IN 'banana') returns 2.
POSITION( 'ana' IN 'banana') returns 2.
POSITION( 'A' IN 'banana') returns 0.
POSITION( 'M' IN 'banana') returns 0.
POSITION( " IN 'banana') returns 1.
POSITION( NULL IN 'banana') returns NULL.

**See also**

“SUBSTRING” on page 162 extracts a character or substring from a string.

**PRIOR_VALUE**

This scalar function returns the prior value of a column, alias, or expression.

**Syntax**

PRIOR_VALUE( columnName )

**Parameters**

- columnName
  
  Column or alias of any data type to evaluate.

**Return type**

Same data type as argument.

**Remarks**

Returns a NULL if there is no prior value, the first time the function is called on the columnName.

Use PRIOR_VALUE() when the data (data streams) enter the system grouped and ordered.

This function is not permitted in the “WHERE clause” on page 319 of a view definition.
Example

Consider this query that identifies how long a task took to complete [as a percentage of an hour] based on minutes since the previous task completed:

```
SELECT Task, CAST((TIMESTAMP_DIFF(PRIOR_VALUE(Completed),Completed,
                  MINUTE ) /60 , DECIMAL(5,4)
              ) AS Hours
FROM Tasks_Completed
```

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>0.0887</td>
</tr>
<tr>
<td>Initialize</td>
<td>0.1012</td>
</tr>
<tr>
<td>Begin job</td>
<td>0.3243</td>
</tr>
<tr>
<td>Finish job</td>
<td>0.2500</td>
</tr>
<tr>
<td>Clean up</td>
<td>0.1285</td>
</tr>
<tr>
<td>Shut down</td>
<td>0.6667</td>
</tr>
<tr>
<td>Have milk shake</td>
<td>0.6667</td>
</tr>
</tbody>
</table>

See also

“PREV” on page 173 returns a value from the next to last row in a set.

**RPAD**

This scalar function adds one or more instances of a string to the end of another string.

**Syntax**

```
RPAD( string, length, [ padChar ]
```

**Parameters**

- **string**
  Character or string to alter.
- **length**
  The display length of the returned string. When using multi-byte characters, the length is the count of characters that display or print, not the count of multi-bytes.
- **padChar**
  Character or string to append. Default is a single space or blank character (" ").

**Return type**

VARCHAR.

**Remarks**

When length is smaller than the length of string, returns the string truncated to length.
Examples
RPAD('ABC', 6, 'x') returns 'ABCxxx'.
RPAD('ABC', 6, 'xo') returns 'ABCxox'.
RPAD('ABC', 4) returns 'ABC '.
RPAD('ABC', 2, 'x') returns 'AB'.

See also
“LPAD” on page 158 inserts characters to the start of a string.

RTRIM
This scalar function removes characters from the end of a string.

Syntax
RTRIM( sourceString [, setString ] )

Parameters
• sourceString
  String to trim.
• setString
  Characters to remove; default is a single blank space (' ').

Return type
VARCHAR.

Remarks
Recursively removes all instances of setString from the end of sourceString until
sourceString no longer ends with set, and returns the result.

Examples
RTRIM('ZZZ ') returns 'ZZZ'.
RTRIM('ZZZaaa', 'a') returns 'ZZZ'.
RTRIM('ZZZababab', 'ab') returns 'ZZZ'.
RTRIM('ZZZababc', 'abc') returns 'ZZZab'.

See also
“LTRIM” on page 159 removes characters from the start of a string.

SUBSTRING
This scalar function returns a substring of a specified string.
Syntax

```
SUBSTRING( string, position, [ length ] )
```

Alternate forms:
```
SUBSTR( string, position, [ length ] )
SUBSTRING( string FROM position [ FOR length ] )
```

Parameters

- **string**
  Character string to search.
- **position**
  Starting position of the substring, where 1 is the first character at the start of the string, and -1 is the last character. Negative values count backwards from the end of the string. Using zero (0) is the same as using 1. A position not within a string returns an empty string.
- **length**
  Length of the substring to extract. Omitting length returns all characters from position through the end of the string. Specifying a value greater than the remainder of the string returns all characters from position through the end of the string, and pads the difference with space characters to achieve the specified length. A negative value or zero (0) returns an empty string.

Return type

VARCHAR.

Examples

```
SUBSTR('breakfast', 1) returns 'breakfast'.
SUBSTR('breakfast', 0) returns 'breakfast'.
SUBSTR('breakfast', 30) returns '' (empty string).
SUBSTR('breakfast', 1, 2) returns 'br'.
SUBSTR('breakfast', 3, 2) returns 'eak'.
SUBSTR('breakfast', 3, 8) returns 'breakfast'.
SUBSTR('breakfast', -5, 4) returns 'kfas'.
SUBSTR('breakfast', 1, -1) returns '' (empty string).
```

See also

- "CHAR_LENGTH" on page 155 returns the length of a character string.
- "POSITION" on page 159 locates a character within a string.
**UPPER**

This scalar function converts all lowercase characters in a string to uppercase.

**Syntax**

`UPPER( string )`

**Parameters**

- `string`
  Character string (VARCHAR) to convert.

**Return type**

VARCHAR.

**Example**

`UPPER('Volta')` returns 'VOLTA'.

**See also**

"LOWER" on page 157 converts to all lowercase.

---

**Time series and set aggregate**

The time series and set aggregate functions are used for operations that involve moving window sets and sets of numbers.

**See also** MOV_function“ on page 126.

**AVG**

This set function returns the average value (arithmetic mean) of a set of numeric values.

**Syntax**

`AVG( numeric )`

**Parameters**

- `numeric`
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view.

**Return type**

Numeric, same data type as numeric argument.

**Remarks**

Calculates the average numeric in all rows in the referenced view. When using a GROUP BY clause (see “GROUP BY clause” on page 320), the average applies to the numeric in each group.

```
SELECT AVG( pr_price ) "Average price" FROM Products
```

Average price

-------------
Example

The following example uses moving averages to produce results similar to a Moving Average Convergence/Divergence (MACD) indicator. (This example is not a true MACD because it does not use an exponential moving average.) In securities trading, the basic MACD trading rule is to sell when the MACD falls below its nine day average and to buy when the MACD rises above the nine day average. This can be accomplished by defining rules similar to this example:

* Raise SELL when MACD > Nine_Day_MA
* Lower SELL when MACD < Nine_Day_MA
* Raise BUY when MACD < Nine_Day_MA
* Lower BUY when MACD > Nine_Day_MA

To get these values you need two views:

* MACD_Base_View tracks the moving averages for each security symbol in the data stream. The Nine_Day_MA formula repeats the formulas for the other two averages, because you cannot reference an alias in another column of the same view.

```sql
SELECT
    StockQuotes.SYMBOL AS Symbol,
    MOV_AVG(StockQuotes_DataStream.CLOSE, Day, 26, StockQuotes.DATE) AS Twentysix_Day_MA,
    MOV_AVG(StockQuotes_DataStream.CLOSE, Day, 12, StockQuotes.DATE) AS Twelve_Day_MA,
    MOV_AVG((MOV_AVG(StockQuotes.CLOSE, Day, 12, StockQuotes.DATE) - MOV_AVG(StockQuotes.CLOSE, Day, 26, StockQuotes.DATE)), Day, 9, StockQuotes.DATE ) AS Nine_Day_MA
FROM StockQuotes
GROUP BY StockQuotes.SYMBOL
```

* MACD_View contains the last MACD values for each security stored in the base view:

```sql
SELECT MACD_Base_View.Symbol AS Symbol,
        MACD_Base_View.Nine_Day_MA AS Nine_Day_MA,
        (MACD_Base_View.Twentysix_Day_MA - MACD_Base_View.Twelve_Day_MA) AS MACD
FROM MACD_Base_View
```

See also

* "median" on page 196 returns the median (middle) number in a set.
* "mode" on page 197 returns the most frequently occurring number in a set.
* "MOV_AVG" on page 168 returns the moving average for a set.
* "TUMBLE_AVG" on page 175 returns the tumbling average for a set.

**COUNT**

This set function returns the count of rows in a view or set.
Syntax
COUNT(*)

Return type
INTEGER

Remarks
Returns zero (0) if the view or set is empty.
Also known as the count star function.
Rows that include NULLs are counted.

CURRENT
This set function returns a value from the latest or last row in a set.

Syntax
CURRENT(columnName)

Parameters
- columnName
  Column or alias to retrieve.

Return type
Same data type as argument.

Remarks
Returns a value from the latest row in the set based on the timestamp on the data stream. When all rows in the set have the same timestamp, the CURRENT function returns the value from the last row in the set.

Moving set semantics
Cannot be used with a moving or tumbling set.

Example
Gather all stock feed bids and group them by stock symbol. The "current" row is always the last one received, and as such, contains the current bid price:

```sql
SELECT symbol, CURRENT(bid) AS Bid, MAX(bid) AS High, MIN(bid) AS LOW
FROM Stock_feed GROUP BY symbol
```

<table>
<thead>
<tr>
<th>symbol</th>
<th>Bid</th>
<th>High</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>31.25</td>
<td>31.28</td>
<td>30.72</td>
</tr>
<tr>
<td>IBM</td>
<td>80.79</td>
<td>80.04</td>
<td>82.55</td>
</tr>
<tr>
<td>VCLR</td>
<td>22.60</td>
<td>24.42</td>
<td>22.00</td>
</tr>
</tbody>
</table>
See also

“PREV” on page 173 returns a value from the row previous to the current one.

MAX

This set function returns the maximum value from a set.

Syntax

MAX( expression )

Parameters

• expression

   An expression that evaluates to any data type and which cannot reference a rank function. Typically the argument is a column in a view.

Return type

Same data type as expression argument.

Remarks

For Boolean, True is greater than False.

For String, ‘z’ is greater than ‘A’.

Moving set semantics

When used as a MOV function (see “MOV_function” on page 126) returns the maximum value for the moving set.

MOV_MAX( numeric, window, size [,timestampColumn] )

Example

Return the maximum price from all the rows in Sales:

SELECT MAX( price ) FROM Sales;

PRICE
-------
770.00

See also

“MOV_MAX” on page 170 returns the maximum value from a moving window set.

“TUMBLE_MAX” on page 177 returns the maximum value from a tumbling window set.

“MIN” returns the minimum value from a set.

“GREATEST” on page 135 returns the maximum value from a list.

MIN

This set function returns the minimum value from a set.
**Syntax**

\[
\text{MIN( expression )}
\]

**Parameters**

- expression
  
  An expression that evaluates to any data type and which cannot reference a rank function. Typically the argument is a column in a view.

**Return type**

Same data type as expression argument.

**Remarks**

For Boolean, True is greater than False.

For String 'z' is greater than 'A'.

**Moving set semantics**

When used as a MOV_function, returns the minimum value for the moving set.

\[
\text{MOV_MIN( numeric, window, size [,timestampColumn] )}
\]

**Example**

Return the minimum price from all the rows in the set Sales:

```sql
SELECT MIN( price ) FROM Sales;
```

```
PRICE
---------
100.00
```

**See also**

- "MOV_MIN" on page 171 returns the minimum value from a moving window set.
- "TUMBLE_MIN" on page 179 returns the minimum value from a tumbling window set.
- "MAX" on page 162 returns the maximum value from a set.
- "LEAST" on page 136 returns the smallest value from a list.

**MOV_AVG**

This moving set function returns the moving average value (arithmetic mean) of a moving window set of numeric values.

**Syntax**

\[
\text{MOV_AVG( numeric, window, size [,timestampColumn] )}
\]

**Parameters**

- numeric
An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  Duration or count of the window to use for determining the size of the set. Must be an integer greater than zero (0).

- **timestampColumn**
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

Numeric, same data type as numeric argument.

**Remarks**

Returns NULL if the group is empty.

**Example**

Return the average price of all the data streams that arrive within a seven day interval:

\[
\text{MOV_AVG(price, DAY, 7, trade_time) AS Avg_7_day_price}
\]

**See also**

- "Function types" on page 115 for a discussion of moving sets.
- "AVG" on page 164 returns the mean average for a set.
- "TUMBLE_AVG" on page 175 returns the tumbling average for a set.

**MOV_COUNT**

This moving set function returns the count of rows in a moving window set.

**Syntax**

\[
\text{MOV_COUNT( *, window, size [,timestampColumn] )}
\]

**Parameters**

- **window**
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  Duration or count of the window to use for determining the size of the set. Must be an integer greater than zero (0).

- **timestampColumn**
(Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

Return type

INTEGER.

Remarks

Returns zero (0) if the set is empty.

Rows that include NULLs are counted.

When used with a GROUP BY, returns the count of rows in the group set. For more information, see “GROUP BY interactions” on page 129.

Example

Return the count of all the data streams that arrive within the current 8 hour interval:

\[
\text{MOV\_COUNT}(*, \text{HOUR}, 8, \text{trade\_time}) \text{ AS Total}
\]

The previous function is shorthand for this in-line window expression:

\[
\text{COUNT}(*) \text{ AS Total OVER (ORDER BY trade\_time '8' HOUR)}
\]

This expression is equivalent to the following after entering all default values:

\[
\text{COUNT}(*) \text{ AS Total OVER (ORDER BY trade\_time \\
RANGE INTERVAL '8' HOUR PRECEDING \\
REFERENCE OPERATOR)}
\]

The eight hour window begins when the first data stream arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE '2003-03-05 00:00:00.000'.

See also

“Function types” on page 115 discusses moving sets.

“COUNT” on page 165 returns the count of a view or set.

“TUMBLE\_COUNT” on page 176 returns the count of a tumbling window set.

**MOV\_MAX**

This moving set function returns the maximum value from a moving window set.

Syntax

\[
\text{MOV\_MAX( numeric, window, size [,timestampColumn] )}
\]

Parameters

- numeric
An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).

- **timestampColumn**
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

Same data type as expression argument.

**Remarks**

For Boolean, True is greater than False.

For String, 'z' is greater than 'A'.

**Example**

Return the maximum price of all the data streams that arrive within a seven day interval:

```
MOV_MAX(price, DAY, 7, trade_time) AS Max_7_day_price
```

**See also**

- "MAX" on page 167 returns the maximum value from a moving window set.
- "TUMBLE_MAX" on page 177 returns the maximum value from a tumbling window set.
- "MIN" on page 167 returns the minimum value from a set.
- "GREATEST" on page 135 returns the maximum value from a list.
- "Function types" on page 115 for a discussion of moving sets.

**MOV_MIN**

This moving set function returns the minimum value from a moving window set.

**Syntax**

```
MOV_MIN( numeric, window, size [,timestampColumn] )
```

**Parameters**

- numeric
An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).

- **timestampColumn**
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

Same data type as expression argument.

**Remarks**

For Boolean, True is greater than False.

For String, 'z' is greater than 'A'.

**Example**

Return the minimum price of all the data streams that arrive within a seven day interval:

```
MOV_MIN(price, DAY, 7, trade_time) AS Min_7_day_price
```

**See also**

- [MIN](#) on page 167 returns the minimum value from a view or set.
- [TUMBLE_MIN](#) on page 179 returns the minimum value from a tumbling window set.
- [MAX](#) on page 162 returns the maximum value from a set.
- [LEAST](#) on page 136 returns the smallest value from a list.
- [Function types](#) on page 115 discusses moving sets.

**MOV_SUM**

This moving set function returns the sum of a moving window set of numeric values.

**Syntax**

```
MOV_SUM( numeric, window, size [,timestampColumn] )
```

**Parameters**

- numeric
An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).

- **timestampColumn**
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

“C-SQL data types for numeric values” on page 65 same data type as numeric argument. Returns NULL if the set is empty.

**Example**

Total the price of all data streams that arrive in current hour:

```
MOV_SUM(Price, HOUR, 1)
```

Which is shorthand for this in-line window expression:

```
SUM(Price) OVER (RANGE '1' HOUR PRECEDING
    REFERENCE OPERATOR)
```

**See also**

“SUM” on page 174 returns the sum of a view or set.

“TUMBLE_SUM” on page 180 returns the sum of a tumbling window set.

“Function types” on page 115 discusses moving sets.

**PREV**

This set function returns a value from the next to last row in a set.

**Syntax**

```
PREV( columnName )
```

**Parameters**

- **columnName**
  Column or alias of any data type to evaluate.

**Return type**

Same data type as argument.
Remarks
Returns values from the row before the "current" row in a set, where the current row is the latest row in the set based on timestamp of the data stream, or when all have the same timestamp, is last row in the set.

Moving set semantics
Cannot be used with a moving or tumbling set.

Example
Gather all stock feed closing prices and group them by stock symbol. The "current" row is always the last one received, and as such, contains the current closing price. The previous row is the close of the previous day:

```
SELECT symbol, CURRENT(close) AS "Last Trade",
       PREV(close) AS "Prev Cls",
       (CURRENT(close) - PREV(close)) AS Change
FROM Stock_feed
GROUP BY symbol
```

<table>
<thead>
<tr>
<th>symbol</th>
<th>Last Trade</th>
<th>Prev Cls</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>31.25</td>
<td>31.28</td>
<td>-0.03</td>
</tr>
<tr>
<td>IBM</td>
<td>80.79</td>
<td>80.04</td>
<td>0.75</td>
</tr>
<tr>
<td>VCLR</td>
<td>24.42</td>
<td>22.60</td>
<td>1.82</td>
</tr>
</tbody>
</table>

See also
- "CURRENT" on page 166 returns the value from the latest or last row in a set.
- "PRIOR_VALUE" on page 136 returns the prior value of a column, alias, or expression.
- "Function types" on page 115 discusses moving sets.

SUM
This set function returns the sum of a set of numeric values.

Syntax
```
SUM( numeric )
```

Parameters
- numeric
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view.

Return type
Numeric, same data type as numeric argument. Returns NULL if the set is empty.
Moving set semantics

When used as a MOV_function (see "MOV_function" on page 126), returns the moving sum for the moving set.

MOV_SUM( numeric, window, size [,timestampColumn])

Last value in the set

When the moving set size is a single data stream, MOV_SUM() maintains the sum of the last order prices for each customer, for all the customers that placed an order since the system startup time.

SELECT os.os_cust_id, MOV_SUM(os.os_price, EVENT, 1)
   FROM order_status os
   GROUP BY os.os_cust_id

Example

Total the invOnHand column for all rows in the stock table:

SELECT SUM(invOnHand) "Total on hand"
   FROM stock;

Total on hand
-------------
     2

See also

"MOV_SUM" on page 172 returns the sum of a moving window set.

"TUMBLE_SUM" on page 180 returns the sum of a tumbling window set.

TUMBLE_AVG

This tumbling set function returns the average value (arithmetic mean) of a tumbling window set.

Syntax

TUMBLE_AVG( numeric, window, size [,timestampColumn])

Parameters

- numeric
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window
  Determines whether size is the count of data streams in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).
- size
  Duration or count of window to use for determining the size of the set. Must be a positive integer.
- timestampColumn
(Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

Numeric, same data type as numeric argument.

**Remarks**

Returns NULL if the group is empty.

**Example**

Return the average price of all the data streams that arrive within a seven day interval:

```
TUMBLE_AVG(price, DAY, 7, trade_time) AS Avg_7_day_price
```

The previous function is shorthand for this in-line window expression:

```
AVG(price) AS Avg_7_day_price OVER (  
    ORDER BY trade_time RANGE INTERVAL '7' DAY PRECEDING  
    SLIDE)
```

To determine the average price of the previous seven days, not including the current, use a window instead of a TUMBLE_AVG(), for example:

```
AVG(price) AS Avg_prev_7_day_price OVER (  
    ORDER BY trade_time  
    RANGE BETWEEN INTERVAL '8' DAY PRECEDING  
    AND INTERVAL '1' DAY PRECEDING  
    SLIDE INTERVAL '7' DAY  
    INITIALIZE TIMESTAMP '1963-02-18 00:00:00.000'  
    REFERENCE OPERATOR)
```

**See also**

- “[AVG](#)” on page 164 returns the mean average for a set.
- “[MOV_AVG” on page 168 returns the average for a moving window set.
- “[Tumbling windows” on page 375 discusses tumbling window sets.

**TUMBLE_COUNT**

This tumbling set function returns the count of rows in a tumbling window set.

**Syntax**

```
TUMBLE_COUNT( *, window, size [,timestampColumn] )
```

**Parameters**

- `window`
  
  Determines whether size is the count of data streams in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- `size`
Duration or count of window to use for determining the size of the set. Must be a positive integer.

- timestampColumn
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

INTEGER.

**Remarks**

Returns zero (0) if the set is empty.

When using an EVENT window, this function returns an integer less than or equal to the window value.

Rows that include NULLs are counted.

When used with a GROUP BY, returns the count of rows in the group set.

**Example**

Return the count of all the data streams that arrive within an 8 hour interval:

```
TUMBLE_COUNT(*, HOUR, 8, trade_time) AS Total
```

The previous function is shorthand for this in-line window expression:

```
COUNT(*) AS Total OVER (ORDER BY trade_time
               RANGE INTERVAL '8' HOUR PRECEDING SLIDE)
```

Which in turn is equivalent to the following after entering in all default values:

```
COUNT(*) AS Total OVER ( ORDER BY trade_time
               RANGE INTERVAL '8' HOUR PRECEDING
               SLIDE INTERVAL '8' HOUR
               REFERENCE OPERATOR)
```

The eight hour window begins when the first data stream arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE '2003-03-05 00:00:00.000'.

**See also**

"COUNT" on page 165 returns the count of a view or set.

"MOV_COUNT" on page 169 returns the count of a tumbling window set.

**TUMBLE_MAX**

This tumbling set function returns the maximum value from a tumbling window set.

**Syntax**

```
TUMBLE_MAX( numeric, window, size [,timestampColumn])
```
**Parameters**

- **numeric**
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  Duration or count of window to use for determining the size of the set. Must be a positive integer.

- **timestampColumn**
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

Same data type as expression argument.

**Remarks**

For Boolean, True is greater than False.

For String, ’z’ is greater than ’A’.

**Example**

Return the maximum price of all the data streams that arrive within a 1 hour interval:

```
TUMBLE_MAX(price, HOUR, 1, trade_time) AS Max_price
```

The previous function is shorthand for this in-line window expression:

```
MAX(price) AS Max_price OVER (ORDER BY trade_time
RANGE INTERVAL '1' HOUR PRECEDING SLIDE)
```

Which in turn is equivalent to the following after entering in all default values:

```
MAX(price) AS Max_price OVER (ORDER BY trade_time
RANGE INTERVAL '1' HOUR PRECEDING
SLIDE INTERVAL '1' HOUR
REFERENCE OPERATOR)
```

The one hour window begins when the first data stream arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE TIMESTAMP ’2003-03-05 00:00:00.000’.

The function TUMBLE_MAX(price, EVENT, 5) is the shorthand for this complete window:

```
MAX(price) OVER (ORDER BY trade_time
EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
SLIDE 5)
```
See also

"MAX" on page 167 returns the maximum value from a moving window set.

"MOV_MAX" on page 170 returns the maximum value from a tumbling window set.

"MIN" on page 167 returns the minimum value from a set.

"GREATEST" on page 135 returns the maximum value from a list.

"Tumbling windows" on page 375 discusses tumbling window sets.

TUMBLE_MIN
This tumbling set function returns the minimum value from a tumbling window set.

Syntax
TUMBLE_MIN( numeric, window, size [,timestampColumn]
)

Parameters
• numeric
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
• window
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).
• size
  Duration or count of window to use for determining the size of the set. Must be a positive integer.
• timestampColumn
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

Return type
Same data type as expression argument.

Remarks
For Boolean, True is greater than False.

For String, 'z' is greater than 'A'.

Example
Return the minimum price of all the data streams that arrive within a 1 hour interval:
TUMBLE_MIN(price, HOUR, 1, trade_time) AS Min_price

The previous function is shorthand for this in-line window expression:

MIN(price) AS Min_price OVER (ORDER BY trade_time
   RANGE INTERVAL '1' HOUR PRECEDING SLIDE)

Which in turn is equivalent to the following after entering in all default values:

MIN(price) AS Min_price OVER ( ORDER BY trade_time
   RANGE INTERVAL '1' HOUR PRECEDING
   SLIDE INTERVAL '1' HOUR
   REFERENCE OPERATOR)

The one hour window begins when the first data stream arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE TIMESTAMP '2003-03-05 00:00:00.000'.

The function TUMBLE_MIN(price, EVENT, 5) is the shorthand for this complete window:

MIN(price) OVER ( ORDER BY trade_time
   EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
   SLIDE 5
   REFERENCE OPERATOR)

See also

“MIN” on page 167 returns the minimum value from a moving window set.

“MOV_MIN” on page 171 returns the minimum value from a tumbling window set.

“MAX” on page 167 returns the maximum value from a set.

“LEAST” on page 136 returns the smallest value from a list.

“Tumbling windows” on page 375 discusses tumbling window sets.

TUMBLE_SUM

This tumbling set function returns the sum of a tumbling window set of numeric values.

Syntax

TUMBLE_SUM( numeric, window, size [,timestampColumn])

Parameters

• numeric
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

• window
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

• size
Duration or count of window to use for determining the size of the set. Must be a positive integer.

- timestampColumn

(Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the first data stream in the view as the basis.

**Return type**

Numeric, same data type as numeric argument. Returns NULL if the set is empty.

**Example**

This tumbling sum expression sums all the data streams that arrive within a 1 hour interval:

```
TUMBLE_SUM(price, HOUR, 1, trade_time) AS Total
```

The previous function is shorthand for this in-line window expression:

```
SUM(price) AS Total OVER (ORDER BY trade_time
   RANGE INTERVAL '1' HOUR PRECEDING SLIDE)
```

Which in turn is equivalent to the following after entering in all default values:

```
SUM(price) AS Total OVER ( ORDER BY trade_time
   RANGE INTERVAL '1' HOUR PRECEDING
   SLIDE INTERVAL '1' HOUR
   REFERENCE OPERATOR)
```

The one hour window begins when the first data stream arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE TIMESTAMP '2003-03-05 00:00:00.000'.

The function TUMBLE_SUM(price, EVENT, 5) is the shorthand for this complete window:

```
SUM(price) OVER ( ORDER BY trade_time
   EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
   SLIDE 5
   REFERENCE OPERATOR)
```

**See also**

- “SUM” on page 174 returns the sum of a view or set.
- “MOV_SUM” on page 172 returns the sum of a moving window set.
- “Tumbling windows” on page 375 discusses tumbling window sets.

**Time series and statistical**

The time series and statistical functions are used for operations that involve statistics and moving window sets of numbers.

See also “MOV_function” on page 126.
**MOV_STD_DEVIATION**

This moving set function returns sample standard deviation of a moving window set of numbers.

**Syntax**

```sql
MOV_STD_DEVIATION( numeric, window, size [,timestampColumn])
```

**Parameters**

- **numeric**
  
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  
  Determines whether size is the count of data streams in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  
  Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).

- **timestampColumn**
  
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

DOUBLE PRECISION.

**Remarks**

Returns 0 when there is only a single row of input. Returns NULL if the set is empty.

\[
\sqrt{\frac{\sum (X_i - \bar{X})^2}{n}}
\]

The result is computed by using the formula, where \( n \) is the number of elements in the sample and \( \bar{X} \) is the sample mean.

**Example**

Return the standard deviation in salaries for the last year's worth of data streams:

```sql
SELECT MOV_STD_DEVIATION(salary, YEAR, 1) AS "Dev. for the last year"
FROM employees;
```

Which in turn is equivalent to the following after entering in all default values:

```sql
SELECT STD_DEVIATION(salary) AS "Dev. for the last year"
OVER (RANGE INTERVAL '1' YEAR PRECEDING
REFERENCE OPERATOR)
```
MOV_VARIANCE

This moving set function returns the square of the sample standard deviation of a moving window set of numbers.

Syntax

MOV_STD_DEVIATION( numeric, window, size [,timestampColumn] )

Parameters

- numeric
  - An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window
  - Determines whether size is the count of data streams in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).
- size
  - Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn
  - (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

Return type

DOUBLE PRECISION.

Remarks

Returns zero (0) when the expression set contains only one element. Returns NULL if the set is empty.

\[ \sum \frac{(x_i - \bar{x})^2}{n} \]

The result is computed by using the formula \( \sum \frac{(x_i - \bar{x})^2}{n} \), where \( n \) is the number of elements in the sample and \( \bar{x} \) is the sample mean.

Example

Return the variation in salaries for each calendar year:
SELECT MOV_STD_DEVIATION(salary, YEAR, 1) AS "Variation for last year"
FROM employees;

Which in turn is equivalent to the following after entering in all default values:

SELECT VARIATION(salary) AS "Variation for the last year"
OVER (RANGE INTERVAL '1' YEAR PRECEDING REFERENCE OPERATOR)

See also

“VARIANCE” on page 192 returns the variance of a view or set.
“STD_DEVIATION” on page 188 returns a standard deviation.
“TUMBLE_VARIANCE” on page 191 returns the variance of a tumbling window set.
“Function types” on page 115 discusses moving sets.

NTILE

This rank function determines the tier rank of each value in a set with respect to the entire set.

Syntax
NTILE( toRank, tiers )

Parameters
• toRank
  An expression of any data type, and which typically references a column of values to rank.
• tiers
  Count of tiers in which to partition the results; an integer greater than zero (0). When this value is greater than the count of items to rank, all items are given the same rank.

Return type
INTEGER.

Remarks

Returns an integer for each row in the set that represents the tier that the row belongs to, where one (1) is the highest tier that holds the greatest value. When toRank results in NULL, that result is assigned to the lowest rank.

This function cannot be used as an argument in a set function, moving set function, or rank function. For example, SUM(NTILE()) is illegal.

An n tile function ranks rows by attempting to evenly distribute them throughout a fixed set of tiers. For example, when there is a set of six expression results {D, B,
E, C, A, and B) to rank into two tiers, NTILE() assigns each a rank of either 1 (for C, D, and E) or 2 (for A, B, and B). Results with the same value are always placed in the same tier.

When a set of values is not divisible by the tiers, the function evenly distributes any leftover rows into higher-level groups. For example, the following table demonstrates how the letter items are distributed into various counts of tiers:

<table>
<thead>
<tr>
<th>Tiers:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example**

The following query ranks sales of coffee and tea products into six tiers by their sales rankings. The ranking is in sixths, so each product name receives a value from 1 to 6. This example requires that there is one unique entry for each product:

```sql
SELECT prod_name, NTILE( dollars, 6) AS sales_rank
FROM (lineitem INNER JOIN product
       ON lineitem.item_id=product.productid)
WHERE product.classkey IN (1, 2, 4, 5);
```

<table>
<thead>
<tr>
<th>PROD_NAME</th>
<th>SALES_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demitasse M</td>
<td>1</td>
</tr>
<tr>
<td>Xalapa Lapa</td>
<td>1</td>
</tr>
<tr>
<td>Cafe Au Lait</td>
<td>2</td>
</tr>
<tr>
<td>Aroma Roma</td>
<td>2</td>
</tr>
<tr>
<td>Veracruzano</td>
<td>3</td>
</tr>
<tr>
<td>Colombiano</td>
<td>3</td>
</tr>
<tr>
<td>Darjeeling Special</td>
<td>4</td>
</tr>
<tr>
<td>Irish Breakfast</td>
<td>4</td>
</tr>
<tr>
<td>English Breakfast</td>
<td>5</td>
</tr>
<tr>
<td>Earl Grey</td>
<td>5</td>
</tr>
<tr>
<td>Gold Tips</td>
<td>6</td>
</tr>
</tbody>
</table>

**See also**

"RANK" ranks rows within the entire set.
"Function types" on page 115 discusses moving sets.

**RANK**

This rank function determines the rank of each value in a set with respect to the entire set.
### Syntax

RANK( expression )

### Parameters

- expression
  
  An expression of any data type, and which typically references a column.

### Return type

INTEGER.

### Remarks

Returns an integer for each row in the set that is the ranking of the row within the entire set, where the greatest value is ranked 1. When expression results in NULL it is ranked last in the result list. For example, the ranking of (10, NULL, 20) ranks the 10 as 2, the 20 as 1, and NULL as 3.

When the values to be ranked are equal, they are assigned the same rank, and the next rank is skipped. For example values 4.5, 4.5, 1.0 is assigned rank values of 1, 1, and 3.

This function cannot be used as an argument in a "set function", "rank function", or "moving set function". For example, SUM(RANK(...)) is illegal. Nor can RANK() be used on a "stateless view".

When the set contains only one row, RANK() returns 1. For example, RANK(SUM(sales)) = 1.

### Examples

Rank product sales by region:

```sql
SELECT RANK( SUM(sales)) AS R, SUM(sales) AS S, region
FROM product_orders
GROUP BY region
```

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100000</td>
<td>north</td>
</tr>
<tr>
<td>2</td>
<td>50000</td>
<td>south</td>
</tr>
</tbody>
</table>

Rank product sales by product:

```sql
SELECT prod_name, SUM(dollars) AS prod_sales,
RANK( SUM(dollars) ) AS prod_rank
FROM product, lineitem
WHERE lineitem.classkey = product.classkey
AND lineitem.prodkey = product.prodkey
GROUP BY prod_name;
```

<table>
<thead>
<tr>
<th>PROD_NAME</th>
<th>PROD_SALES</th>
<th>PROD_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demitasse Ms</td>
<td>656401.50</td>
<td>1</td>
</tr>
<tr>
<td>Xalapa Lapa</td>
<td>577450.00</td>
<td>2</td>
</tr>
<tr>
<td>Aroma Roma</td>
<td>479330.25</td>
<td>5</td>
</tr>
<tr>
<td>Verona</td>
<td>467234.00</td>
<td>6</td>
</tr>
<tr>
<td>NA Lite</td>
<td>557655.00</td>
<td>3</td>
</tr>
</tbody>
</table>
See also

“NTILE” on page 184 ranks rows and places them in a finite set of tiers.

“Function types” on page 115 discusses moving sets.

RATIO_TO_REPORT

This rank function calculates the ratio of a value to the sum of the values for the entire set.

Syntax

\[
\text{RATIO\_TO\_REPORT}( \text{numeric} )
\]

Parameters

- numeric
  
  Any numeric data type expression, typically a reference to a numeric column.

Return Type

DOUBLE PRECISION.

Remarks

Returns a number for each row in the set that is the ratio of the row to the sum of the entire set. When the expression results in NULL, the function returns NULL. When the sum of the set is zero (0), the ratio is also zero.

This function cannot be used as an argument in a set function, moving set function, or rank function. For example, SUM(RATIO_TO_REPORT(...)) is illegal.

Example

Determine what percentage each product sales is to the total sales of all products for the last 20 weeks:

\[
\begin{align*}
\text{SELECT} & \quad \text{prod\_description DESC}, \\
& \quad \text{SUM(dollars) as sales}, \\
& \quad \text{RATIO\_TO\_REPORT( SUM( li\_amount ) ) * 100 AS ratio\_dollars} \\
\text{FROM} & \quad \text{lineitem, product} \\
\text{WHERE} & \quad \text{lineitem.li\_prod\_id = product.prod\_id} \\
\text{GROUP BY} & \quad \text{prod\_description};
\end{align*}
\]

<table>
<thead>
<tr>
<th>DESC</th>
<th>SALES</th>
<th>RATIO_DOLLARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Widget</td>
<td>896931.15</td>
<td>12.88</td>
</tr>
<tr>
<td>Basket</td>
<td>514830.00</td>
<td>7.39</td>
</tr>
<tr>
<td>Football</td>
<td>507022.35</td>
<td>7.28</td>
</tr>
<tr>
<td>Oil Drum</td>
<td>503493.10</td>
<td>7.23</td>
</tr>
<tr>
<td>Computer</td>
<td>437863.00</td>
<td>6.29</td>
</tr>
<tr>
<td>Chair</td>
<td>429637.75</td>
<td>6.17</td>
</tr>
<tr>
<td>Desk</td>
<td>424215.00</td>
<td>6.09</td>
</tr>
<tr>
<td>Mesh Bag</td>
<td>421205.75</td>
<td>6.05</td>
</tr>
</tbody>
</table>
### STDDEV

This set function returns a sample standard deviation of a set of numbers.

#### Syntax

```
STDDEV( number )
```

#### Parameters
- `number`

  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view.

#### Return type

DOUBLE PRECISION.

#### Remarks

Returns 0 when there is only a single row of input. Returns NULL if the set is empty.

\[
\sqrt{ \frac{\sum (X_i - \bar{X})^2}{n} }
\]

The result is computed by using the formula above, where \( n \) is the number of elements in the sample and \( \bar{X} \) is the sample mean.

#### Example

```
SELECT STDDEV(salary) "Deviation"
FROM employees;
```

<table>
<thead>
<tr>
<th>Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3909.36575</td>
</tr>
</tbody>
</table>

#### See also

- "MOV_STDDEV" on page 182 returns the standard deviation of a moving window set.
- "TUMBLE_STDDEV" on page 189 returns the standard deviation of a tumbling window set.
- "VARIANCE" on page 192 returns the square of the standard deviation.
**SUM_OVER_GROUPS**

A rank function that passes two parameters. The parameters are: a numeric value and an ordering parameter. When the groups are ordered by the ordering parameter, the function returns a running sum of the numeric values.

**Syntax**

```
SUM_OVER_GROUPS([numeric value],[ordering parameter])
```

**Parameters**

- **numeric value**
  The values to sum.
- **ordering parameter**
  The order of the values.

**Return type**

Running sum of the numeric values provided in the arguments.

**Example**

Consider the following data set

```
grp  data
--------
1    1
2    3
3   10
4  200
```

If the following was issued:

```
SELECT grp, SUM_OVER_GROUPS(data, grp) as running_sum
GROUP BY grp
```

The result would be:

```
grp  running_sum
--------
1    1
2    4
3   14
4  214
```

**TUMBLE_STD_DEVIATION**

This tumbling set function returns sample standard deviation of a tumbling window set of numbers.

**Syntax**

```
TUMBLE_STD_DEVIATION( numeric, window, size [,timestampColumn] )
```

**Parameters**

- **numeric**
An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  Determines whether size is the count of data streams in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR).

- **size**
  Duration or count of window to use for determining the size of the set. Must be a positive integer.

- **timestampColumn**
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

DOUBLE PRECISION.

**Remarks**

Returns 0 when there is only a single row of input. Returns NULL if the set is empty.

\[
\sqrt{\frac{\sum (X_i - \bar{X})^2}{n}}
\]

The result is computed by using the formula \(\sqrt{\frac{\sum (X_i - \bar{X})^2}{n}}\), where \(n\) is the number of elements in the sample and \(\bar{X}\) is the sample mean.

**Example**

Return the standard deviation in salaries for each calendar year:

\[
\text{SELECT TUMBLE_STD_DEVIATION(salary, YEAR, 1) AS "Deviation per year"}
\]

\[
\text{FROM employees;}
\]

Which in turn is equivalent to the following after entering in all default values:

\[
\text{SELECT STD_DEVIATION(salary) AS "Deviation per year"}
\]

\[
\text{OVER (}
\]

\[
\text{RANGE INTERVAL '1' YEAR PRECEDING}
\]

\[
\text{SLIDE INTERVAL '1' YEAR}
\]

\[
\text{REFERENCE OPERATOR)}
\]

You can use INITIALIZE to declare a fiscal year. And consider using PARTITION BY to get the deviations for different pay grades. For example:

\[
\text{SELECT STD_DEVIATION(salary) AS "Deviation per year"}
\]

\[
\text{OVER (}
\]

\[
\text{PARTITION BY pay_grade}
\]

\[
\text{RANGE INTERVAL '1' YEAR PRECEDING}
\]

\[
\text{SLIDE INTERVAL '1' YEAR}
\]

\[
\text{INITIALIZE '1963-07-01 00:00:00.000'}
\]

\[
\text{REFERENCE OPERATOR)}
\]
See also

“STD_DEVIATION” on page 188 returns the standard deviation of a view or set.

“MOV(STD_DEVIATION)” on page 182 returns the standard deviation of a tumbling window set.

“VARIANCE” on page 192 returns the square of the standard deviation.

“Tumbling windows” on page 375 discusses tumbling window sets.

**TUMBLE_VARIANCE**

This tumbling set function returns the square of the sample standard deviation of a tumbling window set of numbers.

**Syntax**

```c
TUMBLE_VARIANCE( numeric, window, size [,timestampColumn] )
```

**Parameters**

- **numeric**
  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.

- **window**
  Determines whether size is the count of data streams in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).

- **size**
  Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).

- **timestampColumn**
  (Optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

**Return type**

DOUBLE PRECISION.

**Remarks**

Returns zero (0) when the expression set contains only one element. Returns NULL if the set is empty.

\[
\sum_{i=1}^{n} \left( \frac{(x_i - \bar{x})^2}{n} \right)
\]

The result is computed by using the formula, where \( n \) is the number of elements in the sample and \( \bar{x} \) is the sample mean.
Examples

Return the variation in salaries for each calendar year:

```
SELECT TUMBLE_VARIATION(salary, YEAR, 1)
    AS "Variation for last year"
FROM employees;
```

Which in turn is equivalent to the following after entering in all default values:

```
SELECT VARIANCE(price) AS "Variation for last year"
    OVER ( ORDER BY trade_time
             RANGE INTERVAL '1' YEAR PRECEDING
             SLIDE INTERVAL '1' YEAR
             REFERENCE OPERATOR)
FROM employees;
```

See also

“MOV_VARIANCE” on page 183 returns the variance of a moving window set.

“STD_DEVIATION” on page 188 returns a standard deviation.

“VARIANCE” returns the variance of a view or set.

“Tumbling windows” on page 375 discusses moving sets.

VARIANCE

This set function returns the square of the sample standard deviation of a set of numbers.

Syntax

```
VARIANCE( numeric )
```

Parameters

- numeric

  An expression that evaluates to a numeric and which cannot reference a rank function. Typically the argument is a column in a view.

Return type

DOUBLE PRECISION.

Remarks

Returns zero (0) when the expression set contains only one element. Returns NULL if the set is empty.

\[
\sum \left[ \frac{(x_i - \bar{x})^2}{n} \right],
\]

where \( n \) is the number of elements in the sample and \( \bar{x} \) is the sample mean.
Example

```
SELECT VARIANCE(salary) "Variance"
FROM employees;

Variance
----------
15283140.5
```

See also

- "MOV_VARIANCE" on page 183 returns the variance of a moving window set.
- "STD_DEVIATION" on page 188 returns a standard deviation.
- "TUMBLE_VARIANCE" on page 191 returns the variance of a tumbling window set.

Functions provided as sample UDFs

These functions are provided as sample UDFs.

To use one of these functions, you must load the function from the directory manifest in the Cognos Real-time Monitoring program location. The path is similar to `realtime\webcontent\sdk\udf\jar\com\cognos\obi\manifest`. For more information, see Chapter 35, “User-defined functions,” on page 333.

**concatList**

This scalar function returns a string that is the concatenation of a list of characters or strings.

**Syntax**

```
concatList( string1, string2 [, ... stringN ] )
```

**Parameters**

- `string`
  
  An expression that evaluates to a VARCHAR

**Return type**

VARCHAR.

**Remarks**

Returns string2 appended to the end of string1, string3 appended to string2, and so on.

Ignores NULL values unless all values are NULL, in which case it returns an empty string.

**Examples**

```
concatList('a','b','c') returns 'abc'.
```
See also

"CONCAT" on page 155 returns a string that is the concatenation of two characters or strings.

"concatSet" returns an alphabetically ordered set of strings.

"String operators" on page 271 describes the || operator.

**concatSet**

Returns an alphabetically ordered set of strings.

**Syntax**

```
concatSet( stringExp )
```

**Parameters**

- `stringExp`
  
  An expression that evaluates to a VARCHAR. Typically the argument is a column in a view.

**Return type**

VARCHAR.

**Remarks**

Returns a string that is the ordered set of all the strings passed into the function.

Ignores NULL values unless all values are NULL, in which case it returns an empty string.

**Examples**

Consider this statement:

```
SELECT concatSet(item) AS item_list FROM GroceryList
```

If the items in GroceryList are presented as follows in this order:

- 'banana'
- 'egg'
- 'apple'
- 'donut'
- NULL
- 'carrot'

The order in item_list in the new view is:

- 'apple,banana,carrot,donut,egg'

Subsequently, if 'bagel' is added to GroceryList, the new order in the new view is:

- 'apple,bagel,banana,carrot,donut,egg'
See also

"CONCAT" on page 155 returns a string that is the concatenation of two characters or strings.

"concatList" on page 193 returns a string that is the concatenation of a list of characters or strings.

"String operators" on page 271 describes the || operator.

gammaDist
This scalar function returns the gamma distribution of a value.

Syntax

\[
gammaDist(\text{number, alphaNumber, betaNumber, isCumulative})
\]

Parameter

- number Positive number to evaluate, can be zero (0).
- alphaNumber Alpha parameter (positive number, can be zero) to the gamma distribution equation.
- betaNumber Beta parameter (positive number, can be zero) to the gamma distribution equation.
- isCumulative Boolean that determines the form of the function of number based on alphaNumber and betaNumber:
  - TRUE uses the cumulative distribution function.
  - FALSE uses the probability mass function.

Return type

DOUBLE PRECISION.

Remarks

When alphaNumber is one (1), returns an exponential distribution.

When alphaNumber is a positive integer, the result is an Erlang distribution.

Example

Populate a view with the probability mass for a gamma distribution of alpha=9 and beta=2:

\[
\text{SELECT TimeToFail,}
\quad \text{gammaDist}(\text{TimeToFail, 9, 2, FALSE}) \text{ AS GammaDist}
\]

FROM UnitTests
ORDER BY TimeToFail

See also

"logNormDist" returns the cumulative lognormal distribution of a value.

logNormDist
This scalar function returns the cumulative lognormal distribution of a value.
**Syntax**

\[ \text{logNormDist}( \text{number}, \text{meanNumber}, \text{stdNumber} ) \]

**Parameters**

- **number**
  Value to evaluate.
- **meanNumber**
  Mean average of ln(number).
- **stdNumber**
  Standard deviation of ln(number).

**Return type**

DOUBLE PRECISION.

**Remarks**

Returns the cumulative lognormal distribution of a value, where ln(number) is normally distributed with mean and standard deviation.

**median**

This set function returns the median (middle) number in a set.

**Syntax**

\[ \text{median}( \text{numericExp} ) \]

**Parameters**

- **numericExp**
  An expression that evaluates to numeric and which cannot reference a rank function. Typically the argument is a column in a view.

**Return type**

INTEGER when all results of numericExp are integer; otherwise DOUBLE PRECISION when any of the results are decimal.

**Remarks**

This function sorts the values in the set and then returns the median of the ordered set.

When the count of values is odd, the median is the middle number of the set. For example, the median of 2,1,5 is 2: the middle value of the ordered set.

Otherwise, when the count is even, the median is the average value of the two middle numbers in the set. For example, the median of 2,1,5,4 is 3: the average of 2 and 4. Further, when the result of the average is a decimal value, the result is “floored” to the integer: the median of 2 and 3 is 2, which is floor(2.5).

Ignores NULL values.
Examples

Consider this statement:
```
SELECT median(Value) AS MedianV FROM NumberList
```

The result is 1.5 (the average of 1.0 and 2.0 after ignoring the NULLs) when the items in NumberList are presented in this order:
```
3.0
NULL
0.0
2.0
1.0
NULL
NULL
```

The result is 1 if the set is
```
1
2
```

Because the values are integers, the result must also be an integer. The average, which is 1.5, is floored to 1.

See also

“AVG” on page 164 returns the mean average value of a set.

“mode” returns the most frequently occurring number in a set.

mode

This set function returns the most frequently occurring number in a set.

Syntax

```
mode( numericExp )
```

Parameters

- numericExp
  
  An expression that evaluates to numeric and which cannot reference a rank function. Typically the argument is a column in a view.

Return type

Same type as numericExp result.

Remarks

When multiple different values occur with the same frequency, mode() returns the first one it encountered. See the example.

Ignores NULL values.
Examples

Consider this set of numbers, fed into mode() in this order:

1
3
4
1
3

The mode() function returns 1 because it occurs with the most frequency, and is encountered before 3, which occurs with the same frequency. Had the set been fed into mode() in reverse order, it returns 3.

For this set of numbers, mode returns 1.0:

1
3
NULL
2.0
NULL
NULL
1

The NULLs are ignored, and the 2.0 causes mode() to return a DOUBLE PRECISION value.

See also

"AVG" on page 164 returns the mean average number in a set.

"median" on page 196 returns the median (middle value) for a set.

yield

This function computes the yield given a set of test results (fail=0, pass=1) and calculates the yield. That is, total passed/total units. Null values are ignored.

Syntax

yield(pass, unit)

Parameters

• pass
  Indicates whether to calculate the yield based on the number of units that pass or the number of units that fail. Specify pass to use the number of units that pass the test to calculate the yield. Specify fail to use the number of units that fail the test to calculate the yield.

• unit
  Specifies the total number of units to test.

Return type

Returns NULL if the yield is less than zero; otherwise, DOUBLE PRECISION.
Example

If SELECT YIELD(pass, 4) FROM E is passed the following values:

0, 1 1, 0 0, 1 NULL, NULL

The result is 0.5 because test 1 and 3 pass but test 2 fails. Test 4 is ignored.

If SELECT YIELD(fail, 4) FROM E is passed the same values, the result is 0.25 because only one of the four tests passes.
Chapter 16. HTTP Post

Data stream tables for HTTP Post receive events from an HTTP Post action, either as the result of an HTML form sent from a browser or from data encoded in a URL that connects to the table.

How HTTP Post works

The event data for HTTP Post arrives as part of a form submitted through HTTP. The internal agent extracts the fields from the form and streams each Post into the data stream table.

The HTTP Post can occur as the result of a user that submits an HTML form that contains <INPUT> fields, or the post can be created by an application that uses HTTP as its communication protocol.

Data stream tables for HTTP Post

A data stream table for HTTP Post receives new events from an HTTP Post action, which is usually the result of an HTML form sent from a browser.

In an HTML form, each <INPUT> element maps to a column in the data stream table. Event data can also be published in the URL that passes the fields to the system. For examples, see “Post to a data stream table from HTTP Post” on page 204.

Before creating a data stream table for HTTP Post, you must have Create permission for tables. (For more information about permissions, see “Granting permission to create objects” on page 278). The following table lists the attributes for a data stream table for HTTP Post.
Table 60. Data stream table attributes for HTTP Post

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the data stream table. The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which the data stream table is to be stored. Use the Choose Folder button to select or create a folder. If you do not specify a folder, the default is Public Folder.</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Post To URL</td>
<td>URL in which to send the posted information. For examples, see “Post to a data stream table from HTTP Post” on page 204.</td>
</tr>
<tr>
<td>Process data in the order of arrival</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn off this attribute. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
<tr>
<td>Disable data stream after this number of consecutive errors</td>
<td>Disables the data stream when a consecutive count of errors occurs. For example, if set to 5, the data stream is disabled after 5 consecutive errors. However, if 4 errors occur, then no errors are followed by 2 more errors, the data stream remains enabled. The default is Do not disable.</td>
</tr>
<tr>
<td>Column Information</td>
<td>The Column Information tab defines how to map the fields from HTTP Post into columns in the data stream table. There is one column for every field in the data stream table. For more information, see “HTTP Post column information” on page 203.</td>
</tr>
<tr>
<td>Clear State Interval</td>
<td>This tab contains several options for clearing persisted event data. For more information, see “Clear state interval” on page 54.</td>
</tr>
</tbody>
</table>

Creating a data stream table for HTTP Post

You can create a data stream table from HTTP Post.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > Data Stream.
4. In the New Data Stream window, select Single Data Stream.
5. Select HTTP Post as the source type for the data stream table.
6. Assign a name and define the columns of the data stream table in the Column Information tab.
7. Save the HTTP Post table as enabled.

**HTTP Post column information**

The Column Information tab defines how to map the fields from the HTTP Post message into columns in the data stream table.

There is one field for every column in the data stream table, each with the attributes described in the following table:

*Table 61. HTTP Post column information*

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Name</td>
<td>Name of the column in the data stream table.</td>
</tr>
<tr>
<td>Message Name</td>
<td>Name of the field in the message. On an HTML form, it is the NAME attribute assigned to each form element. For examples, see “Post to a data stream table from HTTP Post” on page 204. When mapping a Flat File field, the name for each embedded field is N/A and uneditable.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Data type of the column in the data stream table.</td>
</tr>
<tr>
<td>Format</td>
<td>(Optional) Format of the column in the data stream table for VARCHAR (string) and DECIMAL values.</td>
</tr>
</tbody>
</table>

Each field in the message can be a simple field that maps directly to a column of the data stream table, or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table. Complex fields are treated as flat files in either delimited (CSV), fixed-width, or XML formats. For more information, see Chapter 13, “Flat files,” on page 93.

Message fields can contain more than one row of data. However, each row is part of the same event, unlike flat file imports that treat each row in the file as a unique event.

**Adding a message field**

You can add a message field to provide more information.

For more information about source types, see:
- “Fixed-width files” on page 100
- “Delimited files” on page 99
- “XML file support” on page 100

For more information about sending data to flat file fields, see “Multiple events of input” on page 206.
For more information about flat file attributes, see “Data stream tables from flat files” on page 94 for details.

**Procedure**

1. On the **Column Information** tab, click **Add Flat File Field**.
2. Choose the flat file type of the message field.
   - For fixed-width and delimited files, you have the option of identifying a sample file from a real data file to assist in mapping the columns. Data from this file appears in the next step to assist you as you map the event data into the table.
3. For fixed-width files, define the positions of the data columns with the **Set Field Widths** box.
4. Identify the flat file attributes.
5. Define the format-specific column information.
6. Click **Save Data Stream**.

**Editing the definition of a message field**

You can edit the definition of a message field.

**Procedure**

1. Open an existing data stream.
2. Click **Edit**.
3. On the **Column Information** tab, under **Formatting**, change the value to `<Change Formatting>`.
   - When editing a message field, the sample file option for delimited and fixed-width file types is not available.

---

**Post to a data stream table from HTTP Post**

Most events from HTTP Post are generated from an HTML form. When defining the event, you define one column for each named `<INPUT>` element.

For example, consider this HTML form:

```html
<FORM action="http://.../now/postservlet?eventname=Example"
   method="post">
  <p>
    <LABEL for="name">Name: </LABEL>
    <INPUT type="text" name="name"/><BR>
    <LABEL for="name">Date (yyyy-mm-dd): </LABEL>
    <INPUT type="text" name="date"/><BR>
    <LABEL for="amt">Amount: </LABEL>
    <INPUT type="text" name="amt"/><BR>
    <LABEL for="switch">Switch: </LABEL>
    <INPUT type="radio" name="switch" value="FALSE" checked >Off
    <INPUT type="radio" name="switch" value="TRUE">On<BR>
    <INPUT type="submit" value="Send">
    <INPUT type="reset">
  </p>
</FORM>
```
The four form fields map to the four columns of the data stream table shown in the following illustration:

![Data Stream Table](image)

Figure 18. Four form fields that map to the four columns of the data stream table

The date field maps to a VARCHAR, not a TIMESTAMP. In the views that are derived from this data stream, you cast the values to a date-time, similar to the following example:

```
CAST(httpDataStream."Date" AS TIMESTAMP)
```

**Post to message fields**

To pass data into a message field, you can either embed the information in the URL or use an HTML `<TEXTAREA>` element and enter the flat-file data into that field.

For example, your HTML form might have the following declaration:

```
<LABEL for="flatfile">Flat file text: </LABEL>

  <TEXTAREA name="MessageField" rows="20" cols="80"></TEXTAREA>
```

In the browser, you can either enter the data manually or copy the data from a flat file and paste it into the field. Remember that the data must be in the format of the declared Flat File Field, such as delimited.

For information about embedding the information in the URL, see "Values posted in the URL."

**Attention:**

- When the data stream table contains more than just the message field column, you can enter only one row of data into the `<TEXTAREA>` field. If the message field is the only column, then you can enter multiple rows of data from the event.
- Multiple rows passed through a `<TEXTAREA>` element are considered part of the same event, unlike a text file where each row is a unique event.

**Values posted in the URL**

When passing the field values directly in the URL, name and assign a value to each, separating them with ampersands (&).

For example:

```
...?eventName=Example&name="Skyler"&date=2003-03-05&amt=9.21&switch=TRUE
```
However, if passing the values to a delimited flat file field, name the field and separate the values with the separator character (which is usually a comma). For example:

```plaintext
...?eventname=Example&msgFile="Skyler",2003-03-05,9.21,TRUE
```

**Multiple events of input**

To send multiple events to a flat file field, you can separate them with the \%0D\%0A (the MIME transmission for an end-of-line: "CR LF").

For example:

```plaintext
...="Skyler",2003-03-05,9.21,TRUE\n\nMike",1963-02-18,9.01,FALSE
```

If you intend to send multiple lines, the **Flat File** field must be the only field in the column list of the data stream table. When the list includes other columns, only one line of input is permitted.
Chapter 17. Java Messaging Service

Java Messaging Service (JMS) provides access to messages produced by Java applications.

The producer application publishes messages to topics or queue destinations that IBM Cognos Real-time Monitoring agent subscribes to. Each new published topic or queue message is mapped to a new event in the associated data stream table.

IBM Cognos Real-time Monitoring JMS agents communicate with JMS topic factories or queue factories managed by web application servers. When you define the agent, you tell it how to connect to the factory. When you define a JMS data stream table, you tell it to subscribe to a JMS topic or queue managed by the factory that the agent talks to. Then, when the topic publishes a new message, or when the queue receives a new message, the agent receives it and passes it to the data stream table.

Cognos Real-time Monitoring JMS agents support JMS MapMessage and TextMessage body types for both topic and queue messages. This JMS MapMessage type consists of name-value pairs, where the names are strings and the values are wrappers to Java types. For more information, see “Java Messaging Service data types” on page 210.

JMS data streams

JMS data streams receives new event data from a Java application that publishes messages to the topic or sends messages to the queue that the table subscribes to. Each new topic or queue message is a new event in the table.

If the message contains a complex string that is CSV (delimited) or fixed-width text, it is helpful to have a sample file that contains data in the format of the actual event string. You can use this sample when you create the data stream to ensure that the fields map correctly into the data stream table by seeing how the data lines up in the columns.

The following table describes the attributes for a JMS data stream.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the data stream table.</td>
</tr>
<tr>
<td></td>
<td>The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which the data stream object is to be stored. Use the Choose Folder button to select or create a folder. If you do not specify a folder, the default is Public Folder.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that may contain any text characters.</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Process data in the order of arrival</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn this off. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
<tr>
<td>JMS Agent</td>
<td>An existing JMS agent that connects to the JMS message stream. Create a JMS Queue or JMS Topic agent with the Workbench Activities button. For more information, see “Java Messaging Service queue agents” on page 29 or “JMS topic agents” on page 32.</td>
</tr>
<tr>
<td>Topic or Queue</td>
<td>Identifies the topic or queue on which the message is being sent and defined by the message publisher. This is a JNDI address similar to com.obi.myjmstopic on Oracle WebLogic and topic/com.obi.myjmstopic on JBoss.</td>
</tr>
</tbody>
</table>
**Attribute** | **Description**
--- | ---
**Message selector** | A Boolean expression that puts a filter condition on the messages the publisher sends. The syntax of the condition is the same as that of the SELECT command’s WHERE clause (see “WHERE clause” on page 319). For example, this filter only accepts messages where the Supplier property contains one of three values: Supplier IN ('Xyz, Corp', 'Ink, Inc', 'Gizmos')

**Column Information** | The **Column Information** fields define how to map the fields from the JMS message into columns in the data stream table. There is one column for every field in the data stream table. For more information, see “Java Messaging Service column information” on page 210.

**Clear State Interval** | This tab contains several options for clearing persisted event data. For more information, see “Clear state interval” on page 54.

---

**Note:** The JMS agent supports JMS MapMessage and TextMessage body types only. MapMessage consists of name-value pairs, where the names are strings and the values are wrappers to Java types. TextMessage consists of a single unnamed text string, which can be interpreted as a fixed-width or delimited file or an XML file. See “Java Messaging Service data types” on page 210 for details.

**Creating a data stream table from a Java Messaging Service connection**

You can create a data stream table from a JMS connection.

**Before you begin**

Before creating a data stream from a JMS agent, you must have Create permission for lookup tables and data streams and Read-only access permission on the agent that provides data to the table. You must also obtain access to a JMS topic or queue as identified by the JNDI location of the address factory.

For more information, see “Granting permission to create objects” on page 278.

**Procedure**

1. Open the Workbench tab.
2. Click Activities.
3. Click **Create New > Data Stream**.
4. Select JMS as the source type.
5. Define the attribute values of the data stream table.
6. On the **Column Information** tab, define the columns of the data stream table.
7. Save the JMS data stream as enabled.
Java Messaging Service column information

The column information fields define how to map the fields from the JMS message into columns in the data stream table. There is one field for every column in the data stream table.

Each field in the message can be a simple field that maps directly into an event column, or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table. Complex fields are treated as flat files in either delimited, fixed-width, or XML formats. See Chapter 13, “Flat files,” on page 93 for detailed descriptions of these file types.

Each column in the data stream table has the attributes described in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Name</td>
<td>Name of the column in the data stream table.</td>
</tr>
<tr>
<td>Message Name</td>
<td>Name of the field in the message. When mapping a message field, the name for each embedded field is N/A and uneditable. Specifying a message name indicates that the message is a map message and the name is the name of the message. If you do not specify a message name, you are asked to confirm that the message is a JMS text message payload when you save.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Data type of the column. For more information, see “Java Messaging Service data types.”</td>
</tr>
<tr>
<td>Format</td>
<td>Format of the column for VARCHAR (string) and DECIMAL values. The Format attribute is optional.</td>
</tr>
</tbody>
</table>

Java Messaging Service data types

The JMS mapped message data types map to IBM Cognos Real-time Monitoring.

The following example shows the JMS mapped message data types mapped to Cognos Real-time Monitoring data types. For more information, see Chapter 9, “Data types,” on page 63.

<table>
<thead>
<tr>
<th>Java data type</th>
<th>Cognos Real-time Monitoring data type</th>
<th>Java data type</th>
<th>Cognos Real-time Monitoring data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Boolean</td>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>byte</td>
<td>Integer</td>
<td>float</td>
<td>Double</td>
</tr>
<tr>
<td>short</td>
<td>Integer</td>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Varchar</td>
<td>String</td>
<td>Varchar</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>byte[]</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
Chapter 18. Java Database Connectivity

JDBC (Java Database Connectivity) is a Java interface for application programming that makes it possible to access standard SQL databases from Java programs.

IBM Cognos Real-time Monitoring uses JDBC to perform the following actions:
- Retrieve lookup table data from a relational database (DBMS), as described in "Java Database Connectivity tables" and "Java Database Connectivity agents" on page 221.
- Access the metadata database that Cognos Real-time Monitoring uses to store object and state definitions.
- Allow other Java applications to access the business view data in memory. For more information, see Chapter 19, "Java Database Connectivity access to view and cube data," on page 227.

Java Database Connectivity tables

JDBC data stream and lookup tables receive their data from external relational database systems (DBMS). The data is retrieved by either making a query on the database or by calling a stored procedure in the DBMS.

Lookup tables

For lookup tables, new data is retrieved only when a new event requires it. Then the agent passes the query data to the DBMS, which returns the result from DBMS.

For more information about lookup tables, see Chapter 20, “Lookup tables,” on page 247.
Example of a lookup table

Consider a view that joins a data stream table with a lookup table, such as the following example:

```sql
SELECT DataStream.ID, Lookup.Name, Lookup.BDate AS Birth_Date
FROM DataStream, Lookup WHERE DataStream.ID = Lookup.ID
```

The lookup table output used by this view might look like the following.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>JDBC Data Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING</td>
<td>Varchar</td>
</tr>
<tr>
<td>BDate</td>
<td>DATE</td>
<td>Timestamp</td>
</tr>
</tbody>
</table>

Creating a Java Database Connectivity source for a data stream table

You can create a JDBC source for a data stream table.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the **Workbench** tab.
2. Click **Activities**.
3. Click **Create New > Data Stream**.
4. Select either **Single Data Stream** or **Consolidated Data Stream**.
5. Select JDBC as the source type for the data stream.
6. Select either **Query** source or **Stored Procedure** source.
   - A stored procedure source calls a stored procedure in the DBMS to locate the data.
   - A query source makes a SELECT SQL query on the database in the native database language. Enter the SELECT statement in the query field. When you click **Continue**, the Workbench issues the query to validate the query and to determine the return columns.
7. Save the JDBC source as enabled.
Creating a Java Database Connectivity source for a lookup table

You can create a JDBC source for a lookup table.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > Lookup Table.
4. Choose JDBC as the source type for the lookup table.
5. Click Browse to select the JDBC connection.
6. Choose either Query or Stored Procedure as a source.
   - A stored procedure source calls a stored procedure in the DBMS to locate the data.
   - A query source makes a SELECT SQL query on the database in the native database language. Enter the SELECT statement in the query field. When you click Continue, the Workbench issues the query to validate the query and to determine the return columns.
7. Save the JDBC source as enabled.

Data stream tables

For data stream tables, the agent periodically polls the DBMS to see if new events are available, then retrieves them for inclusion in the data streams table. Each event returned is processed individually, regardless of the count of events returned as a result of the polling query.

Before creating a data stream to a JDBC source table, you need:

- An agent
  - An existing JDBC agent defined with sufficient access rights to query the database, or call the stored procedure. Create an agent by clicking the Activities button and selecting Create New then Data Stream from the drop-down lists. For more information, see "Java Database Connectivity agents" on page 221.
- For queries
  - The schemas of the tables to query.
- For query data streams
  - A column in the source table must be an incrementing value that identifies when new events are available. For more information, see "Polling the Java Database Connectivity source" on page 216.
- For stored procedures
To define the procedure in the RDBMS and provide a list of the input and (result set) output fields, and their data types. For more information, see “Stored procedure source” on page 218.

- Permissions
  Create permissions (see Chapter 23, “Permissions,” on page 275) for tables (see “Granting permission to create objects” on page 278) and Read-only access permission on the agent that provides data to the table.

A JDBC table has the attributes described in the following table:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the table. The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Process data in the order of arrival (consolidated data streams only)</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn off this attribute. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
<tr>
<td>JDBC Agent</td>
<td>An existing JDBC source agent that accesses an RDBMS. Create an agent by clicking the Activities button and selecting Create New then Data Stream from the drop-down lists. For more information about agents, see “Java Database Connectivity agents” on page 221.</td>
</tr>
<tr>
<td>JDBC Query</td>
<td>A SELECT statement made against the database in the native database language. If you change the query, click Resubmit Query to validate it. You cannot save with an invalid query. For more information about the SELECT command, see the reference documentation for the DBMS.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Disable lookup table after this number of consecutive errors</td>
<td>Count of consecutive errors to receive before the system disables the lookup table. Once disabled, a lookup table must be re-enabled manually.</td>
</tr>
<tr>
<td>Treat all rows in the result set as a single event</td>
<td>All rows returned in the result set are considered a single event. Otherwise, every row returned from the table is considered a separate event.</td>
</tr>
</tbody>
</table>

**Field Information**

Columns to populate in the data stream or lookup table. The field names are derived from the result of the JDBC query. When the system validates the query, it populates this field list and identifies the JDBC data type of each return value. You specify the associated data type of the column in the table and the format for the data type (when applicable). The field names are the same as defined in the DBMS schema unless you alias them with the AS operator in the select list of the SELECT statement.

**Event Key** (data streams only)

For fixed-width and delimited files, identifies key field columns for multi-row events. For more information, see “Multi-row events” on page 97.

**Index** (lookup tables only)

Builds an index for the column when checked. It is critical that you select the right index in order to have good performance when prefetch is enabled for data caching in join conditions or key fields in dimensions. Select Index for those columns you use in join conditions. If you are not using prefetch caching, selecting Index has no effect.

**Caching** (lookup tables only)

Stores query results in memory and future requests retrieve data from memory, reducing the number of queries to the DBMS. For more information, see “Caching lookup table queries” on page 250.

**Polling** (data streams only)

How frequently to call the stored procedure or to query the DBMS for new events. For more information, see “Polling the Java Database Connectivity source” on page 216.

**Clear State Interval**

Options for clearing persisted event data. For more information, see “Clear state interval” on page 54.

**Upsert** (data streams only)

Options for enabling the upsert functionality for JDBC data streams. Upsert combines the actions of updating, inserting, and deleting data from data stream tables. For more information, see “Upsert” on page 217.

---

**Query source**

A query source makes a SELECT query on the database in the source DBMS.
Queries are used for both lookup tables and data stream tables, and are in the native syntax used by the DBMS. For specific syntax information, see the DBMS documentation.

**Sybase limitations**
When making a query to a Sybase database, be aware of these limitations.

- All names, including tables and columns, are case-sensitive.
- All queries must be in the SELECT * FROM table form; you cannot include any SELECT clauses. To filter the results, load them into a business view, then filter that view.

**Example of a data stream table that uses a polling query**

This topic provides an example of a data stream table that uses a polling query.

Consider the following query for a data stream table.

```sql
SELECT * FROM sales
```

The resulting field information might look like the following example:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>JDBC Data Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales_ID</td>
<td>INT</td>
<td>Integer</td>
</tr>
<tr>
<td>Product_ID</td>
<td>INT</td>
<td>Integer</td>
</tr>
<tr>
<td>Customer_ID</td>
<td>STRING</td>
<td>Varchar</td>
</tr>
</tbody>
</table>

The polling incrementing field is most likely "Sales_ID." For more information, see "Polling the Java Database Connectivity source."

**Polling the Java Database Connectivity source**

Polling tells the object how frequently to call the stored procedure or to query the DBMS for new events.

Polling has the parameters described in the following table:

*Table 65. JDBC source polling parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polling on interval</td>
<td>How often to call the procedure or issue the query.</td>
</tr>
<tr>
<td>Disable data stream after this number of consecutive errors</td>
<td>Disables the object (stops polling) after consecutive errors occur. By default, the polling stops after five consecutive errors. To re-enable the object, change its object status (see &quot;Object status&quot; on page 253) to enabled. Set this option to zero (0) to avoid disabling the Object automatically.</td>
</tr>
<tr>
<td>Incrementing Field (data stream queries only)</td>
<td>Identifies the column in the source table that contains a value that increments for every event. For example, if the table that is queried contains unique, ascending ID values, that field is the one used by the query that uses the logic &quot;where ID greater that maximum ID from last query.&quot;</td>
</tr>
</tbody>
</table>
Table 65. JDBC source polling parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Value (data stream queries only)</td>
<td>The beginning values assigned to the incrementing field; however, this initial value is not included in the result. The first value included in the result is the next smallest value greater than the initial value. For example, you if you specify an initial ID value of 500, the first value included is 501. For subsequent queries, the values are greater than the biggest incrementing field value returned from the last query. If you are using a SQLServer 2005 for the lookup table database and a 2005 Microsoft driver, initial date values must be in the form yyyy-mm-dd. Otherwise, an error might be generated.</td>
</tr>
</tbody>
</table>

**Upsert**

The upsert functionality combines the actions of updating, inserting, and deleting data from data stream tables. You can enable this feature for JDBC data streams, flat file streams, JMS streams, and others.

The upsert functionality dynamically updates your database. As new events are added to the database tables, old events are replaced or deleted, and the database always contains only the required data.

To enable the upsert functionality, you need a JDBC agent specifically for upsert. This agent manages the data required for the upsert functionality. You can create the upsert JDBC agent before or when creating the data stream. By default, upsert uses the same JDBC agent that you specify for the creation of a JDBC data stream.

You enable the upsert functionality when creating the JDBC data stream. Do not enable this functionality when the data stream uses the event key.

The following parameters are associated with this functionality:

Table 66. Upsert parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Updates/Inserts/Deletes (Upsert)</td>
<td>Enables the upsert functionality for a JDBC data stream.</td>
</tr>
<tr>
<td>JDBC Agent</td>
<td>Specifies the JDBC Agent. You can either accept the default agent that you selected for the data stream, or click Browse and specify a different agent.</td>
</tr>
<tr>
<td>Table Name for Upsert Data Storage</td>
<td>Specifies the name of the table that stores the data required for the upsert functionality. This table is created in the database associated with the JDBC agent. Delete this table if you delete or disable the upsert stream.</td>
</tr>
</tbody>
</table>
Table 66. Upsert parameters (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum age of upsert data</td>
<td>Specifies the period to support upsert.</td>
</tr>
<tr>
<td></td>
<td>The update or delete action is not valid for an event older than this period.</td>
</tr>
<tr>
<td>Key Column</td>
<td>Specifies the column that is used to identify the events to be deleted or updated.</td>
</tr>
<tr>
<td>Action Column</td>
<td>Specifies the column that contains the &quot;I/D/U&quot; flags for the Insert/Delete/Update actions.</td>
</tr>
<tr>
<td></td>
<td>You can select multiple columns. However, do not select the column that is already selected as Action Column.</td>
</tr>
</tbody>
</table>

**Stored procedure source**

The JDBC stored procedure source requires specific attributes.

These attributes are described in the following table:

Table 67. JDBC stored procedure source attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure name</td>
<td>Name of the stored procedure in the DBMS. JDBC data streams do not support multiple result set or stored procedure output parameters. For Oracle, this means that only Oracle functions are supported because only they return a single result.</td>
</tr>
<tr>
<td>Outputs</td>
<td>Columns in the data stream or lookup table, their data types, and optional formatting. The procedure returns a result set whose values map to the columns in the order they appear in this list. The data type identifies the type of the column in the table and is automatically converted from the JDBC type as defined in &quot;Java Database Connectivity data types map&quot; on page 220.</td>
</tr>
</tbody>
</table>
Table 67. JDBC stored procedure source attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>(Optional for data streams.) Parameters passed to the stored procedure and their data types. The parameters contain values to look up in the DBMS table. Inputs are passed as arguments to the procedure in the order that they appear in the list. The data type identifies the IBM Cognos Real-time Monitoring data type of the value that is passed to the procedure. For more information, see “Java Database Connectivity data types map” on page 220. Further, for data streams, the procedure usually queries the DBMS looking for events inserted since the last time the procedure was called. This is done by identifying fields in the table that contain some incrementing or increasing values. For example, if the table that is queried contains unique, ascending ID values, that field is the one used by the procedure by using the logic “where ID greater than maximum ID from last query”. For data streams, the <strong>Initial Polling Value</strong> specifies the value to use the first time the procedure queries the DBMS for events. For example, you might specify ID values that start with 500. For subsequent queries, the <strong>Subsequent Polling Value</strong> identifies a field that contains the maximum value from the last query. This value is an <strong>Output field</strong> from the previous result.</td>
</tr>
<tr>
<td>Polling</td>
<td>How often to call the stored procedure. For more information, see “Polling the Java Database Connectivity source” on page 216.</td>
</tr>
</tbody>
</table>

When making a query to a Sybase database, be aware that the names, including tables and columns, are case-sensitive.

**Example of receiving lookup table data using a stored procedure**

You can see an example of a view that joins a data stream table with a lookup table.

Consider the following example:

```sql
SELECT DataStream.ID, Lookup.Name
FROM DataStream, Lookup
WHERE DataStream.ID = Lookup.ID
```

The lookup table output for this view is the Name column, and the Input to the procedure is the ID column.
Example of receiving an event using a stored procedure

You can see an example of receiving an event using a stored procedure.

Consider an event with the following fields. Event_Timestamp is the field with the unique and increasing value. Each event record has a timestamp assigned by the DBMS.

The data stream input identifies a parameter that passes the value to the query. The name of the input must be unique to the list but is otherwise insignificant. In the following example, the field name is "IN1".

The first time the stored procedure queries the DBMS, it issues a query semantically similar to the following example:

```
SELECT * FROM datastreams
WHERE datastream_time >= "2003-03-05 19:45:00"
```

Subsequent queries use the results from the previous query as the starting point for new events.

Java Database Connectivity data types map

The data types of the DBMS columns are displayed as JDBC data types and map to IBM Cognos Real-time Monitoring data types.

For more information, see Chapter 9, “Data types,” on page 63.

The following table shows this mapping.

<table>
<thead>
<tr>
<th>JDBC data type</th>
<th>Character</th>
<th>Integer</th>
<th>Double</th>
<th>Decima</th>
<th>Timestamp</th>
<th>Boolean</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 68. JDBC data types map (continued)

<table>
<thead>
<tr>
<th>JDBC data type</th>
<th>Character</th>
<th>Integer</th>
<th>Double</th>
<th>Decima</th>
<th>Timestamp</th>
<th>Boolean</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMERIC</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>BINARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARBINARY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

Do not use Cognos Real-time Monitoring Boolean data types in a WHERE predicate passed to the JDBC source. Boolean values can be included in the Select list.

Java Database Connectivity agents

A Java database connectivity agent communicates with a relational database (DBMS) by either making a query on the database, or calling a stored procedure in the DBMS. The DBMS then returns one or more rows of data, which the agent passes on to the requesting data stream or lookup table.

JDBC agents are synchronous, they retrieve event messages and lookup table data as the result of a specific request as summarized in the following table. For lookup tables, the agents access the DBMS when a new event requires lookup table data. For data stream tables, the agent periodically polls the DBMS to see if new events are available, then retrieves them for inclusion in the data stream table. Each event returned is processed individually, regardless of the count of events returned as a result of the polling query.

Table 69. JDBC agents data retrieval

<table>
<thead>
<tr>
<th>Data Stream Push</th>
<th>Data Stream Pull</th>
<th>Lookup Table Pull</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Creating a JDBC agent

You can create a JDBC agent.

JDBC agents have the following attributes:

Table 70. JDBC agent attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Identifies the agent. The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see &quot;Object namespace&quot; on page 261.</td>
</tr>
<tr>
<td><strong>Save in</strong></td>
<td>Specifies the folder in which to save the agent. The default is <strong>Public Folders</strong>. Click the <strong>Choose Folder</strong> button to select a folder.</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>An optional description that can contain any text characters.</td>
</tr>
<tr>
<td><strong>Database Type</strong></td>
<td>Identifies the DBMS vendor as DB2®, Oracle, SQL Server, Sybase, MySQL, PostgreSQL, or Teradata.</td>
</tr>
<tr>
<td><strong>User name</strong></td>
<td>Specifies the user name to use to connect to the DBMS. This user must have query access rights.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Sets the password for the user name. If you omit this option, the agent uses the password specified in the JDBC Source configuration definition in the application server.</td>
</tr>
<tr>
<td><strong>Max Rows Per Query</strong></td>
<td>Specifies the maximum count of rows to return as the query result. Useful to keep users from returning exceptionally large results that affect the DBMS.</td>
</tr>
<tr>
<td><strong>Type of JDBC Connection</strong></td>
<td>Defines how to connect to the JDBC in the application server. The connection can have one of the following sources: Data source-based, which connects to a JDBC database pool through a JNDI connection. URL-based, which connects to a JDBC source through a URL.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **JNDI Name for JDBC Source**  
(datasource-based only) | Identifies the name of the data source to use as a connection to the database. The name is in JNDI form, such as com.cognos.obi.products.ProductSource. A source gets its connection from a pool of connections maintained by the application server. That server keeps the connections open to reduce delays when establishing a connection. 

The connection pool must be configured as a non-transactional pool; non-TxT on WebLogic. 

You can make a fiscal calendar table available by specifying com.cognos.obi.calendarDatasource. For more information, see "Fiscal calendar table" on page 224. |
| **JDBC URL**  
(URL-based only) | Indicates the URL that maps to the JDBC connection configured in the application server that runs IBM Cognos Real-time Monitoring. For example, a URL might look like the following example: jdbc:oracle:thin:some_context/context@v480:1521:symbols |
| **JDBC Driver Class**  
(URL-based only) | Specifies the JDBC driver to use. This driver must reside in the classpath of the application that runs Cognos Real-time Monitoring. Include the complete classname, such as oracle.jdbc.driver.OracleDriver, in the classpath. |
| **JNDI Properties**  
(datasource-based only) | Specifies the optional and additional Java naming and directory interface (JNDI) properties necessary to make or maintain the agent to the JDBC source. You can use these name-value pairs to specify JDBC properties. The names are either one of the following short cuts or a JNDI recognized property. The agent recognizes the following names as short cuts to JNDI properties. 

- factory maps to INITIAL_CONTEXT_FACTORY. 
- provider maps to PROVIDER_URL. 
- security_credentials maps to SECURITY_CREDENTIALS. 
- security_principal maps to SECURITY_PRINCIPAL. |
| **Connection Properties**  
(URL-based only) | Specifies the optional and additional connection properties necessary to make or maintain the agent to the JDBC source. You can use these name-value pairs to specify properties for the JDBC connection. |
Before you begin

Before creating a JDBC agent:
- You must have Create permission for agents.
- You must have a JDBC data source defined and managed by the application server, preferably an application server that pools connections. Configure the connection pool as documented in your application server documentation. Additionally, in the definition of the pool:
  - Set the maximum number of open connections to the database to be at least 200.
  - Set a refresh rate to be greater than 0, preferably to 1 or 2 minutes, so that the database can go down and come back up without losing the connection from the pool. Further, you can set the pool to test for the existence of a physical table in the database.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the **Workbench** tab.
2. Click **Activities**.
3. Select **Create New > Agent**.
4. In the Create Agent window, choose JDBC as the source type.
5. Specify the attributes as needed.
6. Save the agent as enabled.

Fiscal calendar table

A fiscal calendar table is available that you can access by supplying the JDNI name com.cognos.obi.calendarDatasource when creating a JDBC agent.

The fiscal calendar table uses the following schema table when creating data streams and lookup tables associated with the JDBC agent in IBM Cognos Real-time Monitoring Workbench.

```sql
CREATE TABLE STDLKUP.FISCAL_CALENDAR
(
    DATE_VALUE TIMESTAMP ,
    DAY_NAME VARCHAR(20) ,
    DAY_OF_MONTH INTEGER ,
    DAY_OF_WEEK INTEGER ,
    DAY_OF_YEAR INTEGER ,
    MONTH_NAME VARCHAR(15) ,
    MONTH_OF_QUARTER INTEGER ,
    MONTH_OF_YEAR INTEGER ,
    QUARTER_NAME VARCHAR(10) ,
    QUARTER_OF_YEAR INTEGER ,
    F_MONTH_NAME VARCHAR(15) ,
    F_MONTH_OF_QUARTER INTEGER ,
    F_MONTH_OF_YEAR INTEGER ,
    F_QUARTER_NAME VARCHAR(10) ,
    F_QUARTER_OF_YEAR INTEGER ,
    RUNNING_DAY INTEGER ,
```
RUNNING_MONTH INTEGER ,
RUNNING_QUARTER INTEGER ,
RUNNING_WEEK INTEGER ,
RUNNING_YEAR INTEGER ,
F_RUNNING_MONTH INTEGER ,
F_RUNNING_QUARTER INTEGER ,
F_RUNNING_WEEK INTEGER ,
F_RUNNING_YEAR INTEGER ,
F_RUNNING_MONTH INTEGER ,
F_RUNNING_QUARTER INTEGER ,
F_RUNNING_WEEK INTEGER ,
F_RUNNING_YEAR INTEGER ,
TYPE_OF_DAY VARCHAR(10) ,
WEEK_NAME VARCHAR(15) ,
WEEK_OF_YEAR INTEGER ,
F_WEEK_NAME VARCHAR(15) ,
F_WEEK_OF_YEAR INTEGER ,
YEAR INTEGER ,
FYEAR INTEGER
);
Chapter 19. Java Database Connectivity access to view and cube data

IBM Cognos Real-time Monitoring provides an application programming interface (API) that allows JDBC 2.0 applications to retrieve data from a view or cube and to retrieve the metadata that describes the views or cubes in the installation.

- You can view data from the recent view. If the view contains a moving set window, the window data is returned.
- With special property settings in the JDBC URL, you also see the following columns:
  - Event identifier (VC_EVENT_ID), which identifies the event that produced the most recent row included in the view.
  - Latest event identifier (VC_LATEST_EVENT_ID), which identifies the last event that caused the view to update, though data from that event might not be included in the view.
  - Event timestamp (VC_TIMESTAMP), which identifies when the last event was included in the view.

For more information, see "Java Database Connectivity URLs."

Classpath

The classpath to IBM Cognos Real-time Monitoring JDBC driver (cqjdbcclient.jar) and commons-logging-1.1.jar must be added to the client JDBC application.

The classpath is:
```
java -classpath .;commons-logging_1.1.jar;
cqjdbcclient.jar JDBCAccessor
```

You can obtain commons-logging-1.1.jar from Apache.org.

The cqjdbcclient.jar is in the directory <cognos_installdir>/realtime/sdk/java/lib.

Java Database Connectivity URLs

The JDBC URL has a specific format.
```
jdbc:cognos:obi://<host>:<port>[;<property>=<value>][;<property>=<value>]...
```

The default port number is 2669. You can set the JDBC port through the System Control settings in the System Settings of the Administration Console.

The properties and default values are as follows:
### Table 71. Properties and default values for JDBC URLs

<table>
<thead>
<tr>
<th>Properties</th>
<th>Default values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicitColumns</td>
<td>false</td>
<td>Specifies whether the return implicit columns in addition to the normal review data. If true, the columns VC_EVENT_ID, VC_LATEST_EVENT_ID and VC_TIMESTAMP are included in the result set.</td>
</tr>
<tr>
<td>C8passport</td>
<td></td>
<td>If IBM Cognos 8 authenticates a user, the user can use the JDBC driver by passing the C8passport to the driver.</td>
</tr>
<tr>
<td>User name</td>
<td></td>
<td>The database user on whose behalf the connection is being made.</td>
</tr>
<tr>
<td>password</td>
<td></td>
<td>The user’s password.</td>
</tr>
</tbody>
</table>

The property settings have the following effects:

- If the C8passport property is set, the user name and password properties are ignored.
- Password property is ignored and the user is logged in as Anonymous, if the following conditions are met:
  - The C8passport property is not set.
  - Anonymous access is enabled.
  - The user name property is set to Anonymous or not provided.
- If the C8passport property is not set and Anonymous user is not enabled, the values provided for the user name and password properties are used.

If the value of implicitColumns is true, the View data from the recent view, the event identifier, the latest event identifier, and event timestamp is the same information that is written to a database when persisting views. For more information, see “Persisting views to a database” on page 345.

---

**Java Database Connectivity view interfaces**

JDBC 2.0 defines interfaces for accessing data.

IBM Cognos Real-time Monitoring implements the interfaces for accessing data.

For complete details about the interfaces, see the JDBC documentation at http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html.

The Cognos Real-time Monitoring JDBC driver does not provide full support for standard JDBC API. For information about the Real-time Monitoring implementation of the JDBC API, see Appendix A, “Java Database Connectivity methods,” on page 393.

For a list of the Cognos Real-time Monitoring JDBC driver classes (Driver, Connections, Statement, DatabaseMetaData, ResultSet, and ResultSetMetaData) and their methods provided by the Cognos Real-time Monitoring JDBC driver, see Chapter 19, “Java Database Connectivity access to view and cube data,” on page 227.
Data type mappings

The get functions in this class returns values from IBM Cognos Real-time Monitoring columns.

This matrix indicates which functions are used for the various Real-time Monitoring data types.

Table 72. Data type map

<table>
<thead>
<tr>
<th>Data type</th>
<th>getBoolean</th>
<th>getInt</th>
<th>getLong</th>
<th>getDouble</th>
<th>getBigDecimal</th>
<th>getString</th>
<th>getTimestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Yes</td>
<td>No</td>
<td>NA</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>Integer</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Long</td>
<td>NA</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>Double</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Decimal</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Varchar</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>Timestamp</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>No</td>
<td>Yes</td>
<td>NA</td>
</tr>
</tbody>
</table>

Yes indicates that the function returns a value compatible with Real-time Monitoring data type.

No indicates data types that might be compatible, but whose conversion is not recommended.

getColumns() column summary

A summary of the Java documentation for DatabaseMetaData.getColumns().

See the Java documentation for a complete list.

Table 73. getColumns() column summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_CAT</td>
<td>String</td>
<td>Table catalog (can be null).</td>
</tr>
<tr>
<td>TABLE_SCHEM</td>
<td>String</td>
<td>Table schema (can be null).</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>String</td>
<td>Table name.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>String</td>
<td>Column name.</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>int</td>
<td>SQL type from java.sql.Types. See &quot;DATA_TYPE return values&quot; on page 231 for a summary.</td>
</tr>
<tr>
<td>TYPE_NAME</td>
<td>String</td>
<td>Data source dependent type name, for a UDT the type name is fully qualified.</td>
</tr>
<tr>
<td>COLUMN_SIZE</td>
<td>int</td>
<td>Column size. For char or date types, this is the maximum number of characters. For numeric or decimal types this is precision.</td>
</tr>
<tr>
<td>BUFFER_LENGTH</td>
<td>String</td>
<td>Not used.</td>
</tr>
<tr>
<td>DECIMAL_DIGITS</td>
<td>int</td>
<td>Count of fractional digits.</td>
</tr>
<tr>
<td>NUM_PREC_RADIX</td>
<td>int</td>
<td>Radix (typically either 10 or 2).</td>
</tr>
</tbody>
</table>
Table 73. getColumns() column summary (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULLABLE</td>
<td>int</td>
<td>Is NULL allowed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>columnNoNulls means might not allow NULL values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>columnNullable means definitely allows NULL values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>columnNullableUnknown means Nullability unknown.</td>
</tr>
<tr>
<td>REMARKS</td>
<td>String</td>
<td>Comment describing column (can be null).</td>
</tr>
<tr>
<td>COLUMN_DEF</td>
<td>String</td>
<td>Default value (can be null).</td>
</tr>
<tr>
<td>SQL_DATA_TYPE</td>
<td>int</td>
<td>Not used.</td>
</tr>
<tr>
<td>SQL_DATETIME_SUB</td>
<td>int</td>
<td>Not used.</td>
</tr>
<tr>
<td>CHAR_OCTET_LENGTH</td>
<td>int</td>
<td>For char types the maximum number of bytes in the column.</td>
</tr>
<tr>
<td>ORDINAL_POSITION</td>
<td>int</td>
<td>Index of column in table (starting at 1).</td>
</tr>
<tr>
<td>IS_NULLABLE</td>
<td>String</td>
<td>&quot;NO&quot; means column definitely does not allow NULL values; &quot;YES&quot; means that the column might allow NULL values. An empty string means that nobody knows.</td>
</tr>
</tbody>
</table>

getTables() column summary

A summary of the Java documentation for DatabaseMetaData.getTables().

See the Java documentation for a complete list.

Table 74. getTables() column summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_CAT</td>
<td>String</td>
<td>Table catalog. (Can be null).</td>
</tr>
<tr>
<td>TABLE_SCHEM</td>
<td>String</td>
<td>Table schema. (Can be null).</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>String</td>
<td>Table name.</td>
</tr>
<tr>
<td>TABLE_TYPE</td>
<td>String</td>
<td>Table type. Typical types are &quot;TABLE&quot;, &quot;VIEW&quot;, &quot;SYSTEM TABLE&quot;, &quot;GLOBAL TEMPORARY&quot;, &quot;LOCAL TEMPORARY&quot;, &quot;ALIAS&quot;, &quot;SYNONYM&quot;.</td>
</tr>
<tr>
<td>REMARKS</td>
<td>String</td>
<td>Used by IBM Cognos Real-time Monitoring for modifying a column timestamp.</td>
</tr>
<tr>
<td>TYPE_CAT</td>
<td>String</td>
<td>Types catalog. (Can be null).</td>
</tr>
<tr>
<td>TYPE_SHEM</td>
<td>String</td>
<td>Types schema. (Can be null).</td>
</tr>
<tr>
<td>TYPE_NAME</td>
<td>String</td>
<td>Type name. (Can be null).</td>
</tr>
<tr>
<td>SELF_REFERENCING_COLUMN_NAME</td>
<td>String</td>
<td>Name of the designated &quot;identifier&quot; column of a typed table (Can be null).</td>
</tr>
</tbody>
</table>
Table 74. `getTables()` column summary (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REF_GENERATION</td>
<td>String</td>
<td>Specifies how values in SELFREFERENCING_COL_NAME are created. Values are &quot;SYSTEM&quot;, &quot;USER&quot;, &quot;DERIVED&quot;. (Can be null).</td>
</tr>
</tbody>
</table>

**DATA_TYPE return values**

The DATA_TYPE column returns an int value that identifies the Java data type.

See the java.sql.Types file for details. The following table summarizes those values.

Table 75. DATA_TYPE return values

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>-7</td>
</tr>
<tr>
<td>TINYINT</td>
<td>-6</td>
</tr>
<tr>
<td>BIGINT</td>
<td>-5</td>
</tr>
<tr>
<td>LONGVARBINARY</td>
<td>-4</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>-3</td>
</tr>
<tr>
<td>BINARY</td>
<td>-2</td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>-1</td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>2</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>3</td>
</tr>
<tr>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>5</td>
</tr>
<tr>
<td>FLOAT</td>
<td>6</td>
</tr>
<tr>
<td>REAL</td>
<td>7</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>12</td>
</tr>
<tr>
<td>DATE</td>
<td>91</td>
</tr>
<tr>
<td>TIME</td>
<td>92</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>93</td>
</tr>
<tr>
<td>OTHER</td>
<td>1111</td>
</tr>
<tr>
<td>JAVA_OBJECT</td>
<td>2000</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>2001</td>
</tr>
<tr>
<td>STRUCT</td>
<td>2002</td>
</tr>
<tr>
<td>ARRAY</td>
<td>2003</td>
</tr>
<tr>
<td>BLOB</td>
<td>2004</td>
</tr>
<tr>
<td>CLOB</td>
<td>2005</td>
</tr>
<tr>
<td>REF</td>
<td>2006</td>
</tr>
</tbody>
</table>
**JDBC accessor examples**

These examples demonstrate how to connect to the IBM Cognos Real-time Monitoring servers and query view data and metadata about the views and columns defined in the installation.

**Java**

Access to the JDBC driver depends on the java.sql.* classes.

Be sure to include the following import in your applications;

```java
import java.sql.*;
```

**Complete sample**

To compile the application, use this command.

```java
javac -classpath . JDBCAccessor.java
```

To run the program, load it into the application server.

The results from the examples print to the standard output, with errors going to standard error.

**Example: establishing a connection to the IBM Cognos Real-time Monitoring servers**

This example shows how to establish a connection to the JDBC driver of IBM Cognos Real-time Monitoring servers.

Establish the connection by creating a Connection object, similar to the following example:

```java
Connection connection = null;
connection = DriverManager.getConnection(url, userName, password);
```

The user name and password parameters identify a Cognos Real-time Monitoring account. When querying a specific object (such as a view), the user account must have at least Read Only access permission. Otherwise, if the account has No Access to the view, the query fails as if the view did not exist.

The following is a more detailed example. However, to see the complete listing, examine the JDBCAccessor.main() member.

```java
// Common connection properties
String userName = "RTMadmin";
String password = "manager";
String url = "jdbc:cognos:obi://localhost:2669";

// Verify the JDBC driver in the application
try {
    Class.forName("com.cognos.obi.jdbc.driver.Driver");
} catch (ClassNotFoundException e) {
    handleError("Could not find the JDBC driver class.", e);
    return;
}

// Establish the connection to the JDBC driver
Connection connection = null;
```
try {
    connection = DriverManager.getConnection(url, userName, password);
} catch (SQLException e) {
    handleError("Could not connect to the JDBC driver.", e);
    return;
}

Example: querying the contents of a view

This snippet shows how to query the entire contents of a view.

The executeQuery() call passes the query to the driver, which returns the view contents in a ResultSet object. All columns in the query result are included in this list. If you set implicitColumns to true, three additional columns are included. For more information, see "Java Database Connectivity URLs" on page 227.
Additionally, metadata about the view is retrieved in a ResultSetMetaData object to determine the count of columns in the view.

See the JDBCAccesser.PrintViewContents() sample for a complete code listing.

/* Query all contents of a view. */
// Connection has already been established, and the view name defined.
String queryString = "SELECT * FROM " + VIEW_NAME ;
ResultSet rs; // Table to hold the query results.
ResultSetMetaData rmd; // Metadata about the result set.
// Query the view, and get its data and metadata.
Statement stmt = connection.createStatement();
rs = stmt.executeQuery( queryString );
rmd = rs.getMetaData();
// Print the contents of the entire view, row by row.
int columnCount = rmd.getColumnCount();
boolean isEmpty = true;
while (rs.next()) {
    isEmpty = false;
    System.out.print(" Row: ");
    for (int i=0;i<columnCount;i++) {
        // Show the column value, or "NULL"
        String ts = rs.getString(i+1);
        System.out.print( (rs.wasNull() ? "NULL":ts + " ");
    }
    System.out.println(); // Line break
}
if (isEmpty) {
    System.out.println("\n *** The view is empty ***");
}
The result might look like the following if implicitColumns is set to false:

Contents of view [OrderProductTotals]:
Row: Hardware Lag bolts 16400.00 41 65000.00 30000.00 100000.00
30000.00 100000.00
Row: Hardware Nuts 337875.00 159 280000.00 240000.00 320000.00
240000.00 320000.00
Row: Lumber Plywood 304800.00 127 250000.00 200000.00 300000.00
200000.00 300000.00
Row: Hardware Washers 122400.00 72 170000.00 120000.00 220000.00
120000.00 220000.00
Row: Hardware Hinges 132300.00 49 130000.00 90000.00 170000.00
90000.00 170000.00
Row: Hardware Nails 129600.00 48 150000.00 100000.00 200000.00
100000.00 200000.00
Row: Lumber Chip board 277200.00 126 250000.00 200000.00 300000.00
200000.00 300000.00
Row: Hardware Screws 60000.00 30 80000.00 60000.00 100000.00
60000.00 100000.00

The result might look like the following if implicitColumns is set to true:

Contents of view [OrderProductTotals]:
Row: Hardware Lag bolts 16400.00 41 65000.00 30000.00 100000.00
30000.00 100000.00 20 2007-11-01 15:58:09.794 27
Row: Hardware Nuts 337875.00 159 280000.00 240000.00 320000.00
240000.00 320000.00 26 2007-11-01 15:58:09.794 27
Row: Lumber Plywood 304800.00 127 250000.00 200000.00 300000.00
200000.00 300000.00 9 2007-11-01 15:58:09.794 27
Row: Hardware Washers 122400.00 72 170000.00 120000.00 220000.00
120000.00 220000.00 22 2007-11-01 15:58:09.794 27
Row: Hardware Hinges 132300.00 49 130000.00 90000.00 170000.00
90000.00 170000.00 27 2007-11-01 15:58:09.794 27
Row: Hardware Nails 129600.00 48 150000.00 100000.00 200000.00
100000.00 200000.00 22 2007-11-01 15:58:09.794 27
Row: Lumber Chip board 277200.00 126 250000.00 200000.00 300000.00
200000.00 300000.00 17 2007-11-01 15:58:09.794 27
Row: Hardware Screws 60000.00 30 80000.00 60000.00 100000.00
60000.00 100000.00 19 2007-11-01 15:58:09.794 27

Example: querying the column specifications of a view

This snippet shows how to query the user-defined specifications about the columns in a view.

First it shows all of the metadata available for column specifications, then it shows specifications for each column in the view, including the internal system columns. The results appear in the order that the columns appear in the view, followed by the internal columns.

See the JDBCAccessor.PrintColumns() sample for a complete code listing.

/* Query a view's column specifications.
   * Connection has already been established, and the view name defined.
   */
   // Query the table to identify the columns to report on. Because the
   // view contents are irrelevant here, omit them by declaring
   // 'WHERE false' as the query condition.
   String queryString = "SELECT * FROM " + VIEW_NAME +" WHERE false";
   ResultSet rs; // Table to hold the query results.
ResultSetMetaData rmd; // Metadata about the result set.
// Query the view, and then get its metadata.
Statement stmt = connection.createStatement();
rs = stmt.executeQuery(queryString);
rmd = rs.getMetaData();

// Print the metadata about the columns in the view.
System.out.println("Column details for view [" +
    rmd.getTableName(1) +"]:");
// Walk through and show the interesting
metadata available for
// each column in the view. Include labels to identify
what we see
// in the result.
for (int i=0;i<rmd.getColumnCount();i++){
    System.out.println(" " + Integer.toString(i+1) +
        ". Name [" + rmd.getColumnName(i+1) +
        "] Type [" + Integer.toString(rmd.getColumnType(i+1)) +
        "] Precision [" + Integer.toString(rmd.getPrecision(i+1)) +
        "] Scale [" + Integer.toString(rmd.getScale(i+1)) +"]");
}
The results might look like the following example:
Column details for view [ORDERPRODUCTTOTALS]:

Example: querying column metadata
These snippets show how to retrieve metadata about the columns in a view.

To retrieve the column specifications instead, see "Example: querying the column
specifications of a view" on page 234.

See the JDBCAccessor.PrintColumnMeta() sample for a complete code listing.

/* Retrieve the metadata about the columns
   of a defined view.
*/
// NOTE: Specify 'null' instead of 'VIEW_NAME' to get
// the metadata
// for ALL columns in the installation.
rs = meta.getColumns(null,null,VIEW_NAME,null);
rmd = rs.getMetaData();
// Show the metadata available for view columns.
int columnCount = rmd.getColumnCount();
for (int i=0;i<columnCount;i++) {
    System.out.print(rmd.getColumnName(i+1) +
            ((i+1)==columnCount ? "" : ","));
}

The results first lists the metadata column names (see \[getColumns() column summary\] on page 229 for a description of the columns):

```
TABLE_CAT,TABLE_SCHEM,TABLE_NAME,COLUMN_NAME,DATA_TYPE,TYPE_NAME,COLUMN_SIZE,
BUFFER_LENGTH,DECIMAL_DIGITS,NUM_PREC_RADIX,NULLABLE,REMARKS,COLUMN_DEF,
SQL_DATA_TYPE,SQL_DATETIME_SUB,CHAR_OCTET_LENGTH,ORDINAL_POSITION,IS_NULLABLE,
SCOPE_CATALOG,SCOPE_SCHEMA,SCOPE_TABLE,SOURCE_DATA_TYPE
```

Next, show all of the metadata about the columns in a specific view. Columns do not appear in the order that they appear in the view.

// Show the metadata values for the columns in the view.
boolean isEmpty = true;
while (rs.next()) {
    isEmpty = false;
    for (int i=0;i<columnCount;i++) {
        String ts = rs.getString(i+1); // Metadata value
        if (rs.wasNull())
            System.out.print(",");
        else
            System.out.print(ts +
                    ((i+1)==columnCount ? "" : ","));
    }
    System.out.println();
}
if (isEmpty) {
    System.out.println("\n *** Either there are no columns defined " +
            "for this view (unlikely), or the view is not " + "defined (probably).\";
}

The following is a sample listing of the metadata for the OrderProductTotals view. For a mapping of the data types from Java, see \["Data type mappings" on page 229\].

Again, the columns do not appear in any particular order.

```
IBM Cognos RTM,IBM Cognos
RTM,OrderProductTotals,Family,12,VCVarchar,20,129,0,10,1,1,YES,....
IBM Cognos RTM,IBM Cognos
RTM,OrderProductTotals,TargetMin,3,VCDecimal,15,129,2,10,1,2,YES,....
IBM Cognos RTM,IBM Cognos
RTM,OrderProductTotals,TargetMax,3,VCDecimal,15,129,2,10,1,3,YES,....
IBM Cognos RTM,IBM Cognos
RTM,OrderProductTotals,Target,3,VCDecimal,15,129,2,10,1,4,YES,....
IBM Cognos RTM,IBM Cognos
RTM,OrderProductTotals,Sales,3,VCDecimal,25,129,2,10,1,5,YES,....
IBM Cognos RTM,IBM Cognos
RTM,OrderProductTotals,PRODUCT,12,VCVarchar,50,129,0,10,1,6,YES,....
```
Example: querying view metadata

These snippets show how to query view metadata.

First it shows the metadata available for views, then it shows how to find all of the views defined in the system.

See the JDBCAccessor.PrintAllViewsMeta() sample for a complete code listing.

```java
/* Retrieve metadata about views. */
ResultSet rs;
ResultSetMetaData rmd;
// Uses 'null' for the 3rd parameter to retrieve information about
//the entire system, not just a view.
DatabaseMetaData meta = connection.getMetaData();
rs = meta.getTables(null,null,null,new String[]{"VIEWS"});
rmd = rs.getMetaData();
for (int i=0;i<rmd.getColumnCount();i++) {
    System.out.println(" "+rmd.getColumnName(i+1) +
        "] Type "+ rmd.getColumnType(i+1)+
        "] Precision "+ rmd.getPrecision(i+1)) +
    "] Scale "+ rmd.getScale(i+1))
}
```

The results first describe the metadata that is available:

```
[TABLE_CAT] Type [12] Precision [255] Scale [0]
[TABLE_SCHEM] Type [12] Precision [255] Scale [0]
[TABLE_NAME] Type [12] Precision [255] Scale [0]
[TABLE_TYPE] Type [12] Precision [255] Scale [0]
[REMARKS] Type [12] Precision [255] Scale [0]
```

Next, list the views (table names and table types) defined in the system:

```
// Use the metadata to list all of the views in the system.
final int GT_TABLE_NAME = 3; // View name
final int GT_TABLE_TYPE = 4; // 'VIEWS' or 'CUBES'
boolean isEmpty = true;
while (rs.next()) {
    isEmpty = false;
    System.out.println(" " + rs.getString(GT_TABLE_NAME) + ", " + 
        rs.getString(GT_TABLE_TYPE));
    if (isEmpty) {
        System.out.println(" *** No views are defined.
        ***");
    
```
The results look similar to the following example:
"30DAYORDERS", VIEWS
"8WEEKORDERS", VIEWS
"8WEEKORDERSAVG", VIEWS
INVENTORYCHANGEDTAL DETAILS, VIEWS
ORDERCHANGEDETAILS, VIEWS
ORDERPRODUCTTOTALS, VIEWS
ORDERQTYDEMAND_DRILLBACK_VIEW, VIEWS
ORDERSALESGRANDTOTAL, VIEWS
ORDERTOTALS, VIEWS
SUPPLIERALTERNATES, VIEWS
VC_SYSTEM VIEW, VIEWS
VC_TASK VIEW, VIEWS

Java Database Connectivity cube interfaces

The API for cubes is based on the draft olap4j specification version 0.5.

The olap4j specification was not fully developed at the time of the implementation of this interface, and this interface does not confirm strictly to the olap4j specification.

The following sections describe the API for accessing cubes. For the complete API, see the Javadoc HTML documentation for the Package com.cognos.obi.jdbc.api.

IOlapStatement

The IOlapStatement uses interface members to execute queries.

They are:
- executeOlapQuery
  Executes an MDX query.
- executeOlapQueries
  Executes MDX queries. The returned resultSets are event-consistent; that is, they reflect a snapshot up to the same event.
- executeCubeQuery
  Executes the cube query in the proprietary XML format of IBM Cognos Real-time Monitoring server.

IOlapDatabaseMetadata

Provides information about the cube definitions defined in the IBM Cognos Real-time Monitoring installation.

The IOlapDatabaseMetadata interface has the following members:
- getCubes
  Retrieves a description of a cube.
- getDimensions
  Retrieves a result set describing the dimensions within a particular cube.
- getHierarchies
  Retrieves a result set describing each hierarchy within a particular dimension.
- getLevels
Retrieves a result set describing each level within a particular hierarchy.

- **getMembers**
  Retrieves a result set describing the members within a particular level.

- **getMemberChildren**
  Retrieves a result set describing the children members of a member.

- **getMeasures**
  Retrieves a result set describing each measure within a particular cube.

**Example: querying all cubes**

These snippets show how to query cube metadata.

First it shows the metadata available for cubes, then it shows how to find all of the cubes defined in the system.

See the `JDBCAccessor.PrintAllCubesMeta()` sample for a complete code listing.

```java
IOlapDatabaseMetaData meta = (IOlapDatabaseMetaData) connection.getMetaData();
rs = meta.getCubes(null, null, null);
rmd = rs.getMetaData();
for (int i=0;i<rmd.getColumnCount();i++) {
    System.out.println(" + rmd.getColumnName(i+1) + " + rmd.getColumnType(i+1)+" + rmd.getPrecision(i+1)+" + rmd.getScale(i+1)+")");
}
```

The results first describe the metadata that is available:

- `TABLE_CAT` Type [12] Precision [255] Scale [0]
- `TABLE_SCHEM` Type [12] Precision [255] Scale [0]
- `TABLE_NAME` Type [12] Precision [255] Scale [0]
- `TABLE_TYPE` Type [12] Precision [255] Scale [0]
- `REMARKS` Type [12] Precision [255] Scale [0]

Next, list the cubes (table names, table types, and last modification time) defined in the system:

```java
// Use the metadata to list all of the views in the system.
final int GT_TABLE_NAME = 3; // View name
final int GT_TABLE_TYPE = 4; // 'VIEWS' or 'CUBES'
final int GT_REMARKS = 5; // If type is 'CUBES',
// Remarks is the Last modification
// timestamp of the cube.
// Otherwise, it is empty.

boolean isEmpty = true;
while (rs.next()) {
    isEmpty = false;
    System.out.println(" + rs.getString(GT_TABLE_NAME)+" + rs.getString(GT_TABLE_TYPE)
};
```

Chapter 19. Java Database Connectivity access to view and cube data 239
if (isEmpty) {
    System.out.println(" *** No views are defined. ***");
}

The results look similar to the following example:
ORDERCUBE, CUBES, 2007-11-07 13:00:07.107
ORDERQTYDEMAND_DRILLBACK_CUBE, CUBES, 2007-11-07 13:00:11.2

Example: querying the metadata of a cube

This example shows how to retrieve the dimensions, dimension hierarchies, hierarchy levels, level members, and measures of a cube.

```
ResultSet rs;
IOLapDatabaseMetaData meta = (IOLapDatabaseMetaData)connection.getMetaData();
rs = meta.getDimensions(CUBE_NAME);
System.out.println();
System.out.println("*" + rs.getMetaData().getColumnName(1) + ":");
boolean isEmpty = true;
while (rs.next()) {
    isEmpty = false;
    String ts = rs.getString(1);
    System.out.print(" "+ts);
    System.out.println();
    ResultSet hierarchies = meta.getHierarchies(rs.getString(1));
    System.out.println(" *" +
        hierarchies.getMetaData().getColumnName(1) + ":");
    while (hierarchies.next()){
        System.out.print(" ");
        String hierarchy = hierarchies.getString(1);
        System.out.print(hierarchy);
        System.out.println();
        ResultSet levels = meta.getLevels(rs.getString(1),
            hierarchies.getString(1));
        System.out.println(" *" +
            levels.getMetaData().getColumnName(1) + ":");
        while (levels.next()){
            System.out.print(" ");
            String level = levels.getString(1);
            System.out.print(level);
            System.out.println();
            ResultSet members = meta.getMembers(rs.getString(1),
                hierarchies.getString(1),levels.getString(1));
            System.out.println(" *" +
                members.getMetaData().getColumnName(1) + ":");
            while (members.next()){
                System.out.print(" ");
                String member = members.getString(1);
                System.out.print(member);
```
System.out.println();
}
}
}
rs = meta.getMeasures(CUBE_NAME);
System.out.println("*" + rs.getMetaData().getColumn(1) + ":");
while (rs.next()) {
    isEmpty = false;
    String ts = rs.getString(1);
    System.out.print(" "+ ts);
    System.out.println();
}
if (isEmpty) {
    System.out.println("\n*** The cube was not found.");
}

The results look similar to the following example:

*DIMENSION_NAME:
    DateTime Dimensions
*HIERARCHY_NAME:
    DEFAULT HIERARCHY
*LEVEL_NAME:
    All DateTime Dimensions level
*MEMBER_NAME:
    All DateTime Dimensions
Year
    *MEMBER_NAME:
        2003
        2004
        2005
Quarter
    *MEMBER_NAME:
        Q1
        Q2
        Q3
        Q4
Month
    *MEMBER_NAME:
        January
        February
        ...
        November
        December
Week Number
    *MEMBER_NAME:
        1
        2
...  
52  
53  
Day  
MEMBER_NAME:  
Monday  
Tuesday  
...  
Saturday  
Sunday  
Location Dimensions  
MEMBER_NAME:  
"DEFAULT HIERARCHY"  
LEVEL_NAME:  
All Location Dimensions level  
MEMBER_NAME:  
All Location Dimensions  
Region  
MEMBER_NAME:  
All  
Central  
East  
South  
West  
State  
MEMBER_NAME:  
California  
Florida  
...  
Rhode Island  
Texas  
City  
MEMBER_NAME:  
Austin  
Lodi  
..  
Tug Hollow  
Wallaby Ranch  
Product Dimensions  
MEMBER_NAME:  
DEFAULT HIERARCHY  
LEVEL_NAME:  
All Product Dimensions level  
MEMBER_NAME:  
All Product Dimensions  
Family  
MEMBER_NAME:  

Hardware
Lumber
Tape
Product
*MEMBER_NAME:
Chip board
Duct tape
...
Studs
Washers

*MEASURE_NAME:
AverageOrderSales
TotalOrderSales

Example: retrieving the children members of a level member in a cube

These snippets show how to retrieve the children members of a level member in a cube.

In this example, the level member is specified in MDX syntax.

See the JDBCAccess.testGetMemberChildren() sample for a complete code listing.

```java
ResultSet rs;
IOLapDatabaseMetaData meta = (IOLapDatabaseMetaData)connection.getMetaData();
String member = "[store_dimension].[All_store_dimension_level].[All_store_dimension].[USA]";//a member name specified in MDX syntax.
rs = meta.getMemberChildren(member);
int columnCount = rs.getMetaData().getColumnCount();
for (int i=0;i<columnCount;i++) {
    System.out.print(rs.getMetaData().getColumnName(i+1));
    System.out.print(' ');
}
System.out.println();
while (rs.next()) {
    System.out.print(" Row: ");
    for (int i=0;i<columnCount;i++) {
        // Show the column value, or "NULL"
        String ts = rs.getString(i+1);
        System.out.print( (rs.wasNull() ? "NULL ": ts + " ");
    }
    System.out.println(); // Line break
}
```

The results look similar to the following example:

MEMBER_CHILDREN
Row: Hinges
Row: Lag bolts
Row: Nails
Example: querying cube data using query cube XML

These snippets show how to retrieve the cube data using query cube XML, which is a proprietary format for the IBM Cognos Real-time Monitoring server.

See the JDBCAccess.testCubeQuery() sample for a complete code listing.

```java
ResultSet rs; // Table to hold the query results.
Statement stmt = connection.createStatement();
String s = "<queryCube xsi:schemaLocation="http://obi.cognos.com/ ../xsd/
XMLSchema-instance">" + 
"<cube xml:space="preserve">OrderCube</cube>" + 
"<measures>" + 
"<list>" + 
"<name xml:space="preserve">TotalOrderSales</name>" + 
"</list>" + 
"<list>" + 
"<name xml:space="preserve">AverageOrderSales</name>" + 
"</list>" + 
"</measures>" + 
"<dimensionLevels>" + 
"<selectedLevelsAndFilters>" + 
"<selectedLevels>" + 
"<list>" + 
"<dimensionName xml:space="preserve">"Location Dimensions"</dimensionName>" + 
"<memberName xml:space="preserve">Region</memberName>" + 
"<selected xml:space="preserve">true</selected>" + 
"</list>" + 
"</selectedLevels>" + 
"</selectedLevelsAndFilters>" + 
"</dimensionLevels>" + 
"</queryCube>";
rs = (ResultSet)((IOlapStatement)stmt).executeCubeQuery(s);
int columnCount = rs.getMetaData().getColumnCount();
for (int i=0;i<columnCount;i++) {
    System.out.print(rs.getMetaData().getColumnName(i+1));
    System.out.print(' ');
}
System.out.println();
while (rs.next()) {
    System.out.print(" Row: ");
    for (int i=0;i<columnCount;i++) {
        // Show the column value, or "NULL"
    }
}
```
Example: data retrieval from a cube by using MDX queries

These snippets show how to retrieve the cube data by using MDX queries.

See the JDBCAccesser.testMDXQuery() sample for a complete code listing.

```java
ResultSet rs; // Table to hold the query results.
Statement stmt = connection.createStatement();
String s = "SELECT Measures.[TotalOrderSales] ON ROWS,
 [Location Dimensions].[Region].[West] ON COLUMNS FROM [OrderCube]
where [Product Dimensions].[Family].[Hardware]";
String s2 = "SELECT Measures.[TotalOrderSales] ON ROWS,
 [Location Dimensions].[Region].[West].children ON COLUMNS FROM [OrderCube]
where [Product Dimensions].[Family].[Hardware]";
IOlapResultSet[] rss = ((IOlapStatement)stmt).executeOlapQueries(new
String[]{s, s2});
for (int j = 0; j < 2; j++)
{
    System.out.println("mdx" + j + ":");
    rs = rss[j];
    int columnCount = rs.getMetaData().getColumnCount();
    for (int i=0;i<columnCount;i++) {
        System.out.print(rs.getMetaData().getColumnName(i+1));
        System.out.print(" ");
    }
    System.out.println();
    while (rs.next()){
        System.out.println(" Row ");
        for (int i=0;i<columnCount;i++) {
            // Show the column value, or "NULL"
            String ts = rs.getString(i+1);
            System.out.print( (rs.wasNull() ? "NULL ":
            ts + " "));
        }
        System.out.println(); // Line break
    }
}
```
The results look similar to the following example:

```
mdx0:
"LOCATION DIMENSIONS".REGION ORDERCUBE.TOTALORDERSALES
 Row: West 610650.00
mdx1:
"LOCATION DIMENSIONS".REGION "LOCATION DIMENSIONS".STATE ORDERCUBE.TOTALORDERSALES
 Row: West California 428750.00
 Row: West Nevada 181900.00
```
Chapter 20. Lookup tables

Lookup tables augment the processing of information from information sources by providing additional information about the event.

Lookup tables include business information stored in databases, data warehouses, or provided by Web services. For more information, see Chapter 37, “Web services,” on page 349. Lookup tables receive data from agents that communicate with information sources. For more information, see Chapter 4, “Agents,” on page 21. When you define a lookup table, you also instruct the agent how to identify the information from the source.

How lookup tables work

When a business view requires lookup table information, it retrieves the information based on data that is already in the view.

For example, a view that is processing a purchase order might receive a product identification number along with the event data. If the view also requires the suppliers of that product, it would retrieve the supplier names from a lookup table that contains the names that match the ID. In the view definition, a WHERE clause would join the lookup table to the event, similar to the following example:

```
WHERE event.product_id = lookuptable.suppliers_of_product_id
```

When the view performs this join, it passes the ID from the event to the lookup table. If the matching supplier data is already in the lookup table cache, the table uses that data and passes it to the business view. If the data is not already in memory, the ID is passed (either as an SQL query or by value for a stored procedure) to an agent, which sends data to the DBMS or Web service for processing. The result of the query is then loaded into the lookup table and later included in the business view.

The information source can be databases accessed through a JDBC (see Chapter 18, “Java Database Connectivity,” on page 211) or business applications accessed through web services (see Chapter 37, “Web services,” on page 349). An information source can also be a text file. For details about these types of sources,
Creating lookup tables

Every lookup table has a name, description, status attribute, and agent.

Tip:

Specify indexes for lookup tables on the join columns used by views or dimensions. If you do not specify indexes, finding the right value to join to requires searching through all of the rows in the lookup table. Indexing increases the speed with which the row containing the right value is located.

Use prefetch caching for JDBC lookup tables so that the server does not have to connect to the database when each event occurs. Instead, data is looked up in bulk at regular intervals. Be aware, however, that the use of prefetch caching can increase memory usage.

These attributes are defined in the following table.

**Table 76. Lookup table attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the table and is the name accessed by the Business views that depend on this table (see Chapter 36, “Views,” on page 337). The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Description</td>
<td>(Optional) Description of the table.</td>
</tr>
<tr>
<td>Status</td>
<td>Whether or not the object is enabled (able to receive and pass data) or disabled (not receiving or passing data).</td>
</tr>
<tr>
<td>Save in</td>
<td>The folder in which to store the lookup table. The default folder is Public Folders. Click the Choose Folder button to select a folder.</td>
</tr>
<tr>
<td>Agent</td>
<td>The agent that retrieves the information for the lookup table, and passes the data to the data stream or lookup table object. See Chapter 4, “Agents,” on page 21 for information about agent types.</td>
</tr>
<tr>
<td>Disable lookup table after this number of consecutive errors</td>
<td>The number of consecutive errors that is received from the system before it disables the lookup table. Once disabled, a lookup table must be re-enabled manually. The default is 5.</td>
</tr>
</tbody>
</table>

For more information about source types, see the following example:

- Chapter 18, “Java Database Connectivity,” on page 211
- Chapter 37, “Web services,” on page 349
- Chapter 29, “Salesforce,” on page 303
- Chapter 12, “Cognos Real-time Monitoring file system,” on page 87
Before you begin

Before creating a lookup table, you must have Create permissions for tables (see Chapter 23, “Permissions,” on page 275 and “Granting permission to create objects” on page 278) and Read-only access permission on the agent that retrieves the information for the lookup table.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > Lookup Table.
4. Select the source type.
5. Complete the fields in the New Lookup Table window.
6. Save the table as enabled.

Editing lookup tables

Editing the attributes of a lookup table causes the object to lose state, and possibly invalidates dependent views.

For example, if you remove a column, any view or rule that references that column becomes invalid. (However, if you redefine the column in the table, the dependent views are automatically revalidated.)

Before editing a lookup table, you must have Read and Write Chapter 23, “Permissions,” on page 275 for tables (see “Access permissions” on page 275), and Read Only access permission on the agent that feeds the table.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Select the lookup table in either the Workbench Object pane or detail pane.
3. If you selected the lookup table in the detail pane, click Activities then select Edit from the drop-down menu. If you selected the lookup table from the Workbench Object pane, click the edit button.
4. Change the definitions in the Configure Lookup Table form. Each type has its own specific attributes. For more information, see the following topics:
   - Chapter 18, “Java Database Connectivity,” on page 211
   - Chapter 37, “Web services,” on page 349
   - Chapter 13, “Flat files,” on page 93
5. Save the table as enabled.

Column limitations in queries for lookup tables

Data for a lookup table can be retrieved with no limitations from a JDBC query source. However, some limitations apply when retrieving data for a lookup table from a JDBC stored procedure source or a Web service source.

These limitations include:

- When the lookup table column is referenced as part of a query, somewhere in the WHERE or FROM clause it must appear in an equality expression and then only as an atomic predicate (no other operators on the same side of the equal sign). For example, the following is permitted:
WHERE lookup_column = 10*event_column

But the following is not permitted because the left-side predicate, which contains the lookup table column, is an expression that includes an operator (/):
WHERE lookup_column/10 = event_column

• The required atomic reference cannot appear in a disjunct (OR) expression. The following fails:
  WHERE (lookup_column = event_column OR A > B)

However, it can appear in a conjunct (AND) expression:
WHERE (lookup_column = event_column AND A > B)

• Once there is at least one equality reference in the query, you can use the column in any other way. For example, the following two queries are permitted:
WHERE (lookup_column = event_column AND
  lookup_column/10 = other_event_column)
WHERE ((lookup_column = event_column AND
  lookup_column >= other_event_column) OR
  (A > B))

But the following fails because there is no equality reference in the query:
WHERE lookup_column >= event_column

• The required equality expression cannot reference another context column in the same table. For example:
  t1.lookup_column = t1.other_lookup_column

• However, the equality expression can reference a context column in another table, for example:
  t1.lookup_column = t2.other_lookup_column

**Sybase limitations**

When making a query to a Sybase database, be aware of these limitations:

• All names are case-sensitive, including tables and columns.

• All queries must be in the form SELECT * FROM table only. You cannot include any SELECT clauses. To filter the results, load them into a business view then filter that view.

---

**Caching lookup table queries**

Caching allows you to store the results of lookup table queries in memory.

Subsequent requests for the same information are then retrieved from memory instead of impacting the DBMS with a redundant query. When caching is active, and a view requests lookup table data, it searches the cache first. If the data that you want is not in the cache, IBM Cognos Real-time Monitoring issues a query to the database if on-demand caching is enabled. However, if prefetch caching is enabled, Real-time Monitoring issues a query to the prefetch cache.

For more information about caching methods, see “On-demand caching” on page 251 and “Prefetch caching” on page 251.

The lookup table cache has these parameters:
• **Cache data for this lookup table**
  Either cache or do not cache results from the query on the lookup table to the recent query cache. This parameter must be set to enable the remaining parameters to be set. You must also add at least one invalidation schedule when selecting this parameter.

• **Enable prefetch**
  Either enable or disable prefetch of data from the lookup table. This option caches the entire external lookup table into memory from the external data source and becomes a replacement for the external query source.

• **Number of results to cache**
  Count of query results to cache in memory. Each set of results can contain one or more rows of lookup table data related to the event.

• **Invalidation schedule**
  Identifies when to invalidate the cache and discard all information currently in the cache, such as 6:00 a.m. every Monday.

If you select any caching options, on-demand caching is in effect. For example, if you select **In Memory** or **Prefetch**, you have on-demand caching as well as prefetch caching.

**On-demand caching**

With on-demand caching, a recent-query cache is created that maintains results on a least recently used (LRU) basis.

This cache tracks when each result set was last requested. When the cache is full, it keeps the most recently accessed rows and discards the rows that have not been accessed in the longest period of time. If data is requested that is not in the recent query cache, the data is retrieved from the external data source.

On-demand caching occurs when you select **Cache data for this lookup table** on the Data Caching tab and you do not select **Enable prefetch** for JDBC or web service sources.

Rows containing frequently requested data remain in the recent query cache the longest to reduce impact on the database. However, if details about the information can change often, define an invalidation schedule to account for the changes and invalidate the cache.

When a scheduled invalidation occurs, the recent query cache is cleared and updated on subsequent queries. If lookup table data is not rapidly changing, it is best to invalidate the recent query cache less often. For example, if the lookup table is fairly static, you might want to invalidate the cache weekly or monthly. However, if the lookup table database is updated nightly, you might want to invalidate the recent query cache nightly as well to ensure the latest data.

On-demand caching can deliver better performance than if no caching is used. However, you might be able to improve performance further by using prefetch caching.

**Prefetch caching**

When you enable the prefetch cache, the entire external lookup table is cached into memory from the external data source and becomes a replacement for the external query source, referred to as on-demand caching.
However, when data is requested that is not in the on-demand cache, the data is retrieved from the prefetch cache instead of the external data source.

When an invalidation occurs, according to your specified invalidation schedule, the on-demand cache is cleared, and a query is issued to the external data source to update the in-memory copy of the lookup table. The on-demand cache is updated from the prefetch cache on subsequent queries.

For information about on-demand cache functions, see “On-demand caching” on page 251.

When enabling the cache for prefetch data, consider the following effects:

- More memory is used because the information from the external data source is stored in memory, and an index is built for each column.
- The prefetch cache is only updated at the scheduled invalidation time. If data is not in the prefetch cache, no data is returned. Updates to the external data source are not reflected in the prefetch cache until after the next scheduled invalidation.

The prefetch cache again tries up to five times to refresh on the invalidation schedule if an exception occurs. If it cannot refresh from the external data source, the recent query cache is then used until the next scheduled refresh.

**Java Database Connectivity store caching**

You can specify that the data in the flat file is read then stored in the cache by the JDBC agent.

The fetch-and-store mechanism uses two tables to store the data.

Only one of the two tables used by the fetch-and-store mechanism is used at a time for queries. The table that the system is not currently accessing is purged and reloaded according to the invalidation schedule. After that file is reloaded with data from the flat file, queries are redirected to the freshly updated table, and the previous table are purged and updated at the next invalidation in the schedule. For example, table A is currently being used for queries, and the invalidation is set to occur at 12 am. At 12 am, table B is purged and reloaded then queries are redirected to Table B. At the next scheduled invalidation, table A is purged and reloaded and then queries are redirected to it. Table B then waits for the next scheduled invalidation.

Specify that the data in the flat file is read then stored in the cache by the JDBC agent by selecting **JDBC store** as the type of data caching.
Chapter 21. Objects in Cognos Real-time Monitoring Workbench

Every object has a name, optional description, and a status that determines whether it is able to work with specific data. Every object also has properties that define the behavior of the object.

Object status

Every object has one of four states that determine its ability to operate.

The states are described in the following table:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🟢 raining</td>
<td>Enabled</td>
</tr>
<tr>
<td>🟢 ⬅️</td>
<td>The object is accepting and processing new data. You can enable only valid objects that do not depend on disabled objects. When you enable an object, you have the choice of enabling just that object, or that object and all objects that depend on that object. When enabling an object and its dependents, the dependent objects must be valid. If any dependent object is invalid, no objects are enabled.</td>
</tr>
<tr>
<td>🟥</td>
<td>Disabled</td>
</tr>
<tr>
<td>🟥 ⬅️</td>
<td>The object is not accepting new data. Disabling an object does not affect the definition or existence of that object. Rather, it just keeps new data from flowing into the object and to all objects that depend on the object. For example, disabling a view also disables all rules that monitor the view, and also disables all associated alerts and reportlets.</td>
</tr>
<tr>
<td>🟥 ⬅️</td>
<td>Disabled dependent</td>
</tr>
<tr>
<td>🟥 ⬅️</td>
<td>The object is not accepting new data because an object that it depends on is disabled. Enabling the referenced object also enables this object.</td>
</tr>
<tr>
<td>🟥 ⬅️ ⚠️</td>
<td>Invalid</td>
</tr>
<tr>
<td>🟥 ⬅️ ⚠️</td>
<td>The object has a reference that is not valid. For example, one view references a column in another view, but that column no longer exists in the referenced view. An object can be invalid because a referenced object does not exist or because some attribute of the object does not match the requirements of the dependent. For example, a data type might not match, or a column name is missing. An invalid object also invalidates all objects that depend on the object. This happens when you delete an object that has dependencies, or you change the definition of an object.</td>
</tr>
</tbody>
</table>

When you view a list of objects, you can see the valid or invalid state of each object and the enabled or disabled status. You can select an object from the list and enable it through the Activities menu.

Object names

All object names must be either regular identifiers or delimited identifiers.
A regular identifier must begin with a letter from a-z or A-Z, and all subsequent
caracters can be an underscore (_) or character from a-z, A-Z, or 0-9. A regular
identifier cannot be a reserved word.

A delimited identifier must start and end with a double quotation mark ("'). The
body of a delimited identifier must be non-empty and can contain any SQL
language characters including: the Regular Identifier characters; underscore (_);
space ( ), percent (%); ampersand (&); single quotation mark (''); left parenthesis (();
right parenthesis ()); asterisk (*); plus sign (+); comma (,); minus sign (-), slash (/);
colon (;); semicolon (;); equals operator (=); question mark (?); vertical bar (|); or
double quotation mark ("") (escaped with another double quotation mark).

Names for cubes cannot contain periods (.)

All names must be unique among all objects (including dashboard objects and
dashboards) in the same folder. The root folder also contains global objects such as
user, roles, and business activities. For more information, see "Object namespace"
on page 261 Names cannot be identical to reserved words. For example, you
cannot have a view named by the regular identifier select, though you can have a
delimited view named select.

Reserved words

All identifiers beginning with VC_ are reserved system names and cannot be used.
Further, all reserved words in the SQL-99 standard are reserved in C-SQL.

The following table lists the IBM Cognos Real-time Monitoring reserved words.

<table>
<thead>
<tr>
<th>Reserved word</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
</tr>
<tr>
<td>absolute</td>
</tr>
<tr>
<td>acked</td>
</tr>
<tr>
<td>action</td>
</tr>
<tr>
<td>add</td>
</tr>
<tr>
<td>all</td>
</tr>
<tr>
<td>allocate</td>
</tr>
<tr>
<td>alter</td>
</tr>
<tr>
<td>and</td>
</tr>
<tr>
<td>any</td>
</tr>
<tr>
<td>are</td>
</tr>
<tr>
<td>as</td>
</tr>
<tr>
<td>asc</td>
</tr>
<tr>
<td>assertion</td>
</tr>
<tr>
<td>at</td>
</tr>
<tr>
<td>authorization</td>
</tr>
<tr>
<td>avg</td>
</tr>
<tr>
<td>begin</td>
</tr>
<tr>
<td>between</td>
</tr>
<tr>
<td>Reserved word</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>bit</td>
</tr>
<tr>
<td>bit_length</td>
</tr>
<tr>
<td>boolean</td>
</tr>
<tr>
<td>both</td>
</tr>
<tr>
<td>by</td>
</tr>
<tr>
<td>cascade</td>
</tr>
<tr>
<td>cascaded</td>
</tr>
<tr>
<td>case</td>
</tr>
<tr>
<td>cast</td>
</tr>
<tr>
<td>catalog</td>
</tr>
<tr>
<td>ceil</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>char_length</td>
</tr>
<tr>
<td>character</td>
</tr>
<tr>
<td>character_length</td>
</tr>
<tr>
<td>check</td>
</tr>
<tr>
<td>close</td>
</tr>
<tr>
<td>coalesce</td>
</tr>
<tr>
<td>collate</td>
</tr>
<tr>
<td>collation</td>
</tr>
<tr>
<td>column</td>
</tr>
<tr>
<td>commit</td>
</tr>
<tr>
<td>concat</td>
</tr>
<tr>
<td>connect</td>
</tr>
<tr>
<td>connection</td>
</tr>
<tr>
<td>constraint</td>
</tr>
<tr>
<td>constraints</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>convert</td>
</tr>
<tr>
<td>corresponding</td>
</tr>
<tr>
<td>count</td>
</tr>
<tr>
<td>create</td>
</tr>
<tr>
<td>cross</td>
</tr>
<tr>
<td>current</td>
</tr>
<tr>
<td>current_date</td>
</tr>
<tr>
<td>current_time</td>
</tr>
<tr>
<td>current_timestamp</td>
</tr>
<tr>
<td>current_user</td>
</tr>
<tr>
<td>cursor</td>
</tr>
<tr>
<td>date</td>
</tr>
<tr>
<td>Reserved word</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>date_add</td>
</tr>
<tr>
<td>date_diff</td>
</tr>
<tr>
<td>day</td>
</tr>
<tr>
<td>deallocate</td>
</tr>
<tr>
<td>dec</td>
</tr>
<tr>
<td>decimal</td>
</tr>
<tr>
<td>declare</td>
</tr>
<tr>
<td>default</td>
</tr>
<tr>
<td>deferrable</td>
</tr>
<tr>
<td>deferred</td>
</tr>
<tr>
<td>delete</td>
</tr>
<tr>
<td>desc</td>
</tr>
<tr>
<td>describe</td>
</tr>
<tr>
<td>descriptor</td>
</tr>
<tr>
<td>diagnostics</td>
</tr>
<tr>
<td>disconnect</td>
</tr>
<tr>
<td>distinct</td>
</tr>
<tr>
<td>domain</td>
</tr>
<tr>
<td>double</td>
</tr>
<tr>
<td>drop</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end-exec</td>
</tr>
<tr>
<td>escape</td>
</tr>
<tr>
<td>event</td>
</tr>
<tr>
<td>except</td>
</tr>
<tr>
<td>exception</td>
</tr>
<tr>
<td>exec</td>
</tr>
<tr>
<td>execute</td>
</tr>
<tr>
<td>exists</td>
</tr>
<tr>
<td>exp</td>
</tr>
<tr>
<td>external</td>
</tr>
<tr>
<td>extract</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>fetch</td>
</tr>
<tr>
<td>first</td>
</tr>
<tr>
<td>float</td>
</tr>
<tr>
<td>floor</td>
</tr>
<tr>
<td>for</td>
</tr>
<tr>
<td>foreign</td>
</tr>
</tbody>
</table>
Table 78. Reserved words (continued)

<table>
<thead>
<tr>
<th>Reserved word</th>
</tr>
</thead>
<tbody>
<tr>
<td>found</td>
</tr>
<tr>
<td>from</td>
</tr>
<tr>
<td>full</td>
</tr>
<tr>
<td>get</td>
</tr>
<tr>
<td>global</td>
</tr>
<tr>
<td>go</td>
</tr>
<tr>
<td>goto</td>
</tr>
<tr>
<td>grant</td>
</tr>
<tr>
<td>greatest</td>
</tr>
<tr>
<td>group</td>
</tr>
<tr>
<td>having</td>
</tr>
<tr>
<td>hour</td>
</tr>
<tr>
<td>identity</td>
</tr>
<tr>
<td>immediate</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>indicator</td>
</tr>
<tr>
<td>initially</td>
</tr>
<tr>
<td>inner</td>
</tr>
<tr>
<td>input</td>
</tr>
<tr>
<td>insensitive</td>
</tr>
<tr>
<td>insert</td>
</tr>
<tr>
<td>int</td>
</tr>
<tr>
<td>integer</td>
</tr>
<tr>
<td>intersect</td>
</tr>
<tr>
<td>interval</td>
</tr>
<tr>
<td>into</td>
</tr>
<tr>
<td>is</td>
</tr>
<tr>
<td>is_raised</td>
</tr>
<tr>
<td>isolation</td>
</tr>
<tr>
<td>join</td>
</tr>
<tr>
<td>key</td>
</tr>
<tr>
<td>language</td>
</tr>
<tr>
<td>last</td>
</tr>
<tr>
<td>last_day</td>
</tr>
<tr>
<td>leading</td>
</tr>
<tr>
<td>least</td>
</tr>
<tr>
<td>left</td>
</tr>
<tr>
<td>level</td>
</tr>
<tr>
<td>like</td>
</tr>
<tr>
<td>local</td>
</tr>
<tr>
<td>Reserved word</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>log</td>
</tr>
<tr>
<td>lower</td>
</tr>
<tr>
<td>lowered</td>
</tr>
<tr>
<td>lpad</td>
</tr>
<tr>
<td>ltrim</td>
</tr>
<tr>
<td>match</td>
</tr>
<tr>
<td>max</td>
</tr>
<tr>
<td>min</td>
</tr>
<tr>
<td>minute</td>
</tr>
<tr>
<td>mod</td>
</tr>
<tr>
<td>module</td>
</tr>
<tr>
<td>month</td>
</tr>
<tr>
<td>mov_avg</td>
</tr>
<tr>
<td>mov_count</td>
</tr>
<tr>
<td>mov_max</td>
</tr>
<tr>
<td>mov_min</td>
</tr>
<tr>
<td>mov_ntile</td>
</tr>
<tr>
<td>mov_rank</td>
</tr>
<tr>
<td>mov_ratio_to_report</td>
</tr>
<tr>
<td>mov_std_deviation</td>
</tr>
<tr>
<td>mov_sum</td>
</tr>
<tr>
<td>mov_variance</td>
</tr>
<tr>
<td>names</td>
</tr>
<tr>
<td>national</td>
</tr>
<tr>
<td>natural</td>
</tr>
<tr>
<td>nchar</td>
</tr>
<tr>
<td>next</td>
</tr>
<tr>
<td>no</td>
</tr>
<tr>
<td>not</td>
</tr>
<tr>
<td>ntile</td>
</tr>
<tr>
<td>null</td>
</tr>
<tr>
<td>nullif</td>
</tr>
<tr>
<td>numeric</td>
</tr>
<tr>
<td>octet_length</td>
</tr>
<tr>
<td>of</td>
</tr>
<tr>
<td>on</td>
</tr>
<tr>
<td>only</td>
</tr>
<tr>
<td>open</td>
</tr>
<tr>
<td>option</td>
</tr>
<tr>
<td>or</td>
</tr>
<tr>
<td>Reserved word</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>order</td>
</tr>
<tr>
<td>outer</td>
</tr>
<tr>
<td>output</td>
</tr>
<tr>
<td>overlaps</td>
</tr>
<tr>
<td>pad</td>
</tr>
<tr>
<td>partial</td>
</tr>
<tr>
<td>position</td>
</tr>
<tr>
<td>power</td>
</tr>
<tr>
<td>precision</td>
</tr>
<tr>
<td>prepare</td>
</tr>
<tr>
<td>preserve</td>
</tr>
<tr>
<td>prev</td>
</tr>
<tr>
<td>primary</td>
</tr>
<tr>
<td>prior</td>
</tr>
<tr>
<td>prior_value</td>
</tr>
<tr>
<td>privileges</td>
</tr>
<tr>
<td>procedure</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>raised</td>
</tr>
<tr>
<td>rank</td>
</tr>
<tr>
<td>ratio_to_report</td>
</tr>
<tr>
<td>read</td>
</tr>
<tr>
<td>real</td>
</tr>
<tr>
<td>references</td>
</tr>
<tr>
<td>relative</td>
</tr>
<tr>
<td>restrict</td>
</tr>
<tr>
<td>revoke</td>
</tr>
<tr>
<td>right</td>
</tr>
<tr>
<td>rollback</td>
</tr>
<tr>
<td>round</td>
</tr>
<tr>
<td>rows</td>
</tr>
<tr>
<td>rpad</td>
</tr>
<tr>
<td>rtrim</td>
</tr>
<tr>
<td>schema</td>
</tr>
<tr>
<td>scroll</td>
</tr>
<tr>
<td>second</td>
</tr>
<tr>
<td>section</td>
</tr>
<tr>
<td>select</td>
</tr>
<tr>
<td>session</td>
</tr>
<tr>
<td>session_user</td>
</tr>
<tr>
<td>Reserved word</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>set</td>
</tr>
<tr>
<td>sign</td>
</tr>
<tr>
<td>size</td>
</tr>
<tr>
<td>smallint</td>
</tr>
<tr>
<td>some</td>
</tr>
<tr>
<td>space</td>
</tr>
<tr>
<td>sql</td>
</tr>
<tr>
<td>sqlcode</td>
</tr>
<tr>
<td>sqlerror</td>
</tr>
<tr>
<td>sqlstate</td>
</tr>
<tr>
<td>sqrt</td>
</tr>
<tr>
<td>std_deviation</td>
</tr>
<tr>
<td>substr</td>
</tr>
<tr>
<td>substring</td>
</tr>
<tr>
<td>sum</td>
</tr>
<tr>
<td>system_user</td>
</tr>
<tr>
<td>table</td>
</tr>
<tr>
<td>temporary</td>
</tr>
<tr>
<td>then</td>
</tr>
<tr>
<td>time</td>
</tr>
<tr>
<td>timestamp</td>
</tr>
<tr>
<td>timestamp_diff</td>
</tr>
<tr>
<td>timezone_hour</td>
</tr>
<tr>
<td>timezone_minute</td>
</tr>
<tr>
<td>to</td>
</tr>
<tr>
<td>to_char</td>
</tr>
<tr>
<td>to_date</td>
</tr>
<tr>
<td>trailing</td>
</tr>
<tr>
<td>transaction</td>
</tr>
<tr>
<td>translate</td>
</tr>
<tr>
<td>translation</td>
</tr>
<tr>
<td>trim</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>trunc</td>
</tr>
<tr>
<td>union</td>
</tr>
<tr>
<td>unique</td>
</tr>
<tr>
<td>unknown</td>
</tr>
<tr>
<td>update</td>
</tr>
<tr>
<td>upper</td>
</tr>
<tr>
<td>usage</td>
</tr>
</tbody>
</table>
Object namespace

The namespace controls how objects are named.

Generally, object names must be unique among other objects of the same type within the same container. However, here are some exceptions:

- Alerts, rules, and reportlets can share names within the containing scenario. You can use the same name for one alert, one rule, and one reportlet within a scenario. Further, each object within a scenario can share the same name as an object of the same type in another scenario.
- Profiles must be unique within a single user. Multiple users can share profile names.
- Views, data streams, lookup tables, and other such objects cannot share the same name. Their names must be unique within the tables class. See the following outline for details.
- Agents must be unique within the agents class.
- Users and business activities cannot share the same name. Their names must be unique within the containers class. Also, users and folders cannot share the same name because they exist within the same namespace. The following outline summarizes the namespace constraints.

<table>
<thead>
<tr>
<th>/containers</th>
<th>/Business activities</th>
<th>Unique among /containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Scenarios</td>
<td>/Alerts</td>
<td></td>
</tr>
<tr>
<td>/Alerts</td>
<td>/Rules</td>
<td></td>
</tr>
<tr>
<td>/Rules</td>
<td>/Reportlets</td>
<td></td>
</tr>
<tr>
<td>/Users and Roles</td>
<td>/E-mail profiles</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unique within a user</td>
</tr>
</tbody>
</table>
/RTD (Excel) profiles Unique within a user

/tables
/Data streams Unique among /tables
/Lookup tables Unique among /tables
/Consolidated data streams Unique among /tables
/Cube Unique among /tables
/Dimensions Unique among /tables
/Views Unique among /tables

/agents
/Flat (text) files Unique among /agents
/SOAP (Web services) Unique among /agents
/JMS Unique among /agents
/JDBC Unique among /agents
/Rendezvous (TIBCO) Unique among /agents

/External actions (processes)
/Join relationships

Selective export of objects

You can export selected objects. This method is useful when exporting objects from a development environment to a production environment.

You can export a single object, an object and its dependencies, an object and its requirements, or an object and both its requirements and dependencies. The dependencies of an object are other objects that depend on your selected object in order for them to be valid. The requirements of an object are the objects that your selected object requires to be valid.

The following diagram shows the relationships between three dashboards, a number of dashboard objects, three views, two cubes, one dimension, three lookup tables, three data streams, and three agents. In the diagram, the dependencies of View 1 are Dashboard 1, Dashboard Object 1, Cube 1, and Dashboard Object 2. The requirements for View 1 are Data Stream 1 and Agent 1. The arrows between objects indicate the direction of the dependency.
The following aspects of selective export are discussed in this section:

- "Example - exporting objects with dependencies"
- "Example - exporting objects with requirements" on page 265
- "Example - exporting objects with requirements and dependencies" on page 266
- "Exporting selected objects" on page 268

For information about export and import of Cognos Real-time Monitoring metadata, see the IBM Cognos Real-time Monitoring Workbench User Guide.

**Example - exporting objects with dependencies**

You can export an object with its dependencies. In some cases, objects included in this type of export might become invalid after you import them because the requirements of the dependencies might not be included in the export.

The following diagram shows two scenarios of export with dependencies.
Scenario 1

You export Data Stream 1 with its dependencies. The following objects are included or excluded from the export:

- View 1, Dashboard Object 1, Cube 1, and Dashboard Object 2 are exported.
- Dimension 1, Lookup Table 1, and Agent 1 are not exported.
  These objects are requirements for Cube 1. The system traverses only the paths of these objects. As a result, Cube 1 is invalid after the import because its requirements are not included.
- Dashboard Object 1 and Dashboard Object 2 are exported.
- Dashboard 1 is not exported, even though it contains Dashboard Object 1.

Scenario 2

You export Agent 2 with its dependencies. The following objects are included or excluded from the export:

- The dashboard objects in Dashboard 2 and Dashboard 3 are exported, but the dashboards themselves are not exported.
- View 2 and View 3 are exported.
- Data Stream 2 and Agent 1, which are requirements for View 2, are not exported.
  This makes View 2 invalid after the import.
• Agent 3 and Lookup Table 3, which are requirements for View 3, are not exported.
  This makes View 3 invalid after the import.

**Example - exporting objects with requirements**
You can export an object with its requirements.

The following diagram shows a few examples of this type of export.

![Diagram showing examples of objects exported with requirements](image)

*Figure 26. Examples of objects exported with requirements*

When exporting View 1 with its requirements, Data Stream 1 and Agent 1 are also exported.

When exporting Cube 2 with its requirements, Dimension, Lookup Table 1, Agent 1, View 2, Data Stream 2, Lookup Table 2, and Agent 2 are also exported.

When exporting Dashboard 3 with its requirements, Dashboard Object 4, Dashboard Object 5, and Dashboard Object 6 are also exported. Also, the requirements of the three dashboard objects, View 3, Data Stream 3, Lookup Table 3, Agent 2 and Agent 3, are exported.
Example - exporting objects with requirements and dependencies

In some cases, you may want to export both the dependencies and the requirements of an object.

The following scenarios explain the implications of this type of export.

Scenario 1

You want to export Dimension 1 with its dependencies and requirements, as shown in the following diagram.

The following objects are included or excluded from the export:

- Dashboard Object 2, Dashboard Object 3, Cube 1, and Cube 2 are exported.
  These are the dependencies of Dimension 1.
- Lookup Table 1 and Agent 1 are exported.
  These are the requirements for Dimension 1.
- View 1 and View 2 are not exported.
  These are the requirements of Cube 1 and Cube 2. Not including these objects makes Cube 1 and Cube 2 invalid after the import. Only one of Cube 1 and Cube 2 requirements, Dimension 1, is exported.
Scenario 2

You want to export Dimension 1 with its dependencies and the requirements of those dependencies, as shown in the following diagram.

Tip: To do this, use the Export sub-project option.

![Diagram showing objects and their dependencies]

The following objects are included or excluded from the export:

- Dashboard Object 2, Dashboard Object 3, Cube 1, and Cube 2 are exported. These are all dependencies of Dimension 1.
- Lookup Table 1 and Agent 1 are exported. These are the requirements for Dimension 1.
- View 1, View 2, Data Stream 1, Data Stream 2, Lookup Table 2, and Agent 2 are exported. These are the requirements needed to make Cube 1 and Cube 2 valid.
- The dependencies of View 1 and Agent 2 are not exported.

Scenario 3

You want to export Dimension 1 with its all dependencies and requirements, and the dependencies and requirements of all related objects, as shown in the following diagram.

Tip: To do that, use the Export entire project option.
When using this scenario, all objects are valid after you import them.

All objects in Project One are exported. Objects in Project Two are not exported because none of the objects in Project One is connected to any of the objects in Project Two.

**Exporting selected objects**

You can export selected objects.

The export action writes the object information to one of the following file types:

- XML files in a directory on the server.
- A single compressed JAR file on the server or local client computer.

For information about export and import of IBM Cognos Real-time Monitoring metadata, see the *IBM Cognos Real-time Monitoring Workbench User Guide*.

**Before you begin**

Before exporting the objects, ensure that you understand object dependencies and requirements.
Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Objects Library.
3. Select an object.
4. Click Activities > Export objects.
5. In the Selective Export window, from the Operation menu, select one of the following types of export:
   - **Export metadata to a directory on the server**
     Selecting this option exports the metadata to the directory that you specify in the Server Location field. Type the full path to the directory where you want to export the metadata. The directory must exist, and you must have permission to write to the specified directory. Also, two directories are created in your specified directory. One subdirectory is named latest. The other is the timestamp of your export. If the directories exist, they are reused.
   - **Export metadata to a JAR file on the server**
     Selecting this option exports the metadata to a JAR file in the directory you specify in the Server Location field. Type the full path to the directory where you want to export the metadata. The directory must exist, and you must have permission to write to the specified directory.
     The file name on the server is _export.jar. If _export.jar exists, it is overwritten.
   - **Export metadata to a JAR file (download)**
     Selecting this option exports the metadata to a JAR file on the client. Use the browser download box to identify the location (and optionally change the file name) on your local machine.
6. From the Export Options menu, specify one of the following options:
   - **Export entire project**
     Exports the selected objects, and the dependencies and requirements of all related objects.
   - **Export sub-project**
     Exports the selected objects with their requirements and dependencies, and the requirements of those dependencies.
   - **Export with requirements**
     Exports the selected objects and their requirements.
   - **Export with dependencies**
     Exports the selected objects and their dependencies. The requirements of dependencies are not exported.
   - **Export with dependencies and requirements**
     Exports the selected objects with their dependencies and requirements. The requirements of dependencies are not exported.
   - **Export only selected objects**
     Exports only the objects that you select. The dependencies and requirements of those objects are not exported.
7. Optionally, select **Include permissions on these objects**.
   This option is useful when you want to avoid reassigning permissions for your objects after you import them.
8. Click **Preview Export List** to ensure that you selected the proper export options.
9. Click **OK**.
Chapter 22. Operators and constants

IBM Cognos Real-time Monitoring and C-SQL support these operators and constants in expressions and arguments.

Numeric operators

There are two classes of numeric operators: prefix and infix.

Prefix operators

Prefix operators control the arithmetical sign of numeric values:

<table>
<thead>
<tr>
<th>Prefix operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Unary plus</td>
</tr>
<tr>
<td>-</td>
<td>Unary minus</td>
</tr>
</tbody>
</table>

Infix operators

Numeric operators perform arithmetical operations on numeric values:

<table>
<thead>
<tr>
<th>Numeric operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>11 + 3 returns 14</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>11 - 3 returns 8</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>11 * 3 returns 33</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>11 / 3 returns 3</td>
</tr>
</tbody>
</table>

To perform arithmetic operations on date-time values, use DATE_ADD and DATE_DIFF. For more information, see “DATE_ADD” on page 133 and “DATE_DIFF” on page 134.

String operators

Concatenation (|| fn_||) is the only string operator and it appends the right-side string to the end of the left-side string. For example 'a' || 'b' returns 'ab'.

The behavior is identical to that of the CONCAT function. For more information, see “CONCAT” on page 155.
Comparison operators

Comparison operators compare two or more values of the same data type and return a Boolean value.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>1=1 returns TRUE</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>'A'&lt;&gt;'a' returns TRUE</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>CURRENT_DATE()&gt;TO_DATE('02/28/1963') returns TRUE</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Age&lt;21 returns UNKNOWN when Age is NULL</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>TRUE&gt;=FALSE returns TRUE</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>NULL&lt;=NULL returns NULL</td>
</tr>
<tr>
<td>IN</td>
<td>Is a member of a list</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Symbol IN (IBM, 'MSFT', 'VCLR') or Count NOT IN (5, 10, 15, 20)</td>
</tr>
<tr>
<td>BETWEEN /AND</td>
<td>Is within a range</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>SalePrice BETWEEN 50.0 AND (90.0) or NOT BETWEEN 'M' AND 'O'</td>
</tr>
<tr>
<td>LIKE</td>
<td>Pattern matching</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Title LIKE 'MR_'</td>
</tr>
</tbody>
</table>

LIKE operator

The LIKE operator matches a pattern of characters. A percent sign (%) in the pattern is a wildcard for zero or more characters, and an underscore (_) is a wildcard for exactly one character.
WHERE Title LIKE 'MR_'
WHERE E_Mail NOT LIKE '%.edu'

To include either a percentage sign (%) or an underscore (_) in the search string, use the keyword ESCAPE to designate an escape character. A % or _ following an escape character is treated as a literal. The escape character cannot be used elsewhere in the search string. The following example looks for 10% anywhere in Discount:
WHERE Discount LIKE '%10%' ESCAPE '§'

An escape character prefixing anything other than an escape or special character is ignored.

Be careful about using LIKE when comparing against numeric types. LIKE is a string operator, and searching a numeric first performs an implicit cast of the numeric value to a string. For more information, see “Implicit cast” on page 63. When casting numerics to strings, be aware of the following considerations:

- For DECIMAL numbers, casting to string zero pads the decimal values to match the precision and scale defined for the column. So, for example, if a column is defined as precision 5 and scale 4, a value of 1.1 in the column is cast as '1.1000', and so searches for single digit decimals must be done as LIKE '__000'.
- For DOUBLE PRECISION numbers, the 'e' is cast to uppercase. So, for example, +1e11 is converted to '1.0E11'.

## Logical operators

Logical operators compare Boolean values, such as the result of a comparison operation.

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Description</th>
</tr>
</thead>
</table>
| AND | Both true  
|     | Example:  
|     | (SalesPrice>500) AND (OnSale) |
| OR | One must be true  
|    | Example:  
|    | (ZipCode = '90210') OR (City = 'Lodi') |
| NOT | Inverse  
|     | Example:  
|     | NOT OnSale |
| IS | Test of Boolean fn_is not  
|    | Example:  
|    | IS OnSale or IS NOT OnSale |

The truth table for the equal sign (=) operator is equivalent to IS. For more information, see “Boolean operators” on page 75.
The C-SQL Boolean constants are TRUE, FALSE, and UNKNOWN.

For more information, see "Boolean operators" on page 75.

<table>
<thead>
<tr>
<th>Constants</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>True.</td>
</tr>
<tr>
<td>FALSE</td>
<td>Not true.</td>
</tr>
<tr>
<td>NULL</td>
<td>No data.</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Test for Boolean value that is NULL, or where a comparison cannot be determined, for example when comparing null to null.</td>
</tr>
</tbody>
</table>

NULL is ignored when computing set function, moving set function, and rank function values. For example, the average of (3, NULL, 3) is 3, not NULL and it is not 2.

**NULL** is a null value. Any non-Boolean column which does not have an entry is considered NULL. For more information, see "Boolean operators" on page 75.

WHERE SalesPrice = NULL      Returns TRUE
WHERE SalesPrice = UNKNOWN   Error, cannot cast Numeric to Boolean

However, testing a Boolean column returns UNKNOWN when the column is empty. For more information, see "Boolean operators" on page 75.

WHERE OnSale = NULL          Returns UNKNOWN (null = null)
WHERE OnSale = UNKNOWN       Returns UNKNOWN (null = unknown)
WHERE OnSale IS NULL         Returns TRUE
WHERE OnSale IS UNKNOWN      Returns TRUE
Chapter 23. Permissions

Permissions control which users can see, create, and edit objects and user accounts.

Permissions can be set in three places:

• From the Access Permissions tab in the Edit User window. You access this window from the Administration Console tab by selecting Users and double-clicking an object name. From the Access Permissions tab, you can view and assign the type level permissions for the selected user. When you assign a type level permission, it is the minimum permission that the user has to all objects of this type.

• From the Access Permissions tab in the Edit Role window. You can assign type level permissions to a role by double-clicking the role and assigning the permission. If you assign permissions to a role, all the users associated with that role implicitly have those permissions.

• From the Permissions window. You can access this window by clicking Permissions in the detail view of an object.

When a user belongs to one or more roles, the highest level of access between the roles and the user’s assigned permissions is the effective permission for that user. For more information, see Chapter 27, “Roles,” on page 295. In the Permissions window, only explicitly assigned permissions are shown.

By default, every new user has No Access permissions (except where they have Read-only permissions on the properties: user, roles, and global system properties). However, even with this minimal set of permissions, a user can receive and view alert notifications and reportlets generated as the result of mandatory subscriptions.

The rtmadmin user always has full permissions to every object in the installation. For information about the rtmadmin user, see Chapter 34, “Users,” on page 329.

Access permissions

Access permissions specify the level of access a user has to an object. Permissions can be assigned to a class of object types or to a specific object.

The access permissions are:

Table B4. Access permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Object type</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Access</td>
<td>Cannot see any objects of these types unless granted read on specific objects.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Filtered/Read-Only</td>
<td>Not applicable</td>
<td>Limits the rows in a view or cube that the user can see based on an access filter. For more information, see Chapter 2, “Access filters,” on page 3.</td>
</tr>
</tbody>
</table>
Table 84. Access permissions (continued)

<table>
<thead>
<tr>
<th>Permission</th>
<th>Object type</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-only</td>
<td>Can see all objects of these types.</td>
<td>Can see the object.</td>
</tr>
<tr>
<td>Read and Write</td>
<td>Can see and edit all objects of these types.</td>
<td>Can see and edit the object.</td>
</tr>
</tbody>
</table>

You cannot assign a permission to an object that is greater than your own for the same object.

**Type level access permissions**

All type level permissions are assigned to a user or role. To see or change a type level permission, you must first edit the user’s account or role definition.

When viewing the list of users or roles, do not use the Permissions button. That button defines which users and roles can access the specific user accounts or role definitions in the list. For more information, see “Specific object access permissions” on page 277.

**Changing user permissions for a class of objects**

You can change user permissions for a class of objects.

For more information, see “Access permissions tab” on page 331.

**Procedure**

1. On the Administration Console tab, click Users.
2. Double-click the account of the user in the list.
3. In the Edit User window, click the Access Permissions tab.
   This tab shows the effective permissions, including permissions assigned to the roles to which the user belongs.
4. Click Edit next to the permission to change. The permission window has three fields:
   - **Role-Granted Permissions** shows the permission assigned by the roles that the user is a member of. When the user belongs to multiple roles, the greatest level of access among them is applied.
   - **Additional User-Specific Permission** is the permission that you are assigning for the user for the class. While you can assign a permission lower than the role permissions, doing so does not lower the permission for the user.
   - **Effective Permissions** is the greatest level of permission assigned by the other two fields, and is the permission assigned to the user for this class of object types.
5. Save the changes to immediately apply them to the user.

**Changing role permissions for a class of objects**

You can change role permissions for a class of objects.

**Procedure**

1. On the Administration Console tab, click Roles.
2. Double-click the account of the user in the list.
3. On the **Access Permissions** tab, click **Edit** next to the permission to change. Set the permission to the class for this role.

4. Save the changes.

**Changing user permissions for creating reports**

You can change user permissions for creating reports.

**Procedure**

1. On the **Administration Console** tab, click **Users**.
2. Double-click the account of the user in the list.
3. In the Edit User window, click the **Access Permissions** tab.
4. Click **Edit** next to Create Reports. The Allowed to Create Reports window displays. In this window, you can set the access for creating a **Report** and an **Advanced Report**. With Access permission, users can create a report. With Access with Grant permission, users can create a report and grant another user permission to create a report.
5. Click **OK**.

**Specific object access permissions**

You access the permissions to specific objects by selecting the object in the list of objects and clicking the Permissions button.

This applies to the lists of users and roles as well. You can assign access permissions to specific user accounts and roles in the same way that you assign access to specific views or agents.

**Viewing user permissions for one or more specific objects**

You can view user permissions for one or more specific objects.

**Procedure**

1. Select one or more objects, such as a specific view or views in a list.
2. Click **Activities**.
3. Select **Permissions**. The Permissions window provides the following information:
   - **Your permissions**
     The effective permissions that you, as the user currently logged in to IBM Cognos Real-time Monitoring Workbench, have on this object. The effective permissions are a combination of the permissions assigned to you explicitly or implicitly through the roles to which you belong for this object and this object's type.
   - **Name**
     The names of the users and roles that have permissions on this object or this object's type. For users, only those with explicitly assigned permissions are listed.
   - **Type**
     The type indicates whether the Name is associated with a **Role** or a **User**.
   - **Permissions**
     The permissions explicitly assigned to the Role or User on this object. Click the **Change Permissions** button to set the Permissions on a User or Role.
   - **Type Level Permissions**
     The permissions of this object's type explicitly assigned to the User or Role.
Changing user permissions for one or more specific objects
You can change user permissions for one or more specific objects.

Procedure
1. Select one or more objects, then select Permissions from the Activities drop-down list. Filtered/Read-Only permission can be assigned only to one view or cube at a time.
   For a single object, you can also click the Permissions button in the detail view for that object.
2. Select the users whose access permissions you want to change and click Change Permissions.
   You cannot change the permissions for users who do not meet the criteria listed in "Permission restrictions" on page 279. Users whose permissions you cannot change have the "No" symbol displayed next to their names.
3. Set the permissions on the Basic tab. When you assign permissions to a view or cube, you have the option of choosing a Filtered/Read-Only permission. See Chapter 2, "Access filters," on page 3 for information about this permission.
   Save the permissions and they are immediately applied to the objects.

Adding a user or role to the permissions of one or more objects
You can add a user or role to the permissions of one or more objects.

Procedure
1. Select one or more objects, then select Permissions from the Activities drop-down list.
   For a single object, you can click the Permissions button in the detail view for that object.
2. In the Permissions box, click the Add User button to add a user or click the Add Role button to add a role.
3. Click the check box next to the name you want to add to the permissions.

Granting permission to create objects
Create permissions specify which classes of objects a user can create.

When you create an object, you have Read and Write, and Grant Access permissions for that object. This allows you to grant any Access or Grant permissions to any other users for that object.

Once you have created an object, any other user with Grant permissions on the object can reassign permissions, overriding any permissions you assigned.

When you create a user, the user does not have access to anything. You can assign permissions to that user for an entire class of objects on the user's Access permissions tab. For more information, see "Access permissions tab" on page 331.

Viewing which types of objects you can create
You can view which types of objects you can create.
Procedure

Click Account Settings and view the Access permissions tab for your account. This shows the effective permissions that you get explicitly or implicitly through the roles to which you belong. For more information, see “Access permissions tab” on page 331.

Grant permissions

The grant permissions allow you to assign permissions to other users.

The grant permissions are:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Ability to grant</td>
<td>Cannot grant permissions.</td>
</tr>
<tr>
<td>Ability to grant read-only permissions</td>
<td>Can grant Read-only permission.</td>
</tr>
<tr>
<td>Ability to grant read access granting</td>
<td>Can grant ability to grant Read-only permission.</td>
</tr>
<tr>
<td>Ability to grant read and write permission</td>
<td>Can grant read/write permission.</td>
</tr>
<tr>
<td>Ability to grant read and write access granting</td>
<td>Can grant ability to grant read/write permission.</td>
</tr>
</tbody>
</table>

The user rtmadmin is the only user that is always guaranteed to have full permissions on all objects.

Procedure

1. Select the objects and click Permissions.
2. Select the users to modify and click Change Permissions then choose the Grant permissions on the Advanced tab.

Permission restrictions

You must be aware of certain restrictions when granting permissions.

These restrictions are as follows:

- You cannot lower another user’s permission on an object for which they have a higher permission than you.
- You cannot raise another user’s permission on an object to be higher than your own permission on that object. You usually encounter this restriction when attempting to assign permissions on multiple objects at the same time, where your permission on one of the objects is less than your permission for the others.

Permission inheritance and dependencies

Objects that track permissions control access to the object and to objects they can contain. Other objects inherit their permissions from the object in which they are contained.

The objects that control permissions are:

- Agents
- Business activities (which control access to contained scenarios, rules, alerts, and reportlets)
• Roles
• Tables, including data streams, lookup tables, and consolidated data streams
• Users (controls access to user accounts).
• Views, cubes, and dimensions
• Dashboards
• Dashboard objects
• External processes
• Create reports

Regardless of what permissions users have to an alert or reportlet, they can always see the information in alert notifications and reportlets that are sent to them.

Dependencies
When you create or edit objects, you are limited by the permissions of any dependent objects and by the permissions of any containing object.

The following are the objects that have dependent requirements for create or edit:

**Table**  You must have Read permission to the agent that feeds the table.

**View**  You must have **Read** permission to the data stream table or base view and any lookup table that feed the view.

**Cubes**  You must have **Read** permission to the data stream table or base view and any dimension that feeds the view.

**Business activity**
To see the definitions of any objects contained in a business activity, you need **Read** permission on that business activity. To see definitions for contained scenarios, rules, alerts, and reportlets you need **Read** permission on the underlying view. To create a contained object, you need **Read and Write** permission on the business activity, and **Read** permission on the underlying view.

Permissions on one object can affect a user’s access to another, especially for restrictions on views. For example, you might be able to edit an alert but not the reportlet for the alert when the view for the reportlet is **No Access**, but the view for the alert is **Read Only**.
Chapter 24. Portal server integration

You can integrate IBM Cognos Real-time Monitoring servers with one of the supported portal servers, in accordance with the JSR-168 specification.

Prerequisites for portal server integration

Several prerequisites must be in place before you begin the procedure for portal server integration.

These prerequisites include:

- IBM Cognos Real-time Monitoring servers must be started and running on any of the supported application server implementations.
- One of the following supported portal servers must be installed and running:

<table>
<thead>
<tr>
<th>Portal server</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEA WebLogic portal server</td>
<td>The version bundled with WebLogic Application Server v 8.1 SP5. Be sure to also install the bundled WebLogic Workshop, as this will be required to complete several steps in the implementation.</td>
</tr>
<tr>
<td>IBM WebSphere portal server</td>
<td>Version 5.1.0.1 This product is bundled with WebSphere Application Server version 5.1.13. (DB2 only) If you are using WebSphere Application Server with a DB2 database, you also require the DB2 Universal V 8.1 or 8.2 fix pack 10.</td>
</tr>
<tr>
<td>JBoss portal server</td>
<td>Version 2.2.1 SP3 This product is bundled with the JBoss 4.0.3 SP1 application server.</td>
</tr>
</tbody>
</table>

Integrating with BEA WebLogic portal server

This section describes how to convert the lavaJSR168.war file and deploy it into a WebLogic portal server.

Converting the lavaJSR168.war file

To integrate with WebLogic portal server, you must first convert the lavaJSR168.war file to the WebLogic portal specification using the Portlet Preparer Tool utility available from BEA.

Procedure

1. From the Portlet Preparer Tool utility, extract the portletConverter.jar file to a temporary directory; that is, $CONVERTER_ROOT$.
2. Copy the lavaJSR168.war file to the same directory.
3. In the Portlet Preparer Tool utility, set WEBLOGIC_HOME in the env.sh or env.bat file (depending upon the platform) to the same temporary directory.
4. Run the following command:
   ```
   run ant -Dwar.file=lavaJSR168.war
   ```
   This creates a subdirectory called tempDir.
5. Using WebLogic Workshop, import the tempDir as a portal web project.
Deploying the lavaJSR168.war file into WebLogic portal server

After converting the lavaJSR168.war file, you can deploy it.

**Procedure**

1. Install WebLogic Portal Server, if it is not installed already.
2. Create a new portal domain instance and start the new portal domain server.
3. Using WebLogic Workshop, create a new portal application.
4. Into the new portal application, import the tempDir portal web project you created in the previous section.
   Before proceeding, ensure that the lavaJSR168.war file is visible under the WEB-INF/lib directory of the web project and that the cqjsr168.jar file is visible under the portlet section.
5. Launch the Portal Administration Console and log in.
   Before proceeding, ensure that the correct web application is selected in the console. Ensure also that the dashboard displays under portlets in the modules node.
6. Create a new page and add the dashboard.
7. Create a new portal.
8. Under the new portal, create a new desktop.
9. Under the new desktop, create a new blank book and add the page you previously created.
10. View the desktop to launch the portal application.
11. In the dashboard portlet, go to Edit mode and log in to the running IBM Cognos Real-time Monitoring servers.
    This action displays the available metrics.
12. Select the metrics that you want to be displayed in the portal page.
13. Click OK.
    This action displays the dashboard objects that are available for the portlet.

Integrating with IBM WebSphere portal server

You can integrate IBM Cognos Real-time Monitoring with IBM WebSphere portal server.

To perform the following procedure, review the WebSphere Portal documentation.

**Procedure**

1. Install WebSphere portal server, if it is not installed already.
2. Log in to the portal administration console.
4. Create a new virtual portal server with an appropriate name.
5. From the newly created virtual portal server, create a new page.
6. When configuring page layout:
   - Select a two-column layout.
   - Add the dashboard portlet.
7. When completed, the portal displays the page name as a link in the left column. When you click the Page link, the right column displays the dashboard portlet.
8. Click the Page link, and go to the Edit mode of the dashboard portlet.
9. Access IBM Cognos Real-time Monitoring servers by entering the user name, password, and URL of the running instance.
   This action displays the available metrics.
10. Select the metrics that you want to be displayed in the portal page.
11. Click OK.
    This action displays the dashboard objects that are available for the portlet.

---

**Integrating with JBoss portal server**

You can integrate IBM Cognos Real-time Monitoring with the JBoss portal server.

**Procedure**

1. Install JBoss portal server, if it is not installed already.
2. Shut down the JBoss application server, if it is running.
3. Copy the `lavaJSR168.war` file to the deployment directory in the JBoss application server installation.
4. Restart the JBoss application server:
   - `...\$JBOSS_HOME\bin\run.bat` (Windows)
   - `.../$JBOSS_HOME/bin/run.sh` (UNIX)
5. Start the View page in a browser.
   For example, the URL can be `http://[host]:[port]/portal/portal/default/[page name]`.
   The page name value can be customized by modifying the `<page-name>` element in the `celequest-objects.xml` file in the WAR package. The default is `celequestpage`.
6. Go to the Edit mode of the dashboard portlet.
7. Access IBM Cognos Real-time Monitoring servers by entering the user name, password, and URL of the running instance.
   This action displays the available metrics.
8. Select the metrics that you want to be displayed in the portal page.
9. Click OK.
   This action displays the dashboard objects that are available for the portlet.

---

**Multiple portlet instances on a JBoss Portal Server**

You can create multiple dashboard portlet instances on the JBoss portal server. You can create multiple portal pages and have different sets of dashboards displayed on them.

**Create multiple instances of CelequestDashboardPortletInstance**

You can create dashboard portlet instances on the JBoss portal server.

**Procedure**

1. Log in to JBoss Portal with the user name admin and password admin.
2. Choose the Admin Portal page on the Tab.
3. Click the Manage Instances Link.
4. Select LavaDashboardPortletInstance.
5. Give the instance a new name and click Create New Instance.
6. Repeat steps 4 and 5 to create multiple instances of LavaDashboardPortletInstance.
Creating multiple pages on the JBoss portal server:

You can create multiple portal pages on the JBoss portal server.

Procedure
1. Click Manage Portal Link.
2. Click the root folder in the tree.
3. Click the default page under the root tree.
4. Create a new page (for example, celPage1) in the left frame, then click that page.
5. Now, associate the new page with a LavaDashboardPortletInstance instance on the center of the page.
6. Click Preview Link for the page in step 5 and edit the IBM Cognos Real-time Monitoring dashboard portlet. For example, you can add Dashboard objects to the page.
7. Repeat steps 4, 5, and 6 to create Cognos Real-time Monitoring dashboard portlets for each LavaDashboardPortletInstance instance.
Chapter 25. Processes

A process is the set of steps that accomplish a task. A real transaction through a process is a process instance.

The following example shows a four-step process for making a request for approval. A request is made, reviewed, and then approved or rejected. A real transaction, such as a specific request for approval, is a process instance, such as a specific request for approval.

Business process management (BPM) systems create and manage business processes and instances. When a BPM is managing a process instance, it sends details about each step of the transaction to IBM Cognos Real-time Monitoring, which then develops statistics about the entire process. For example, the system might determine how long, on average, it takes to complete the entire process, whether the process is getting faster over time, what percentage of requests are rejected, or how long the review step takes.

How processes work

Process instance details arrive in IBM Cognos Real-time Monitoring as events. The Cognos Real-time Monitoring server aggregates the details in views or cubes to generate the statistics. Cognos Real-time Monitoring Dashboard then presents the process as a diagram, and includes the statistics in a table. Other objects can display other metrics, such as charts that present statistics over time.

Cognos Real-time Monitoring Dashboard also provides an optional query to the BPM to get the details about a specific process instance, such as where it is in the process. When a user makes such a request, Cognos Real-time Monitoring Dashboard queries a lookup table in Cognos Real-time Monitoring servers, and that table queries the BPM for the specific instance details. The results are then returned to Cognos Real-time Monitoring Dashboard for display.
The process management system generates a process definition file that describes the process in XML. A process definition in IBM Cognos Real-time Monitoring Workbench then associates that definition file with an aggregate view or cube and optionally with a search lookup table. Cognos Real-time Monitoring Dashboard uses that object as the source for a process chart and to identify the associated aggregate view or cube and search lookup table.

Creating and using processes

To create and use processes, the external business process system must perform several tasks.

These tasks include:
- Generate a process definition file in an XML format recognized by IBM Cognos Real-time Monitoring.
- Publish process step statistics as event data into Cognos Real-time Monitoring servers.
- Provide an interface for the process instance queries from Cognos Real-time Monitoring lookup tables. This task is optional

Procedure
1. Create an agent (if necessary) and a data stream table to receive the process events from the BPM.
2. Create one view or cube per process.
3. (Optional) Create a lookup table to query the BPM.
4. Create a process definition object.
5. Create a process diagram.
Data stream tables
Data stream tables receive and aggregate the instance statistics of a process. Event data is received in a data stream table, usually arriving through an agent. The aggregate views are based on the data stream table.

For more information about these components, see the following topics:
- Chapter 4, “Agents,” on page 21
- Chapter 10, “Data streams,” on page 77
- Chapter 36, “Views,” on page 337
- Chapter 8, “Cubes,” on page 57

The following requirements also apply:
- There must be only one agent and data stream table per BPM.
- There must be one view or cube per process. Use a WHERE clause to distinguish the process events from other processes in the data stream table, such as 'Process Name'='Request Approval'. For more information, see process definitions in the IBM Cognos Real-time Monitoring Workbench User Guide.

Lookup table search
The lookup table generates a query to the BPM whenever an IBM Cognos Real-time Monitoring Dashboard user makes an ad hoc query about a specific process instance.

For more information, see Chapter 20, “Lookup tables,” on page 247.

When the lookup table receives a query from Cognos Real-time Monitoring Dashboard, it first looks for the details in the lookup table cache. If the instance is not in the cache, the table then queries the BPM for the details. Be sure to define a reasonable invalidation schedule for your business, or disable the cache if the queries need to return the most up-to-date information about the process instance.

Process definition files
To create a process definition, you must have a process definition file that describes your Business Process Model (BPM). This process definition file must conform to XML Process Definition Language (XPDL) 2.0.

IBM Cognos Real-time Monitoring supports:
- Multiple workflowProcess elements within an XPDL document
- Default icons for all event and gateway types.

All XPDL documents must specify 2.0 in the Package.PackageHeader.XPDLVersion element.

The following are the exceptions to the XPDL 2.0 standard in the rendering of process diagrams by Cognos Real-time Monitoring:
- ActivitySet elements and activities and transitions defined within an ActivitySet element render on top of all other objects within a workset. (Activity sets render on top of other objects because they are expected to depict a subprocess on top of the current process.) Other objects render according to their location in the XPDL file, where the later an object is specified in your XPDL definition, the more likely it is rendered on top of other objects. The current order from top to bottom is:
- Transitions
- Activities
- Artifacts
- Associations
- Messages
- Lanes
- Pools

According to the XPDL 2.0 standard, the NodeInfoGraphicsInfo element is optional for each node. However, Cognos Real-time Monitoring requires a NodeInfoGraphicsInfo element for each node in the process diagram. If more than one NodeInfoGraphicsInfo element is associated with a single node, only the element with the property ToolID set to "Cognos Real-time Monitoring" is read. Also, one Coordinates element is expected with each NodeGraphicsInfo element. The X and Y coordinates must correspond to the upper left corner of the bounding box of the node. Also, specify the height and width.

According to the XPDL 2.0 standard, the ConnectorGraphicsInfo element is not required for each connector-type element. However, If you do not specify ConnectorGraphicsInfo elements, a straight route is used from one node to another. Transitions that specify a Condition type of CONDITION render a conditional diamond at the starting point of the transition. A Condition type of OTHERWISE renders a backslash (\) at the beginning of the transition. When specifying a ConnectorGraphicsInfo element, you must specify coordinates from start to finish and not from finish to start.

Events are rendered only as specified by the XPLD 2.0 standard. Deprecated attributes marking a start or end process are not supported by Cognos Real-time Monitoring.

Connectors and nodes that contain a Name attribute render with the text of the Name attribute as a label.

When specifying URLs to custom icons, the URL can be specified either in a Shape attribute of a NodeGraphicsInfo element or in the Icon subelement of an Activity element. You can use either a full URL or a URL relative to the URL of your Cognos Real-time Monitoring Dashboard. Custom icons are not skewed to match the width and height specified in the NodeGraphicsInfo element. Custom icons retain their aspect ratio.

**Creating process definitions**

You can create process definitions.

For details about this task, see process definitions in the IBM Cognos Real-time Monitoring Workbench User Guide.

**Before you begin**

Before creating a process definition, you must have:

- A process definition file. This file must conform to the XML Process Definition Language (XPDL) 2.0.
- Read-only access permission on the view or cube that aggregates the process events.
- A view or cube that aggregates the process-instance data, where one column specifies the steps in the process.
It is optional to have Read-only access permission on the search table for the lookup table.

**Procedure**

1. In IBM Cognos Real-time Monitoring Workbench, click the **Workbench** tab.
2. Click **Activities**.
3. Click **Create New > Process Definition**.
4. Name the process definition, specify a folder, and provide a description. The default folder is **Public Folders**. A description is optional.
5. Click **Upload Process Definition File** to upload the process definition file generated by the BPM.
   The process name from the definition file appears in the **Process Name** field after uploading the file. For more information, see “Process definition files” on page 287.
6. Click **Select Data Source** to select the aggregate view or cube.
7. Use the **Step Column** list to select the step column. For example, each step in the process might be identified by a name or ID number.
8. To create a detail search, click **Select Data Source**, and select the column to search from the **Search Column** list. You can enter text in the **Descriptive Name** field. This name appears in IBM Cognos Real-time Monitoring Dashboard.
9. Click **Save**.
   You can now create process diagrams based on this definition.

**Creating process diagrams**

You can create process diagrams in IBM Cognos Real-time Monitoring Dashboard using the process definitions that you created in IBM Cognos Real-time Monitoring Workbench.

For details about this task, see the section on process diagrams in the IBM Cognos Real-time Monitoring Dashboard User Guide.

**Before you begin**

Before creating a process diagram, process definitions must already be defined in IBM Cognos Real-time Monitoring Workbench. Also, you must have Read-only access permission on the aggregate view or cube that provides the statistics.

**Procedure**

1. In IBM Cognos Real-time Monitoring Workbench, open the **Process Diagrams** manager.
2. Click **Create Diagram**.
3. Select the process definition as defined in IBM Cognos Real-time Monitoring Workbench.
4. Select the source columns to present as statistics.
5. Name the process diagram and save it.
   IBM Cognos Real-time Monitoring Dashboard immediately presents the process diagram. The statistics update as events arrive for the specific process.
Chapter 26. Reportlets

Reportlets describe the contents of a view and present that information in a report that is either attached to an alert message or presented by an external system.

For example, when inventory is low for a product, and a restock shipment is overdue, alerts might notify purchasing managers of that state and a reportlet attached to the alert might list the alternative suppliers for that product. Reportlets are attached to all subscribers of the associated alert. For more information, see Chapter 5, “Alerts,” on page 37.

There are two types of reportlets:

- Internal reportlets are the visual representation of the information in a view when the alert generated the reportlet. The presentation is a table formatted in either text or HTML that contains all of the information that was in the view.
- External (third-party) reportlets are produced by external reporting systems. External reportlets present a report based on view data passed to them when the user clicks a link to the external system. That system is responsible for generating and presenting the report.

Creating reportlets

You can create reportlets to provide information about an event.

For information about reportlet attributes, see “Reportlet attributes” on page 292 or “External reportlet attributes” on page 292.

Before you begin

You must have specific permissions to create a reportlet:

- Create permission. For more information, see Chapter 23, “Permissions,” on page 275 and “Granting permission to create objects” on page 278.
- Read/write permission on the business activity that contains the reportlet. For more information, see Chapter 23, “Permissions,” on page 275.
- Read-only permission on the view that provides data to the reportlet

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Scenario Modeler tab.
2. Under Business Activities, click an existing scenario that will contain the reportlet.
3. Click the Reportlets tab.
4. Click New Reportlet.
5. Select the type of reportlet to create.
   - Reportlets are formatted as HTML tables.
   - External (third-party) reportlets are defined and produced by external reporting systems based on the data passed to them.
The external reportlets option is available only when external links are defined. For more information, see the IBM Cognos Real-time Monitoring Workbench User Guide.

6. Complete the attribute fields in the Create Reportlet window.
7. Save the reportlet as enabled.

You can also create a reportlet when creating or editing an alert. Doing so automatically attaches the reportlet to that alert. For more information, see Chapter 5, “Alerts,” on page 37.

## Reportlet attributes

Reportlets are formatted as HTML tables that have particular attributes.

### Table 86. Reportlet attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the reportlet. The name can contain letters and numerals only. This name must be unique among reportlets within the same scenario. See &quot;Object namespace&quot; on page 261 for details.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Status</td>
<td>Whether the reportlet is enabled or disabled. When the containing scenario is disabled, you cannot make the reportlet enabled. The scenario must be enabled before the reportlet can be enabled.</td>
</tr>
<tr>
<td>View</td>
<td>Business view from which the report draws its data. For more information, see Chapter 36, “Views,” on page 337.</td>
</tr>
</tbody>
</table>

## External reportlet attributes

External reportlets present reports based on view data passed to them when the user clicks a link to the external (third-party) reporting system.

That system is responsible for generating and presenting the report. The external reportlet definition identifies the external link and the view information to pass to the external system which uses the information to identify the report to present. For example, an external report might present a PDF file that contains the complete description of a product identified in an alert.

### Table 87. External reportlet attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reportlet Name</td>
<td>Identifies the reportlet. The name can contain letters and numerals only. This name must be unique among reportlets within the same scenario. See &quot;Object namespace” on page 261 for details.</td>
</tr>
<tr>
<td>Status</td>
<td>Whether the reportlet is enabled or disabled. When the containing scenario is disabled, you cannot make the reportlet enabled. The scenario must be enabled before the reportlet may be enabled.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that may contain any text characters.</td>
</tr>
<tr>
<td>Data from View</td>
<td>Business view from which the report draws its data. Contains the column data to send to the external report. For more information, see Chapter 36, “Views,” on page 337.</td>
</tr>
<tr>
<td>Report Name</td>
<td>Name of the report in the external system. This is the DocName element in the URL that communicates with the external report system: <a href="http://localhost.com?DocName=">http://localhost.com?DocName=</a>&lt;Report Name&gt;</td>
</tr>
</tbody>
</table>
### Table 87. External reportlet attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| Report Parameters  | Parameters to pass to the external system. Each parameter corresponds to a column in the view. The reportlet substitutes the value of each named column into the URL. For example, the URL is defined as follows:  
...?DocName=<Report Name>&Parameter1=PROD_ID&...

   It looks similar to this when sent to the external system:  
...?DocName=<Report Name>&Product="product_id"&...

| Report Arguments   | This panel specifies argument values that can be assigned to arguments in the URL specified in the external link. For example, consider the case in which the external link defines a base URL for an IBM Cognos report, and the base URL needs the following arguments:  
ui.tool=CognosViewer&ui.object=content/folder[@name='Demo Report']/report[@name='Inventory Levels by Retailers']ui.action=run &run.outputFormat=PDF

   The base URL is http://server_name/ibmcognos/cgi-bin/cognos.cgi?b_action=xts.run&m=portal/launch.xts&

   When the Associated Report is called from the dashboard object with the values specified for the arguments in the Report Arguments panel, the actual URL request is constructed as follows:

   http://server-name/ibmcognos/cgi-bin/cognos.cgi?b_action=xts.run &m=portal/launch.xts&ui.tool=CognosViewer&ui.object=content/folder[@name='Demo Report']/report[@name='Inventory Levels by Retailers']ui.action=run&run.outputFormat=PDF

| Display Link       | Shows the complete, qualified URL that will appear in the alert message and is the link to the external report system.                                                                                                                                                 |

The URLs used to communicate with the external report system begin with the string defined for the report in the External Links list on the **Workbench** tab in the Cognos Real-time Monitoring Workbench. For more information, see the IBM Cognos Real-time Monitoring **Workbench User Guide**.

**Reportlet views**

Reportlets retrieve their information from the business view that the alert is based on or from any view derived from the same data stream.

For example, consider a rule that generated the customer alert based on the view InventoryLow. Another view, AvailableSuppliers, adds context by indicating alternative suppliers. The reportlet attached to the alert can draw information from either of these views. Because ShippingNotices is derived from the same data stream, you could also retrieve information from it. However, you cannot retrieve information from OrderDetails because it is on a different data stream.
For more details about the information that appears in the reportlet view, see "Reportlet filtering" on page 44.

The reportlet view cannot be a synchronized join. For more information, see "Synchronized joins" on page 340.
Chapter 27. Roles

Roles define the minimum sets of permissions associated with users.

Using roles, you can quickly assign the same permissions to objects or sets of objects. For example, an operator role might provide full permissions to agents, but not to data streams or business activities, while an application developer role might have full permissions on all objects, except agents.

Users can belong to none, one, or several roles. To see which roles a user belongs to, edit the user account and view the User details tab. To see which roles you belong to, click Account Settings and view the tab. For more information, see “User details tab” on page 329.

Permissions for a particular operation depend on permissions associated with that user’s roles and individual permissions assigned to the user. Consider a user with two roles: one role has read-only access to the views class, and the other role has read and write access to the same views class. The user has read and write permissions for the views class, and can edit any view in this class.

Similarly, if a user is a member of one role, and that role has read-only permissions for all views, but read and write permissions for a particular view, that user can edit only that particular view.

Roles, like all IBM Cognos Real-time Monitoring objects, are protected by permissions. Only users with specific permissions on the roles can modify or delete them. For example, to add users to a role, you need read and write permissions for that role.

The Roles list in the Administration Console shows the current roles in the system.

For more information about specific access levels for roles, see Chapter 23, “Permissions,” on page 275.

Creating roles

You can create roles to define the minimum sets of permissions associated with users.

Before you begin

To create a role, you need to create permissions for roles. For each object class, you can assign up to the greatest permission that you have for that class.

For example, if you have create permissions for a class, you can assign any of the following permissions:

- No Access
- Read-only
- Read and Write
- Create
Procedure
1. On the Administration Console tab, click Roles to see the list of all currently defined roles.
2. Click New Role.
3. Specify the role attributes, assign access permissions, identify the members of the role, and save the new role.

Role attributes
Each role object has attributes.

Table 88. Role attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Specifies the role name. The name must be unique among all objects in the root folder, such as business activities, users, roles, dashboard objects, and so on.</td>
</tr>
<tr>
<td>Description</td>
<td>Provides an optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Access Permissions</td>
<td>Specifies access permissions for each class of objects assigned to this role. These are the minimum permissions. A user can have more permissions assigned individually to an object or a class of objects.</td>
</tr>
<tr>
<td>Members</td>
<td>Specifies the users associated with this role.</td>
</tr>
</tbody>
</table>

The access permissions tab
Access permissions are the global permissions that a user or role can have for an object or a class of objects.

On this tab, you can specify permissions for:
- Business activities
- Views, cubes, and dimensions
- Lookup tables and data streams
- Users
- Roles
- Agents
- Global system properties
- Dashboards
- Dashboard objects
- External processes

New users and roles have no access permissions for any object. As a result, the users and roles receive and view alerts and reportlets allowed only by mandatory subscriptions.

For more information about assigning permissions, see Chapter 23, “Permissions,” on page 275.
Rules monitor business activities by analyzing business views looking for metrics that meet specific conditions.

Rule conditions are spreadsheet-like formulas that evaluate the changing business metrics looking for exceptional conditions. When a condition is found to exist, an alert is sent to key personnel.

You can create rules that send alerts every time the condition is found to exist (fire), rules that send alerts once and ignore subsequent events until the initial condition is resolved (raise), or rules that reset (lower) previously raised rules.

In earlier versions of the product, the rule was able to monitor only views derived from the scenario default view. This restriction no longer applies.

Creating rules

You can create a new rule.

**Before you begin**

You must have Create permission for business activities, and Read and Write permission on the business activity that contains the rule, and Read-only permission on the view that provides data to the rule.

For more information about permissions, see "Granting permission to create objects" on page 278.

**Procedure**

1. In IBM Cognos Real-time Monitoring Workbench, click the Scenario Modeler tab.
2. Under Business Activities, click an existing scenario that will contain the rule.
3. Open an existing scenario.
5. Choose the data source for the rule to monitor.
6. Enter the rule attributes in the Create Rule window. Additionally:
If the scenario has a default view, that view appears selected. You can choose another source to monitor by clicking **Select data source**.

For a view, choose the view.

For a cube, choose the dimension level in a cube. You can also apply a filter that further restricts the data that the rule monitors.

If the source contains data, that data provides a sample of what to expect. When the source is empty, the window displays only the column names and the message No Data Available.

To identify the alert to activate when the rule condition is met, either select an existing alert by clicking this alert in the Rule effect field, or click **Next** to access the Alert Definition window.

7. Save the rule as enabled.

**Cloning rules**

You can clone and modify existing rules.

**Before you begin**

You must have Create permission for business activities, and Read and Write permission on the business activity that contains the rule and Read-only permission on the view that provides data to the rule.

For more information about permissions, see “Granting permission to create objects” on page 278.

**Procedure**

1. Edit the rule you want to clone.
2. Change the rule name, and change the other attributes that differ from the original rule.
3. Click **Save as New Rule**.

**Rule attributes**

Every rule has attributes.

*Table 89. Rule attributes*

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Data Source</td>
<td>Identifies the view or cube that the rule monitors. Click the <strong>Select Data Source</strong> button to change the data source for the rule.</td>
</tr>
<tr>
<td>Rule Name</td>
<td>Identifies the rule object. The name can contain letters and numerals only. This name must be unique among rules within the same scenario. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the rule is enabled (receiving new information) or disabled. When the containing scenario is disabled, you cannot make the rule enabled. The scenario must be enabled before the rule can be enabled.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
</tbody>
</table>
### Table 89. Rule attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Condition</strong></td>
<td>The spreadsheet-like true or false formula that is evaluated against the associated business view. When the formula evaluates to True, the condition exists and the system then sends the alert. For more information, see &quot;Rule conditions.&quot; When the condition exists, an action is taken. Rules can send alerts every time the condition is found to exist (fire), send alerts once and ignore subsequent events until the initial condition is resolved (raise), or reset (lower) previously raised rules. For information about actions taken when a condition is met, see &quot;Rule actions&quot; on page 300.</td>
</tr>
<tr>
<td><strong>Holds for</strong></td>
<td>Sends the alert only when the condition holds true for a specified length of time. No alert is sent if the condition becomes false at any time during the wait. If you omit the duration, the system sends the alert as soon as the condition exists. You can specify a value using one of the following methods: • Specify a number to indicate the length of time to wait. • Specify the name of a column in the view that contains the number. When you use this option, the rule takes the value from the event in the view. In locales where daylight saving time is observed, durations of days, months, and years are adjusted accordingly. While 1 day is typically 24 hours long, it can be 23 or 25 hours depending on the time of year.</td>
</tr>
<tr>
<td><strong>Basic</strong></td>
<td>Use the <strong>Basic</strong> tab to specify a single alert to send when the rule condition is met.</td>
</tr>
<tr>
<td><strong>Advanced</strong></td>
<td>Use the <strong>Advanced</strong> tab to select multiple alert options for the rule. In addition to selecting the alert to use, you can specify the type of action to take when the condition is met, maintain alert states for a specific column in a view, or consolidate multiple messages for a single event.</td>
</tr>
</tbody>
</table>

### Rule conditions

A rule condition is a formula that tests the row in the associated business view looking for a specific condition.

When the condition exists, the rule action activates an alert. Formulas can be simple tests for a value in a column in the view, such as Status='Resolved', or they can be complex Boolean expressions with functions, operators, and parenthesis groupings, such as (Status='Resolved' OR Status='Assigned') AND UPPER(cust_tier)='HIGH'.

A rule condition formula contains any number of column references, operators, and functions. However, the formula must conform to the following requirements:
• The formula result must be Boolean, returning true or false.
• All column references must be in the associated business views.
Only scalar functions (functions that apply to a single row in a view) can be used. To see which functions are available, click More Functions when entering the rule condition. For more information, see Chapter 15, “C-SQL functions,” on page 123.

Rule actions

Rules can have one of three effects. They can send alerts every time a condition is found to exist, send alerts once and ignore subsequent events until the condition is resolved, or reset previously raised rules to enable the rule to send alerts.

- Send alerts every time the condition is found to exist (fire). A fire action sends an alert every time a rule identifies an exceptional condition. For example, in a customer support center that tracks customer problems as ticket events, an alert might be fired every time a new ticket is opened.

- Send alerts once and ignore subsequent events until the initial condition is resolved (raise). A raise action sends an alert message when the rule’s condition applies but ignores subsequent events until after the initial condition is resolved. A raise action is useful when you do not want multiple alerts for situations where the rule condition is true for multiple, related events. For example, if an open customer problem ticket is edited, you do not want another alert for the edit event, even though the status of the second event is still open.

  You can send alerts once for each specific occurrence of the named column. For example, to send an alert each time a new problem ticket is opened, you might identify Ticket as the specific occurrence column. Now one alert is sent for the open event of each ticket, but the alert is ignored for all subsequent events to that ticket while its status remains open.

- Reset (lower) previously raised rules to allow them to send alerts.

Rule examples

When a Raise rule activates an alert, the alert does not activate again. Subsequent Raise rules for the alert are ignored until a Lower rule first resets it.

When a rule condition is Status=Open, the first event in the following example activates the alert, but the subsequent events are ignored unless the state of the alert is lowered. For example:

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>.......</td>
</tr>
<tr>
<td>0703</td>
<td>Open</td>
</tr>
<tr>
<td>0706</td>
<td>Open</td>
</tr>
<tr>
<td>0704</td>
<td>Open</td>
</tr>
<tr>
<td>0705</td>
<td>Open</td>
</tr>
</tbody>
</table>

In this example, it is more likely that you want an alert for each Open event. You can use a specific condition, which activates an instance of the alert for each unique occurrence of the values in the specified columns:

Raise someAlert when Status = 'Open' for unique occurrences of Ticket.

By identifying Ticket as the specific column, an alert instance is activated, and a message sent, every time for each Open event that does not already have an instance for the specific ticket number. Similarly, you can reset each alert instance individually with specific Lower rules.
Lower `someAlert` when `Status = 'Open'` for unique occurrences of `Ticket`.

If you ignore the specific fields in the Lower rule, the rule resets all instances of the alert that are raised.

**Rules that monitor alerts**

Rules typically analyze business views looking for metrics that meet specific conditions.

However, rules can also monitor generated alerts looking for conditions that require further attention with the `IS_RAISED()` function.

For more information about the `IS_RAISED` function, see “`IS_RAISED`” on page 153.

Monitor an alert and send a second one when the first remains raised.

---

**Figure 33. A first rule condition is met**

For example, if an alert was sent 4 hours earlier and is still in a raised state, another rule might notice that fact and generate a new, escalated alert. Consider these rule descriptions, where `EscalateAlert` is raised only when `OpenAlert` remains raised for at least 4 hours:

```
Raise `OpenAlert` when `Status = 'Open'`
Reset `OpenAlert` when `Status <> 'Open'`
Raise `EscalateAlert` when `IS_RAISED('OpenAlert')` holds for 4 HOUR.
Reset `EscalateAlert` when NOT `IS_RAISED('OpenAlert')`
```

**Specific alerts**

When an alert is generated for unique occurrences of fields, the system tracks each alert by those field values.

For example, the following rule raises alerts and tracks the open ones by the unique values of the `Ticket` field:

```
Raise `OpenAlert` when `Status = 'Open'` for unique occurrences of `Ticket`
```

To properly track this alert, the rule with the `IS_RAISED()` function needs the same specific condition:

```
Raise `EscalateAlert` when `IS_RAISED('OpenAlert')` holds for 4 HOUR for unique occurrences of `Ticket`
```
If you omit the specific field condition, the EscalateAlert is raised for the first OpenAlert only. Similarly, you must reset the alerts with rule conditions specific to the same fields, or else you reset all of the raised alerts.

- **Reset** OpenAlert when Status
  
  `<> 'Open' for unique occurrences of Ticket`

- **Reset** EscalateAlert when NOT
  
  `IS_RAISED('OpenAlert')`  
  
  `for unique occurrences of Ticket`

---

**The system log**

You can build a rule that monitors the system log messages looking for high priority error conditions and reports them to key administrators.

To do this, follow the steps for monitoring the logs in the IBM Cognos Real-time Monitoring Workbench documentation.

Cognos Real-time Monitoring generates messages that facilitate software service and maintenance by producing reports suitable for analysis by users, system administrators, support engineers, and software development teams. For details about the logging system, see the documentation for Cognos Real-time Monitoring Workbench.
Chapter 29. Salesforce

You can access and import tables from a Salesforce customer relationship management (CRM) system.

This is a two-step process. The first step is the creation of a Salesforce agent that connects to Salesforce tables. The second step is the use of a separate extraction wizard to import Salesforce data stream and lookup tables.

The imported Salesforce objects can then be used as data sources for dashboard objects, such as business views and cubes.

Creating a Salesforce agent

A Salesforce agent imports Salesforce tables for use in data stream and lookup tables.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > Agent.
4. Choose Salesforce as the agent type.
5. Enter a Name value and set Status to enabled.
6. Enter a User name and Password.
7. Click OK.

You can now use a wizard to import Salesforce data stream and lookup tables into IBM Cognos Real-time Monitoring Workbench.

Importing data streams and lookup tables from Salesforce

You can use the Import Salesforce Object wizard.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Administration Console tab.
2. Under Administration, click Wizards.
3. On the Wizards tab, click Import Salesforce Object Wizard.
4. Select a Salesforce agent.
5. Select the users to import from List of Users.
6. Select the tables to import from List of Tables for the data stream and lookup tables and click Add.
   When you add a table, it is removed from the List of Tables and placed in the Selected Tables list. To move it back, click Remove.
7. Click OK.

The data stream and lookup tables are created from the selected tables with _events (data stream) and _context (lookup table) added to the table name under Tables and Views. For example, if you import a Salesforce table Account into a data stream table, it has the data stream name Account_event.
If a table in the **Selected Tables** list is being used for a data stream or a lookup table, you cannot remove it from **Selected Tables**. You must first delete the data stream or lookup table.

### Salesforce flattening function

A function is available for use with Salesforce agents that flattens a Salesforce table when creating a lookup table from a Salesforce source.

However, the table can be flattened only if the table represents a tree structure, where the top level represents the root element of the tree. After the table is flattened, each row in the new table represents a leaf node of the original tree.

For example, consider the following table:

**Table 90. Table of nodes**

<table>
<thead>
<tr>
<th>NodeName</th>
<th>NodeID</th>
<th>NodeParentId</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>NULL</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

This table can be represented by the following tree. The tree has four levels where Level 0 has the root element A.

![Tree structure](image)

**Figure 34. Tree structure, where the top level represents the root element of the tree**

After flattening, the rows in the new table represents the leaf nodes C, D, F, and G.

**Table 91. Table rows that represent leaf nodes**

<table>
<thead>
<tr>
<th>Level0</th>
<th>Level1</th>
<th>Level2</th>
<th>Level3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>NULL</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td>G</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Syntax

The flattening function has the following syntax:

```plaintext
FLATTEN( tableName, idColumnName, parentIdColumnName,
nameColumn, prefixForLevelColumns )
```

Parameters

The parameters of the function are defined as follows:

- **tableName**
  The name of the table to flatten.

- **idColumnName**
  The ID column of the table (that is, the column to which `parentIdColumnName` joins). For example, in the previous description, the `idColumnName` is `NodeID`.

- **parentIdColumnName**
  The column that contains the node IDs of the parent nodes for each row. For example, in the previous description, node A has an ID of 1 and is the parent of node B; therefore, the parent ID in the column `NodeParentID` for Node B is 1. Likewise, the parent node of node D is node B; therefore, the `NodeParentID` for D is 2 because the node ID of node B is 2.

- **nameColumn**
  The column that represents the data to be shown in the flattened table.

- **prefixForLevelColumns**
  The string prefix that is added to the column. In the previous description, the prefix is `Level`.

Remarks

You must have data caching with prefetch enabled in your lookup table for the flatten function to work properly. See "Caching lookup table queries" on page 250 for details about enabling caching.

Example

The following example shows the table before flattening.

![Table before flattening](image)

Figure 35. Table before flattening

The following example shows the table after it is flattened with the function:

```plaintext
flatten(USERROLE, ID, ParentRoleID, Name, cq_)
```
Salesforce picklist function

In addition to the flatten function, there is also a picklist function available for use with Salesforce agents. The picklist function returns a single column data set of the possible values for a picklist typed column in Salesforce.

Syntax

The picklist function has the following syntax:

PICKLIST( tableName, pickListColumn )

Parameters

The parameters of the function are defined as follows:

- tableName
  The name of the table from which to return the single column data set.
- pickListColumn
  The column in the table specified by tableName to return.

Remarks

You must have data caching with prefetch enabled in your lookup table for the picklist function to work properly. For details about enabling caching, see "Caching lookup table queries" on page 250.

The column to return must be a picklist typed column.

Example

The following picklist function returns the data set of possible values from the LeadSource column in the Opportunity table.

picklist(opportunity, leadsource)

The following illustration shows the Opportunity table:
The picklist function retrieves the LeadSource column.

Salesforce administration console

To obtain a Salesforce.com account you must have access to the IBM Cognos Real-time Monitoring for AppExchange administration console.

To access the console, enter the following address in your browser:
http://[localhost]:[port]/[install location]/jsp/salesforceadmin.jsp

On the page that displays, enter the information requested.

After you click Submit, you are sent an email that confirms your request was successfully submitted. It also informs you that a representative will contact you to confirm the submission and provide you with the information about the next steps for completing the configuration process.
Chapter 30. SAP connectivity

You can access and import fact tables in the form of ODS objects and cubes in the form of OLAP cubes from an SAP system.

This is a three step process. The first step is the creation of a standard JDBC agent that connects to the SAP database. The second is the creation of an ERP agent that uses an SAP metadata JDBC agent to access the SAP system as a source type. The third step is the use of a separate extraction wizard to import ODS objects or OLAP cubes.

The imported ODS objects and cubes can then be used as data sources for dashboard objects like business views and cubes.

Creating an SAP agent

An SAP agent is an enterprise resource planning (ERP) agent that uses a JDBC agent to access the database for the SAP system.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Select Create New > Agent.
4. Choose ERP as the agent type.
   When you select the agent type, the subsequent settings redisplay to show settings appropriate to that type.
5. Enter a Name value and set Status to Enabled.
6. For ERP Source Type, select SAP-BW.
7. For Data Agent, select SAP Application Layer. You can also use a JDBC agent to connect to the SAP database. For instructions on creating a JDBC agent, see "Java Database Connectivity agents" on page 221.
8. Complete the Connectivity Settings as appropriate, filling in the fields:
   • User Name
   • Password
   • Language
   • ASHost
   • Client
   • System Number
   • Gateway Host
   • Gateway Server
9. Click OK.

You can now use a wizard to import ODS objects (see "Importing operational data store objects" on page 310) and OLAP cubes (see "Importing OLAP cubes" on page 310).
Importing operational data store objects

You can import operational data store (ODS) objects as either a data stream or a lookup table.

For more information, see Chapter 10, “Data streams,” on page 77 and Chapter 20, “Lookup tables,” on page 247.

Procedure
1. Open the Workbench Administration Console.
2. Under Administration, click Wizards.
3. On the Wizards tab, click Import ODS Object Wizard.
4. Select an existing ERP agent.
5. Select the Info Group that contains the ODS object that you want.
6. In the Table to Import drop-down list, select the fact table that you want.
7. Choose whether to import the ODS object as a data stream or a lookup table.
8. Click OK.

Importing OLAP cubes

You can use the Import ERP Cube wizard.

Procedure
1. Open the Workbench Administration Console.
2. Under Administration, click Wizards.
3. On the Wizards tab, click Import ERP Cube Wizard.
4. Select an ERP agent.
5. Select the cube that you want to import.
6. In the Import as field, accept the automatic default (the same names as in SAP) or enter a new name by which the imported cube will be identified in the Workbench.
7. Click OK.
Chapter 31. Scenarios

A scenario is a collection of rules, alerts, and reportlets that identify exceptional business conditions.

The rules in the scenario are the tests that determine when the exceptional condition exists or when it no longer exists.

Deleting or disabling a scenario deletes or disables its contained rules, alerts, and reportlets.

Every scenario has certain attributes.

Table 92. Scenario attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder Status</td>
<td>Specifies if the rule is enabled (receiving new data) or disabled. When a scenario is disabled, all of its rules, alerts, and reportlets are also disabled. When the containing business activity is disabled, you cannot make the scenario enabled. The business activity must be enabled before the scenario can be enabled.</td>
</tr>
<tr>
<td>Scenario Name</td>
<td>Identifies the scenario object. The name can contain letters and numerals only. This name must be unique among scenarios. See &quot;Object namespace&quot; on page 261 for details.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description of the scenario that may contain any text characters.</td>
</tr>
<tr>
<td>Default View</td>
<td>Identifies the default view (see Chapter 36, &quot;Views,&quot; on page 337) that the rules of this scenario monitor, and reportlets report on. Selecting the No Default View requires that you pick a view for rules and reportlets when you create them. Selecting the Existing View button allows you to select an existing view from IBM Cognos Real-time Monitoring Workbench.</td>
</tr>
<tr>
<td>View Name</td>
<td>The name of the view to use for the scenario. This attribute is only available if Existing View is selected. Click Browse to select the view.</td>
</tr>
</tbody>
</table>
Creating scenarios

You can create a scenario.

You can also delete scenarios.

Attention: Deleting a scenario deletes the rules, alerts, and reportlets that the scenario contains. After deletion, the rules, alerts, and reportlets cannot be restored.

Before you begin

To create a scenario, you must have the following permissions:

- Create permissions (see Chapter 23, “Permissions,” on page 275) for business activities (see “Granting permission to create objects” on page 278 for details)
- Read and Write permissions (see Chapter 23, “Permissions,” on page 275) on the business activity that will contain the scenario
- (Optional) Read-only permission on the default view for the scenario

Procedure

1. Open the Scenario Modeler.
2. Select one of the Business Activities to contain the scenario.
3. Click New Scenario.
4. Complete the fields in the Create Scenario window. To delete a scenario, select the scenario to delete and click Delete scenario(s).
Chapter 32. SELECT statements

C-SQL SELECT statements define the views that manage information in IBM Cognos Real-time Monitoring.

IBM Cognos Real-time Monitoring Workbench constructs SELECT statements based on the views you define in the graphical user interface, and then passes them to Real-time Monitoring server or servers for instantiation.

You can see the complete SELECT statement that defines a view in Cognos Real-time Monitoring Workbench by clicking Display SQL Expression when creating or editing a view.

Some of the advanced features of the SELECT command cannot be expressed by the options in Real-time Monitoring Workbench. For example, complex join conditions, query windows, and table expressions must be entered in the user interface.

This section describes the syntax and features of the C-SQL SELECT command in detail.

The C-SQL SELECT command is a subset and extension of ANSI SQL-99, a query language standard. The C-SQL implementation supports outer joins using the ANSI outer join (left outer join and right outer join) syntax, and aggregation functions in the select clause. Each individual statement is treated as a transaction and is committed as soon as it executes.

Syntax

The SELECT specification is defined by using several operators.

These operators are:

```
SELECT selectList
    FROM joinClauses
    [ WHERE searchCondition ]
    [ GROUP BY groupClause ]
    [ WINDOW windowClause ]
    [ ORDER BY orderClause ]
```

The operators are applied in the following order:

1. The “FROM clause” on page 316 specifies the base tables or views that provide data to this view.
2. The “WHERE clause” on page 319 (optional) filters the input to match specified criteria.
3. The “GROUP BY clause” on page 320 (optional) groups the resulting table on one or more columns.
4. “Select list” on page 314 defines the columns to appear in the resulting table.
5. Chapter 38, “Query windows,” on page 361 (optional) defines windows used by aggregate functions in the Select list.
6. "Stateful view semantics" on page 321 ORDER BY (optional) orders (sorts) the resulting table.

## Select list

The Select list statement defines which columns appear in the resulting virtual tables.

The select list immediately follows the SELECT keyword and has two forms:

- An asterisk (*) to choose all columns that are part of the source table.
  
  ```sql
  SELECT * ...
  ```

  The resulting view contains the columns of each of the input tables or views, in the order that they occur in the source, and in the order listed in the FROM clause (see "FROM clause" on page 316). For outer joins (see "Outer joins" on page 317), resulting columns that do not exist in both references are assigned NULL values.

- A list of unique column names or derived columns.

  ```sql
  SELECT columnNameList
  ```

  Where each columnName is separated by a comma (,) and is defined as
  
  ```sql
  columnName[[AS] aliasName [ OVER (windowClause) ]]
  ```

  The AS option assigns a new name to the associated column. The literal "AS" is optional.

  The OVER option defines an in-line window. See Chapter 38, “Query windows,” on page 361 for details.

  Where each columnName has one of the following sources:

  - simple column reference (the column name).
    
    ```sql
    current_rental_price
    ```

  - qualified column reference (one prefaced by the table name).
    
    ```sql
    movie_titles.current_rental_price
    ```

  To select all columns from one table while selecting some columns from other tables, use a qualified column reference and specify an asterisk (*) for the column name. For example, the following code selects all movie_title columns, and two columns from the media table:

  ```sql
  movie_titles.*, media.media_type, media.name
  ```

  - derived column (an expression, possibly a case expression), see "CASE expression" for details:
    
    ```sql
    MAX((movie_titles.current_rental_price/2)) AS Half_Price
    ```

  For derived columns, the aliasName is required.

## CASE expression

A CASE expression returns the result of an expression that corresponds to a matching true condition. Optionally, each condition can return NULL instead.

If no condition is found to be true, the expression returns the result of the ELSE condition or NULL when ELSE is omitted.

There are two forms of CASE expressions:

- Simple condition
Evaluates the caseExpression and compares it against the result of each equalsExpression until one matches, then returns the corresponding resultExpression result. Each of the equalsExpression must of a type comparable to the caseExpression.

```
CASE
  caseExpression
      [ { WHEN equalsExpression THEN { resultExpression | NULL }}...]
      [ ELSE { resultExpression | NULL } ]
END
```

- **Search condition**
  Evaluates each searchCondition until one is found to be true, then returns the corresponding resultExpression result.

```
CASE WHEN
  searchExpression THEN { resultExpression | NULL }
      [ { WHEN searchExpression THEN { resultExpression | NULL }}...]
      [ ELSE { resultExpression | NULL } ]
END
```

These CASE expressions have the same result:

```sql
SELECT Tier AS
    CASE WHEN Tier = 'High' THEN 'Priority customer'
        ...
SELECT Tier AS
    CASE Tier WHEN 'High' THEN 'Priority customer'
        ...
```

The following code is an example that generates running totals for each ticket status at each tier level:

```sql
CREATE VIEW VTotal_Tickets AS
    SELECT Tier,
        SUM(CASE Status WHEN Open THEN 1 ELSE 0 END) AS Opens,
        SUM(CASE Status WHEN Reopen THEN 1 ELSE 0 END) AS Reopens,
        SUM(CASE Status WHEN Resolved THEN 1 ELSE 0 END) AS Closes,
        (Opens+Reopens-Closes) AS Pending
    FROM VCustomerTickets
    GROUP BY Tier
```

The following table shows the possible results:

Table 93. CASE expression results

<table>
<thead>
<tr>
<th>Tier</th>
<th>Opens</th>
<th>Reopens</th>
<th>Closes</th>
<th>Pending</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Medium</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Low</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
FROM clause

The FROM clause specifies the tables and views that the new view is built from.
FROM reference [[ AS ] aliasName ]
[ , reference [[ AS ] aliasName ] ... ]

Where a reference is:
• Simple reference:
  tableOrView
• Join operation (see “Join operations,” for details):
  ( reference [(LEFT | RIGHT) [ OUTER ] | INNER ]
     JOIN reference ON searchCondition )
• Table expression (see “Table expressions” on page 317 for details):
  ( SELECT selectList FROM joinClauses [ WHERE
     searchCondition ]
     [ GROUP BY groupClause ] [ WINDOW windowClause ])

Specifying a single, simple reference creates a view that is a snapshot view of the source table or view. Including more than one reference specifies a join operation.

View constraints

Views have constraints of the sources to the FROM clause.

These constraints include:
• A view can be derived from a data stream table or another view.
• A view can join a data stream table or view, and one or more lookup tables.
• A view cannot join two or more data stream tables, or views based on different data stream tables. (A consolidated data stream is a special-case join of data stream tables.)
• A view cannot be derived from lookup tables only.

Join operations

The C-SQL SELECT supports several join operations.

These join operations include:
• "Cross joins"
• "Inner joins" on page 317
• "Outer joins" on page 317
• "Nested joins" on page 317

Including more than one reference causes a join operation. The resulting view is cross join unless you either use the JOIN operator to specify another type of join condition or include a WHERE clause (see “WHERE clause” on page 319) that specifies a join condition.

Cross joins

If you omit the JOIN operator, you define a view that is a cross join of the input views (also know as a Cartesian product).

The following example shows a join of the Product and Manufacturer views:
FROM Product AS P, Manufacturer AS M

Do not create a cross join unless you are sure that is what you want. A cross join creates a view whose count of rows is equal to the count of rows in the first view times the count in the second view (rows join = rows view1 * rows view2). This severely impacts the system and usually does not produce the view that you want.

Instead of creating a cross join, specify another type of join with the JOIN clause:
reference [ joinType ] JOIN reference ON searchCondition

**Inner joins**

The JOIN clause performs an inner join unless you specify a joinType variable. An inner join is one where the rows in the result table are the rows from the first table that meet the specified criteria, combined with the corresponding rows from the second table that meet the specified criteria.

FROM (Product AS P INNER JOIN Manufacturer AS M
ON P.productName = M.ProductName)

Inner joins are sometimes called equijoins.

**Outer joins**

An outer join is one where the rows in the result table are the rows that result from an inner join, plus the rows from the first table (LEFT OUTER JOIN) or the second table (RIGHT OUTER JOIN) that have no matches in the other table.

For example:
FROM (Product AS P LEFT OUTER JOIN Manufacturer AS M
ON P.productName = M.ProductName)

The first table in a LEFT OUTER JOIN and the second table in a RIGHT OUTER JOIN must be a data stream table or a view. The first table cannot be a lookup table.

Resulting columns that do not exist in both references are assigned NULL values.

**Nested joins**

Joins can be nested. There is no limit on the maximum level of nesting.

For example:
FROM (Product AS P LEFT OUTER JOIN Manufacturer AS M
  ON P.productName = M.ProductName)
  AS Temp, inventoryContext AS INVvt

**Table expressions**

A table expression, also called an in-line view, is a subquery that creates a view that can be referenced by the containing query. It is essentially a SELECT statement, bounded by parenthesis, and appearing in the FROM clause.

For example, the following code is a table expression contained in a query:
SELECT *
FROM Warehouse AS wh,
(SELECT *
    FROM WarehouseQtyChange AS wqc,
    Product AS pr
    WHERE wqc.wprod_id = pr.pprod_id
AS sv,
    WHERE sv.warehouse_id = wh.wh_region_id
)

The previous example has the same result as WHRegionView in this example:

**SummaryView:**

```sql
SELECT *
FROM WarehouseQtyChange AS wqc,
    Product AS pr
WHERE wqc.wprod_id = pr.pprod_id
```

**WHRegionView:**

```sql
SELECT *
FROM Warehouse AS wh,
SummaryView AS sv
WHERE sv.warehouse_id = wh.wh_region_id
```

### Syntax

A table expression is a limited SELECT statement, enclosed in parentheses, and with restrictions.

```sql
( SELECT selectList FROM joinClauses
    [ WHERE searchCondition ]
    [ GROUP BY groupClause ]
    [ WINDOW windowClause ] ) AS aliasName
```

### Restrictions

In-line views have the same semantic restrictions as standard views.

For example, a derived column in an in-line view cannot have the same name or alias as a column in the containing selectList list. They also have restrictions and limitations.

Restrictions and limitations include:

- Must be enclosed in parentheses
- Must have the same base data stream table as the other views and tables in the containing query
- Must be defined in the FROM clause only

### HAVING clause example

Some SQL implementations include a HAVING clause you can use to filter the aggregate results of a view. C-SQL does not include a HAVING clause. However, you can construct a HAVING clause by using a WHERE clause.

A WHERE clause (see "WHERE clause" on page 319) filters the results of an in-line view. Essentially:

```sql
SELECT * FROM (inlineView) WHERE filterCondition
```

For example, to create a view that filters the result of an aggregation, you first need to perform the aggregation in an in-line view, then filter the results with the
containing view. The following in-line view sums the total sales for each product line, and then the containing view displays in descending order only those results greater than $1,000,000.

```
SELECT FamilyTotals.family AS "Product Line",
       FamilyTotals.sales_for_family AS "Total Sales"
FROM (SELECT family,
           SUM(total_price) AS sales_for_family
               FROM OrdQtyDemand
           GROUP BY Family
         ) AS FamilyTotals
WHERE "Total Sales" > 1000000
ORDER BY "Total Sales" DESC;
```

In the previous example, as new events enter the OrdQtyDemand view, the totals are updated and the order of product families can change. New families can enter the view as their sales totals exceed a million.

**WHERE clause**

A WHERE clause examines each row in the input and accepts only those that match the specified condition.

The syntax is:

```
WHERE searchCondition
```

A searchCondition is a combination of Boolean predicates that together make a test. Only those input rows that pass the test are inserted into the new view. Rows that do not meet the condition are discarded, not tracked, and not included in the calculations of a set function, moving set function, or rank function.

All dependent views update and their functions recalculate, regardless of whether the input met the condition. See "Updating views through event propagation" on page 343 for more information.

**Predicates**

A predicate is a Boolean expression that asserts a fact about values. Each expression can be stated alone or compared to one of the Boolean test values.

For example, these expressions are equivalent:

```
WHERE (Age >= 21)
WHERE (Age >= 21) IS TRUE
```

The predicates that the WHERE clause supports are listed in Chapter 22, "Operators and constants," on page 271.

Predicates can include functions, but functions that reference columns can reference only event columns.

**Aliases**

C-SQL extends the ANSI standard to permit alias references in the WHERE clause.

For example:

```
SELECT user_age AS Age
```
FROM user_list
WHERE (Age >= 21)

GROUP BY clause

The GROUP BY clause groups the resulting virtual table as one or more columns.

The syntax is:
GROUP BY [tableName.]columnName[ , [tableName.]columnName
... ]

Where tableName is a source table or view. All columnName variables in the select list that are not referred directly by a rank function or scalar function must appear in the GROUP BY list.

Another and more powerful way to group data is with the windows PARTITION clause. See "Window partitions" on page 371 for details.

The GROUP BY option produces summary information for groups of rows whose values in the selected fields are the same. Consider this set of data:

<table>
<thead>
<tr>
<th>Name</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nano Webber</td>
<td>10</td>
</tr>
<tr>
<td>Fizzy Lifter</td>
<td>700</td>
</tr>
<tr>
<td>Nano Webber</td>
<td>50</td>
</tr>
<tr>
<td>Nano Webber</td>
<td>20</td>
</tr>
<tr>
<td>Nano Webber</td>
<td>15</td>
</tr>
<tr>
<td>Smoke Shifter</td>
<td>310</td>
</tr>
</tbody>
</table>

If you create a view that groups by name and determines the sum of the quantity for each group, it would look like this:

SELECT product.name AS Name, SUM(product.quantity) AS Qsum
FROM product
GROUP BY product.name

<table>
<thead>
<tr>
<th>Name</th>
<th>Qsum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nano Webber</td>
<td>95</td>
</tr>
<tr>
<td>Fizzy Lifter</td>
<td>700</td>
</tr>
<tr>
<td>Smoke Shifter</td>
<td>310</td>
</tr>
</tbody>
</table>

You can also group on multiple fields. For example:

SELECT product.name AS Name, product.location AS Locale,
       SUM(product.quantity) AS Qsum
FROM product
GROUP BY product.name, product.location

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Qsum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nano Webber</td>
<td>West</td>
<td>10</td>
</tr>
<tr>
<td>Fizzy Lifter</td>
<td>East</td>
<td>700</td>
</tr>
<tr>
<td>Nano Webber</td>
<td>East</td>
<td>85</td>
</tr>
</tbody>
</table>
When the select list includes a moving set function, each group contains a result for the moving set.

**Aliases**

C-SQL extends the ANSI standard to permit alias references in the GROUP BY clause.

For example:

```sql
SELECT product.name AS Name, product.location AS Locale,
       SUM(product.quantity) AS Qsum
FROM product
GROUP BY Name, Locale
```

**Derived views**

When a view is defined with a GROUP BY clause, any view derived from that view has an implicit GROUP BY clause. This is known as view merging or view expansion.

For example, in the following illustration, even though the SELECT statement for View2 does not include a GROUP BY clause, its results include the same groups as View1.

In the illustration, the tables Data_Stream and Lookup_Table are used in the following SELECT statement to create View1.

```sql
SELECT Name, Qty, Cost FROM Data_Stream, Prices GROUP BY Name WHERE Data_Stream.Name=Lookup_Table.Name
```

View2 is then created from View1 with the following statement:

```sql
SELECT Name, (Qty*Cost) AS Total FROM View1
```

**Stateful view semantics**

When a view contains a GROUP BY clause, that view is a stateful view. It maintains information from previous events, not just the most recent event.

In "Derived views," View2 is stateful even though its SELECT definition does not contain a set function or an explicit GROUP BY clause; rather, it is stateful because it is derived from a stateful view.

For more information, see "Stateless and stateful views" on page 343.
ORDER BY clause

The ORDER BY clause orders (sorts) the resulting view based on column names or on expression results.

The syntax is:
ORDER BY columnName [(ASC|DESC)] [, columnName [(ASC|DESC)]] ...

Without this clause, there is no guarantee that the same query will produce rows in the same order on subsequent queries.

Any sort key mentioned in the ORDER BY clause must refer to a column name in the select list.

By default, the view is ordered in ascending order (ASC). To order in descending order, specify the DESC option.

The following code is an example that orders the view first by supplier name in ascending order and then by price in descending order within each supplier:

```
SELECT Product.prod_id AS ProductID,  
  orderStatusDataStream.OS_PRICE AS Price,  
  Supplier.supp_name AS SupplierName  
FROM orderStatusDataStream, Product, Supplier  
WHERE orderStatusDataStream.OS_PROD_ID = Product.prod_id  
AND  
  Product.prod_supp_id = Supplier.supp_id  
ORDER BY Supplier.supp_name  
ASC, orderStatusDataStream.OS_PRICE DESC
```
Chapter 33. TIBCO Rendezvous

TIBCO Rendezvous is a messaging system for business applications.

Business applications publish messages to the stream managed by TIBCO Rendezvous transport servers. Each message has a name that identifies the subject of the message.

Other applications monitor the stream that looks for messages that, when found, are provided to other applications, such as IBM Cognos Real-time Monitoring TIBCO Rendezvous agents (see “TIBCO Rendezvous agents” on page 34).

How TIBCO Rendezvous works

IBM Cognos Real-time Monitoring data stream tables receive TIBCO Rendezvous messages as events. Each data stream table corresponds to a single message subject. The tables identify the message subjects to a listening daemon application through the agent. When the daemon locates a new message of the requested subject, it passes the message to the table through the agent. The table definition then maps the message into the table as a new event.

TIBCO Rendezvous tables

A data stream table for TIBCO Rendezvous receives messages from a business application through a TIBCO Rendezvous message stream.

Each message is identified by subject, and each new message for a subject is a new event. When the table receives a new message, it first maps the message data into the data types for the data stream table.

Limitations

All messages for an event subject must be in the same form: every message must have the same fields, though a field can be empty.

Some TIBCO Rendezvous data types are not supported and cannot be mapped into a data stream table.

For more information, see “TIBCO Rendezvous data types” on page 327.
Creating a data stream table for TIBCO Rendezvous

This section describes how to create a data stream table for TIBCO Rendezvous.

For more information, see "TIBCO column information" on page 325.

Before you begin

Before creating a TIBCO Rendezvous data stream, you need the following prerequisites:

- Permission
  Create permissions (see Chapter 23, "Permissions," on page 275) for tables (see "Granting permission to create objects" on page 278) and Read-Only access permission on the agent that provides data to the table.

- An agent
  An existing TIBCO Rendezvous agent that connects to the TIBCO Rendezvous message stream. Create an agent with the Workbench Administration tab. See "TIBCO Rendezvous agents" on page 34 for details.

- Subject name
  Each TIBCO Rendezvous message has a subject name that identifies the event source. You identify the subject name, and the agent monitors the message stream and looks for the messages. When the agent finds a message with that subject, the agent passes the message information to the data stream table. Subject names consist of one or more elements separated by dot characters (periods), such as: SUPPORT.TICKETS.

- Message format
  Each TIBCO Rendezvous message contains fields of information. This information can contain nested messages, a subject, and ‘reply to’ headers. For more information, see "TIBCO column information" on page 325.

- A sample file (Optional)
  If the message tibco_complex_string contains a complex string, it is helpful to have a sample file that contains data in the format of the actual event string. You can use this sample when you create the data stream to ensure that the fields map correctly into the data stream table by seeing how the data lines up in the columns.

For the details of the subject name and message format, consult the IT specialist who maintains your TIBCO Rendezvous system. A TIBCO Rendezvous data stream table has the attributes described in the following table:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the data stream object.</td>
</tr>
<tr>
<td></td>
<td>The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see &quot;Object namespace&quot; on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
</tbody>
</table>
Table 94. TIBCO Rendezvous data stream table attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save in</td>
<td>Specifies the folder in which the data stream object is to be stored. Use the Choose Folder button to select or create a folder. If you do not specify a folder, the default is Public Folder.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Process data in the order of arrival</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn off this attribute. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
<tr>
<td>TIBCO Rendezvous agent</td>
<td>An existing agent that connects to the TIBCO Rendezvous message stream. Create an agent with the Workbench Administration tab. See “TIBCO Rendezvous agents” on page 34 for details.</td>
</tr>
<tr>
<td>Subscription</td>
<td>Identifies the subject on which the message is being sent, and defined by the message publisher. Typically, this string looks similar to: com.cognos.obi.mytibcotopic.</td>
</tr>
<tr>
<td>Column Information</td>
<td>The Column Information fields define how to map the fields from the TIBCO Rendezvous message into columns in the data stream table. There is one column for every field in the data stream table. For more information, see “TIBCO column information.”</td>
</tr>
<tr>
<td>Clear State Interval</td>
<td>This tab contains several options for clearing persisted event data. For more information, see “Clear state interval” on page 34.</td>
</tr>
</tbody>
</table>

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > Data Stream.
4. Select TIBCO RV as the data stream type.
5. Identify the subject of the message events to collect.
6. Select an existing TIBCO Rendezvous agent.
7. Identify the fields in the message, and how they map to IBM Cognos Real-time Monitoring data types.
8. Optionally, set the clear state interval on the Clear State Interval tab.
9. Save the TIBCO Rendezvous table.

TIBCO column information

The Column Information fields define how to map the fields from the TIBCO Rendezvous message into columns in the data stream table.
There is one column for every field in the data stream table.

Each field in the message can be a simple field that maps directly into a column of a data stream table, or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table. Complex fields are treated as flat files in either delimited (CSV), fixed-width, or XML formats. For a detailed description of these file types, see Chapter 13, “Flat files,” on page 93.

Each column in the data stream table has the attributes described in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Name</td>
<td>Name of the column in the data stream table.</td>
</tr>
<tr>
<td>Message Name</td>
<td>Name of the field in the message. When mapping a message field, the name for each embedded field is N/A and uneditable.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Data type of the column. For more information, see “TIBCO Rendezvous data types” on page 327.</td>
</tr>
<tr>
<td>Format</td>
<td>(Optional) Format of the event column for VARCHAR (string) and DECIMAL values.</td>
</tr>
</tbody>
</table>

Add columns by clicking Add Field or Add Flat File Field.

You can add columns for the message subject and reply. To map the subject into a column in the data stream table, specify <tibcosubject> as the field name. For replies, specify <tibcoreply>.

In cases were a field is contained within a nested message, you can specify the field to map to the column in the data stream table with dot notation. For example, to map field d within message c that is nested within message a, which is nested in message b, you would specify the field name as <a.b.c.d>.

**Add a message field**

Each field in the message can be a simple field that maps directly into a column of a data stream table, or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table.

Complex fields are treated as flat files in either delimited (CSV), fixed-width, or XML formats. For a detailed description of these file types, see Chapter 13, “Flat files,” on page 93.

**Procedure**

1. Click Add Flat File Field.

   A flat file field creates a message field of embedded fields, each of which maps to a column in the data stream table.

2. Choose the flat file type of the message field.

   Identify a sample file to assist in mapping the columns. This file is a sample of the real data file. Data from this file appears in the next step to assist you as you map the data stream data into the table. This is optional for fixed-width and delimited files.

3. For fixed-width files, define the positions of the data columns in the Set Field Widths window.
Identify the flat-file attributes. See “Data stream tables from flat files” on page 94 for details.

Define the format-specific column information. For details about the source type, see these topics:
- “Fixed-width files” on page 100
- “Delimited files” on page 99
- “XML file support” on page 100

Click Save Data Stream to save the message field definition.

**Edit the definition of a message field**
Each field in the message can be a simple field that maps directly into a column of a data stream table, or it can be a complex field (a flat file field) that contains several fields that each map into columns in the table.

Complex fields are treated as flat files in either delimited (CSV), fixed-width, or XML formats. For a detailed description of these file types, see Chapter 13, “Flat files,” on page 93.

**Procedure**
In the Format column of the field, change the value from Flat File: file type to <Change Formatting>.
When editing a message field, the sample file option for delimited and fixed-width file types is not available.

**TIBCO Rendezvous data types**
Each message is a set of fields that each contain one data item of a specific data type. You can identify each field by a defined name and specify the associated data type of the field.

The TIBCO Rendezvous data types map to Cognos Real-time Monitoring data types (see Chapter 9, “Data types,” on page 63), as shown in the following table:

<table>
<thead>
<tr>
<th>TIBCO Rendezvous type</th>
<th>Cognos Real-time Monitoring type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Data Types</td>
<td>Not Supported</td>
</tr>
<tr>
<td>BOOL</td>
<td>Boolean</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Timestamp</td>
</tr>
<tr>
<td>F32</td>
<td>Double</td>
</tr>
<tr>
<td>F32ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>F64</td>
<td>Double</td>
</tr>
<tr>
<td>F64ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>I16</td>
<td>Integer</td>
</tr>
<tr>
<td>I16ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>I32</td>
<td>Integer</td>
</tr>
<tr>
<td>I32ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>I64</td>
<td>Long</td>
</tr>
<tr>
<td>I64ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>I8</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Table 96. TIBCO Rendezvous data types mapping (continued)

<table>
<thead>
<tr>
<th>TIBCO Rendezvous type</th>
<th>Cognos Real-time Monitoring type</th>
</tr>
</thead>
<tbody>
<tr>
<td>I8ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>IPADDR32</td>
<td>Not Supported</td>
</tr>
<tr>
<td>IPPORT16</td>
<td>Not Supported</td>
</tr>
<tr>
<td>MSG</td>
<td>Not Supported</td>
</tr>
<tr>
<td>OPAQUE</td>
<td>Not Supported</td>
</tr>
<tr>
<td>STRING</td>
<td>Varchar</td>
</tr>
<tr>
<td>U16</td>
<td>Integer</td>
</tr>
<tr>
<td>U16ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>U32</td>
<td>Long</td>
</tr>
<tr>
<td>U32ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>U64ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>U64</td>
<td>Decimal</td>
</tr>
<tr>
<td>U8ARRAY</td>
<td>Not Supported</td>
</tr>
<tr>
<td>XML</td>
<td>Varchar</td>
</tr>
</tbody>
</table>
Chapter 34. Users

Each user that interacts with IBM Cognos Real-time Monitoring is known to the system by their user account information.

When administrators create or edit user accounts, or when users view their account settings, they access specific account information from tabs.

rtadmin user

Every installation has a default user named rtadmin. This user has all permissions. The rtadmin user creates other users and assigns permissions. Among the permissions that the rtadmin user can grant is the ability to create users. Contact your administrator to learn the default password of the rtadmin user for your installation.

Edit your own account information

You can edit your own account information.

Procedure

Click Account Settings at any time.

Create, delete, or edit a user account

You can create, delete, or edit a user account.

You cannot delete the system user or yourself.

Procedure

1. Open the Administration Console.
2. Click Users to see a list of all users currently defined in the system. From this page, you can do the following actions:
   • Edit an existing user’s account information by double-clicking the name in the list.

User details tab

The User details tab lists user attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>Specifies the user login name. The name must be unique among all objects in the root folder, such as business activities, users, roles, dashboard objects, and so on.</td>
</tr>
<tr>
<td>Password</td>
<td>Specifies the user’s password. The password can contain any combination of letters, numerals, and characters.</td>
</tr>
<tr>
<td>Roles</td>
<td>Specifies the roles to which the user belongs.</td>
</tr>
</tbody>
</table>
Delivery profiles tab

Delivery profiles specify where and how to deliver alerts and data feeds to the user.

Users can have multiple profiles. In the Alert Manager of IBM Cognos Real-time Monitoring Dashboard, users can identify which profiles receive which alerts. Also, every user has at least one profile: Dashboard Profile. Dashboard Profile sends notifications to the Alert Manager in the Real-time Monitoring Dashboard.

At least one of the profiles must be designated as the one to use when subscribing to an alert. All profile flagged as Auto are automatically added to new alert subscriptions.

The profile types are the possible delivery mechanisms available in the installation, and they include:

- **Dashboard Profile**
  
  The Alert Manager in the Dashboard. You cannot delete this profile.

- **Email**
  
  An email account to receive the generated alert message.

- **Web Service**
  
  A web services method that receives the alert notification and attached reportlet as XML data. For more information, see Chapter 37, "Web services," on page 349.

**Email**

Email is an address where the user receives email messages.

It has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile Name</td>
<td>Specifies the name that identifies the profile. The name can contain letters and numerals only.</td>
</tr>
<tr>
<td>Email address</td>
<td>Specifies the user's email address.</td>
</tr>
</tbody>
</table>

**Web service**

Web service receives alert notifications and any attached reportlets as XML data.

This service publishes a Simple Object Access Protocol (SOAP) doc-style message, not a Remote Procedure Call (RPC) style message.

Web service has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web service URL</td>
<td>Specifies the HTTP location of the application that provides the DOC (SOAP) service. RPC style messages are not supported.</td>
</tr>
<tr>
<td>Method</td>
<td>Specifies the method of the web service to use.</td>
</tr>
<tr>
<td>Username</td>
<td>Specifies the user account name to use when connecting to the service. This attribute is optional.</td>
</tr>
</tbody>
</table>
Table 99. Web service attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>Specifies the password for the account. This attribute is optional.</td>
</tr>
<tr>
<td>UDF</td>
<td>Specifies the user-defined function (UDF) used to launch the web service. If a UDF is not specified, the default WSDL is used to invoke the web service. For more information, see Chapter 35, “User-defined functions,” on page 333 and the sections about working with UDFs and JAR files in the IBM Cognos Real-time Monitoring Workbench User Guide. This attribute is optional.</td>
</tr>
</tbody>
</table>

To use a web service, your administrator must reference the AlertMessage.xsd file, and the AlertService.wsdl file, which describes the subscription service, the data it provides, and how to exchange data with the service.

Your administrator can locate the file in this directory:

```
install location/realtime/sdk/api/wsalert/api/wsalert
```

For more information about the web service, see Chapter 37, “Web services,” on page 349.

---

**Access permissions tab**

Access permissions are the global permissions that a user or role may have for an object or a class of objects.

On the **Access permissions** tab, you can specify permissions for:

- Business activities
- Views, cubes, and dimensions
- Lookup tables and data streams
- Users
- Roles
- Agents
- Global system properties
- Dashboard
- Dashboard objects
- External processes

New users and roles have no access permissions for any object. As a result, the users and roles receive and view alerts and reportlets allowed only by mandatory subscriptions.

For more information about assigning permissions, see Chapter 23, “Permissions,” on page 275.
Chapter 35. User-defined functions

You can use user-defined functions (UDFs) to extend C-SQL by defining your own functions for use in queries, views, and rules.

With this feature, you can define a scalar function or set function by implementing the appropriate IBM Cognos Real-time Monitoring Java interfaces.

UDFs are Java programs that take arguments and return a value, just like the internal Cognos Real-time Monitoring functions. For example, you might have a UDF that takes a set of values and concatenates them alphabetically while ignoring NULL values. You would use that UDF in an expression like the following example:

```java
ConcatSet(Product.Name)
```

After compiling the Java program, you deploy (load) it into Cognos Real-time Monitoring, where it is then available to all users who can create or edit queries, views, and rules.

For information about creating user-defined functions, see "Creating and using a user-designed function" on page 334. For more details about the interfaces, see "com.cognos.obi.api.function" in the Javadoc documentation.

User-defined function restrictions

User-defined functions have several restrictions.

These restrictions are as follows:

- Scalar and set functions only; no rank functions.
  UDFs can define scalar functions by implementing the IUDScalarFunction interface, or set functions by implementing the IUDAggregateFunction and IUDAggregateState interfaces. See "com.cognos.obi.api.function" in the Javadoc documentation for details.

- UDFs are not exposed to IBM Cognos Real-time Monitoring Workbench.
  The formula editor does not recognize UDFs. You are responsible for tracking which UDFs are registered with the system.

- All users have access to all UDFs.
  You cannot apply access permissions to a UDF. Similarly, multiple users can define different UDFs with the same name. In that case, the system uses the first one it finds.

- Set functions must implement object serialization and maintain compatibility with earlier versions.
  Failure to implement meaningful serialization and deserialization routines can result in unpredictable behavior in many areas, including checkpoint and recovery, and parallel execution.

- UDFs are never pushed as predicates to a remote source.
  Essentially, UDFs are never sent to a DBMS for evaluation. See "Column limitations in queries for lookup tables" on page 249 for more details.
Creating and using a user-designed function

These instructions use the ConcatSet sample UDF.

The sample is at the following location: \realtime\webcontent\sdk\udf. For more information about the sample, see the README.txt file in this directory.

Procedure
1. Create your UDF by implementing the appropriate com.cognos.obi.api.function interfaces.
   See the Javadoc documentation for details about the interfaces.
2. Compile your implementation.
   When you compile your UDF implementation, include the cognosAPI.jar file. The file is in the following location: \realtime\webcontent\sdk\udf. For more information about the sample, see the README.txt file in this directory.
3. Create a manifest for the UDF JAR file.
   A manifest is an XML file that describes the UDF JAR file to Cognos Real-time Monitoring. For a description of the file and a sample listing, see "Manifest files."
4. Create a JAR file that contains your classes (such as ConcatSet.class) and the manifest (manifest.xml). The manifest must appear under com/cognos/obi/manifest in the JAR. For example:
   jar -cvf udf.jar samples/* com/cognos/obi/manifest/manifest.xml
5. Add the UDF to the list of available UDFs.
   - In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
   - Click the User Defined Functions folder and choose New User Defined Function.
   - Specify or load the JAR file that contains the function, and click Continue.
   - Select the functions to add, and optionally assign new names to them.
   - Click Finish to add the UDFs to the list.
   After the UDF is deployed, you can use it in formulas, for example:
   ConcatSet(Product.Name)

Altering an existing user-designed function

You can alter an existing UDF.

Procedure
1. Change the implementation and create an updated JAR file.
2. Upload the JAR over the existing one.
3. Readd the UDFs to the list of user-defined functions.

Manifest files

A manifest is an XML file that describes the contents of the UDF JAR to IBM Cognos Real-time Monitoring.

For every function in your JAR file, define the <UDF> and <name> elements. For every data type that the function can return, define a <UDFDescriptor> element. If subsequent calls to the same function with the same argument values can return different values, set the <isVariant> element to true. For example, the
CURRENT_TIMESTAMP internal function (see page 132) takes no arguments but returns a different result each time it is called. That function is "variant".

The following sample is a manifest for the ConcatSet() function. This listing is adapted from the file in the samples directory at /samples/udf/jar/com/cognos/obi/manifest.xml.

Sample manifest.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <jarManifestXSDVersion>1</jarManifestXSDVersion>
  <author>by Martin Handwerker &lt;martin@handwerkers.com&gt;</author>
  <UserDefinedFunctions>
    <UDF>
      <name>ConcatSet</name>
      <description>Concatenate a set of values alphabetically into a large string. Ignores nulls.</description>
      <implementor>com.udfobi.concatset.ConcatSet</implementor>
      <UDFDescriptors>
        <UDFDescriptor>
          <result>VARCHAR</result>
          <argument>
            <type>ANY</type>
          </argument>
        </UDFDescriptor>
      </UDFDescriptors>
      <isVariant>false</isVariant>
    </UDF>
  </UserDefinedFunctions>
</jarManifest>
```
Chapter 36. Views

Business views are data models that provide a real-time picture of a business activity.

Records of changes and transactions in your business enter IBM Cognos Real-time Monitoring as events. Each new event drives an immediate update of the views (the business models) derived from that event, providing a real-time picture of the business metrics. After a view is updated, the system evaluates the rules associated with the view and looks for exceptional business conditions that require attention.

A business view is a virtual table that resides in memory. The contents of the view come from sources defined by a C-SQL SELECT query statement. You define views in IBM Cognos Real-time Monitoring Workbench. That system then constructs a well-formed SELECT statement before passing it to the Cognos Real-time Monitoring servers for execution and maintenance. For more information about the SELECT statement and its syntax and usage, see Chapter 32, “SELECT statements,” on page 313.

You can limit the rows that a user sees by associating an access filter to the view, and by applying the filter to users or roles that see the view. For more information, see Chapter 2, “Access filters,” on page 3 for complete details.

The data in the views on lookup tables are static or slow-changing. Changes in the lookup tables do not cause the query engine to update the view. However, changes in the lookup table are reflected in the view when the lookup table is joined with the next row in the data stream table.

Creating views

You can create views that are updated by changes and transactions in your business.

Tip:

Try to use stateless views whenever possible. Stateless views use much less memory than stateful views, because only the last event is held in memory. Use stateless views to provide data for cubes and stateful views.

Use the minimum number of groups necessary. Grouping by the event key can create too many groups, so group by region, group by minute, and so on. Do not group by customer or timestamp, unless you lower the precision first.

Perform complex calculations only once. Instead of performing a timestamp truncation several times in one view, create a column in the view where that calculation is done once, and then use that value in the child views.

TO_DATE() and other timestamp functions are very processor-intensive. Restrict their use to a minimum.

Design your windows properly. Truncate your timestamps down to the minute or hour granularity, based on how you will be analyzing the data. Also, use the appropriate window definitions. For example, the longer form of SUM(columnName)
OVER (ORDER BY TruncatedTimeStampColumn RANGE INTERVAL '24' HOUR PRECEDING
SLIDE INTERVAL '1' HOUR REFERENCE OPERATOR) is preferable to the simplified
syntax MOV_SUM(NumericColumn).

Persist data from a stateless. Do not try to hold all rows in memory. Enable View
Persistence to store the view so you can look it up again later using the Drill Back
to Detail option.

**Before you begin**

You must have Create permission for views and Read-only access permission on
the data stream table that provides data to the new view.

For more information, see "Granting permission to create objects" on page 278.

**Procedure**

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > View.
4. In the New View window, select the data stream or derivative view on which
to base your new view. Use Browse to select existing data streams or views
from other folders.
5. Click OK.
6. In the Configure View window, specify the attributes you want.

**Copying a view**

You can copy the definition of an existing view to a new view.

**Procedure**

1. Edit the view you want to copy.
2. Change the view name and any other attributes that differ from the original
view.
3. Click Save as New View.

**View attributes**

You can configure a view to specify the attributes that you need.

**Table 100. Attributes of a view**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the view object. The name can contain letters and numerals only. This name must be unique among views, data streams, lookup tables, and consolidated data streams. For more information, see &quot;Object namespace&quot; on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the view is enabled (ready to receive data) or disabled. When a view is disabled, all objects that depend on the view are also disabled, including rules, alerts, and reportlets.</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which the data stream table is to be stored. Click Choose Folder to select or create a folder. If you do not specify a folder, the default is Public Folder.</td>
</tr>
</tbody>
</table>
### Table 100. Attributes of a view (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Workset</td>
<td>The data stream tables and views on which the view is derived.</td>
</tr>
</tbody>
</table>
| Drag field from workset          | The fields selected from the workset are the columns to include in the view, including columns whose values are derived from formulas. These columns make up the select list in the underlying SELECT statement.  
  The **Group By** option identifies columns on which to group the results. This is the GROUP BY clause in the underlying SELECT statement. |
| Maintain in view                  | Allows tracking of past data stream information for stateless views. For more information, see “Maintaining events in stateless views” on page 344. |
| From Clause                      | How to join the information from multiple tables and views in the workset. This data makes up the FROM clause in the underlying SELECT statement. |
| Where Clause                     | Identifies which source information to include in the new view. Data streams that do not meet the specification are not included in the view. This data makes up the WHERE clause in the underlying SELECT statement.  
  Even though information from a data stream might be discarded, derived views are updated, though they too do not contain the data stream information. For more information, see “WHERE clause” on page 319 and “Updating views through event propagation” on page 343. |
| Window Clause                    | Defines windows for aggregating sets of rows in the view. For more information, see Chapter 38, “Query windows,” on page 361.                  |
| Order by Clause                  | Sorts the resulting view based on column names or on expression results. This data makes up the ORDER BY clause in the underlying SELECT statement. |
| Advanced                         | Use the **Advanced** tab to specify the following options:                                                                                   |
|                                  |  • **View Persistence**                                                                                                                      |
|                                  | Saves view data to a database for later analysis. For more information, see “Persisting views to a database” on page 345.                   |
|                                  |  • **Drill Back to Detail**                                                                                                                  |
|                                  | Enables the user to see the details about the data presented by a dashboard object. For more information, see “Enabling drill back to detail” on page 346. |

---

**View constraints**

IBM Cognos Real-time Monitoring views have constraints.

These constraints are:

• A view can be derived from a data stream table or another view.
A view cannot join different data streams tables. (A consolidated data stream is a union of two identical data stream tables. For more information, see “Consolidated data streams” on page 342).

A view can join two views of the same data stream table. (A synchronized join is a view derived from multiple views based on the same data stream table. For more information, see “Synchronized joins”).

A view can join a data stream table and one or more lookup tables.

A view can join a view and one or more lookup tables.

A view cannot be derived from lookup tables only.

### Synchronized joins

A synchronized join is a view derived from two views based on the same originating data stream table.

In a synchronized join, the resulting view contains rows that are the combination of the same data stream tables in the source views.

![Synchronized join view](image)

Figure 42. Synchronized join view

Synchronized joins are always based on the internal ID of a data stream table. There is an implicit join condition on the internal data stream table column. However, you must still define a WHERE clause or join condition in the FROM clause to avoid a possible cross join result. For more information, see “Cross joins” on page 316.

A synchronized join stream always results in a stateless view.

**Restriction**

A synchronized join requires that events be processed in the order that they arrive.

The source data stream object must have the option **Process data in the order of arrival** enabled.

**Example of a synchronized join**

A synchronized join is shown that determines the ratio of total sales by region.

One view (*SalesTotal*) determines the total sales for all events, while the other view (*SalesByRegion*) determines the totals for each region. Finally, the synchronized join (*SalesRatiosByRegion*) determines the percentage of each region by joining the two total views and dividing the region totals into the grand total.
The previous synchronized join is a cross join, which in this case is acceptable. However, consider this next example that determines the ratio of each marketing representative relative to the total sales:

```sql
SELECT SUM (SalesEvents.Amount) AS TotalSales
FROM SalesEvents
```

The `SalesRatioByRegion` view determines the ratio of sales that each region represents.

```sql
SELECT SalesByRegion.Region AS Region,
       (SalesByRegion.TotalForRegion / SalesTotal.TotalSales) AS RatioOfTotal
FROM SalesByRegion, SalesTotal
```

Figure 43. Example of a synchronized join

Figure 44. Another example of a synchronized join

```sql
SELECT SalesEvents.Region AS Region,
       SalesEvents.Rep AS Rep,
       SUM (SalesEvents.Amount) AS Amount
FROM SalesEvents
GROUP BY Region, Rep
```

The `SalesRatioPerRepRegion` view determines a representative sales as a ratio of the total.

```sql
SELECT SalesTotalsByRepRegion.Rep AS Rep,
       SalesByRegion.Region AS Region,
       SalesTotalsByRepRegion.Amount / SalesByRegion.TotalForRegion AS RatioPerRep
FROM SalesTotalsByRepRegion
INNER Join SalesByRegion
ON SalesByRegion.Region = SalesTotalsByRepRegion.Region
```

```sql
SELECT SalesEvents.Region AS Region,
       SUM (SalesEvents.Amount) AS TotalForRegion
FROM SalesEvents
GROUP BY Region
```
In the previous illustration, you do not want a cross join because it creates one row for each marketing representative in every region, including the regions that the representatives do not belong to. Instead, the SalesRatiosPerRepRegion view declares an inner join to limit the results by region.

### Consolidated data streams

Consolidated data streams are views that accept events from two different event sources.

For more information, see the *IBM Cognos Real-time Monitoring Workbench User Guide*.

### Aggregate views

A key power of business views is the ability to aggregate data stream and lookup table information to extract, analyze, and combine the information into meaningful business metrics. Aggregate views have at least one field definition that includes an aggregation or GROUP BY clause.

For example, consider this simple view that tracks the total count of events that arrived in the last hour. Every time a new event arrives, the MOV_COUNT() function recalculates the count of all events in the view, providing a real-time metric about the event:

```sql
SELECT MOV_COUNT( *, HOUR, 1) AS "Data Streams in the last hour"
FROM Events
```

You can make the previous example more complex by limiting the events that the view sees. For example, to count only those events whose Status value is Open:

```sql
SELECT MOV_COUNT( *, HOUR, 1) AS "Opens in the last hour"
FROM Events
WHERE Status='Open'
```

By combining the aggregate information with other lookup tables, you can generate more meaningful metrics. For example, this view reports the count of events whose Status value is Open, and groups them by Feature:

```sql
SELECT Context.Topic AS Topic,
       MOV_COUNT( *, HOUR, 3) AS "Opens in 3 hours"
FROM Events INNER JOIN Context ON Event.Feature = Context.Feature
WHERE (Status='Open')
GROUP BY Feature
```

<table>
<thead>
<tr>
<th>Feature</th>
<th>Opens in 3 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>12</td>
</tr>
<tr>
<td>Install</td>
<td>3</td>
</tr>
<tr>
<td>Servers</td>
<td>6</td>
</tr>
</tbody>
</table>

For more information about aggregate and moving set functions, see “Function types” on page 115.
Updating views through event propagation

When a view receives a new event, it attempts to update itself with the new information. If the update occurs, the view then notifies all dependent views to update as well.

However, the following exceptions can keep the view from updating:

- If a view is empty when it receives an event, and it remains empty after processing the event, it never notifies the dependent views.
- If a stateful view becomes empty as a result of a deletion, such as when an existing event is discarded from a moving set function set, all dependent views are notified to update as well.
- An update might cause a stateless view to become empty, and any dependent views to also become empty. If a subsequent event also results in an empty view, the view appears to not update, even though it does.
- If the new event is discarded because it does not meet some criteria, the stateful view is not updated. However, it still publishes a snapshot of itself to all dependent views, which can cause dependent moving set functions to update their views.

If an error occurs when processing an event while updating a view, all rows related to the entire event are discarded, and the view remains valid and enabled.

Stateless and stateful views

All views in IBM Cognos Real-time Monitoring are either stateless or stateful.

Stateful views contain the results of aggregations derived from past events in a single row. A view is stateful if it:

- contains a set function, or moving set function in the SELECT clause, or
- contains a GROUP BY clause (in which case each group contains only one row), or
- is derived from a stateful view.

Stateless views are any views that are not stateful.

Generally, a stateless view shows the information about a single event, such as a single purchase order. A stateful view, however, shows the aggregate information about multiple events, such as the average price of multiple purchase order events.

View initialization

Each view maintains two snapshots of the data it contains.

The two snapshots are:

- Current view
  Data currently in the view. For a stateful view, the snapshot shows all rows in the view. For a stateless view, it shows all the rows corresponding to the last event, which after aggregation might be an empty set.
  - Recent view
    A snapshot of the last non-empty current view. When the view has a moving set window, the recent view contains the last non-empty rows in the window. For example, if the window is two days, the recent view contains the last 2-day set
that was not an empty event if the current view is empty. A recent view is what
appears in IBM Cognos Real-time Monitoring Workbench when editing an object
that displays view results, and what is used by derived views during view
initialization.

When you create or enable a view, it is initialized to a state based on the data in
the base view as follows:

- **Stateful view**
  The new view is initialized with the data in the current view snapshot of the
  base view. For example, consider a stateful base view which tracks sales by
  region:

  ```
  SELECT
  region, SUM(sales) AS region_sales GROUP BY region
  ```

  When you derive a new view from sales_by_region, the new view is
  immediately populated with the data in the current view of sales_by_region.

  ```
  SELECT SUM(region_sales) AS total_sales FROM sales_by_region
  ```

- **Stateless view**
  The new view is initialized with the data in the recent view snapshot of the base
  view. For example, consider this stateless view. The current snapshot of this
  view is empty when no sales are greater than 1,000,000.

  ```
  SELECT region,
  sales WHERE
  sales > 1000000
  ```

  However, a view derived from this view is initialized with the data in the recent
  view snapshot of the view.

- **Data stream table**
  The new view is empty. Data stream tables do not maintain snapshots.

---

**Maintaining events in stateless views**

By default, a stateless view contains only rows representing the last event and
which satisfied the view condition; rows from previous events are discarded.

If the event does not meet the condition, the view is empty. You can use the
option **Maintaining in view** to specify a set containing recent non-empty event
information to maintain in the view.

Use this option to include the recent events in the following situations:

- When displaying the view’s contents on the **Results** tab. This tab displays the
current rows in the view.
- When using external applications that receive the view as a real-time data feed.
The external application can perform trend or historical analysis when this
option is enabled.

**Procedure**

1. Edit the view that you want to persist.
2. In the Configure View window, check **Maintain in view**.
3. Enter either the count or time-span of events to retain.
   - An event count is the maximum number of non-empty events to maintain.
The view discards the oldest event rows that do not fit in the specified size.
   - A time interval defines a set of the most recent events. The count of events in
   the view varies depending on the number of events in the interval when the
view was updated. For example, if an event arrived that does not meet the view criteria, it is excluded from the view, but the view recalculates the interval for that time.

The set of events is determined when the last event was inserted, not at the current time. For example, an interval of one hour shows all the events that arrived in the view for the hour previous to the last update. If no events were inserted in the last day, the view might still show an hour of events from the previous day. However, as soon as the new event arrives at the view, all those events are discarded.

For more information about performing aggregations on sets of recent events, see “Moving set functions” on page 117.

### Persisting views to a database

IBM Cognos Real-time Monitoring can persist business view data to an external DBMS for future reporting by third-party tools.

The information in the table is sufficient for the reporting tools to recreate a complete snapshot of the view. When persisting, the view information is written to a table in the DBMS at a rate following a policy that you define.

Cognos Real-time Monitoring can create the table automatically, or you can predefine the table in the DBMS. For more information, see “View persistence attributes.”

#### Before you begin

An application specialist must first define a JDBC agent to the RDBMS that stores the view data. For more information, see “Java Database Connectivity agents” on page 221.

#### Procedure

1. Edit the view that you want to persist.
2. In the Configure View window, click the Advanced tab.
3. Under View Persistence, click Define.
4. Complete the fields in the Define View Persistence window. For more information, see “View persistence attributes.”

The view begins persisting data as soon as it is enabled.

#### View persistence attributes

You can specify these attributes when you persist business view data to an external DBMS.

**Table 101. View persistence attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC Agent</td>
<td>Agent to the RDBMS defined in the task.</td>
</tr>
</tbody>
</table>
Table 101. View persistence attributes (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Target table in the RDBMS to receive the persisted data. If you omit this name, the table takes the same name as the business view. The columns in the target table must have the same names as the columns in the view, appear in the same order as in the view, and must be at least the same width as the columns in the view. When a column width in the target is smaller, the RDBMS either truncates the data to fit or generates an error. Similarly, the target table must support row lengths at least as long as the rows in business view.</td>
</tr>
<tr>
<td>Create this table if it does not exist</td>
<td>Indicates that the application specialist or database administrator has not already created the named table. If this option is selected, IBM Cognos Real-time Monitoring attempts to create the target table using a CREATE TABLE command in the JDBC user's default table space.</td>
</tr>
<tr>
<td>Persistence Policy</td>
<td>Persistence depends on whether the view is stateless or stateful. Stateful views are written as snapshots that are persisted based on the specified interval, and only the snapshots are persisted. Stateless views are written as snapshots as well, but are also logged so that the condition between snapshots is captured as well.</td>
</tr>
<tr>
<td>Number of Events</td>
<td>Specifies the number of events to write to the database. For example, if the number of events is set to 10, then 10 events must occur before they are written to the database. Setting the number of events to 1 causes a write with every event. An event is a row in a database. Events are represented by data stream objects in Cognos Real-time Monitoring.</td>
</tr>
<tr>
<td>Time interval</td>
<td>Specifies the interval at which to write to the database.</td>
</tr>
<tr>
<td>Disable after this number of consecutive errors</td>
<td>Specifies how many consecutive errors to write to the error log before disabling persistence. This option prevents the server from repeating the same error.</td>
</tr>
</tbody>
</table>

View columns to persist

The database receives all of the columns and rows currently in the view.

Additionally, each row contains these additional internal columns:

Table 102. Internal columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC_EVENT_ID</td>
<td>Event identifier that identifies the event that produced the most recent row included in the view.</td>
</tr>
<tr>
<td>VC_LATEST_EVENT_ID</td>
<td>Latest event identifier that identifies the last event that caused the view to update, although data from that event might not be included in the view.</td>
</tr>
<tr>
<td>VC_TIMESTAMP</td>
<td>Event timestamp that identifies when the last event was included in the view.</td>
</tr>
</tbody>
</table>

Enabling drill back to detail

You can create views that make it possible to drill back to detail on a chart.
Users can use this option to see details about the data presented by a dashboard object. **Drill Back to Detail** has the following requirements:

- The data source of the dashboard object must be a cube.
- The view on which the cube is built must be stateless. For more information, see “Maintaining events in stateless views” on page 344.
- The dimensions of the cube must come from a single external database and the database must be the same as the measures for the cube.
- If there is more than one table in a view, you can use a persisted table for the drill back to detail table. For more information, see “Persisting views to a database” on page 345.

**Drill Back to Detail** has the following limitations:

- You cannot have spaces or special characters in table and column names in views that are used for drill back with view persistence.
- Column names for views should be in capital letters as a best practice when using drill back. Otherwise, drill back to detail might not work.
- A drill back with a filter on a dimension column of type TIMESTAMP does not work if the dimension context is stored in an Oracle database.
- If the user who creates the cube has filtered access to the fact view, the cube is not enabled for drill back.
- Drill back to detail is only supported on cubes that are built on a stateless view.

**Before you begin**

An application specialist must first define a JDBC agent to the RDBMS that has access to the view data. For more information, see “Java Database Connectivity agents” on page 221.

**Procedure**

1. Open the view editor on a view for which you want to enable **Drill Back to Detail**.
   - On an existing view, choose **Edit This View**.
   - When creating a new view, choose **New View**.
2. Click the **Advanced** tab and under **Drill Back to Detail**, click **Define**.
3. In the **JDBC Agent** field, select the agent to the RDBMS.
4. In the **Table** field, specify the table in the RDBMS to use for drill back to detail. This can be either a fact table or the same table as the one specified for **View Persistence**.
5. Click **OK**.
6. Save the view.

---

**IBM Cognos Real-time Monitoring data in IBM Cognos Business Intelligence**

You can use an IBM Cognos Real-time Monitoring cube or view as a data source in IBM Cognos Business Intelligence.

You can use Cognos Real-time Monitoring data in Business Intelligence. For example, you can view Real-time Monitoring data in Query Studio or you can develop and publish a report package in Framework Manager, and view the reports in Cognos Connection.
Viewing IBM Cognos Real-time Monitoring data in IBM Cognos Business Intelligence

You can view IBM Cognos Real-time Monitoring data in IBM Cognos Business Intelligence.

You can view the following types of data:
- Cognos Real-time Monitoring cubes.
- Cognos Real-time Monitoring views.

Before you begin

Before performing this procedure, perform the actions:
- Verify that the Cognos Real-time Monitoring server is running.

  Tip: If you can open Cognos Real-time Monitoring in a web browser, then the Cognos Real-time Monitoring server is running.
- Verify that the Cognos Business Intelligence server is running.

  Tip: Use Cognos Configuration to verify that the Cognos Business Intelligence server is running. For more information, see the online help in Cognos Configuration.
- Configure single signon between Cognos Real-time Monitoring and Cognos Business Intelligence. For more information, see the online help in IBM Cognos Real-time Monitoring Workbench.

Procedure

1. In IBM Cognos Administration, create a data source connection to a Real-time Monitoring cube. For information about how to do this, see the online help in the IBM Cognos Administration console.
2. In Framework Manager, create a new project. For information about how to create a project in Framework Manager, refer to the online help in Framework Manager.
3. In the Metadata Wizard in Framework Manager, ensure that you select the data source that you created a connection for in step 1.
4. In the Publish Wizard in Framework Manager, create and publish the package to the Business Intelligence server.
5. You can now view the data in Business Intelligence. For example, you can view metadata in Query Studio or create reports in Framework Manager and view them in Cognos Connection.
Chapter 37. Web services

A web service is an interface to an application running on a web application server. The service can be a simple lookup script for a database or a complex product for enterprise application integration (EAI), such as products provided by Siebel or SAP.

IBM Cognos Real-time Monitoring can connect to web services to perform the following tasks:

- Receive data as XML. For more information, see "Web service data streams."
- Retrieve lookup tables. For more information, see "Lookup tables for web services" on page 353.
- Publish alert messages:
  - To the delivery profile of a subscriber. For more information, see "Delivery profiles tab" on page 330.
  - On an individual basis, as initiated by a user viewing the message in IBM Cognos Real-time Monitoring Dashboard. For more information, see "Web service external processes" on page 357.

Web service data streams

A web service publishes data as XML through HTTP directly to IBM Cognos Real-time Monitoring servers. All web service publishers use the same URL. Encoded in XML is the name of the data stream to receive the data as well as the Cognos Real-time Monitoring user name and password that has access to the data stream table. When the servers receive the data, they parse the XML, decode the data, and insert the data into the identified data stream table.

When configuring a data stream for a web service, you can choose from one of two types of data streams: web service or polling web service. For more information, see "Data stream attributes for web services" or "Data stream attributes for polling web services" on page 350. The two types of data streams differ. When you select Web Service, the data is pushed to Cognos Real-time Monitoring from the web service. When you select, Polling Web Service, the data is pulled from the specified web service at regular intervals.

Data stream attributes for web services

Each data stream table for a Web service has several attributes.

These attributes are described in the following table.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the table and is the name accessed by the business views that depend on this table. The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which to save the agent. The default is Public Folders. Click the Choose Folder button to select a folder.</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Process data in the order of arrival</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn this off. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
<tr>
<td>Disable data stream after this number of consecutive errors</td>
<td>Disables the data stream when a consecutive count of errors occur. For example, if set to 5, disables the data stream after 5 consecutive errors. However, if 4 errors occur, and then no errors occur followed by 2 errors, the data stream remains enabled. The default is off: Do not disable.</td>
</tr>
<tr>
<td>Column Information</td>
<td>The Column Information fields define the columns in the data stream table and are the same name as the fields in the XML message, as described in the WSDL for the data stream.</td>
</tr>
<tr>
<td>Clear State Interval</td>
<td>This tab contains several options for clearing persisted event data. For more information, see “Clear state interval” on page 54.</td>
</tr>
</tbody>
</table>

### Data stream attributes for polling web services

Each data stream table for a polling web service has several attributes.

These attributes are described in the following table.

**Attention:** When setting schedules, the time is based on the time setting of the server, not local time.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Identifies the table and is the name accessed by the business views that depend on this table. This name must be unique among views, data streams, lookup tables, and consolidated data streams. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Status</td>
<td>Whether the object is enabled (able to receive and pass data) or disabled (not receiving or passing data).</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description of the table.</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which to save the agent. The default is Public Folders. Click Choose Folder to select a folder.</td>
</tr>
<tr>
<td>Web Service Agent</td>
<td>The agent that this data stream uses. Click Change to select the web service agent.</td>
</tr>
<tr>
<td>Port Type</td>
<td>Optionally, specifies the port type needed to disambiguate between operators with the same name. The need to supply a port type is based on the WSDL you are using.</td>
</tr>
<tr>
<td>Method</td>
<td>Method of the web service to use. When the service provides multiple methods, you must choose which one to use.</td>
</tr>
<tr>
<td>Enable recovery</td>
<td>When on, logs data from an event that arrived after the last checkpoint started. This recovery log is used to restore the state of the system after an abnormal shutdown of the servers. For more information, see the checkpoint and recovery information in the IBM Cognos Real-time Monitoring Workbench User Guide.</td>
</tr>
<tr>
<td>Process data in order of arrival</td>
<td>Choose this option when events must be processed in the order received. Otherwise, if events can be processed out of order, turn this off. To join events in a view, the events must be processed in order. Leave this option on to join the events.</td>
</tr>
<tr>
<td>Disable data stream after this number of</td>
<td>Disables the data stream when a consecutive count of errors occurs. For example, if set to 5, disables the data stream after 5 consecutive errors. However, if 4 errors occur, and then no errors occur followed by 2 errors, the data stream remains enabled. The default is Do not disable.</td>
</tr>
<tr>
<td>consecutive errors</td>
<td></td>
</tr>
<tr>
<td>Treat all rows in result set as a single event</td>
<td>All rows returned in the result set are considered a single event. Otherwise, every row returned from the table is considered a separate event.</td>
</tr>
<tr>
<td>XPath Root</td>
<td>Identifies the XPath root of the repeating elements in the output that encapsulates one row of data.</td>
</tr>
<tr>
<td>Column Information</td>
<td>The Column Information fields define the columns in the data stream table. The names are the same as the fields in the XML message, which are described in the WSDL for the data stream.</td>
</tr>
<tr>
<td>Input Arguments</td>
<td>The Input Arguments fields are the values supplied in the input to the method if the method needs input values. If the method does not have any input arguments, you do not need to specify any input argument fields.</td>
</tr>
<tr>
<td>Clear State Interval</td>
<td>This tab contains several options for clearing persisted event data. For more information, see “Clear state interval” on page 54.</td>
</tr>
<tr>
<td>Polling</td>
<td>This table contains options for setting how often the web service is polled for data. The options are:</td>
</tr>
</tbody>
</table>
|                                              | • Poll on an interval  
  Lets you set the interval between polls. The default is 600 seconds.                                                                                                                                                                                                                                                                                               |
|                                              | • Poll on a schedule  
  Select to poll the web service on a schedule. Selecting this option activates the scheduling feature.                                                                                                                                                                                                                                          |
Table 103. Data stream attributes for polling web services (continued)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable data stream after this number of consecutive polling errors.</td>
<td>Disables the data stream when a consecutive count of polling errors occurs. For example, if set to 5, the data stream is disabled after 5 consecutive polling errors. However, if 4 errors occur, and then no errors occur followed by 2 errors, the data stream remains enabled.</td>
</tr>
</tbody>
</table>

Creating a data stream for a web service

Before creating a data stream table for a web service, you need to create permission for tables.

For more information, see Chapter 23, “Permissions,” on page 275 and “Granting permission to create objects” on page 278.

Procedure

1. Create a data stream for a web service in IBM Cognos Real-time Monitoring Workbench.
2. Optionally, create an IBM Cognos Real-time Monitoring user account for the web service to use when publishing the event. The account must have at least read/write permission on the data stream table to publish to the data stream table.
3. Retrieve the event WSDL definition using HTTP.
   
   http://<host:port>/cognos/realtime/wsdl/eventstream.wsdl

   The eventstream.wsdl file describes all defined data stream objects for web services in Cognos Real-time Monitoring. For more information, see the documentation in the WSDL for descriptions of the XML elements and attributes.
4. Create the web service client that conforms to the WSDL in step 3.
   
   This XML example carries data to the OrderWSEvent data stream and specifies the account used to access the data stream (WSInputAccount).
   
   There are four columns of data defined in the <OrderWSEventData> element.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
     <cqesi>
       <eventname>OrderWSEvent</eventname>
       <username>WSInputAccount</username>
       <password>wsPWD</password>
     </cqesi>
     <OrderWSEventData>
       <ProdName>Plywood</ProdName>
       <OrderQuantity>150</OrderQuantity>
       <OrderTotal>987.34</OrderTotal>
       <IsBackordered>true</IsBackordered>
     </OrderWSEventData>
   </soapenv:Body>
</soapenv:Envelope>
```
5. Publish data streams to the URL identified in the eventstream.wsdl file, similar to the following example:
   http://<host:port>/cognos/realtime/webservice/eventstream.wsdl

If you later change the machine that hosts Cognos Real-time Monitoring servers, be sure to requery the WSDL file to determine the correct URL.

---

**Lookup tables for web services**

Business views request rows from a lookup table that match one or more input values, such as a list of suppliers that supply an item, where the item ID is the input. That input is then passed to the web service application through the agent as XML. The application then returns one or more rows of data as XML, which are then mapped into the lookup table. The table then passes the requested data to the requesting business view.

![Diagram of lookup tables for web services](image)

*Figure 46. Lookup tables for web services*

IBM Cognos Real-time Monitoring web service agents are synchronous: they retrieve lookup table data as the result of a specific request. When requesting data, the agent uses Simple Object Access Protocol (SOAP) binding to communicate with the application through an HTTP connection. The application then returns one or more rows of data in XML, following the format of the web service definition language (WSDL), Doc-type format. WSDL RPC-style is not supported.

For more information about lookup table agents, see “Web service agents” on page 35.

Before creating a lookup table for data from a web service, you must have:

- Create permissions for tables. For more information, see Chapter 23, “Permissions,” on page 275 and “Granting permission to create objects” on page 278.
- A web service agent that provides data to the table. For more information, see “Web service agents” on page 35.
- Read-only access permission on the agent.
A lookup table for web services has the following attributes:

Table 104. Web services lookup table attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Lookup table name. The name must be unique among all objects (including dashboard objects and dashboards) in the same folder. The root folder also contains global objects such as users and roles and Business Activities. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies whether the object is enabled (ready to receive data) or disabled (not ready to receive data).</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters.</td>
</tr>
<tr>
<td>Save in</td>
<td>Specifies the folder in which to save the agent. The default is Public Folders. Click the Choose Folder button to select a folder.</td>
</tr>
<tr>
<td>Web Service Agent</td>
<td>An existing web service agent that connects to a web service application. Create an agent with the Workbench Administration tab. For more information, see “Web service agents” on page 35. This value cannot be changed.</td>
</tr>
<tr>
<td>Method</td>
<td>Method of the web service to use. When the service provides multiple methods, you need to choose which one to use. This value cannot be changed in this release.</td>
</tr>
<tr>
<td>Disable lookup table after this number of consecutive errors</td>
<td>The number of consecutive errors that is received from the system before it disables the lookup table. Once disabled, a lookup table must be re-enabled manually. The default is 5.</td>
</tr>
<tr>
<td>XPath Root</td>
<td>Identifies the XPath root of the repeating elements in the output, typically /Envelope/Body. This path is prepended to all paths in the Output Field Name list.</td>
</tr>
<tr>
<td>Outputs</td>
<td>Columns that receive the information from the web service. For more information about output columns, see “Output columns” on page 355.</td>
</tr>
<tr>
<td>Inputs</td>
<td>Columns that contain the data which identify what to look up in the query. For more information about input columns, see “Input columns” on page 355.</td>
</tr>
<tr>
<td>Caching</td>
<td>For information about caching, see “Caching lookup table queries” on page 250.</td>
</tr>
</tbody>
</table>

Creating a lookup table for data from a web service

Use the following procedure to create a lookup table for data from a web service.

Procedure

1. In IBM Cognos Real-time Monitoring Workbench, click the Workbench tab.
2. Click Activities.
3. Click Create New > Lookup Table.
4. In the New Lookup Table window, select Web Service as the source type.
5. Select an existing Web Service agent.
6. Select the service method to use. Each web service provides one or more methods for accessing the data it provides according to its WSDL file definition (the URL location of which you specified when creating the web service agent). Choose the method that performs the query your lookup table needs.
7. Define columns that receive information from the web service, that is, the output columns. By default, the editor defines one column for each element returned by the method. For more information about defining these columns, see “Output columns.”

8. Define the columns that contain the data that identify what to look up in the query, that is, the input columns. By default, the editor defines one column for each element returned by the method. For more information about Input columns, see “Input columns.”

9. Specify how many results to cache, if any. For more information about caching, see “Caching lookup table queries” on page 250.

Save the web services table as enabled.

Output columns
The output columns receive the information from the web service and define the table to receive the data. The editor automatically defines one column for each element returned by the method.

Each column has the following attributes:

Table 105. Output column attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Name</td>
<td>Name of the table column that contains the result returned by the web service. By default, the name is the same as the element in the Output Field Name. You can assign any valid name.</td>
</tr>
<tr>
<td>Output Field Name</td>
<td>Identifies the element in the XML returned by the service. You can view the entire path to the element by opening the Hide/Show box. This attribute cannot be changed.</td>
</tr>
<tr>
<td>XSD Data Type</td>
<td>Identifies the data type of the element in the XML. Only the basic data types are supported: numbers, strings, dates, and boolean. Complex types like ANY and ARRAY and mime types are not supported. This attribute cannot be changed.</td>
</tr>
<tr>
<td>Data Type</td>
<td>IBM Cognos Real-time Monitoring data type of the column in the table. Choose a data type appropriate to the data returned.</td>
</tr>
<tr>
<td>Formatting</td>
<td>Formats the String, Date-Time, or DECIMAL value returned. This option is not available for other data types. For more information about data types, see Chapter 9, “Data types,” on page 63.</td>
</tr>
</tbody>
</table>

To exclude columns from the result, open the Hide/Show box and clear the fields to exclude.

Input columns
The input columns pass information to the web service to identify the information to return (the outputs). The editor automatically defines one column for each element identified by the method. Each column has the following attributes:
Table 106. Input column attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Name</td>
<td>Name of the column that contains the information passed to the web service query. For example, it might contain an ID that identifies a product to look up. This column is populated by the business view that requires the lookup table information. By default, the name is the same as the element in the Input Field Name. You can assign any valid name.</td>
</tr>
<tr>
<td>Input Field Name</td>
<td>Identifies the element in the XML passed to the service. You can view the entire path to the element by opening the Hide/Show box. This attribute cannot be changed.</td>
</tr>
<tr>
<td>String Replacement Text</td>
<td>A string to pass to the service which contains values inserted by the business view requesting the information. For more information, see “String replacement templates.”</td>
</tr>
<tr>
<td>XSD Data Type</td>
<td>Identifies the data type of the element in the XML. Only the basic data types are supported: numbers, strings, dates, and boolean. Complex types like ANY and ARRAY, and mime types are not supported. This attribute cannot be changed.</td>
</tr>
<tr>
<td>Data Type</td>
<td>IBM Cognos Real-time Monitoring data type of the Column Name attribute. Choose a data type appropriate to the data to pass.</td>
</tr>
<tr>
<td>Formatting</td>
<td>Formats the String, Date-Time, or DECIMAL value returned. This option is not available for other data types. For more information, see Chapter 9, “Data types,” on page 63.</td>
</tr>
</tbody>
</table>

To exclude unnecessary columns from the query, open the Hide/Show box and clear the fields to exclude.

**String replacement templates**

String replacement is used when a web service requires data for a parameter where the data for that parameter is provided as more than one parameter in the data stream.

The String Replacement Template allows you to join parameters from the data stream so that they can be passed to the web service as a single field.

String replacements are optional. You use them only as required by your web service.

**Procedure**

1. Click ... to open the template editor.
2. Type the template expression followed by an equal sign (=) and a question mark (?) for the column name you want to replace. To combine column names, enter an ampersand (&) between each expression.
3. Under Replacement fields, enter the name you want to use for each replacement field.
4. Save the template to update the Column Name fields under Inputs. The names you specified for the replacement fields replace the column names.
Web service external processes

External web service processes are methods that receive XML documents that describe the alert message or dashboard object that an IBM Cognos Real-time Monitoring Dashboard user is viewing, and which was sent to the service by the user. The XML document describes all of the data in the item that the user was viewing.

Publish an item to an external web service

You can publish an item to an external web service

Procedure

1. Define the external process to receive the message in IBM Cognos Real-time Monitoring Workbench. For more information, see "Creating an external process."
2. Send the item from Cognos Real-time Monitoring Dashboard. When viewing an alert message, task, or dashboard object, select Take Action > Initiate Process and select the process. The external service receives the item as an XML document and processes it. For more information, see “Implementing the external service” on page 358.

External process attributes

Each external process has several attributes.

These attributes are described in the following table.

Table 107. External process attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>External process name to appear in the Initiate® Process box in IBM Cognos Real-time Monitoring Dashboard. This name must be unique among external processes. For more information, see “Object namespace” on page 261.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description that can contain any text characters. This description appears in the Initiate Process box in Real-time Monitoring Dashboard.</td>
</tr>
<tr>
<td>Status</td>
<td>Whether the process is enabled (sending XML documents) or disabled (not sending documents).</td>
</tr>
<tr>
<td>Web Service URL</td>
<td>HTTP location of the application providing the RPC (SOAP binding) service. DOC style messages are not supported.</td>
</tr>
<tr>
<td>Method</td>
<td>Method of the web service to use.</td>
</tr>
<tr>
<td>Username</td>
<td>Account to use when connecting to the service. This attribute is optional.</td>
</tr>
<tr>
<td>Password</td>
<td>Password for the account. This attribute is optional.</td>
</tr>
</tbody>
</table>

Creating an external process

You can define an external process from the Administration Console tab of IBM Cognos Real-time Monitoring Workbench.

Before you begin

Before creating a web services external process, you must have:
A defined external web service method to receive the published method. You must know the URL for connecting to the service, the name of the method that receives the message, and any user or account name and password required by the service. For more information, see “Implementing the external service.”

Create permission for external processes. For more information, see “Granting permission to create objects” on page 278.

Procedure
1. In IBM Cognos Real-time Monitoring Workbench, click Administration Console.
2. Click External Processes.
4. Define the attributes for the process.
5. Save the process.
   To use the process, in IBM Cognos Real-time Monitoring Dashboard choose Take Action > Initiate Process and select the process.
   To see and use a process from Cognos Real-time Monitoring Dashboard, users must have Read permission for that process. You can assign permissions for the new object by clicking Permissions in the External Processes list, or an administrator can grant Read access to the class of external processes for the users. For more information, see “Access permissions” on page 275.

Implementing the external service
You can implement the external web service.

To implement the external web service, perform the following steps:

Procedure
1. Define the web service to receive a SOAP binding message with the fields in the external action XSD.
2. Create a WSDL (definition file) that follows IBM Cognos Real-time Monitoring target and import requirements.

Message fields
The XML message fields are defined in the invokeExternalAction.xsd definition file.

You can see this file (and all XSD files) in the following location: realtime\sdk\api\metadata.

For more information about XML and XSD files in Cognos Real-time Monitoring, see Chapter 39, “XML and XSD files used by IBM Cognos Real-time Monitoring,” on page 381.

You must have the file common.xsd located in the same directory.

Every message contains the following fields:
• description
  Description of the external process defined in IBM Cognos Real-time Monitoring Workbench
• actionName
Name of the external process defined in the Workbench.

- severity
  Severity of either the original alert message or as chosen by the user that initiates the message.

Other fields are included as necessary, based on the object that the user is viewing in Cognos Real-time Monitoring Dashboard when they initiate the action, such as the subject of an alert message, or the row set of the data in the view on which a chart is presenting. See the XSD file for details.

**Web service WSDL**

When implementing the web service, define the web service to receive a SOAP message and set its attributes.

The web service can have the attributes described in the following table:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>rpc</td>
<td>Do not use &quot;document&quot;.</td>
</tr>
<tr>
<td>target namespace (tns)</td>
<td><a href="http://cognos.obi.com">http://cognos.obi.com</a></td>
<td>—</td>
</tr>
<tr>
<td>encoding (soap:body)</td>
<td>encoded</td>
<td>Do not use &quot;literal&quot;.</td>
</tr>
<tr>
<td>import namespace</td>
<td><a href="http://cognos.obi.com/2">http://cognos.obi.com/2</a></td>
<td>Defines IBM Cognos Real-time Monitoring data types. Alternatively, you can define the types in the WSDL. Type definition is beyond the scope of this document.</td>
</tr>
<tr>
<td>import location</td>
<td>Installation-specific</td>
<td>Location of invokeExternalAction.xsd in your installation.</td>
</tr>
</tbody>
</table>

The following is an example WSDL that handles the invokeExternalAction message on a machine and port named host:80. It imports the invokeExternalAction.xsd definition file.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<definitions
 xmlns:tns="http://cognos.obi.com"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
  xmlns:soap="http://schemas.xmlsoap.org/soap/
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soaphttpenvelope="http://schemas.xmlsoap.org/soap/http"
  xmlns:soaphttp="http://schemas.xmlsoap.org/soap/http"
  xmlns:tns="http://cognos.obi.com"
  targetNamespace="http://cognos.obi.com"
  xmlns="http://schemas.xmlsoap.org/soap/">
  <import namespace="http://cognos.obi.com/5"
    location="/cognos/obi/api/metadata/invokeExternalAction.xsd"/>
  <message name="invokeExternalActionRequest">
    <part name="request" element="impl:invokeExternalAction"/>
  </message>
</definitions>
```
<portType name="invokeExternalActionPortType">
    <operation name="invokeExternalActionOperation">
        <documentation>Receives a IBM Cognos Real-time Monitoring external action.</documentation>
        <input message="tns:invokeExternalActionRequest"/>
    </operation>
</portType>

<binding name="invokeExternalActionBinding" type="tns:invokeExternalActionPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="invokeExternalActionOperation">
        <soap:operation soapAction=""/>
        <input>
            <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://cognos.obi.com" use="encoded"/>
        </input>
    </operation>
</binding>

<service name="invokeExternalActionService">
    <port name="invokeExternalActionService" binding="tns:invokeExternalActionBinding">
        <soap:address location="http://host:80/axis/services/invokeExternalActionService"/>
    </port>
</service>
</definitions>
Chapter 38. Query windows

A query window specifies a set of rows that are used in calculations with respect to the current row under examination in a C-SQL SELECT statement.

For more information about C-SQL SELECT statements, see Chapter 32, “SELECT statements,” on page 313.

Business views and cubes created in IBM Cognos Real-time Monitoring Workbench create C-SQL statements internally that are passed to the servers. The term *event* specifies a row or set of rows arriving into the system at one time. The source of these row or rows is a data stream.

Query windows overview

The calculation using a window can be for computing a moving set function, a join, or expiring rows from a view.

All such calculations use a window. In the default cases for these operations, you do not need to define the window semantics. For example, in C-SQL, set functions perform calculations on sets of rows in a view. The default set of rows for each function is all events because the view is initiated (an “unbounded” set). Consider the following view that calculates the total value of a column named Qty for all events that were ever included in the view:

```sql
SELECT SUM(Qty) AS Total_Qty
FROM Orders
```

Another way to express the SUM() in order to get the same result is the following, which says to sum over the set of all previous events:

```sql
SELECT SUM(Qty) OVER (EVENTS UNBOUNDED PRECEDING) AS Total_Qty
FROM Orders
```

The OVER clause defines a window that identifies the set of rows to include. With a window, you can limit the set to a specific count of event rows or to those events that occurred within a specific time-span. To total the current and last five events, define a window as in the following example:

```sql
SUM(Qty) OVER (EVENTS 5 PRECEDING) AS Total_Qty
```

And, to total the events of the current month, use a time-series window, as in the following example:

```sql
SUM(Qty) OVER (RANGE INTERVAL '1' MONTH PRECEDING) AS Total_Qty
```

Window types

All windows are identified by either the EVENTS or RANGE clause and include an extent definition that defines the size of the window.

```sql
(RANGE | EVENTS )
<window frame extent >
```
The extent syntax is unique to the window type and is described in detail in "Event-series windows" on page 364 and "Time-series windows" on page 365. Other clauses (not shown) control how the window behaves as new events enter the window, which items to include, and how and when the window updates to include new events and discard old ones.

Window declarations and references

There are two ways to define windows and associate them with functions.

In-line

In-line defines the window parameters immediately following the function reference.

For example:
```
SELECT PartName, SUM(Qty) OVER (EVENTS 4 PRECEDING) AS Total_Qty,
FROM Orders
GROUP BY PartName
```

This format is useful when you have only one window per query, though you can also use it with multiple windows. You cannot share these window definitions among functions in the same query.

Reference by name

Reference by name to use a window defined with the WINDOW clause.

For example:
```
SELECT PartName,
    SUM(Qty) OVER Previous4 AS Total_Qty,
    AVG(Qty) OVER Previous4 AS Average_Qty
FROM Orders
GROUP BY PartName
WINDOW Previous4 AS (EVENTS 4 PRECEDING)
```

This format is useful when you have multiple simple window definitions because you can define them all in one place: in the same WINDOW clause definition. This form also allows you to share the definition over multiple functions in the same query (as previously shown) and allows you to use windows that extend the definition of another window. For more information, see "Extending one window definition with another" on page 363.

Functions in the select list associated with a window must have alias names defined with the AS operator, such as AS Total_Qty in the previous examples.

Multiple windows per query

The WINDOW clause defines windows that can be shared throughout the query, and which can be extended by other windows.

The following example defines two windows, each used by a different function in the query:
SELECT PartName, 
  SUM(Qty) OVER Previous4 AS Total_Qty_4, 
  AVG(Qty) OVER Previous10 AS Average_Qty_10 
FROM Orders 
GROUP BY PartName 
WINDOW Previous4 AS (EVENTS 4 PRECEDING), 
Previous10 AS (EVENTS 10 PRECEDING)

Extending one window definition with another

When windows share common traits, you can define those traits in one window definition, then extend (inherit) that definition with other unique aspects in different windows.

For example, the following definition defines one window named Common with the PARTITION BY clause, then defines additional windows that extend the common traits with the range required for the unique windows:

SELECT PartName, 
  SUM(Qty) OVER Events4 AS Total_Of_Qty_4, 
  AVG(Qty) OVER Events10 AS Average_Of_Qty_10 
FROM Orders 
WINDOW Common AS (PARTITION BY PartName), 
  Events4 AS (Common EVENTS 4 PRECEDING), 
  Events10 AS (Common EVENTS 10 PRECEDING)

The previous WINDOW definition is the same as:

Events4 AS (PARTITION BY PartName EVENTS 4 PRECEDING), 
Events10 AS (PARTITION BY PartName EVENTS 10 PRECEDING)

Restrictions

Window extension definitions cannot include properties defined in the base window.

For example, you cannot define an ORDER BY in both the base and extension windows. Further:

- A PARTITION BY clause can appear in the base window definition only; it cannot appear in extensions.
- These clauses can appear in extension definitions only; they cannot appear in the base window:
  BETWEEN/AND 
  CURRENT EVENT 
  EVENTS 
  INITIALIZE 
  RANGE 
  REFERENCE 
  SLIDE
Event-series windows

Event-series windows contain a maximum fixed-set of events.

Initially the window is empty, but fills with new events until it reaches its defined capacity. After that, the oldest events are discarded as the newest events are included.

EVENTS clause

Event-series windows are identified by the EVENTS clause.

There are two ways to express spans of events. The first method is to use the BETWEEN and AND clauses, which specify both the upper and lower boundary of the window. The second method is to use the PRECEDING expression, which specifies only the lower boundary of the window. The upper boundary is the current event in the second case. The PRECEDING expression clarifies that the event rows precede the current one.

```sql
([[PARTITION BY < column >]

EVENTS { BETWEEN {< oldestEvent> | UNBOUNDED} PRECEDING
    AND { < newestEvent> PRECEDING | CURRENT
EVENT }
    |{< oldestEvent > | UNBOUNDED} PRECEDING
}
[SLIDE < distance >]
[REFERENCE {FRAME | OPERATOR} ]
}

PARTITION BY creates one window frame for each <column> of events, similar to a GROUP BY window. See “Window partitions” on page 371 for details.

SLIDE identifies how to advance the window when new events arrive in the view. See “Window advancement” on page 374 for a detailed description of this option.
REFERENCE tells the window when to determine if rows have expired from the window set. For more information about this option, see “Window update reference” on page 377.

Examples of the Events Clause
This window contains rows limited by a count of consecutive events in the view, such as the last five events:

\[ \text{SUM(Qty) OVER (EVENTS 4 PRECEDING) AS Total_Qty} \]

Or the 10 events starting 12 events ago:

\[ \text{SUM(Qty) OVER (EVENTS BETWEEN 11 PREceding AND 2 PREceding) AS Total_Qty} \]

Notice that the size of the window frame is \(<\text{oldestEvent}>-<\text{newestEvent}>+1\), where \(<\text{oldestEvent}>\) is the oldest event and \(<\text{newestEvent}>\) is the newest event. For example, the previous frame contains 10 events (11-2+1).

To include all of the previous events, including the current one, with the UNBOUNDED option, as in the following example:

\[ \text{SUM(Qty) OVER (EVENTS UNBOUNDED PRECEDING) AS Total_Qty} \]

Which is the same behavior as if no window was defined:

\[ \text{SUM(Qty) AS Total_Qty} \]

Current event
The counting is zero-based. Event zero (0) is the current event.

In the examples in “Examples of the Events Clause,” notice that the starting event is numbered 1 less than the starting event that you want.

Another way to express the last five events is:

\[ \text{OVER (EVENTS BETWEEN 4 PRECEDING AND 0 PRECEDING)} \]

Yet another way to express the range of events is to use the CURRENT EVENT literal as follows:

\[ \text{OVER (EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT)} \]

Time-series windows
Time-series windows grow to include all of the events that occur within an interval of time.
For example, the 1-day window, shown in the following illustration, grows as new events arrive during the day.

Similarly, a 3-day window includes all of the events within the current 3-day window.

**RANGE clause**

Time-series windows are identified by the RANGE clause.

These windows contain rows that are limited to a time range in combinations of years, months, days, hours, minutes, or seconds. There are two ways to express spans of events. The first method is to use the BETWEEN and AND clauses, which specify both the upper and lower boundary of the window. (The upper boundary is inclusive, and the lower boundary is exclusive.) The second method is to use the PRECEDING expression, which specifies only the lower boundary of the window. The upper boundary is the latest timestamp in the second case.

```sql
[[PARTITION BY < column >]
 [[ORDER BY {< date-time column > | < integer column}>]
 [ ASC | DESC ] ]
 RANGE (BETWEEN
 | INTERVAL < oldestTime > | < oldestInt>
 | UNBOUNDED) PRECEDING
 AND {INTERVAL < newestTime > | < oldestInt>}
 PRECEDING
 |{INTERVAL < oldestTime > | < oldestInt
 > | UNBOUNDED} PRECEDING
 )
 [SLIDE [INTERVAL < distance >]]
 [REFERENCE (FRAME | OPERATOR) ]
 [INITIALIZE < initTimestamp >]
)
```

When using a RANGE clause, be aware of the following:
• The order-by, "oldest", and "newest" columns are usually date-time data types. However, you can also use an integer that represents a time-series. For more information, see "Integer time-series" on page 370.

• PARTITION creates one window for each < column > of events, similar to a GROUP BY window. For more information, see "Window partitions" on page 371.

• ORDER BY identifies the column used to calculate the time of the event. For more information, see "ORDER BY clause" on page 368.

• SLIDE identifies how to advance the window when new events arrive in the view. For more information, see "Window advancement" on page 374.

• INITIALIZE specifies a common date-time to which to initialize all associated windows in a view. When you use a time-series window it is best to initialize the start time to be midnight for day, month, and year windows; to the first day of the month for month and year windows; and to the first day of the desired range for year windows. For more information, see "Window initialization" on page 378.

• REFERENCE tells the window when to determine if rows have expired from the window set. For more information, see "Window update reference" on page 377.

• ASC specifies that the rows are ordered according to their timestamps from oldest to newest. For more information about ordering, see "Descending" on page 369.

• DESC specifies that rows are ordered according to their timestamps from newest to oldest. For more information about ordering, see "Descending" on page 369.

Examples of the RANGE clause

This example totals the Qty column for the events of the current month.

\[ \text{SUM(Qty) OVER (RANGE INTERVAL '1' MONTH PRECEDING)AS Total_Qty} \]

You can also identify specific ranges, such as this one which starts 18 hours and 15 minutes ago, and stops 45 seconds ago: See "Date-time" on page 69 for detail about the date-time specifications.

\[ \text{OVER (RANGE BETWEEN INTERVAL '18:15' HOUR TO MINUTE PRECEDING}
\]

\[ \text{AND INTERVAL '45' SECOND PRECEDING} \]

Which events are included?

The events to include in a time-series window are determined when a new event enters the window or view.

For more information, see "Window update reference" on page 377.

The interval is inclusive of events that are the size of the range interval from the current event. For example, consider a window with a one-day interval:

\[ \text{SUM(order_qty) OVER (RANGE INTERVAL '1' DAY PRECEDING AS TotalQty} \]

When two events have one day between them, they are both included in the window. Notice the value of TotalQty after the two events:

<table>
<thead>
<tr>
<th>order_qty</th>
<th>TotalQty</th>
<th>Event Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2003-12-01 09:00:00.0</td>
</tr>
</tbody>
</table>
When using the BETWEEN clause, <oldestTime> is excluded and <newestTime> is included. For example, use these definitions to have two windows, one for the current week and one for the week before that:

ThisWeek: RANGE INTERVAL '7' DAY PRECEDING
LastWeek: RANGE BETWEEN INTERVAL '14' DAY PRECEDING AND INTERVAL '7' DAY PRECEDING

Notice that both ranges use '7' as a bounding value. The current week includes everything from now back seven days inclusive, while the previous week includes the seven days before seven days ago. Another way to define the previous windows is to use '0' as the current time; for example:

ThisWeek: RANGE BETWEEN INTERVAL '7' DAY PRECEDING AND INTERVAL '0' DAY PRECEDING
LastWeek: RANGE BETWEEN INTERVAL '14' DAY PRECEDING AND INTERVAL '7' DAY PRECEDING

The row at precisely seven days ago would not be included in this week, but it would be included in last week.

**ORDER BY clause**

Unless defined otherwise, all times are calculated based on the internal timestamp of each event for the arrival of the event.

For example, this window contains events of the last hour in the order that they arrived in the system.

`OVER (RANGE INTERVAL '1' HOUR PRECEDING)`

You can designate any date-time column in the event as key. For example, you might want to use the time that an order was placed:

`OVER (ORDER BY order.order_timestamp RANGE '4' DAY PRECEDING)`

The ORDER BY argument is a single column name reference; you cannot use integers to represent the ordinal position of the column in the SELECT list. Further, the default sort order is ascending (ASC), though descending (DESC) is available also. See "Descending” on page 369 for details. The ORDER BY clause has the following syntax:

`ORDER BY < columnNameReference > [ ASC | DESC ]`

**Out-of-order arrival**

When events are not in the expected order, the query engine attempts to insert the out-of-order event into its correct location in the window frame, and updates all aggregations accordingly.

The query engine always uses the latest time of all the events received before the out-of-order event to determine whether it is included. If the event is not within the latest window frame, it is omitted.
For example, consider a window of 1-hour that receives the following events in the order listed. In the example, the second event is the latest, and only events received after 08:10 (09:10 minus 1 hour) are included in the window. The fourth event, therefore, is rejected because its timestamp is 08:04.

```sql
OVER (ORDER BY Time RANGE INTERVAL '1' HOUR PRECEDING)
```

<table>
<thead>
<tr>
<th>Arrival</th>
<th>Sequence</th>
<th>Time</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>08:45</td>
<td>242.69</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>09:10</td>
<td>103.76</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>08:50</td>
<td>90.20</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>08:04</td>
<td>188.88</td>
</tr>
</tbody>
</table>

After the previous four events are processed, the view that contains them looks like this example:

<table>
<thead>
<tr>
<th>Arrival</th>
<th>Sequence</th>
<th>Time</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>08:45</td>
<td>242.69</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>08:50</td>
<td>90.20</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>09:10</td>
<td>103.76</td>
</tr>
</tbody>
</table>

**Descending**

By default, order is assumed to be ascending: oldest events are processed first.

However, you can specify DESC for descending sort order. When events arrive and they are not already in descending order, they are processed in the same way as noted in ["Out-of-order arrival" on page 368](#).

```sql
OVER (ORDER BY Time DESC RANGE INTERVAL '1' HOUR PRECEDING)
```

<table>
<thead>
<tr>
<th>Arrival</th>
<th>Sequence</th>
<th>Time</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>08:45</td>
<td>242.69</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>09:10</td>
<td>103.76</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>08:50</td>
<td>90.20</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>08:04</td>
<td>188.88</td>
</tr>
</tbody>
</table>

After the previous four events are processed, the view that contains them looks like this example:

<table>
<thead>
<tr>
<th>Arrival</th>
<th>Sequence</th>
<th>Time</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>08:04</td>
<td>188.88</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>08:45</td>
<td>242.69</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>08:50</td>
<td>90.20</td>
</tr>
</tbody>
</table>
**NULL value timestamps**

When the referenced column contains a NULL value for the timestamp, the event is rejected unless the range is UNBOUNDED.

When the window frame is unbounded, all events are included, including the NULL timestamps; though the NULL values are placed last in the set, in the order they arrived. For example:

<table>
<thead>
<tr>
<th>Arrival</th>
<th>Sequence</th>
<th>Time</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>08:04</td>
<td>188.88</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>08:45</td>
<td>242.69</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>103.76</td>
<td>&gt;&gt; First NULL time</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>08:50</td>
<td>90.20</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>157.11</td>
<td>&gt;&gt; Second NULL time</td>
</tr>
</tbody>
</table>

When the previous events are processed in ascending order by Time column, the resulting view looks like the following example:

<table>
<thead>
<tr>
<th>Arrival</th>
<th>Sequence</th>
<th>Time</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>08:04</td>
<td>188.88</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>08:45</td>
<td>242.69</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>08:50</td>
<td>90.20</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>157.11</td>
<td>&gt;&gt; Second NULL time</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>103.76</td>
<td>&gt;&gt; First NULL time</td>
</tr>
</tbody>
</table>

**Integer time-series**

A time-series range is expressed as a range of date-time or date interval values.

However, you can also use an integer that represents a time-series. For example, consider this series of date-time values and matching integer values:

<table>
<thead>
<tr>
<th>Date_time_value</th>
<th>Date_time_int</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003-12-01 09:00:00</td>
<td>3795637500</td>
</tr>
<tr>
<td>2003-12-01 13:00:00</td>
<td>3795654167</td>
</tr>
<tr>
<td>2003-12-02 09:00:00</td>
<td>3795737500</td>
</tr>
<tr>
<td>2003-12-03 09:01:00</td>
<td>3795837569</td>
</tr>
<tr>
<td>2003-12-04 09:00:00</td>
<td>3795937500</td>
</tr>
</tbody>
</table>

The time that the integers represent is entirely arbitrary and not a factor of how IBM Cognos Real-time Monitoring processes the values. You must understand what the values mean. For example, in the previous series, 100,000 represents one day. As such, a window of the previous two days for this data is defined as:

```
ORDER BY Date_time_int
RANGE 200000 PRECEDING
```

Or to see only the previous day:

```
ORDER BY Date_time_int
RANGE BETWEEN 200000 PRECEDING AND 100000 PRECEDING
```
When using an integer time-series, always use the ORDER BY clause and identify the integer column as the series order.

### Window partitions

All events in a view are included in a single set (window) unless defined otherwise by the GROUP BY or the PARTITION BY clause.

These clauses sort events into windows based on a key value, such as a common name or ID. Use partitioned windows to aggregate the events specific to the window. For example, to collect the total volume for all transactions by security, the view definition might look like the following example:

```sql
SELECT Trades.symbol, SUM(Trades.volume) OVER Symbols AS Total_volume
FROM Trades
WINDOW Symbols AS ( PARTITION BY Trades.symbol )
```

This is similar to a view defined with the GROUP BY clause in the following manner:

```sql
SELECT Trades.symbol, SUM(Trades.volume) AS Total_volume
FROM Trades
GROUP BY Trades.symbol
```

Querying the two previous views produces different results. The grouped view returns one row for each group, while the partitioned view, by default, returns just one row containing the result of the last event that entered the view (though the information for each partition is maintained internally). To see more rows from a partitioned view, set the [Maintain in view setting](#) of the view to a size greater than 1. For more information, see "Historical results from partitioned views" on page 373.

### PARTITION BY clause

The PARTITION BY clause defines one or more columns that contain the values that identify a partition window.

```
PARTITION BY < column > [, < column > ... ]
```

Where `<column>` is either the name of a column in the SELECT list, or an ordinal integer that represents the position of a column listed in the SELECT list of columns (the first reference is 1). When you declare a list of columns, one partition is created for each unique value of the set. For example, this declaration creates partitions for individuals based on unique lastname+firstname combinations:

```
PARTITION BY last_name, first_name
```

Consider the illustrations in the following sections that show what happens when a new event arrives that is newer than the most recent event already in the view.

### View update for a simple GROUP BY

When an event arrives in a view with a simple GROUP BY clause, the new event is applied to the associated group.

In this case, the new event is applied to the average for all AAA events ever received.
View update for a partition with frame reference

Now consider the same event entering a view partitioned by name, and where only the partition window frame that identifies the event updates.

In this example, the AAA event is 2 hours newer than the last AAA event. All previous values for the partition expire and are discarded, and only the new event is used. Notice that the other partition is not affected.

View update for a partition with operator reference

Finally, consider the same event entering a partitioned view that updates based on REFERENCE OPERATOR.

The reference tells all partitions to update when a new event enters the window. In this illustration, all existing partitions expire and a new one is created for the new event because none of the events tracked by the existing partitions are within the range of the last hour.
The following query is similar to the one shown in the previous illustration, except that in the previous illustration, only the row corresponding to the last event entered the view is shown in the result:

```
SELECT Name, MOV_AVG(Value,HOUR,1)GROUP BY Name
```

Advantage of partitions over groups

The main advantage of partitions is that you can have multiple partitions based on different columns in the same view, while GROUP BY applies solely to the entire view.

Use of windows to expire GROUP BY

With GROUP BY, you can view the results of all groups in IBM Cognos Real-time Monitoring Dashboard or on the Results tab of the Workbench.

This is an advantage over partitions, which you cannot use to look at the view to see the contents of the partitions.

A GROUP BY maintains results for each group as long as there are data in the group. In the previous illustration, if you use the view constructed with the GROUP BY expression instead of the PARTITION, you are able to view the contents, and groups expire if they had no events in the last hour.

Historical results from partitioned views

When you query a partitioned view, by default, the result is a view with one row containing the result of the last event that entered the view (though the information for each partition is maintained internally).

For example, if you track the average price of securities, partitioned by symbol, querying `SELECT *` on the view would return a result similar to the following:

```
SELECT * FROM AveragePricesView
Symbol  AvgPrice  Date
-------  --------  ----------
JMH      164.35   2003-07-14
```

In the previous results, the last event that the AveragePricesView received was for the JMH symbol.
To see more rows, set the **Maintain in view** setting of the partitioned view to a size greater than 1. If you have many aggregate events, set the value to a large number, such as 500. Thus, querying the view returns up to that many rows, as shown in the following example:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>AvgPrice</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>24.35</td>
<td>2003-03-05</td>
</tr>
<tr>
<td>SRKH</td>
<td>102.07</td>
<td>2003-03-05</td>
</tr>
<tr>
<td>JMH</td>
<td>90.22</td>
<td>2003-03-05</td>
</tr>
<tr>
<td>SRKH</td>
<td>106.88</td>
<td>2003-03-06</td>
</tr>
<tr>
<td>AAA</td>
<td>25.66</td>
<td>2003-03-06</td>
</tr>
<tr>
<td>JMH</td>
<td>94.11</td>
<td>2003-03-06</td>
</tr>
</tbody>
</table>

... 

The results appear in the order that the view produced them: the order of the events that last entered each partition.

To get meaningful historical results, order the new view, such as on Symbol and Date, as shown in the following example:

```sql
SELECT * FROM AveragePricesView ORDER BY Symbol, "Date"
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>AvgPrice</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>24.35</td>
<td>2003-03-05</td>
</tr>
<tr>
<td>AAA</td>
<td>25.66</td>
<td>2003-03-06</td>
</tr>
<tr>
<td>AAA</td>
<td>25.25</td>
<td>2003-03-07</td>
</tr>
<tr>
<td>AAA</td>
<td>24.92</td>
<td>2003-03-08</td>
</tr>
</tbody>
</table>

... 

### Window advancement

When a new event enters a window, the window determines which events to keep and which to discard when the window is full.

A window frame is full if the next row causes an existing row in the frame to expire out of the window. When viewed in the context of future and past events in the information stream, the window can be seen to advance or slide along the information stream as it adds and discards events.
**SLIDE clause**

The window advance clause (SLIDE) specifies the distance to advance when the window is full.

By default, when SLIDE is omitted, event-series windows slide one event along the information stream for each new event, while time-series windows advance to include the latest event and all events within the interval defined in the RANGE clause remain. The rest of the events are discarded.

Including a SLIDE clause advances the window either the entire size of the window (when you omit the argument) or advances the event distance or time interval specified by the argument.

\[
\text{SLIDE} \ [ \ < \text{interval}> \ | \ < \text{distance}> \ ]
\]

For example, consider an event-series window whose size is three events. Declaring SLIDE with no arguments creates the same effect as declaring SLIDE 3.

![Figure 55. An event-series window whose size is three events](image)

When a window advances more than one event, it is tumbling. The TUMBLE functions are shorthand for complete window expressions that use this sliding behavior. For more information, see “Tumble functions” on page 376.

The SLIDE argument does not have to be the same as the window size, but it must be less than or equal to the window size.

**Tumbling windows**

A tumbling window empties the window contents when the window advances to include the newest event.

For example, a tumbling time-series window continues to grow until a new event enters that causes older events to be discarded. When the SLIDE interval is the same as the window size, the window dumps all existing events when a new one arrives and the window is full.
This functionality is useful for tracking a full interval’s worth of events during the interval. For example, if you start an interval on a Sunday and declare a slide interval of seven days, the window empties and advances every Sunday. Use the INITIALIZE clause to set the starting time appropriately. For more information, see “Window initialization” on page 378.

**Trailing tumbling windows**

A trailing tumbling window empties and begins refilling after it slides.

However, it is important to understand that the items that enter the view must fall within the window as identified by the last event. For example, consider this 7-day window that includes values from the previous week and empties every seven days:

```
RANGE BETWEEN INTERVAL '14' DAY PRECEDING
AND INTERVAL '7' DAY PRECEDING
SLIDE INTERVAL '7' DAY
```

This window accepts only values that are older than seven days preceding the last event. Even though there might be a full week’s worth of events, the window contains only those events that are seven days older than the last. For example, when these events are fed into the window, only the first event is included in the window because it is more than seven days older than the last event:

```
Data_Stream_Time
-------------------
2003-12-01 09:00:00 << Only event included in the window
2003-12-01 10:00:00
2003-12-02 08:45:00
2003-12-08 09:10:00 << Last event
```

Each of the first three events is included only after receiving an event after 08:45 on 12-09.

**Tumble functions**

Most of the set functions have associated tumble_ functions, which are shorthand for complete sliding window expressions.

For example, consider this tumbling SUM() expression which sums all the events that arrive within a 1 hour interval:

```
TUMBLE_SUM(price, HOUR, 1, trade_time) AS T_Sum
```

The previous function is shorthand for the following in-line window expression:
SUM(price) OVER (ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING SLIDE) AS T_Sum

Which in turn is equivalent to the following, after entering in all default values:

SUM(price) OVER ( ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING
    SLIDE INTERVAL '1' HOUR
    REFERENCE OPERATOR) AS T_Sum

The function TUMBLE_SUM(price, EVENT, 5) is the shorthand for this complete window:

SUM(price) OVER ( EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
    SLIDE 5
    REFERENCE FRAME) AS T_Sum

See the descriptions of the individual tumble functions for details about their behavior.

Window update reference

When a view receives a new event, one or more of the windows of the view can update to reflect the new information.

Depending on the reference and definition of each window, it is possible for all events in a window to expire and be removed from the window or to not be affected by the update.

Events that are filtered out before they enter the view, such as when excluded by a WHERE clause, do not affect the windows of the view and do not cause the windows to update, regardless of the reference point.

For more information about the WHERE clause, see "WHERE clause" on page 319.

REFERENCE clause

A reference determines when to evaluate the set of events included in a window.

The references are:

- OPERATOR
  The window updates whenever a new event enters the view, whether the event is included in the window or not. For example, if a view tracks securities traded in the last hour, and partitions each security into its own window, each window evaluates its set whenever a new trade enters the view. If a security is not traded in the last hour, its window becomes empty. All other windows include only those securities traded in the last hour; older trades are removed from their windows.

SELECT
Trades.symbol,
    AVG(Trades.price)
OVER (RANGE INTERVAL '1' HOUR PRECEDING REFERENCE OPERATOR)
AS av_price_last_hour
GROUP BY symbol
FROM Trades
This mode is desirable and the default when using time-series aggregations, and you want all windows to include events referenced from the same time; that is, the time of the last event that arrived in the view. This mode is also useful for views where you want event-series windows to expire and reduce memory consumption by the view.

- FRAME
The window updates only when a new event enters the window. For example, if the view tracks the last 10 trades for each security, only the window that receives the new trade updates. All other (security) windows retain their 10 event's worth of events.

```sql
SELECT Trades.symbol,
       AVG(Trades.price) OVER (EVENTS 9 PRECEDING REFERENCE FRAME)
       AS av_price_last_10_trades
GROUP BY symbol
FROM Trades
```

This mode is desirable when you want windows to retain a set of events, regardless of when they arrived, such as for event-based moving aggregates. This mode is the default for windows-based events.

The FRAME reference is also useful for reclaiming server memory.

---

### Window initialization

When using a time-series window, the beginning time for the window frame is set by the first event that arrives in the window.

When a view has several group or partition frames, each might have a different starting time. Consider these two events, which are the first to arrive in the view:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>09:00:00.875</td>
</tr>
<tr>
<td>CQST</td>
<td>09:23:02.111</td>
</tr>
</tbody>
</table>

If the view that receives these events places them in different GROUP BY frames, each view starts at the time of each event and continues to reset based on that initialization time, as shown in the following example:

```sql
SELECT Trades.Symbol, Trades.Time,
       AVG(Trades.Price) OVER An_Hour AS Avg_Price_One_Hour_Tumble
GROUP BY Trades.Symbol, Trades.Time
FROM Trades
WINDOW An_Hour AS (ORDER BY Trades.Time
                   RANGE INTERVAL '1' HOUR PRECEDING SLIDE)
```

With this view definition, an event arriving at 09:10 causes the initial IBM event to expire, but the CQST remains in its window for at least another 13:02 minutes: the time remaining since it entered the view.

---

### INITIALIZE clause

To have all windows begin at the same time, use the INITIALIZE clause.
This clause defines the initialization point for all frames based on the window
definition. For example, to have all windows begin at the same time, initialize
them to a date-time older than the first event that is likely to arrive in the view.

    WINDOW An_Hour AS (ORDER BY Trades.Time
                    RANGE INTERVAL '1' HOUR PRECEDING
                    SLIDE
                    INITIALIZE TIMESTAMP '2003-03-05 00:00:00.000')

With this definition, all windows initialize at the same time: midnight. Each frame
expires at the top of the hour (when minutes is 00:00.000). Because the window
includes the SLIDE clause, all previous trades are discarded when the frame
expires, and, only new events arriving during the current hour are accepted.

This clause acts as a filter in that it excludes all events before the initialization
time.

The initialization time is a date-time literal value. For more information, see
"TIMESTAMP literal" on page 71. Further, the initialization value is static: it cannot
change after the view is created.

**Example of the INITIALIZE clause**

This initialization definition defines the current fiscal year, which begins on 1 July
of the calendar year.

    (RANGE INTERVAL '1' YEAR PRECEDING SLIDE
     INITIALIZE TIMESTAMP '1963-07-01 00:00:00')
Chapter 39. XML and XSD files used by IBM Cognos Real-time Monitoring

All IBM Cognos Real-time Monitoring objects and many system operations can be expressed in XML format and uploaded into the system.

The XML code must be properly formatted, as defined by XML schema files (XSD). For a complete list of the objects defined and operations you can perform, see "Cognos Real-time Monitoring XSD files" on page 388.

Details about XML and XSD are beyond the scope of this documentation. For information about XML and XSD, see http://www.xml.org.

XML and XSD files in Cognos Real-time Monitoring

All XML files have schema files that define the structure of the XML code.

The XSD files and sample XML files are available here: install_location/ realtime/sdk/api/metadata/, where install_location specifies the IBM Cognos Business Intelligence installation location.

In addition to the samples, another way to see properly defined XML files is to first create objects in Cognos Real-time Monitoring Workbench or Cognos Real-time Monitoring Dashboard, then use the Administration Console to export the entire set. All exported objects are written as XML files to the export directory on the server. For more information about this procedure, see the section about importing and exporting metadata in the IBM Cognos Real-time Monitoring Workbench User Guide.

Dependencies

Most of the objects in the system depend on other objects.

When you define a new object, all of its dependences must be defined first. Do that by defining and uploading base objects in the order of dependency, or by defining them in batch as described in "Defining multiple objects with XML" on page 384.

White space

When an XML element value contains multiple, contiguous white space characters that must be retained, direct the system to keep the spaces with xml:space="preserve".

Otherwise, the XML specification says to remove extra spaces. For example, without the preservation directive, the following description would be trimmed of the trailing spaces and would have only one space between the words:

```xml
<description xml:space="preserve">A
 note  </description>
```

Escape characters

The characters "<" and "&" are not valid in XML.
Some other characters are valid but can cause confusion when looking at them. For these characters, use these XML escape entities instead.

**Table 109. XML escape entities**

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Character</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>Apostrophe</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>Quotation mark</td>
</tr>
</tbody>
</table>

For example, when expressing a query that contains a less-than symbol, use &lt instead the less-than symbol, as shown in the following example:

```xml
<query>SELECT c1, c2 FROM event1 WHERE c3&lt;=100 AND c2='CQST'&apos;</query>
```

**Character data**

Instead of using escape characters, another way to express special characters is to use a CDATA tag.

This tag tells the parser to ignore all special characters and treat them as literals.

```xml
<query>&lt;![CDATA[SELECT c1, c2 FROM event1 WHERE c3&lt;=100 AND c2='CQST']></query>
```

**Uploading XML files**

You can upload XML files into IBM Cognos Real-time Monitoring.

There are two ways:

- **Uploading XML files from a command line with the cqupload.jar utility.** For more information, see "Uploading XML files from a command line."
- **Uploading XML files from a web browser with the fileupload.jsp script.** For more information, see "Uploading XML files from a web browser" on page 383.

Both methods require that the Cognos Real-time Monitoring server is running. Also, each method uses a Cognos Real-time Monitoring user account to log in to the server and perform the action. In each case, the account must have Create rights to the class of objects to create, or administration rights to the operations to perform. For more information about user accounts, see Chapter 34, “Users,” on page 329.

**Uploading XML files from a command line**

From a command line, use the cqupload.jar utility to upload files.

You can find the utility at the following location: `realtime\sdk\java\lib`.

This utility has three options:
### Table 110: Upload utility options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-s applicationURL</td>
<td>A URL that locates Cognos Real-time Monitoring.</td>
</tr>
<tr>
<td>-u userName</td>
<td>User account to use. Omit this option to use the default system administrator account.</td>
</tr>
<tr>
<td>-p password</td>
<td>Password for the user account. Required if you include -u.</td>
</tr>
</tbody>
</table>

For example, to upload an XML file using the default system administrator account, you use the following:

```
java -jar /CognosLava/CQupload/cquload.jar
java -jar <CD-ROM>/LavaActivitySuite/CQUpload/cqueload.jar
-s http://applicationServer >/now createUserSkyler.xml
```

To include a user name or password you would use the following example:

```
java -jar cqueload.jar -u skyler -p roo -s
```

A successful operation occurs silently; however, if the operation fails, the utility returns an error message to the command window. Review the error message to identify the problem. For example, this message indicates an error in the XML:

```
Error uploading file: createUserSkyler.xml
```

Further down the message you can find the actual cause: an invalid element:

```
Caused by: javax.xml.bind.UnmarshalException: Unexpected element
{http://cognos.obi.com/5}:nome
```

---

**Uploading XML files from a web browser**

The `fileupload.jsp` script presents a form where you identify the XML file that you want to upload.

The `fileupload.jsp` includes a file picker where you to identify the XML file to load and displays a message with the results of the upload.

**Procedure**

1. Run the `fileupload.jsp` script. Use an address similar to the following URL. Use the localhost only if you are running the browser on the same host as IBM Cognos Real-time Monitoring servers. Otherwise, use the same location that you use to run Cognos Real-time Monitoring Workbench.
   
   `http://localhost/now/jsp/fileupload.jsp`

2. Identify the user name and password to use.
3. Identify the XML file to upload.
4. Choose **Upload**.

   The results page displays the name of the XML file uploaded and its result. If the Status is Failed, review the exception to see what went wrong and correct
the problem. For example, this message reveals that the operation failed because the user object Rolf did not exist.

Exception Encountered
com.cognos.obi.exception.VCException: Cannot alter the [User] named [Rolf] because that object does not exist.

Defining an object with XML

This section describes how to define and upload a single XML definition.

To upload multiple XML files, in particular objects with dependencies, follow the instructions in “Defining multiple objects with XML.”

Procedure
1. Create an XML definition. Use the associated XSD file to determine the valid elements of the XML file. For an example of a complete XML file, see “Example: create user.”
2. Ensure that Cognos Real-time Monitoring is running.
   Use either of the methods described in “Uploading XML files” on page 382.
   To upload the file from the command line with the cqupload.jar utility, use the following example:
   java -jar /CognosNow/CQUpload/cqupload.jar

Example: create user

This example shows an XML definition that creates a user.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<createUser
  xsi:schemaLocation="http://cognos.obi.com/5
                    ../../api/metadata/createUser.xsd"
  xmlns="http://cognos.obi.com/2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
  <name>Skyler</name>
  <description xml:space="preserve">A power user</description>
  <Password>roo</Password>
</createUser>
```

Defining multiple objects with XML

When defining multiple objects, especially objects with dependencies, use the batch mode XML definition object commandBatch.xsd.

When you use the batch mode, include all of the XML in a single file, then upload that file. All of the operations must be valid or none of the operations are accepted.
To define and upload a single XML object, follow the instructions in “Defining an object with XML.”
Procedure

1. Create an XML batch file. Use commandBatch.xsd as the definition. Within the file, nest each definition within a <command> element and place them all in a single <commands> element in the order that the objects must be defined.
   For more information about a listing that defines multiple, dependent objects, see "Example: batch command."

2. Ensure that Cognos Real-time Monitoring is running.

3. Upload the batch XML file.
   Use the cqupload.jar utility to upload the XML file the application server running Real-time Monitoring. For example, the following uploads the example batch file:
   
   ```
   java -jar /CognosNow/CQUpload/cqupload.jar
   -s http://<applicationServer>/now commandBatchSkyler.xml
   ```

Example: batch command

This batch command defines a user account, two delivery profiles for the user, a user preference, and assigns one permission to the user.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<commandBatch
   xsi:schemaLocation="http://cognos.obi.com/5
   .../api/metadata/commandBatch.xsd"
   xmlns="http://cognos.obi.com/5"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
  <commands>
    <command>
      <createUser>
        <name>Skyler</name>
        <description xml:space="preserve">A power user</description>
        <Password>roo</Password>
      </createUser>
    </command>
    <command>
      <createUserProfile>
        <name xml:space="preserve">Work e-mail</name>
        <UserName>Skyler</UserName>
        <isDefault>true</isDefault>
        <EmailProfile>
          <EmailAddress>skyler@email.com</EmailAddress>
        </EmailProfile>
      </createUserProfile>
    </command>
    <command>
      <createUserProfile>
        <name xml:space="preserve">Second profile</name>
        <UserName>Skyler</UserName>
        <isDefault>false</isDefault>
        <EmailProfile>
          <EmailAddress></EmailAddress>
        </EmailProfile>
      </createUserProfile>
    </command>
  </commands>
</commandBatch>
```
Altering an existing object with XML

To alter an existing object's definition, use the same XML and schema as when creating the object, but include an <alterInformation> element to identify the alter operation.

(The <alterInformation> element is defined in common.xsd.) For example, this definition renames a view from OldName to NewName; note that it uses the createView.xsd schema:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<createView xsi:schemaLocation="http://cognos.obi.com/5
/api/metadata/createView.xsd"
   xmlns="http://cognos.obi.com/5"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
   <alterInformation>
     <previousName>OldName</previousName>
   </alterInformation>
   <name>
     newName
   </name>
</createView>
```
No matter what change you are implementing, you must use `<previousName>`. If you are not changing the name of the object, use the same name for both `<previousName>` and `<name>` elements, as shown in the following example:

```xml
<alterInformation>
  <previousName>OldName</previousName>
  <name>OldName</name>
  <description>New description</description>
  <query>SELECT c1, c2 FROM event1 WHERE c3 &lt; 555</query>
</alterInformation>
```

By default, an alter operation fails if the existing object does not exist. However, you can force the object to be created regardless of the existence of the existing object by including a `<createIfNotFound>` element, as shown in the following example:

```xml
<alterInformation>
  <previousName>OldName</previousName>
  <createIfNotFound>true</createIfNotFound>
</alterInformation>
```

**Dependencies**

When you alter an existing object, all other objects that depend on the altered object are evaluated and made invalid if their definition is broken as a result of the change.

You must alter the invalid objects and correct their definitions before they can be re-enabled.

### Issuing commands with XML

Many system operations can be performed with XML commands.

The following table includes some of the common operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable an object</td>
<td>enableObject.xsd</td>
</tr>
<tr>
<td>Disable an object</td>
<td>disableObject.xsd</td>
</tr>
<tr>
<td>Drop (delete) an object</td>
<td>dropObject.xsd</td>
</tr>
<tr>
<td>Set a system property</td>
<td>setProperty.xsd</td>
</tr>
<tr>
<td>Import or export the system metadata</td>
<td>performImportExport.xsd</td>
</tr>
<tr>
<td>Perform a checkpoint</td>
<td>systemCommand.xsd</td>
</tr>
</tbody>
</table>
The following is an example operation.

**Example: enabling an object and its dependencies**

To enable an object and all of its dependencies, use the enableObject.xsd schema.

You must identify the name of the object and its object type. The valid <type> values are defined in common.xsd by the <VCEnableObjectType> element.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<enableObject xsi:schemaLocation="http://cognos.obi.com/5/api/metadata/enableObject.xsd"
xmlns="http://cognos.obi.com/5" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <name>MyView</name>
  <type>VIEW</type>
  <cascade>true</cascade>
</enableObject>
```

This example enables all dependent objects because <cascade> is set to true. Omit this element, or set it to false to enable just the named object.

---

**Cognos Real-time Monitoring XSD files**

IBM Cognos Real-time Monitoring XML schema files can be arranged by categories.

The categories are:
- Users, profiles, and roles
- Dashboard
- Rules, alerts, and reportlets
- Data streams, lookup tables, cubes, and views
- Scenarios and business activities
- Agents
- System administration
- Object management
- Miscellaneous files

The files are located in the following location: realtime\sdk\api\metadata.

**Users, profiles, and roles**

The following table includes a description of schema files for users, profiles, and roles.

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addMembersToRole.xsd</td>
<td>Adds one or more existing users to an existing role.</td>
</tr>
<tr>
<td>addMemberToRole.xsd</td>
<td>Adds an existing user to an existing role.</td>
</tr>
<tr>
<td>createRole.xsd</td>
<td>Creates a user role object.</td>
</tr>
<tr>
<td>createSecurityFilter.xsd</td>
<td>Creates an access filter.</td>
</tr>
</tbody>
</table>
### Table 112. Schema files for users, profiles, and roles (continued)

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createUser.xsd</td>
<td>Creates a user object.</td>
</tr>
<tr>
<td>createUserProfile.xsd</td>
<td>Creates a user delivery profile object.</td>
</tr>
<tr>
<td>setPrivilege.xsd</td>
<td>Sets the permission of a user or role on an object or class of objects.</td>
</tr>
<tr>
<td>setUserPreferences.xsd</td>
<td>Set a user’s preferences.</td>
</tr>
</tbody>
</table>

### Dashboard

The following table includes a description of schema files for dashboards.

### Table 113. Schema files for dashboards

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createBookmarks.xsd</td>
<td>Creates a dashboard bookmark.</td>
</tr>
<tr>
<td>createDashboard.xsd</td>
<td>Creates a dashboard.</td>
</tr>
<tr>
<td>createPlan.xsd</td>
<td>Creates dashboard references and thresholds.</td>
</tr>
<tr>
<td>createPortlet.xsd</td>
<td>Creates a dashboard object.</td>
</tr>
<tr>
<td>createTask.xsd</td>
<td>Creates a dashboard task.</td>
</tr>
<tr>
<td>createTaskMessage.xsd</td>
<td>Creates a dashboard task message.</td>
</tr>
</tbody>
</table>

### Rules, alerts, and reportlets

The following table includes a description of schema files for rules, alerts, and reportlets.

### Table 114. Schema files for rules, alerts, and reportlets

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alertCommon.xsd</td>
<td>Common alert XSD definitions.</td>
</tr>
<tr>
<td>alertMessage.xsd</td>
<td>Alert message to be delivered to a Web Service.</td>
</tr>
<tr>
<td>alterAlertState.xsd</td>
<td>Alters the state of an existing alert.</td>
</tr>
<tr>
<td>alterRule.xsd</td>
<td>Alters an existing rule object.</td>
</tr>
<tr>
<td>createAlert.xsd</td>
<td>Creates an alert object.</td>
</tr>
<tr>
<td>createReportlet.xsd</td>
<td>Creates a reportlet object.</td>
</tr>
<tr>
<td>createRule.xsd</td>
<td>Creates a rule object.</td>
</tr>
<tr>
<td>createRuleBundle.xsd</td>
<td>Specifies the values associated with parameters in a rule template, and generates the rules, alerts, and reportlets based on the template definition.</td>
</tr>
<tr>
<td>createRuleTemplate.xsd</td>
<td>Creates a template of parameterized definitions of a set of rules, the alert used by the rules, and the reportlets associated with the alert.</td>
</tr>
<tr>
<td>ruleCommon.xsd</td>
<td>Contains common rule XSD definitions.</td>
</tr>
</tbody>
</table>

### Data streams, lookup tables, cubes, and views

The following table includes a description of schema files for data streams, lookup tables, cubes, and views.
### Table 115. Schema files for data streams, lookup tables, cubes, and views

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createConsolidatedEventView.xsd</td>
<td>Creates a consolidated data stream view based on an existing data stream and one or more additional data streams or views.</td>
</tr>
<tr>
<td>createContext.xsd</td>
<td>Creates a lookup table object based on an existing source.</td>
</tr>
<tr>
<td>createCube.xsd</td>
<td>Creates a cube object.</td>
</tr>
<tr>
<td>createDimension.xsd</td>
<td>Creates a cube dimension.</td>
</tr>
<tr>
<td>createEventStream.xsd</td>
<td>Creates a data stream object based on an existing source.</td>
</tr>
<tr>
<td>createView.xsd</td>
<td>Creates a view object based on an existing source table (data stream) and other, optional (joined) tables (data stream or lookup table).</td>
</tr>
<tr>
<td>createViewPersistence.xsd</td>
<td>Creates a view persistence definition.</td>
</tr>
<tr>
<td>ffsourceType.xsd</td>
<td>A flat-file data stream object.</td>
</tr>
<tr>
<td>jdbcSource.xsd</td>
<td>A JDBC lookup table object.</td>
</tr>
<tr>
<td>messageSource.xsd</td>
<td>A message object passed from an agent to a data stream or lookup table, used by JMS, TIBCO RV, log4j, and HTTP.</td>
</tr>
<tr>
<td>queryCube.xsd</td>
<td>Issues a query on a cube against measures in dimensions.</td>
</tr>
<tr>
<td>queryInformation.xsd</td>
<td>Contains the elements of a query (SELECT statement).</td>
</tr>
<tr>
<td>sourceDefinition.xsd</td>
<td>Source types (agent types) supported by the system.</td>
</tr>
<tr>
<td>sourceDefinition.xsd</td>
<td>Contains common source definitions for data stream and lookup table object XSD definitions.</td>
</tr>
<tr>
<td>tableDefinition.xsd</td>
<td>Supports the definition of an event source.</td>
</tr>
<tr>
<td>webServiceSource.xsd</td>
<td>A Web service data stream object.</td>
</tr>
</tbody>
</table>

### Scenarios and business activities

The following table includes a description of schema files for scenarios and business activities.

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createBusinessActivity.xsd</td>
<td>Creates a business activity object.</td>
</tr>
<tr>
<td>createScenario.xsd</td>
<td>Creates a scenario object in an existing business view and linked to an existing view.</td>
</tr>
</tbody>
</table>

### Agents

The following table includes a description of schema files for agents.

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFConnection.xsd</td>
<td>A flat file agent.</td>
</tr>
<tr>
<td>jdbcConnection.xsd</td>
<td>A JDBC agent.</td>
</tr>
<tr>
<td>JMSConnection.xsd</td>
<td>A JMS agent.</td>
</tr>
<tr>
<td>log4jConnection.xsd</td>
<td>A log4j messaging agent (used internally for logging).</td>
</tr>
<tr>
<td>procSource.xsd</td>
<td>Stored procedure definitions.</td>
</tr>
</tbody>
</table>
Table 117. Schema files for agents (continued)

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVConnection.xsd</td>
<td>A TIBCO RV agent.</td>
</tr>
<tr>
<td>SOAPConnection.xsd</td>
<td>A web service connection.</td>
</tr>
<tr>
<td>xmlBuffer.xsd</td>
<td>An XML buffer that holds part of a message.</td>
</tr>
</tbody>
</table>

System administration

The following table includes a description of schema files for system administration.

Table 118. Schema files for system administration

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyTypeDefinition.xsd</td>
<td>Describes a system property.</td>
</tr>
<tr>
<td>setLogLeve1.xsd</td>
<td>Sets the logging level for a system logger (logging module).</td>
</tr>
<tr>
<td>setProperty.xsd</td>
<td>Command to set a system property.</td>
</tr>
<tr>
<td>systemCommand.xsd</td>
<td>Performs a checkpoint.</td>
</tr>
</tbody>
</table>

Object management

The following table includes a description of schema files for object management.

Table 119. Schema files for object management

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createKeyRelationship.xsd</td>
<td>Creates a relationship between two objects.</td>
</tr>
<tr>
<td>disableObject.xsd</td>
<td>Disables an object.</td>
</tr>
<tr>
<td>dropObject.xsd</td>
<td>Deletes an object.</td>
</tr>
<tr>
<td>enableObject.xsd</td>
<td>Enables an object.</td>
</tr>
<tr>
<td>setObjectRelation.xsd</td>
<td>Relates two objects.</td>
</tr>
</tbody>
</table>

Miscellaneous files

The following table includes a description of miscellaneous schema files.

Table 120. Miscellaneous schema files

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commandBatch.xsd</td>
<td>Defines multiple objects to be loaded (defined) in batch. All definitions must be valid or no objects are defined.</td>
</tr>
<tr>
<td>common.xsd</td>
<td>Contains common XSD definitions used by most XSD schemas.</td>
</tr>
<tr>
<td>createJar.xsd</td>
<td>Creates a JAR object.</td>
</tr>
<tr>
<td>createUDF.xsd</td>
<td>Creates a UDF object.</td>
</tr>
<tr>
<td>invokeExternalAction.xsd</td>
<td>Describes an external action message sent to a web service. For more information about this file, see “Web service external processes” on page 357.</td>
</tr>
<tr>
<td>jarManifest.xsd</td>
<td>Defines the manifest in a JAR.</td>
</tr>
<tr>
<td>performImportExport.xsd</td>
<td>Command to the system to perform an import or export.</td>
</tr>
<tr>
<td>Schema</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>schedule.xsd</td>
<td>Contains common schedule and schedule interval definitions used by XSD definitions.</td>
</tr>
</tbody>
</table>
Appendix A. Java Database Connectivity methods

This appendix lists the classes and methods of IBM Cognos Real-time Monitoring driver for use with JDBC. The description for each method indicates whether it complies with the standard JDBC API or returns data that is different from the JDBC 2.0 standard.

**Driver**

The JDBC SQL database driver.

The class name is `com.cognos.obi.jdbc.driver.Driver`

`boolean acceptsURL(String url)`

Returns true if the url starts with "jdbc:cognos:obi:".

`Connection connect(String url, Properties info)`

`int getMajorVersion()`

`int getMinorVersion()`

`DriverPropertyInfo[] getPropertyInfo(String url, Properties info)`

Returns null.

`boolean jdbcCompliant()`

Returns false.

**Connection**

A connection to a specific database.

`void clearWarnings()`

Does nothing.

`void close()`

Complies with the standard JDBC API.

`void commit()`

Does nothing. IBM Cognos Real-time Monitoring JDBC driver is read-only. The concept of commit does not apply.

`Statement createStatement()`

Complies with the standard JDBC API.
**Statement**

**createStatement(int resultSetType, int resultSetConcurrency)**

Complies with the standard JDBC API when resultSetType is ResultSet.TYPE_FORWARD_ONLY and resultSetConcurrency is ResultSet.CONCUR_READ_ONLY; otherwise, throws SQLException indicating the method is not implemented.

**Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)**

Not implemented. Throws SQLException.

**boolean getAutoCommit()**

Returns false. IBM Cognos Real-time Monitoring JDBC driver is read-only. The concept of commit does not apply.

**String getCatalog()**

Returns "IBM Cognos Real-time Monitoring"

**int getHoldability()**

Returns ResultSet.HOLD_CURSORS_OVER_COMMIT.

Attention: The method commit() is not operational in the Real-time Monitoring Driver. Therefore, a ResultSet is not closed when Connection.commit() is called.

**DatabaseMetaData getMetaData()**

Compliant with the standard JDBC API. The DatabaseMetaData object returned by this method is also an instance of IOlapMetadataData

**int getTransactionIsolation()**

Returns TRANSACTION_NONE.

**Map getTypeMap()**

Not implemented. Throws SQLException.

**SQLWarning getWarnings()**

Returns null.

---

**Connection**

A connection to a specific database.

**void clearWarnings()**

Does nothing.
void close()
Complies with the standard JDBC API.

void commit()
Does nothing. IBM Cognos Real-time Monitoring JDBC driver is read-only. The concept of commit does not apply.

Statement createStatement()
Complies with the standard JDBC API.

Statement createStatement(int resultSetType, int resultSetConcurrency)
Complies with the standard JDBC API when resultSetType is
ResultSet.TYPE_FORWARD_ONLY and resultSetConcurrency is
ResultSet.CONCUR_READ_ONLY, otherwise throws SQLException indicating the method is not implemented.

Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)
Not implemented. Throws SQLException.

boolean getAutoCommit()
Returns false. The IBM Cognos Real-time Monitoring JDBC driver is read-only. The concept of commit does not apply.

String getCatalog()
Returns "IBM Cognos Real-time Monitoring"

int getHoldability()
Returns ResultSet.HOLD_CURSORS_OVER_COMMIT.

Attention: The method commit() is not operational in the Cognos Real-time Monitoring Driver; therefore, a ResultSet is not closed when Connection.commit() is called.

DatabaseMetaData getMetaData()
Compliant with the standard JDBC API. The DatabaseMetadata object returned by this method is also an instance of IOlapMetadataData

int getTransactionIsolation()
Returns TRANSACTION_NONE.

Map getTypeMap()
Not implemented. Throws SQLException.
SQLWarning getWarnings()

Returns null.

boolean isClosed()

Compliant with the standard JDBC API.

boolean isReadOnly()

Returns true.

String nativeSQL(String sql)

Not implemented. Throws SQLException.

CallableStatement prepareCall(String sql)

Not implemented. Throws SQLException.

CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)

Not implemented. Throws SQLException.

PreparedStatement prepareStatement(String sql)

Complies with the standard JDBC API.

PreparedStatement prepareStatement(String sql, int[] columnIndexes)

The autoGeneratedKeys argument is ignored.

PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

The arguments resultSetType and resultSetConcurrency are ignored.

PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

The arguments resultSetType, resultSetConcurrency, and resultSetHoldability are ignored.

PreparedStatement prepareStatement(String sql, String[] columnNames)

The argument columnNames is ignored.

releaseSavepoint(Savepoint savepoint)

Not implemented. Throws SQLException.
void rollback()

Does nothing.

void rollback(Savepoint savepoint)

Not implemented. Throws SQLException.

void setAutoCommit(boolean autoCommit)

Does nothing.

setCatalog(String catalog)

Does nothing.

void setHoldability(int holdability)

Not implemented. Throws SQLException.

void setReadOnly(boolean readOnly)

Not implemented. Throws SQLException.

Savepoint setSavepoint(String name)

Not implemented. Throws SQLException.

Savepoint setSavepoint(String name)

Not implemented. Throws SQLException.

void setTransactionIsolation(int level)

Not implemented. Throws SQLException.

void setTypeMap(Map map)

Not implemented. Throws SQLException.

### Statement

An SQL statement to pass to the database.

void addBatch(String sql)

Do not use. Reserved for internal use.

void cancel()

Not implemented. Throws SQLException.

void clearBatch()

Do not use. Reserved for internal use.
void clearWarnings()

Does nothing

void close()

Complies with the standard JDBC API.

boolean execute(String sql)

Complies with the standard JDBC API.

boolean execute(String sql, int autoGeneratedKeys)

Not implemented. Throws SQLException.

boolean execute(String sql, int[] columnIndexes)

Not implemented. Throws SQLException.

boolean execute(String sql, int[] columnNames)

Not implemented. Throws SQLException.

int[] executeBatch()

Do not use. Reserved for internal use.

ResultSet executeQuery(String sql)

Complies with the standard JDBC API.

int executeUpdate(String sql)

Not implemented. Throws SQLException.

int executeUpdate(String sql, int autoGeneratedKeys)

Not implemented. Throws SQLException.

int executeUpdate(String sql, int[] columnIndexes)

Not implemented. Throws SQLException.

int executeUpdate(String sql, int[] columnNames)

Not implemented. Throws SQLException.

Connection getConnection()

Complies with the standard JDBC API.

int getFetchDirection()

Not implemented. Throws SQLException.
int getFetchSize()
Complies with the standard JDBC API.

ResultSet getGeneratedKeys()
Not implemented. Throws SQLException.

getMaxFieldSize()
Returns 0.

int getMaxRows()
Complies with the standard JDBC API.

boolean getMoreResults()
Moves to the next result of the Statement object and returns true if it is a ResultSet object.

Attention: When in use with IBM Cognos Real-time Monitoring, the JDBC API does not implicitly close any current ResultSet objects obtained with the ResultSet method.

boolean getMoreResults(int current)
Moves to the next result of the Statement object and returns true if it is a ResultSet object. The value of current is ignored.

Attention: When in use with IBM Cognos Real-time Monitoring, the JDBC API does not implicitly close any current ResultSet object or objects obtained with the ResultSet method.

int getQueryTimeout()
Not implemented. Throws SQLException.

ResultSet getResultSet()
Complies with the standard JDBC API.

int ResultSetConcurrency()
Not implemented. Throws SQLException.

int ResultSetHoldability()
Not implemented. Throws SQLException.

int ResultSetType()
Not implemented. Throws SQLException.
int getUpdateCount()

Returns -1.

SQLException getWarnings()

Returns null if the Statement is not closed.

void setCursorName(String name)

Not implemented. Throws SQLException.

void setEscapeProcessing(boolean enable)

Not implemented. Throws SQLException.

void setFetchDirection(int direction)

Not implemented. Throws SQLException.

void setFetchSize(int rows)

Complies with the standard JDBC API.

void setMaxFieldSize(int max)

Not implemented. Throws SQLException.

void setMaxRows(int max)

Complies with the standard JDBC API.

void setQueryTimeout(int seconds)

Not implemented. Throws SQLException.

DatabaseMetaData

Provides information about the view definitions defined in IBM Cognos Real-time Monitoring installation.

boolean allProceduresAreCallable()

Returns false.

boolean allTablesAreSelectable()

Returns true.

boolean dataDefinitionCausesTransactionCommit()

Returns false.

Note: Real-time Monitoring JDBC Driver does not support data definition.
boolean dataDefinitionIgnoredInTransactions()

Returns true.

Note: IBM Cognos Real-time Monitoring JDBC Driver does not support data definition.

boolean deletesAreDetected(int type)

Returns false.

boolean doesMaxRowSizeIncludeBlobs()

Returns false.

ResultSet getAttributes(String catalog,String schemaPattern, String typeNamePattern, String attributeNamePattern)

Returns an empty ResultSet object.

ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)

Returns an empty ResultSet object.

ResultSet getCatalogs()

Complies with the standard JDBC API.

String getCatalogSeparator()

Complies with the standard JDBC API.

String getCatalogTerm()

Complies with the standard JDBC API.

ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)

Returns an empty ResultSet object.

ResultSet getColumns(String catalog,String schemaPattern, String tableNamePattern, String columnNamePattern)

Retrieves a description of table columns available in the table. Real-time Monitoring JDBC API does not treat the arguments schemaPattern, tableNamePattern, and columnNamePattern as patterns. For any of these three arguments, if its value is null or "%", all names are matched. Otherwise, the argument is treated as an exact name rather than a pattern.

Connection getConnection()

Complies with the standard JDBC API.
ResultSet getCrossReference(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)

Returns an empty ResultSet object.

int getDatabaseMajorVersion()

Complies with the standard JDBC API.

int getDatabaseMajorVersion()

Complies with the standard JDBC API.

String getDatabaseMinorVersion()

Complies with the standard JDBC API.

String getDatabaseProductName()

Complies with the standard JDBC API.

String getDatabaseProductVersion()

Complies with the standard JDBC API.

int getDefaultTransactionIsolation()

Returns Connection.TRANSACTION_NONE.

int getDriverMajorVersion()

Complies with the standard JDBC API.

int getDriverMinorVersion()

Complies with the standard JDBC API.

String getDriverName()

Complies with the standard JDBC API.

String getDriverVersion()

Complies with the standard JDBC API.

ResultSet getExportedKeys(String catalog, String schema, String table)

Returns an empty ResultSet object.

String getExtraNameCharacters()

Returns an empty string.
String getIdentifierQuoteString()

Complies with the standard JDBC API.

ResultSet getImportedKeys(String catalog, String schema, String table)

Returns an empty ResultSet object.

ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)

Returns an empty ResultSet object.

int getJDBCMajorVersion()

Complies with the standard JDBC API.

int getJDBCMinorVersion()

Complies with the standard JDBC API.

int getMaxBinaryLiteralLength()

Returns 0.

int getMaxCatalogNameLength()

Returns 32.

int getMaxCharLiteralLength()

Returns 0.

int getMaxColumnNameLength()

Returns 128.

int getMaxColumnsInGroupBy()

Returns 0.

int getMaxColumnsInIndex()

Returns 0.

int getMaxColumnsInOrderBy()

Returns 0.

int getMaxColumnsInSelect()

Returns 0.
int getMaxColumnsInTable()
Returns 0.

int getMaxConnections()
Returns 0.

int getMaxCursorNameLength()
Returns 0.

int getMaxIndexLength()
Returns 0.

int getMaxProcedureNameLength()
Returns 0.

int getMaxRowSize()
Returns 0.

int getMaxSchemaNameLength()
Returns 0.

int getMaxStatementLength()
Returns 0.

int getMaxStatements()
Returns 0.

int getMaxTableNameLength()
Returns 128.

int getMaxTablesInSelect()
Returns 0.

int getMaxUserNameLength()
Returns 128.

String getNumericFunctions()
Returns "avg, sum, count, max, min".

ResultSet getPrimaryKeys(String catalog, String schema, String table)
Returns an empty ResultSet object.
ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnPattern)

Returns an empty ResultSet object.

ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)

Returns an empty ResultSet object.

String getProcedureTerm()

Returns "Procedure".

int getResultSetHoldability()

Returns ResultSet.HOLD_CURSORS_OVER_COMMIT.

ResultSet getSchemas()

Complies with the standard JDBC API.

String getSchemaTerm()

Returns "schema".

String getSearchStringEscape()

Returns null.

String getSQLKeywords()

Returns an empty String.

int getSQLStateType()

Returns 0.

String getStringFunctions()

Returns an empty String.

ResultSet getSuperTables(String catalog, String schemaPattern, String tableNamePattern)

Returns an empty ResultSet object.

ResultSet getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)

Returns an empty ResultSet object.

String getSystemFunctions()

Returns an empty String.
ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)

Returns an empty ResultSet object.

ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)

The arguments schemaPattern and tableNamePattern as patterns are not treated as patterns. For either of these two arguments, if its value is null or "%", it is matched with all names. Otherwise, the argument is treated as an exact name rather than a pattern.

ResultSet getTableTypes()

Complies with the standard JDBC API.

String getTimeDateFunctions()

Returns an empty String.

ResultSet getTypeInfo()

Returns an empty ResultSet object.

ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)

Returns an empty ResultSet object.

String getURL()

Returns null.

String getUsername()

Complies with the standard JDBC API.

ResultSet getVersionColumns(String catalog, String schema, String table)

Returns an empty ResultSet object.

boolean insertsAreDetected(int type)

Complies with the standard JDBC API.

boolean isCatalogAtStart()

Complies with the standard JDBC API.

boolean isReadOnly()

Complies with the standard JDBC API.
boolean locatorsUpdateCopy()

Returns true.

boolean nullPlusNonNullIsNull()

Complies with the standard JDBC API.

boolean nullsAreSortedAtEnd()

Complies with the standard JDBC API.

boolean nullsAreSortedAtStart()

Complies with the standard JDBC API.

boolean nullsAreSortedHigh()

Complies with the standard JDBC API.

boolean nullsAreSortedLow()

Complies with the standard JDBC API.

boolean othersDeletesAreVisible(int type)

Complies with the standard JDBC API.

boolean othersInsertsAreVisible(int type)

Complies with the standard JDBC API.

boolean othersUpdatesAreVisible(int type)

Complies with the standard JDBC API.

boolean ownDeletesAreVisible(int type)

Complies with the standard JDBC API.

boolean ownInsertsAreVisible(int type)

Complies with the standard JDBC API.

boolean ownUpdatesAreVisible(int type)

Complies with the standard JDBC API.

boolean storesLowerCaseIdentifiers()

boolean storesLowerCaseQuotedIdentifiers()

Complies with the standard JDBC API.
boolean storesMixedCaseIdentifiers()
Complies with the standard JDBC API.

boolean storesMixedCaseQuotedIdentifiers()
Complies with the standard JDBC API.

boolean storesUpperCaseIdentifiers()
Complies with the standard JDBC API.

boolean storesUpperCaseQuotedIdentifiers()
Complies with the standard JDBC API.

boolean supportsAlterTableWithAddColumn()
Returns false.

boolean supportsAlterTableWithDropColumn()
Returns false.

boolean supportsANSI92EntryLevelSQL()
Returns false.

boolean supportsANSI92FullSQL()
Returns false.

boolean supportsANSI92IntermediateSQL()
Returns false.

boolean supportsBatchUpdates()
Returns false.

boolean supportsCatalogsInDataManipulation()
Returns true.

boolean supportsCatalogsInIndexDefinitions()
Returns false.

boolean supportsCatalogsInPrivilegeDefinitions()
Returns false.

boolean supportsCatalogsInProcedureCalls()
Returns false.
boolean supportsCatalogsInTableDefinitions()
Returns true.

boolean supportsColumnAliasing()
Retrieves whether this database supports column aliasing.

boolean supportsConvert(int fromType, int toType)
Returns false.

supportsConvert(int fromType, int toType)
Returns false.

boolean supportsCoreSQLGrammar()
Returns false.

boolean supportsCorrelatedSubqueries()
Returns false.

boolean supportsDataDefinitionAndDataManipulationTransactions()
Returns false.

boolean supportsDataManipulationTransactionsOnly()
Returns false.

boolean supportsDifferentTableCorrelationNames()
Returns false.

boolean supportsExpressionsInOrderBy()
Returns false.

boolean supportsExtendedSQLGrammar()
Returns false.

boolean supportsFullOuterJoins()
Returns false.

boolean supportsGetGeneratedKeys()
Returns false.

boolean supportsGroupBy()
Returns true.
boolean supportsGroupByBeyondSelect()
Returns true.

boolean supportsGroupByUnrelated()
Returns true.

boolean supportsIntegrityEnhancementFacility()
Returns false.

boolean supportsLikeEscapeClause()
Returns true.

boolean supportsLimitedOuterJoins()
Returns true.

boolean supportsMinimumSQLGrammar()
Returns false.

boolean supportsMixedCaseIdentifiers()
Returns false.

boolean supportsMixedCaseQuotedIdentifiers()
Returns false.

boolean supportsMultipleOpenResults()
Returns false.

boolean supportsMultipleResultSets()
Returns false.

boolean supportsMultipleTransactions()
Returns false.

boolean supportsNamedParameters()
Returns false.

boolean supportsNonNullableColumns()
Returns false.

boolean supportsOpenStatementsAcrossCommit()
Returns true.
boolean supportsOpenStatementsAcrossRollback()
Returns true.

boolean supportsOpenStatementsAcrossCommit()
Returns true.

boolean supportsOpenStatementsAcrossRollback()
Returns false.

boolean supportsOrderByUnrelated()
Returns false.

boolean supportsOuterJoins()
Returns true.

boolean supportsPositionedDelete()
Returns false.

boolean supportsPositionedUpdate()
Returns false.

boolean supportsResultSetConcurrency(int type, int concurrency)
Returns false.

boolean supportsResultSetHoldability(int holdability)
Returns true if holdability is ResultSet.HOLD_CURSORS_OVER_COMMIT.

boolean supportsResultSetType(int type)
Returns true if type is ResultSet.TYPE_FORWARD_ONLY.

boolean supportsSavepoints()
Returns false.

boolean supportsSchemasInDataManipulation()
Returns false.

boolean supportsSchemasInIndexDefinitions()
Returns false.

boolean supportsSchemasInPrivilegeDefinitions()
Returns false.
boolean supportsSchemasInProcedureCalls()
Returns false.

boolean supportsSchemasInTableDefinitions()
Returns false.

boolean supportsSelectForUpdate()
Returns false.

boolean supportsStatementPooling()
Returns false.

boolean supportsStoredProcedures()
Returns false.

boolean supportsSubqueriesInComparisons()
Returns false.

boolean supportsSubqueriesInExists()
Returns false.

boolean supportsSubqueriesInIns()
Returns false.

boolean supportsSubqueriesInQuantifieds()
Returns false.

boolean supportsTableCorrelationNames()
Returns true.

boolean supportsTransactionIsolationLevel(int level)
Returns true if level is Connection.TRANSACTION_NONE.

boolean supportsTransactions()
Returns false.

boolean supportsUnion()
Returns false.

boolean supportsUnionAll()
Returns false.
boolean updatesAreDetected(int type)
Returns false.

boolean usesLocalFilePerTable()
Returns false.

boolean usesLocalFiles()
Returns false.

ResultSet
A table of data representing a database result set, which is generated by executing a statement that queries the database.

boolean absolute(int row)
Not implemented. Throws SQLException.

void afterLast()
Not implemented. Throws SQLException.

void beforeFirst()
Not implemented. Throws SQLException.

void cancelRowUpdates()
Not implemented. Throws SQLException.

void clearWarnings()
Not implemented. Throws SQLException.

void close()
Complies with the standard JDBC API.

void deleteRow()
Not implemented. Throws SQLException.

int findColumn(String columnName)
Complies with the standard JDBC API.

boolean first()
Not implemented. Throws SQLException.

Array getArray(int i)
Not implemented. Throws SQLException.
Array getArray(String colName)
Not implemented. Throws SQLException.

InputStream getAsciiStream(int columnIndex)
Not implemented. Throws SQLException.

InputStream getAsciiStream(int columnName)
Not implemented. Throws SQLException.

BigDecimal getBigDecimal(int columnIndex)
Complies with the standard JDBC API.

BigDecimal getBigDecimal(int columnIndex, int scale)
Not implemented. Throws SQLException.

getBigDecimal(String columnName)
Complies with the standard JDBC API.

InputStream getBinaryStream(int columnIndex)
Not implemented. Throws SQLException.

InputStream getBinaryStream(String columnName)
Not implemented. Throws SQLException.

Blob getBlob(int i)
Not implemented. Throws SQLException.

Blob getBlob(String colName)
Not implemented. Throws SQLException.

boolean getBoolean(String columnIndex)
Complies with the standard JDBC API.

boolean getBoolean(String columnName)
Complies with the standard JDBC API.

byte getByte(int columnIndex)
Not implemented. Throws SQLException.

byte getByte(String columnName)
Not implemented. Throws SQLException.
byte[] getBytes(int columnIndex)
Not implemented. Throws SQLException.

byte[] getBytes(String columnName)
Not implemented. Throws SQLException.

Reader getCharacterStream(int columnIndex)
Not implemented. Throws SQLException.

Reader getCharacterStream(int columnName)
Not implemented. Throws SQLException.

Clob getClob(int i)
Not implemented. Throws SQLException.

Clob getClob(String colName)
Not implemented. Throws SQLException.

int getConcurrency()
Not implemented. Throws SQLException.

String getCursorName()
Not implemented. Throws SQLException.

Date getDate(int columnIndex)
Not implemented. Throws SQLException.

Date getDate(int columnName)
Date getDate(int columnIndex, Calendar cal)
Not implemented. Throws SQLException.

double getDouble(int columnIndex)
Complies with the standard JDBC API.

double getDouble(String columnName)
Complies with the standard JDBC API.

int getFetchDirection()
Not implemented. Throws SQLException.
int getFetchSize()
Complies with the standard JDBC API.

float getFloat(int columnIndex)
Not implemented. Throws SQLException.

float getFloat(String columnName)
Not implemented. Throws SQLException.

int getInt(int columnIndex)
Complies with the standard JDBC API.

int getInt(String columnName)
Complies with the standard JDBC API.

long getLong(int columnIndex)
Complies with the standard JDBC API.

ResultSetMetaData getMetaData()
Complies with the standard JDBC API.

Object getObject(int columnIndex)
Complies with the standard JDBC API.

Object getObject(int i, Map map)
Not implemented. Throws SQLException.

Object getObject(String columnName)
Complies with the standard JDBC API.

Object getObject(String colName, Map map)
Not implemented. Throws SQLException.

Ref getRef(int i)
Not implemented. Throws SQLException.

int getRow()
Not implemented. Throws SQLException.

short getShort(int columnIndex)
Complies with the standard JDBC API.
short getStatement()  
Complies with the standard JDBC API.

String getString(int columnIndex)  
Complies with the standard JDBC API.

String getString(String columnName)  
Complies with the standard JDBC API.

Time getTime(int columnIndex)  
Not implemented. Throws SQLException.

Time getTime(int columnName)  
Not implemented. Throws SQLException.

Time getTime(int columnIndex, Calendar cal)  
Not implemented. Throws SQLException.

Timestamp getTimestamp(int columnIndex)  
Complies with the standard JDBC API.

Timestamp getTimestamp(int columnIndex, Calendar cal)  
Not implemented. Throws SQLException.

Timestamp getTimestamp(String columnName)  
Complies with the standard JDBC API.

Timestamp getTimestamp(String columnName, Calendar cal)  
Not implemented. Throws SQLException.

int getType()  
Not implemented. Throws SQLException.

InputStream getUnicodeStream(int columnIndex)  
Not implemented. Throws SQLException.

InputStream getUnicodeStream(int columnName)  
Not implemented. Throws SQLException.

URL getURL(int columnIndex)  
Not implemented. Throws SQLException.
URL getURL(String columnName)
Not implemented. Throws SQLException.

SQLWarning getWarnings()
Not implemented. Throws SQLException.

void insertRow()
Not implemented. Throws SQLException.

boolean isAfterLast()
Not implemented. Throws SQLException.

boolean isFirst()
Not implemented. Throws SQLException.

boolean isLast()
Not implemented. Throws SQLException.

boolean last()
Not implemented. Throws SQLException.

void moveToCurrentRow()
Not implemented. Throws SQLException.

void moveToInsertRow()
Not implemented. Throws SQLException.

boolean next()
Complies with the standard JDBC API.

boolean previous()
Not implemented. Throws SQLException.

void refreshRow()
Not implemented. Throws SQLException.

boolean relative(int rows)
Not implemented. Throws SQLException.
boolean rowDeleted()
Not implemented. Throws SQLException.

boolean rowInserted()
Not implemented. Throws SQLException.

boolean rowUpdated()
Not implemented. Throws SQLException.

void setFetchDirection(int direction)
Not implemented. Throws SQLException.

void setFetchSize(int rows)
Complies with the standard JDBC API.

void updateArray(int columnIndex, Array x)
Not implemented. Throws SQLException.

void updateArray(String columnName, Array x)
Not implemented. Throws SQLException.

void updateAsciiStream(int columnIndex, InputStream x, int length)
Not implemented. Throws SQLException.

void updateAsciiStream(String columnName, InputStream x, int length)
Not implemented. Throws SQLException.

void updateBigDecimal(int columnIndex, BigDecimal x)
Not implemented. Throws SQLException.

void updateBigDecimal(String columnName, BigDecimal x)
Not implemented. Throws SQLException.

void updateBinaryStream(int columnIndex, InputStream x, int length)
Not implemented. Throws SQLException.

void updateBinaryStream(int columnIndex, InputStream x, int length)
Not implemented. Throws SQLException.
void updateBlob(int columnIndex, Blob x)
Not implemented. Throws SQLException.

void updateBlob(String columnName, Blob x)
Not implemented. Throws SQLException.

void updateBoolean(int columnIndex, boolean x)
Not implemented. Throws SQLException.

void updateByte(int columnIndex, byte x)
Not implemented. Throws SQLException.

void updateByte(String columnName, byte x)
Not implemented. Throws SQLException.

void updateBytes(int columnIndex, byte[] x)
Not implemented. Throws SQLException.

void updateBytes(String columnName, byte[] x)
Not implemented. Throws SQLException.

void updateCharacterStream(int columnIndex, Reader x, int length)
Not implemented. Throws SQLException.

void updateCharacterStream(String columnName, Reader reader, int length)
Not implemented. Throws SQLException.

void updateClob(int columnIndex, Clob x)
Not implemented. Throws SQLException.

void updateClob(String columnName, Clob x)
Not implemented. Throws SQLException.

void updateDate(int columnIndex, Date x)
Not implemented. Throws SQLException.

void updateDate(String columnName, Date x)
Not implemented. Throws SQLException.
void updateDouble(int columnIndex, double x)
Not implemented. Throws SQLException.

void updateFloat(int columnIndex, float x)
Not implemented. Throws SQLException.

void updateInt(int columnIndex, int x)
Not implemented. Throws SQLException.

void updateInt(String columnName, int x)
Not implemented. Throws SQLException.

void updateLong(int columnIndex, long x)
Not implemented. Throws SQLException.

void updateLong(String columnName, long x)
Not implemented. Throws SQLException.

void updateNull(int columnIndex)
Not implemented. Throws SQLException.

void updateNull(String columnName)
Not implemented. Throws SQLException.

void updateObject(int columnIndex, Object x)
Not implemented. Throws SQLException.

void updateObject(int columnIndex, Object x, int scale)
Not implemented. Throws SQLException.

void updateObject(String columnName, Object x)
Not implemented. Throws SQLException.

void updateObject(String columnName, Object x, int scale)
Not implemented. Throws SQLException.

void updateRef(int columnIndex, Ref x)
Not implemented. Throws SQLException.

void updateRow()
Not implemented. Throws SQLException.
void updateShort(int columnIndex, short x)
Not implemented. Throws SQLException.

void updateShort(String columnName, short x)
Not implemented. Throws SQLException.

void updateString(int columnIndex, String x)
Not implemented. Throws SQLException.

void updateString(String columnName, String x)
Not implemented. Throws SQLException.

void updateTime(int columnIndex, Time x)
Not implemented. Throws SQLException.

void updateTimestamp(int columnIndex, Timestamp x)
Not implemented. Throws SQLException.

void updateTimestamp(String columnName, Timestamp x)
Not implemented. Throws SQLException.

boolean wasNull()
Not implemented. Throws SQLException.

---

**ResultSetMetaData**

Provides information about the types and properties of the columns in a ResultSet object.

**String getCatalogName(int column)**

Returns "IBM Cognos Real-time Monitoring".

**String getColumnClassName(int column)**

Returns null.

**int getColumnCount()**

Complies with the standard JDBC API.

**int getColumnDisplaySize(int column)**

Complies with the standard JDBC API.

**String getColumnLabel(int column)**

Complies with the standard JDBC API.
String getColumnName(int column)
Complies with the standard JDBC API.

int getColumnType(int column)
Complies with the standard JDBC API.

String getColumnType(int column)
Complies with the standard JDBC API.

int getPrecision(int column)
Complies with the standard JDBC API.

int getScale(int column)
Complies with the standard JDBC API.

String getSchemaName(int column)
Returns "IBM Cognos Real-time Monitoring".

String getTableName(int column)
Complies with the standard JDBC API.

boolean isAutoIncrement(int column)
Returns false.

boolean isCaseSensitive(int column)
Returns false.

boolean isCurrency(int column)
Returns false.

boolean isDefinitelyWritable(int column)
Returns false.

boolean isDefinitelyWritable(int column)
Returns false.

boolean isDefinitelyWritable(int column)
Returns false.

int isNullable(int column)
Returns columnNullable.

isReadOnly(int column)
Returns true.
**isSearchable(int column)**

Complies with the standard JDBC API.

**boolean isSigned(int column)**

Returns true.

**boolean isWritable(int column)**

Returns false.
Appendix B. UM-SSO API

IBM Cognos Real-time Monitoring provides an API for creating user management and single sign-on (UM-SSO) plug-ins that provide user management and single sign-on capabilities to an external application after it has authenticated the users. The API consists of three packages.

The packages are described in the following table.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.obi.ibm.um</td>
<td>This package contains the UMAPI (see “UMAPI” on page 430) interface that provides the methods for implementing a UM-SSO plug-in.</td>
</tr>
<tr>
<td>com.ibm.obi.um.exception</td>
<td>This package defines the exception that the UM-SSO methods can throw if an exception occurs.</td>
</tr>
<tr>
<td>com.ibm.obi.um.vo</td>
<td>This package contains classes for creating objects needed by the UMAPI methods, such as Credential and User.</td>
</tr>
</tbody>
</table>

How UM-SSO works

IBM Cognos Real-time Monitoring can use the same authentication logic as any other external web application by using the framework of the UM-SSO plug-in. Tasks that are specific to the external application are implemented in the plug-in while the framework ensures that all protected access uses the plug-in for authentication.

The actions of the UM-SSO plug-in fall into four basic scenarios:
- Initial log-in to Cognos Real-time Monitoring.
- Log-in to Cognos Real-time Monitoring and navigating to the external application.
- Log-in to the external application and then navigating to Cognos Real-time Monitoring.
- Synchronization of users.

Making an initial log-in to IBM Cognos Real-time Monitoring

When a user logs in to IBM Cognos Real-time Monitoring, the external application calls the doAuthenticate method with a Credential that contains the user name and password for the user. If the credentials are valid, the doAuthenticate(Credential, HttpRequest, HttpResponse) method returns a SessionInfo object that Cognos Real-time Monitoring uses to maintain the session for the user.

For more information, see “doAuthenticate” on page 430, “Credential” on page 434, and “SessionInfo” on page 437.
Logging in to IBM Cognos Real-time Monitoring and navigating to the external application

In this scenario, the login proceeds in the same ways as the initial log-in. However, the doAuthenticate(Credential, HttpRequest, HttpResponse) method must also set HTTP cookies or parameters or both so that the external application can use them to validate the current session.

Logging in to the external application then navigating to IBM Cognos Real-time Monitoring

In this scenario, the log-in to the external application must add an HTTP cookie or parameter that allows the application to validate its own sessions. When the user navigates to IBM Cognos Real-time Monitoring, these cookies and parameters remain in the request and become available to the doAuthenticate(HttpRequest, HttpResponse) method of the plug-in.

Here the plug-in authenticates the session using the same logic that the external application uses. If a SessionInfo (see "SessionInfo" on page 437) object was created previously for this session, then the same object must be returned. This ensures that the Cognos Real-time Monitoring data that was associated with the session is maintained. This scenario is handled in the same way as the scenario where the user logs in to Cognos Real-time Monitoring and then navigates to another Cognos Real-time Monitoring page. If the doAuthenticate method cannot authenticate the user, it throws a typed exception. For example, it throws a SessionTimedOutException because the session times out. For more information, see "SessionTimedOutException" on page 440.

Synchronization of users

IBM Cognos Real-time Monitoring schedules synchronization of users at regular intervals. During synchronization, Cognos Real-time Monitoring calls the fetchRoles and fetchAllUsers methods in the plug-in, which is expected to return all roles and users that are allowed access into Cognos Real-time Monitoring. The fetchRoles(User) method is called when a user is authenticated but is not synchronized to Cognos Real-time Monitoring. The user is synchronized, enabling a log-in of that user.

Installing a UM-SSO plug-in

After you create a UM-SSO plug-in using the UMAP interface, you can install the plug-in.

Procedure

1. Create a JAR file that contains your implementation. This JAR file must contain a manifest file similar to the manifest required for user-defined functions (UDFs).

   The following is an example of the manifest that you must include in the JAR file.

   ```xml
   <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

   - <jarManifest xsi:schemaLocation="http://obi.cognos.com/4 jarManifest.xsd"
     xmlns="http://obi.cognos.com/5" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <jarManifestXSDVersion>1</jarManifestXSDVersion>
   </jarManifest>
   <author>IBM Cognos Real-time Monitoring</author>
   ```
The implementor element specifies the name of the package that contains your plug-in.

2. In IBM Cognos Real-time Monitoring Workbench, click Workbench.
5. In the Choose JAR File window, select Upload a JAR file.
6. Enter the Name, Description, and Path for the JAR.
7. Click Continue.
8. Select Finish to upload the JAR.

setPropertyDetails XML

A UM-SOO plug-in can specify property details that can add additional settings to system settings in the Administration Console of IBM Cognos Real-time Monitoring Workbench.

The details of the properties are specified in XML that the getSettings (see "getSettings" on page 434) method provides to IBM Cognos Real-time Monitoring Workbench. When implementing the getSettings method in a plug-in, you must provide the XML that this method returns as a string. The format of this XML is described in the following sections.

Elements of the setPropertyDetails XML

The top-level element of the Property Details XML is setPropertyDetails. The setPropertyDetails contains two subelements: Group and Property.

Group element

The Group element defines the page that the UM-SOO plug-in adds to the system settings in the Administration Console of IBM Cognos Real-time Monitoring Workbench. A setPropertyDetails element can contain any number of Group elements, and all Group elements must occur at the beginning of the XML document. The Group element contains two subelements: name and nameKey.

name

The name element specifies the property group name for the Group. Any Property element that uses this name is related to the Group. For example, if a Group element specifies a name element with the value "Xyz", any Property element that specifies "Xyz" as the value of its propertyGroup subelement is a member of the Xyz group.

nameKey

The nameKey is the key for the property group name that corresponds to a localized string maintained in a resource bundle returned by the getResourceBundle (see "getResourceBundle" on page 432) method of the UM API
Property element
The Property element defines the controls that display on the page in system settings. All Property elements must follow the Group elements defined in the XML. However, multiple Property elements can refer to the same Group.

A Property element contains the following subelements:

• name
• nameKey
• descriptionKey
• uiProperty

name
The name of the property element functions as an identifier that an implementer of a UM-SSO plug-in can use to find the property.

nameKey
The nameKey is the key for the property that corresponds to a localized string maintained in a resource bundle. The resource bundle is returned by the "getResourceBundle" method of the "UMAPI" interface. The string in the resource bundle that matches the keyName is the name of the label displayed in System Settings for this property.

descriptionKey
The descriptionKey is the key that corresponds to a localized string maintained in the resource bundle for a description of the property. The resource bundle is returned by the "getResourceBundle" method of the "UMAPI" interface. The text of the description displays in hover help when a user moves the pointer over the property in the System Settings page.

uiProperty
The uiProperty element associates the Property element with a Group, specifies the order in which the property displays on the System Settings page if more than one Property is associated with the Group, and whether the Property is mandatory.

The uiProperty element has the following subelements:

• propertyGroup
  The name of the Group that the Property element belongs to. For example, if the propertyGroup element has a value of "Xyz", IBM Cognos Real-time Monitoring associates the Property element with the Group element that has the name "Xyz".
• listOrder
  If more than one Property element is associated with a Group element, listOrder specifies the order in which the properties display on the page in System Settings.
• isMandatory
This is a Boolean element. The value is set to false. Setting the value of the isMandatory element to true indicates that the property must be set the first time IBM Cognos Real-time Monitoring Workbench is opened.

- defaultValue
  Specifies the default value for the property. If you specify this element, you cannot specify a defaultVariable element.
- defaultVariable
  Specifies the default variable for the property. If you specify this element, you cannot specify a defaultValue element.

**propertyTypeDefinition**

The propertyTypeDefinition element specifies the control related to the property that displays on the System Settings page. The control can be a text field or box, an integer field, a radio button, a check box, or a schedule panel.

You can define any one of the following subelements in a propertyTypeDefinition to specify the type of control to display:

- string
  Creates a text field where a user can enter a string. This element has the following optional attributes:
  - maxlength
    The maxLength attribute specifies the maximum length of the string that the user can enter.
  - isPassword
    Specifies that the text field is for entering a password. A clear text algorithm is used for passwords.
- multiString
  Creates a text box where the user can enter multiple strings.
- int
  Specifies a field where the user can enter an integer. This element has two optional attributes:
  - min
    Specifies the minimum integer value that the user can enter. If the min attributes is not specified, the minimum value is -2147483648
  - max
    Specifies the maximum integer value that the user can enter. If the max attribute is not specified, the maximum value is 2147483647.
- enum
  Creates a drop-down list that displays on the System Settings page. You can specify multiple enum elements. The enum element has two attributes:
  - choiceKey
    A key that corresponds to a localized string maintained in a resource bundle. The string is used for the label associated with the radio button. The resource bundle is returned by the getResourceBundle (see “get ResourceBundle” on page 432) method of the UMAPI (see “UMAPI” on page 430) interface.
  - choiceValue.
    Specifies the value returned when the user selects the radio button.
- Boolean
Creates a check box.

- Schedule
  Adds a schedule interface to the page that lets a user select the interval at which to schedule an event, such as synchronization.

**UMAPI**

UMAPI is a Java interface that contains the methods implemented in a UM-SSO plug-in. The interface is contained in the package com.obu.ibm.um.

**doAuthenticate**

Authenticates a user based on an HTTP request and an HTTP response.

A Credential (see “Credential” on page 434) object is included in the method along with the request and response the first time a user logs in or the session of the user expires. This method is called when IBM Cognos Real-time Monitoring wants to perform a log-in while in SSO mode. The method creates a session with the external application.

**Syntax**

```java
```

**Parameters**

credential

A Credential object that contains the user name and password for the user that is being authenticated.

request

An HttpServletRequest object that contains the information for an HTTP request, such as authentication, headers, and cookies.

response

An HttpServletResponse object that contains the information for the HTTP response to the corresponding HTTP request.

**Returns**

The session information for the current session. The SessionInfo object contains a User object and associated attributes.

**Throws**

ApplicationNotAvailableException

If the external application is not available due to a network connection failure, system maintenance, and so on.
InvalidCredentialException

If the external application supplied an invalid Credential when a user attempted to log in.

**doLogout**

Logs out the current user that has a session with the external application. This method destroys the current session.

**Syntax**

```java
```

**Parameters**

*sessionInfo*

The session-specific properties along with the User for whom the session is being maintained.

*request*

An HttpServletRequest object that contains the information for the HTTP request, such as authentication, headers, and cookies.

*response*

An HttpServletResponse object that contains the information for the response to the corresponding request.

**Throws**

*ApplicationNotAvailableException*

If the external application is not available due to a network connection failure, system maintenance, and so on.

*InvalidCredentialException*

If the external application supplied an invalid Credential when a user attempted to log in.

*SessionTimedOutException*

If the session with the external application timed out.

**fetchAllUsers**

Fetch all the relevant users that have permission to interact with IBM Cognos Real-time Monitoring.

**Syntax**

```java
java.util.Iterator fetchAllUsers(int batchSize)
```
Parameters

batchSize

Number of relevant users to fetch.

Returns

An iterator with objects of type User (see "User" on page 438). The number of User objects returned depends on batchSize.

Throws

ApplicationNotAvailableException

If the external application is not available due to a network connection failure, system maintenance, and so on.

SessionTimedOutException

If the session with the external application timed out.

fetchRoles

Fetch all the relevant roles that have rights to interact with IBM Cognos Real-time Monitoring.

Syntax

public java.util.Iterator fetchRoles(int batchSize)

Parameter

batchSize

The maximum number of roles to return.

Returns

An iterator with objects of type Role (see "Role" on page 435). The maximum number of Role objects returned depends on batchSize.

Throws

ApplicationNotAvailableException

If the external application is not available due to a network connection failure, system maintenance, and so on.

SessionTimedOutException

If the session with the external application timed out.

getResourceBundle

Gets localized strings for the attributes or keys used by the plug-in. The strings are returned in the form of a ResourceBundle object.
Syntax
java.util.ResourceBundle getResourceBundle(java.util.Locale locale)

Parameter
locale
Number of relevant roles to fetch.

Returns
A ResourceBundle object that contains the localized strings for attributes or keys used in the plug-in.

Throws
ApplicationNotAvailableException
If the external application is not available due to a network connection failure, system maintenance, and so on.

SessionTimedOutException
If the session with the external application timed out.

degetRoles
Gets all the assigned roles for the specified User in the external application.

Syntax
public java.util.Iterator getRoles(User user)

Parameter
user
The user whose roles the method retrieves.

Returns
An iterator with objects of type Role (see "Role" on page 435).

Throws
ApplicationNotAvailableException
If the external application is not available due to a network connection failure, system maintenance, and so on.

SessionTimedOutException
If the session with the external application timed out.
**getSettings**

Gets the configuration settings as an XML-formatted string. The settings help IBM Cognos Real-time Monitoring register the UM-SSO plug-in. The implementation of the getSettings method determines how the XML is retrieved. For example, the method could retrieve the XML from a text file handled by the external application or have the XML hard-coded inside the method.

**Syntax**

```java
public java.lang.String getSettings()
```

**Returns**

Returns an XML document that specifies configuration settings.

**onPropertyChange**

The onPropertyChange method is a callback method called from IBM Cognos Real-time Monitoring to notify the plug-in about any property that the user changed in the Administration Settings of Real-time Monitoring and relevant to the plug-in. To retrieve the property value, the plug-in can call:

- `VCSystem.getVCProperty().getProperty(propertyName)` or
- `VCSystem.getVCProperty().isPropertySet(propertyName)`.

For example, on a property change of a schedule during synchronization, this method would stop the synchronization process.

**Syntax**

```java
public void onPropertyChange(java.lang.String propertyName)
```

**Parameter**

- **propertyName**

  The name of the property that changes in Cognos Real-time Monitoring.

---

**Credential**

The Credential class is used to create Credential value objects that hold the log-in credentials for authentication: user name and password.

A Credential value object is used in the doAuthenticate (see “doAuthenticate” on page 430) method when the external application needs to log in a user for the first time or re-establish a session for a user.

**getPassword**

Gets the password contained in the credential that is associated with the user name in the credential.

**Syntax**

```java
public java.lang.String getPassword()
```

**Returns**

A password as a string.
**getUserName**

Gets the user name contained in the credential.

**Syntax**

```java
public java.lang.String getUserName()
```

**Returns**

A user name as a string.

**setPassword**

Sets the password string in the credential.

**Syntax**

```java
public void setPassword(java.lang.String password)
```

**Parameters**

- `password`:

The password associated with the user name specified in the Credential.

**setUserName**

Sets the user name in the credential.

**Syntax**

```java
public void setUserNmae(java.lang.String userName)
```

**Parameters**

- `password`:

The user name to be contained in the Credential.

---

**Role**

The Role class is used to create Role value objects that hold the attributes associated with a role along with the users belonging to that role.

**getAttributes**

Gets the attributes specified in the role.

**Syntax**

```java
public java.util.Map getAttributes()
```

**Returns**

The attributes for the role as key-value pairs contained in a Map object.

**getRoleDescription**

Gets the description for the role.
**Syntax**
```
public java.lang.String getRoleDescription()
```

**Returns**
A string that contains the description of the role.

**getRoleName**
Gets the name for the role.

**Syntax**
```
public java.lang.String getRoleName()
```

**Returns**
Returns a string that contains the name of the role.

**getUsers**
Gets the users associated with the role.

**Syntax**
```
public java.util.Iterator getUsers()
```

**Returns**
An iterator with objects of type User (see “User” on page 438).

**setAttributes**
Sets the attributes associated with the Role.

**Syntax**
```
public void setAttributes(java.util.Map attributes)
```

**Parameters**
- attributes
  The attributes associated with the role.

**setRoleDescription**
Sets the description for the role.

**Syntax**
```
public void setRoleDescription(java.lang.String roleDescription)
```

**Parameters**
- roleDescription
  The description of the role.
setRoleName
Sets the name of the role.

Syntax
public void setRoleName(java.lang.String roleName)

Parameters
roleName
The name of the role.

setUsers
Sets the users associated with the role.

Syntax
public void setUsers(java.util.Iterator userIterator)

Parameters
userIternator
An iterator with objects of type User (see "User" on page 438).

SessionInfo
The SessionInfo class is used to create SessionInfo value objects that hold the session-specific properties along with the user for whom the session is being maintained. An external application can use SessionInfo objects to track the current session.

getAttributes
Gets the attributes for the session.

Syntax
public java.util.Map getAttributes()

Returns
Returns a Map object that contains the attributes for the session.

getUser
Returns the user associated with the session.

Syntax
public User getUser()

Returns
Returns a "User" on page 438 object.

setAttributes
Sets the attributes in the SessionInfo object.
Syntax
public void setAttributes(java.util.Map attributes)

Parameters
attributes

A Map object that contains the attributes for the session as key-value pairs.

setUser
Sets the user for the SessionInfo object.

Syntax
public void setUser(User user)

Parameter
user

The User to be associated with the session.

User

The User class is used to create a User value object that holds the attributes associated with a user.

getAttributes
Gets the attributes for the user.

Syntax
public java.util.Map getAttributes()

Returns
A Map object that contains the attributes for the user as key-value pairs.

getUserDescription
Gets the description for the user.

Syntax
public java.lang.String getUserDescription()

Returns
Returns the description of the user as a string.

getUserId
Gets the ID of the user.

Syntax
public java.lang.String getUserId()
Returns
The ID of the user as a string.

**setAttributes**
Sets the attributes for the user.

**Syntax**
```java
public void setAttributes(java.util.Map attributes)
```

**Parameters**

attributes
A Map object that contains the attributes for the user as key-value pairs.

**setUserDescription**
Sets the description for the user.

**Syntax**
```java
public void setUserDescription(java.lang.String userDescription)
```

**Parameters**

userDescription
The description for the user.

**setUserId**
Sets the user ID for the user.

**Syntax**
```java
public void setUserId(java.lang.String userId)
```

**Parameters**

userId
The user ID for the User.

Exceptions
The UM-SOO API defines three exceptions that a plug-in can throw.
- `ApplicationNotAvailableException`
- `InvalidCredentialException`
- `SessionTimedOutException`

These exceptions are defined in the `com.ibm.obi.um.exception` package.
**ApplicationNotAvailableException**

The UM-SSO plug-in throws this exception if the external application is not available for any reason, such as a network connection failure or system maintenance.

**InvalidCredentialException**

The UM-SSO plug-in throws this exception if the external application sends an invalid credential.

**SessionTimedOutException**

The UM-SSO plug-in throws this exception if the session with the external application expires.
Notices

This information was developed for products and services offered worldwide.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service. This document may describe products, services, or features that are not included in the Program or license entitlement that you have purchased.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Software Group
Attention: Licensing
3755 Riverside Dr
Ottawa, ON K1V 1B7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.
Trademarks

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:
- Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Glossary

This glossary includes terms and definitions for IBM Cognos Real-time Monitoring.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

B

business data modeling
A technique for describing the data streams, lookup tables, views, and rules that depict how a business functions.

cascade
An operation that propagates the exact same operation to all dependant objects.

C

consolidated data stream
A data stream table that captures data streams from different, but similar data stream sources and combines them into a single data stream.

D

data stream
An object that represents an event.

delivery profile
A profile that specifies where and how to deliver alerts and data feeds to the user.

disabled
Pertaining to an object that is not accepting new data. Disabling an object does not affect the definition or existence of that object, rather, disabling an object keeps new data from flowing into the object and to all objects that rely on the target object.

E

enabled
Pertaining to an object that is accepting new data and is processing them. When an object is created, it is enabled.

event
A row or a series of rows of data.

I

inner join
The result of a join operation that includes only the matched rows of both tables that are being joined. See also outer join.

invalid
Pertaining to an object that has a reference to another object that cannot be satisfied. A reference can be invalid because an object does not exist or because some attribute of an object does not match the requirements of the dependent (such as a data type mismatch), not because the dependent is disabled. All objects that depend on an invalid object are also invalid.

M

moving set function
A function that performs calculations on a set of the latest rows in a view. The set of rows to include is determined only when a new data stream arrives.

O

outer join
A join whose result consists of the matched rows of the two tables that were joined and the unmatched rows of one or both tables. See also inner join.
query window
A window that specifies a set of rows used in calculations with respect to the current row under examination. The calculation may be for computing a moving set function, a join, or expiring rows from a view.

rank function
A function that computes the scalar result for each value in a set with respect to the entire set. A rank function may only be used in the selection list of a SELECT statement.

recent view
A snapshot of the last non-empty current view. A recent view is what appears in the workbench when editing an object that displays view results.

scalar expression
An expression without a set function.

scalar function
An SQL function that optionally accepts arguments and that returns a single scalar value each time that it is invoked. A scalar function can be referenced in an SQL statement wherever an expression is valid.

scenario
A sequence of events that tests business data models for expected or possible outcomes. The models identify exceptional business conditions.

set function
A function that performs calculations on a column in a set of rows in a view, such as the average value of the cost of some similar product orders. A set function may only be used in the selection list of a SELECT statement.

snapshot view
A view that is a replica of a view at the time a query was executed.

stateful view
A view that contains the results of aggregations derived from past data streams in a single row. A view is stateful if it contains a set function or moving set function in the SELECT clause, or contains a GROUP BY clause (in which case there is one row for each group), or is derived from another stateful view.

stateless view
A view that contains only rows representing the effect of the last event.

terminal rank function
A rank function that has only scalar arguments.

terminal set function
A set function that has only scalar arguments.

tumbling set function
A function that performs calculations on a windowed set of the rows in a view. The set of rows to include is determined when a new data stream arrives, and the set empties when full.

tumbling window
A window that empties its contents when it advances to include the newest event.

U

UDF
See user-defined function

user-defined function (UDF)
A function defined by the user that provides a mechanism for extending C-SQL for use in formulas, including queries, field expressions, and rules.

V

virtual table
A table or view where the rows are derived as they are required in memory.
Index

A
ABS() 140
absolute value 140
access filters 3
assigning 8
behavior 5
condition 3
restrictions 5
user context 4
views, creating on 6
access permissions 275
acknowledged alerts 41
addition, decimal results 68
addMembersToRole.xsd file 388
addMemberToRole.xsd file 388
agents 21
altering 25
creating 24
editing 25
file system 29
flat file 25
JDBC 221, 309
JMS 30, 32
Rendezvous 34
TIBCO Rendezvous 34
Web service 35
XSD file 390
aggregate views 342
aggregation functions 120
alert functions 118
alert states, testing for acknowledged 152
alert states, testing for raised 153
alertCommon.xsd file 389
alertMessage.xsd file 389
alerts 37
acknowledged 41
escalation 42
fire 41
graphics, embedded 39
HTML, embedded 39
importance 38
mandatory subscriptions 40
message text 39
messages, consolidated 42
monitoring with rules 301
optional subscriptions 40
permissions for 280
raised 41
reportlets 44
stateful 41
stateless 41
subscribers 40
XSD file 389
aliases in dimensions 84
Allow Short Rows, delimited files option 99
alterAlert.xsd file 389
alterInformation element 386
alterRule.xsd file 389
AND operator 273
ANSI SQL-99 313
appending string lists 193
appending strings 155, 194
application interface, JDBC API 227
archiving view data 345
AS option 314
average 123, 164
MACD moving average example 164
moving 168
tumbling 175
AVG() 164
B
BEA WebLogic Portal Server 281
BETWEEN operator 272
bookmarks
XSD file 389
Boolean
data types 75
functions 118
business activities
XSD file 390
business rules 49
about 49
creating 50
business views 337
C
C-SQL
data types 63
functions 123
caching context query results 250
Cartesian product 316
CASE expression 131, 314
case statement 130
CAST() 131
casting data types 63
numeric to string 67
numeric types 67
CDATA 382
CEI 51
event tables, creating 54
CEI data stream 52
CEIL() 141
CHAR_LENGTH() 155
characters
length of a string 155
lowercase conversion 157
uppercase conversion 164
checkpoint
XSD file 391
cognoscAPI.jar file 334
column references 314
naming in selection 314
columns, value of previous row 137, 148, 153, 160
combining numeric types 66
comma separated value 99
commandBatch.xsd file 385, 391
commit 313
common event infrastructure 51

© Copyright IBM Corp. 2007, 2013
common.xsd file 391
comparison operators 272
CONCAT() 155
concatenating strings 155
lists 193
sets 194
concatList() 193
concatSet() 194
ConcatSet() sample UDF 334
configXML file 12
jarring 12
Connection JDBC interface 393, 394
consolidated alert messages 42
consolidated events 342
XSD file 390
constants 274
constraints on views 316, 339
cross joins 316
CSV 99
cubes 57
XSD file 390
cumulative distribution function 195
currency, displaying money 141
current date and time 133
current view snapshot 343
CURRENT_TIMESTAMP() 133
CURRENT_USER() 154
CURRENT() 166
dashboard objects
XSD file 389
Dashboard Profile
delivery profile 330
dashboards
XSD files 389
data flow
current view snapshot 343
data from external sources 78
data stream
CEI 52
data streams
table name 350
upsert 217
data type
LONG 65
data types 63
Boolean 75
casting 63
casting order of precedence 64
casting to other types 63
date-time 70, 71
DECIMAL 65
DECIMAL, casting to string 67
DOUBLE 65
DOUBLE, casting to string 67
INTEGER 65
Java int data type values 231
JDBC 65, 220
JMS 210
numeric 65
Rendezvous 327
string 68, 69
TIBCO Rendezvous 327
VARCHAR 68
DATA_TYPE column, values 231
Data-Driven Subscription tab 40
date functions 118
DATE_ADD() 133
DATE_DIFF() 134
date-time data types 70
adding a duration 133
casting 70
current 133
date-time intervals 72
date-time intervals 73
creating
business rules 50
functions (continued)
count star, tumbling window set 176
COUNT() 166
CURRENT_TIMESTAMP() 133
CURRENT_USER() 154
CURRENT() 166
date_add() 133
date_diff() 134
descriptions 123
display_money() 141
exp() 143
fLOOR() 144
formulas, usages in 115
gammaDist() 195
greatest() 135, 144, 156
is_acked() 152
is_raised() 153
last_day() 135
least() 136, 145, 157
list of all 123
log() 145
logNormDist() 196
lower() 157
lpad() 158
ltrim() 159
max() 167
median() 196
min() 168
mod() 146
mode() 197
mov_avg() 168
mov_count() 169
mov_max() 170
mov_min() 171
mov_std_deviation() 182
mov_sum() 172
mov_variance() 183
moving set 117
ntile() 184
position() 159
power() 147
prev() 173
prior_value() 137, 148, 153, 160
rank 117
rank() 186
ratio_to_report() 187
reference 123
round() 149
rpad() 161
rtrim() 162
safe_divide() 149
set 116
sign() 150
sqr() 151
std_deviation() 188
substr() 163
substring() 163
sum_over_groups() 189
sum() 174
timestamp_diff() 137
to_char() 139
to_date() 139
trunc() 151
tumble 376
tumble_avg() 175
tumble_count() 176
tumble_max() 177
functions (continued)
tumble_min() 179
tumble_std_deviation() 189
tumble_sum() 180
tumble_variance() 191
tumbling set 118
types of 115
upper() 164
variance() 192
G
gamma distribution 195
gammaDist() 195
gonography maps 84
categories 84
glossary 445
grant permissions 279
graphics in alerts 39
GREATEST()
date and time 135
numbers 144
text 156
GROUP BY clause
implicit 321
moving sets, interacting with 129
stateful views 321
H
HAVING clause 318
hierarchy of dimension levels 84
holds for attribute, rules 298
HTML
code in alerts 39
forms (event posting) 204
http post event 201
event tables 201
posting data to 204
tables, creating 202
url 204
I
IBM WebSphere Portal Server 282
identifiers 254
implicit casting of data types 63
import
XSD file 391
in
in-line
query windows 362
views 317
individual subscription tab 40
initialize clause 378, 379
inner joins 317
integer data type 65
integer time-series query windows 370
interfaces (API) 228
Interval data type
literal 72
intervals
date-time 72
day-time 73
year-month 72
invalid
  objects 253
  state 253
invokeExternalAction.xsd 358
IS operator 273
IS_ACKED() 152
IS_RAISED() 153

J
JAR
  manifest, XSD file 391
  XSD file 391
JAR Files folder 13
jarManifest.xsd file 391
Java
  data type values 231
  date-time data type 71
  numeric data type 65
  SimpleDateFormat class 74
  string data type 69
Java Database Connectivity 211
Java Messaging Service 207
JBoss Portal Server 283
JDBC 211
  agents 221, 309
  API 227
  context tables 211
  data types 220
  date-time 71
  event tables 211
  interface examples 232
  interfaces 228
  polling 216
JDBC data types
  date-time 71
  numeric 65
jdbcConnection.xsd file 390
jdbcSource.xsd file 390
JMS
  agents 30, 32
  data types 210
  event tables, creating 209
  events 207
JMSTopicConnection.xsd file 390
  joins
    cross, declaring 316
    inner 317
    nested 317
    outer 317
    synchronized 340
JSR-168 portal integration 281

K
key columns
  dimensions 85

L
language codes 142
largest value of a set 135, 144, 156
LAST_DAY() 135
latest row in a set 166
leap second 134
leap year 134
LEAST()
  date and time 136
  numbers 145
  text 157
LIKE operator 272
limitations of views 316, 339
literals
  INTERVAL 72
  TIMESTAMP 71
locale 70
LOG() 145
log4jConnection.xsd file 390
logarithm 145
logging
  XSD file 391
  logical functions 118
  login name
    returning 154
lognormal distribution 196
logNormDist() 196
LONG data type 65
LOWER() 157
lowercase conversion 157
LPAD() 158
LTRIM() 159

M
MACD 164
maintaining events in stateless views 344
mandatory subscriptions 40
manifest files, user-defined functions 334
math functions 118, 119
MAX() 167
mean average 123, 164
  moving 168
  tumbling 175
measures 58
median() 196
message text, alerts 39
messageSource.xsd file 390
metrics, in real-time 342
MIN() 168
mixing numeric types 66
MOD() 146
model() 197
money, displaying currency 141
months, last day of 135
MOV_AVG() 168
MOV_COUNT() 169
MOV_MAX() 170
MOV_MIN() 171
MOV_STD_DEVIATION() 182
MOV_SUM() 172
MOV_VARIANCE() 183
moving averages, MACD example 164
moving sets 126
  functions 117
    GROUP BY, interacting with 129
moving windows of stateless view 344
multiple row events 97
multiplication, decimal results 67
MySQL
  date-time data type 71
  numeric data type 65
  string data type 69
N
name
  data stream 349, 350
event 78
names, restrictions on 254
namespace 261
nested joins 317
new features
  version 10.2.0 xvii
  version 10.2.1 xvii
No Access permission 275
NOT operator 273
NTILE() 184
NULL
  constant, testing for 274
  value semantics 75
number sign 150
numbers
  rounding 141
  rounding down 144
  rounding up 149
  truncating 151
numeric
  casting 67
  combining 66
  data types 65
  decimal precision results 67
  operators 271

O
objects 253
  altering 253
  altering with XML 386
  defining with XML 384
  disabling 253
  enabling 253
  invalidating 253
  management XSD files 391
  name restrictions 254
  name uniqueness constraints 261
  relationship, XSD file 391
  status 253
Objects
  selective export 262
  operators 271
  optional subscriptions 40
OR operator 273
Oracle
  date-time data type 71
  dstring data type 69
  numeric data type 65
ORDER BY (query window) clause 368
ORDER BY (SELECT) clause 322
outer joins 317
permissions (continued)
  grant 279
  inheritance 279
  No Access permission 275
  Read-Only permission 275
  Read-Write permission 275
  restrictions 279
  setting 296, 331
persisting view data 345
plans
  XSD file 389
  POSITION() 159
  PostgreSQL
    numeric data type 65
  PostgreSQL
    date-time data type 71
    string data type 69
POWER() 147
  precision and scale, declaring 67
  PREV() 173
  previous row in a set 173
  PRIOR_VALUE()
    date and time 137
    numbers 148
    other 153
    text 160
  probability mass function 195
  process order of events 77
  processes 285
  procSource.xsd file 390
  profiles 330
  properties
    XSD file 391
    propertyTypeDefinition.xsd file 391

Q
qualified column reference 314
query windows 361
  advancement 374
  declarations 362
  event-series windows 364
  frames 361
  in-line declarations 362
  initialization 378
  integer time-series 370
  partitions 371
  reference, update 377
  references 362
  sliding 374
  time-series windows 366
  tumbling 375
  update references 377
queryCube.xsd file 390
queryInformation.xsd files 390

R
raised alerts 41
RANGE clause 366
rank functions 117
rank values
  as a ration to the set 187
  by percentage of set 187
  into tiers 184
  within the entire set 186
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANK()</td>
<td>186</td>
</tr>
<tr>
<td>ranking functions</td>
<td>119</td>
</tr>
<tr>
<td>RATIO_TO_REPORT()</td>
<td>187</td>
</tr>
<tr>
<td>Read-Only permission</td>
<td>275</td>
</tr>
<tr>
<td>Read-Write permission</td>
<td>275</td>
</tr>
<tr>
<td>real-time metrics</td>
<td>342</td>
</tr>
<tr>
<td>recent view snapshot</td>
<td>343</td>
</tr>
<tr>
<td>recovery file for event data</td>
<td>79, 349, 350</td>
</tr>
<tr>
<td>REFERENCE clause</td>
<td>377</td>
</tr>
<tr>
<td>references</td>
<td></td>
</tr>
<tr>
<td>XSD file</td>
<td>389</td>
</tr>
<tr>
<td>regular identifiers</td>
<td>254</td>
</tr>
<tr>
<td>Relative XPath</td>
<td>101</td>
</tr>
<tr>
<td>replacement fields</td>
<td>356</td>
</tr>
<tr>
<td>reporting on view data</td>
<td>345</td>
</tr>
<tr>
<td>Reportlet data based on option</td>
<td>46</td>
</tr>
<tr>
<td>reportlets</td>
<td>291</td>
</tr>
<tr>
<td>access filters, interaction with</td>
<td>5</td>
</tr>
<tr>
<td>alerts, attached to</td>
<td>44</td>
</tr>
<tr>
<td>Cognos LAVA attributes</td>
<td>292</td>
</tr>
<tr>
<td>creating</td>
<td>291</td>
</tr>
<tr>
<td>external attributes</td>
<td>292</td>
</tr>
<tr>
<td>filtering</td>
<td>44</td>
</tr>
<tr>
<td>permissions for</td>
<td>280</td>
</tr>
<tr>
<td>XSD file</td>
<td>389</td>
</tr>
<tr>
<td>reserved words</td>
<td>254</td>
</tr>
<tr>
<td>roles</td>
<td></td>
</tr>
<tr>
<td>access filters, with</td>
<td>8</td>
</tr>
<tr>
<td>attributes</td>
<td>296</td>
</tr>
<tr>
<td>creating</td>
<td>295</td>
</tr>
<tr>
<td>ROUND()</td>
<td>149</td>
</tr>
<tr>
<td>rounding numbers</td>
<td></td>
</tr>
<tr>
<td>ceiling</td>
<td>141</td>
</tr>
<tr>
<td>down</td>
<td>144</td>
</tr>
<tr>
<td>up</td>
<td>149</td>
</tr>
<tr>
<td>rows</td>
<td></td>
</tr>
<tr>
<td>current</td>
<td>166</td>
</tr>
<tr>
<td>latest</td>
<td>166</td>
</tr>
<tr>
<td>previous</td>
<td>173</td>
</tr>
<tr>
<td>RPAD()</td>
<td>161</td>
</tr>
<tr>
<td>RPC style messages</td>
<td>36</td>
</tr>
<tr>
<td>RPC, WSDL</td>
<td>353</td>
</tr>
<tr>
<td>RTRIM()</td>
<td>162</td>
</tr>
<tr>
<td>ruleCommon.xsd file</td>
<td>389</td>
</tr>
<tr>
<td>rules</td>
<td>297</td>
</tr>
<tr>
<td>access filters, interaction with</td>
<td>5</td>
</tr>
<tr>
<td>action</td>
<td>300</td>
</tr>
<tr>
<td>alerts, monitoring</td>
<td>301</td>
</tr>
<tr>
<td>for a specific clause</td>
<td>300</td>
</tr>
<tr>
<td>functions</td>
<td>119</td>
</tr>
<tr>
<td>Holds for attribute</td>
<td>298</td>
</tr>
<tr>
<td>permissions for</td>
<td>280</td>
</tr>
<tr>
<td>XSD file</td>
<td>389</td>
</tr>
<tr>
<td>RVConnection.xsd file</td>
<td>391</td>
</tr>
</tbody>
</table>

**S**

SAFE_DIVIDE() 149

samples

ConcatSet() UDF 334

XML objects 381

SAP 349

Save as New View option 338

scenarios 311

attributes 311

creating 312

deleting 312

scenarios (continued)

permissions for 280

XSD file 390

schedule.xsd file 392

schedules

XSD file 392

Schema XPath 101

SELECT command 313

FROM clause 316

ORDER BY clause 322

select list 314

WHERE clause 319

select list 314

selective export 262

Separator character 99

set data stream time zone format flat file 98

set functions 116

setLogLevel.xsd file 391

setObjectRelation.xsd file 391

setPrivilege.xsd file 389

setProperty.xsd file 391

sets 126

current row 166

latest row 166

previous row 173

setUserPreferences.xsd file 389

severity 38

Siebel 349

sign of a number 150

SIGN() 150

simple column reference 314

Simple Object Access Protocol 353

SimpleDateFormat, Java class 74

SLIDE clause 374, 375

sliding window 126

SOAP 353

SOAPConnection.xsd file 391

sourceDefinition.xsd file 390

sources of external data 78

space characters in XML files 381

SQL Server

date-time data type 71

string data type 69

SQRT() 151

standard deviation 188

moving window set 182

tumbling window set 189

stateful

alerts 41

views 343

views and GROUP BY 321

stateless

alerts 41

maintaining more than one event 344

views 343

Statement JDBC interface 397

statistical functions 119

AVG() 164

median() 196

mode() 197

status 253

STD_DEVIATION() 188

string replacement templates (web Service) 356
strings
  appending characters 161
  appending lists of 193
  appending to 155
  appending to a set 194
  concatenating 155
  concatenating a list 193
  concatenating a set 194
  converting to non-string data types 69
  data types 68
  dates, converting to 139
  functions 120
  inserting characters 158
  length, determining 155
  lowercase conversion 157
  operators 271
  padding, left 158
  removing characters 159
  removing characters from right 162
  search for a character in a substring 163
  uppercase conversion 164
  width declaration 68
subscribers 40
subscriptions
  mandatory 40
  optional 40
SUBSTRI() 163
SUBSTRING() 163
subtraction, decimal results 68
SUM_OVER_GROUPS() 189
SUM() 174
superuser 329
Sybase
  limitations 250
  numeric data type 65
  string data type 69
  synchronized joins 340
  system
    administration, XSD files 391
    property, XSD file 391
    user 329
  systemCommand.xsd file 391
T
  table names
    data streams 349, 350
    events 78
  tableDefinition.xsd file 390
  tables
    expressions 317
    names, context 249
    virtual, declaring 313
  tasks
    message XSD file 389
    XSD file 389
  TextAgent.xml file 27
  text
    files 87, 93
    functions 120
  Text qualifier 99
  TIBCO Rendezvous 323
    agents 34
    data types 210, 327
    event tables 323, 324
    limitations 323
    messages 326
time
  current date-time 133
  functions 118
time format
  create custom 98
time-series 129
  aggregations, functions for 120
  aggregations, spans 129
  windows 366
  windows, integer based 370
  TIMESTAMP data type
    formatting 74
    interval between two 137
    literal 71
  TIMESTAMP_DIFF() 137
timezone 70
to a set 194
TO_CHAR() 139
TO_DATE() 139
total
  moving 172
  set, of a 174
  tumbling 180
  transaction 313
  TRUE constant 75
  TRUNC() 151
tumble functions 376
  TUMBLE_AVG() 175
  TUMBLE_COUNT() 176
  TUMBLE_MAX() 177
  TUMBLE_MIN() 179
  TUMBLE_STD_DEVIATION() 189
  TUMBLE_SUM() 180
  TUMBLE_VARIANCE() 191
tumbling
  set functions 118
  windows 375
U
  Unicode, character length 155
  UNKNOWN constant 75
  upload utility 382, 384
  UPPER() 164
  uppercase conversion 164
  upsert
    data streams 217
    user name, returning 154
  user-defined functions 333
    altering 334
    creating 334
    manifest file 334
    restrictions 333
    XSD file 391
    users 329
      access filters, with 8
      attributes 329
      context, as 4
      creating 329
      delivery profiles 330
      details 329
      profiles 330
      system user 329
valid state 253
VARCHAR data type 68
    width declaration 68
VARIANCE() 192
VC_EVENT_ID column 346
VC_LATEST_EVENT_ID column 346
VC_string reserved names 254
VC_TIMESTAMP column 346
view expansion 321
view merging 321
views 337, 345
    access filtered, derived from 5
    access, limiting 3
    aggregation 342
    API 227
    attributes 338
    consolidated events 342
    constraints 316, 339
    creating 338
    current snapshot 343
    derived from GROUP BY views 321
    filtered access 3
    functions specific to 121
    in-line views 317
    initialization 343
    limitations 316, 339
    persisting to database 345
    recent snapshot 343
    restricted access 3
    retaining events in stateless views 344
    stateful 343
    stateful when using GROUP BY 321
    stateless 343
    table name, context 249
    tables 78, 349, 350
    updating for new events 343
    workset 338
    XSD file 390
virtual tables
    declaring 313

Web portal server integration 281
web services 349
    context 353
    context, creating 354
    events 349

web services (continued)
    external process 357
    external process, XML format 358
Web services
    agents 35
    delivery profile 330
    webServiceSource.xsd file 390
    WHERE clause 319
    white space in XML files 390
    WINDOW clause 362
    windows 126, 361
        stateless view, of 344
workbench 2
    launching 1
    workset 338
WSDL
    doc format 353
    RPC format 353

XML 381
    altering an object with 386
    CDATA 382
    character literals 382
    defining an object with 384
    escape characters 382
    example files 381
    sample files 381
    schema files 388
    source files (flat file agent) 100
    upload errors 382, 383
    uploading files 382
    white space in files 381
    XSD files 388
XML format
    external web service process 358
xmlBuffer.xsd file 391
XPaths 101
    relative 101
    schema 101
XSD 381
    data types (flat file agent) 102
    files 388

year-month intervals 72