



**Getting Started Guide**

---

# **Sybase CEP Option R4**

DOCUMENT ID: DC01027-01-0400-02

LAST REVISED: April 2011

Copyright © 2011 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. A ® indicates registration in the United States of America.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568

# Contents

<b>Filtering Data</b> .....	<b>1</b>
Sybase CEP Engine Data Flow .....	2
Starting Sybase CEP Studio .....	2
Creating a Project .....	5
Adding an Input Stream .....	6
Defining a Stream Schema .....	8
Attaching an Input Adapter .....	12
Adding an Output Stream .....	17
Writing a Query .....	18
Starting a Project .....	20
Using the Flow View .....	22
Examining Project Files .....	23
Modifying the Query: Exercise .....	23
Wrap-up .....	24
Solutions .....	25
Modifying the Query: Solution .....	25
<b>Maintaining State</b> .....	<b>27</b>
Getting Started: Maintaining State .....	27
Maintaining State: Overview .....	27
Maintaining State .....	28
Maintaining State per Column Value .....	31
Aggregating Values .....	32
Exercises .....	34
Consolidating the Code .....	34
Exploring Window Types .....	35
Exercise .....	37
Maintaining Rows Based on Rank .....	37
Exercises .....	39
Wrap-up .....	40
Solutions .....	40
Aggregating Values .....	40

Exploring Window Types .....	41
Maintaining Rows Based on Value .....	41
<b>Joins .....</b>	<b>43</b>
Combining Data from Multiple Sources .....	43
Before Beginning the Combining Data Tutorial ...	43
Joining a Stream to a Window .....	43
Executing the Join .....	45
Exploring Join Variations: Exercise .....	48
Creating an inner join: Exercise .....	49
Creating an outer join: Exercise .....	50
Combining data tutorial: Conclusion .....	54
Solutions .....	54
Creating an additional join filter: Exercise .....	55
Predicting and Controlling Join Output .....	56
Seeing the Join in Action: Exercise .....	56
Join Example .....	57
.....	57
<b>Event Pattern Matching .....</b>	<b>59</b>
Event pattern matching tutorial overview .....	59
Detecting a sequence of events: Exercise .....	59
Examine and Run the Project .....	60
Detecting One of Several Events .....	64
Subdividing a Sequence: Exercise .....	65
Publishing the Matched Pattern .....	67
Event pattern matching tutorial conclusion .....	69
Solutions .....	70

# Filtering Data

Filtering data enables you to filter incoming data to a specific location such as a stock symbol or an IP address.

Sybase® CEP Engine is an information processing system designed to run continuous queries. This means that instead of having to submit your query each time you want it to produce results, you submit your query to Sybase CEP Server, which runs the query continuously until you tell it to stop. This gives you the power to examine and analyze large volumes of incoming data virtually instantaneously, even if the data is arriving at very high speeds, without having to store the information in a database first.

This tutorial walks you through the process of using Sybase CEP Studio to build a simple Sybase CEP application that filters incoming stock trade data for events related to a specific stock symbol. While the example used in this tutorial is based on stock trading, filtering is a fundamental activity that you will undoubtedly incorporate into your application regardless of business segment. For example, you can filter sensor readings for values that fall outside a particular range, or filter clickstream data for user connections from a particular IP address.

If you would like a conceptual introduction to Sybase CEP Engine before you begin, see *Sybase CEP Engine Data Flow* on page 2. This information is also presented as a recap at the end of the tutorial.

After completing this tutorial, you will be able to:

- Describe the user interface components of Sybase CEP Studio.
- Locate and identify the purpose of Sybase project files.
- List the components of a Sybase application and define the function of each.
- Write and run a simple filtering application.

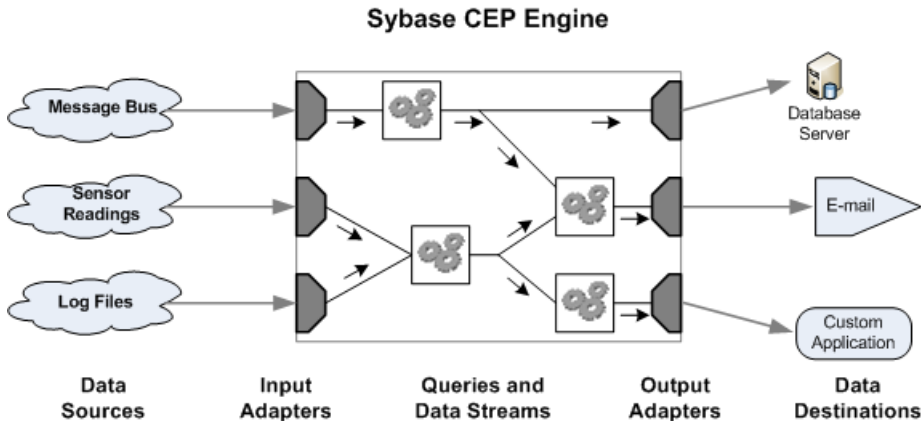
Before you begin the activities in this tutorial, be sure you have access to the following:

- An instance of Sybase CEP Server running on your computer. If you don't have such an instance, follow the directions in "Installing Sybase CEP Engine" in the *Sybase CEP Installation Guide*. Note that attempting to complete the tutorial while connected to a remote instance of Sybase CEP Server will not work because of the data files used in the tutorial.
- Sybase CEP Studio installed on your computer. If you have not installed Sybase CEP Studio, follow the directions in "Installing Sybase CEP Studio" for your platform in the *Sybase CEP Installation Guide*.

## Sybase CEP Engine Data Flow

Illustrates the flow of data through the Sybase CEP Engine.

As shown in the following diagram, Sybase CEP Engine processes incoming external data and publishes the results to external destinations:



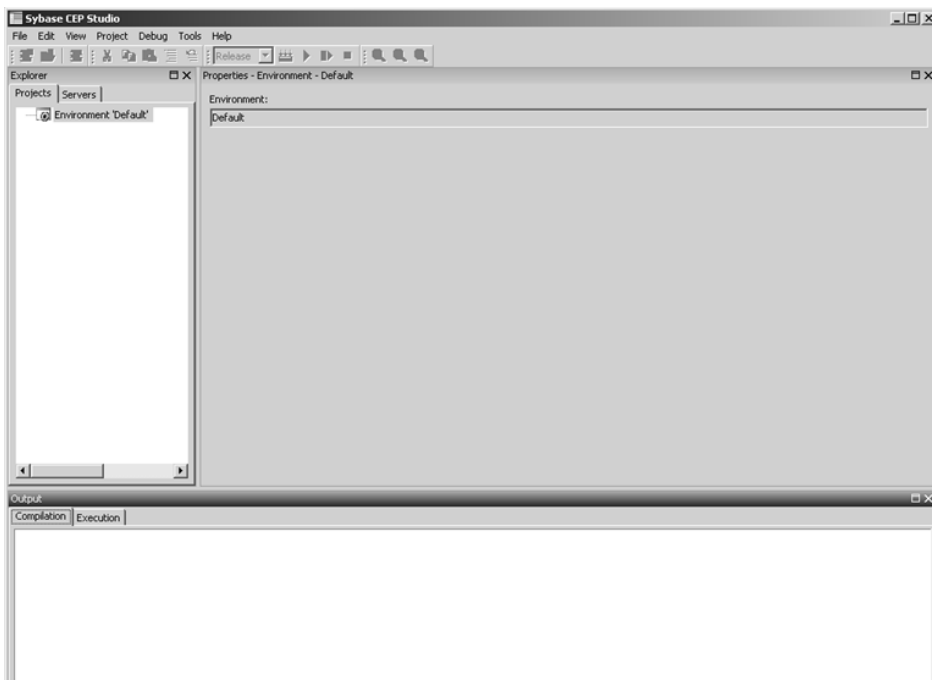
## Starting Sybase CEP Studio

Get started on Sybase CEP Studio by becoming familiar with the user interface.

Sybase CEP Studio is the Sybase CEP development application that interacts with Sybase CEP Server (the execution environment for Sybase CEP applications) and provides a way for you to write, compile, and run Sybase CEP applications. You can use other tools for development such as the Sybase CEP command-line tools or the Sybase CEP Eclipse plug-in, but this tutorial uses Sybase CEP Studio to introduce concepts common to all development methods.

Follow these steps to start Sybase CEP Studio and examine the user interface:

1. Make sure that Sybase CEP Server is running and then start Sybase CEP Studio:

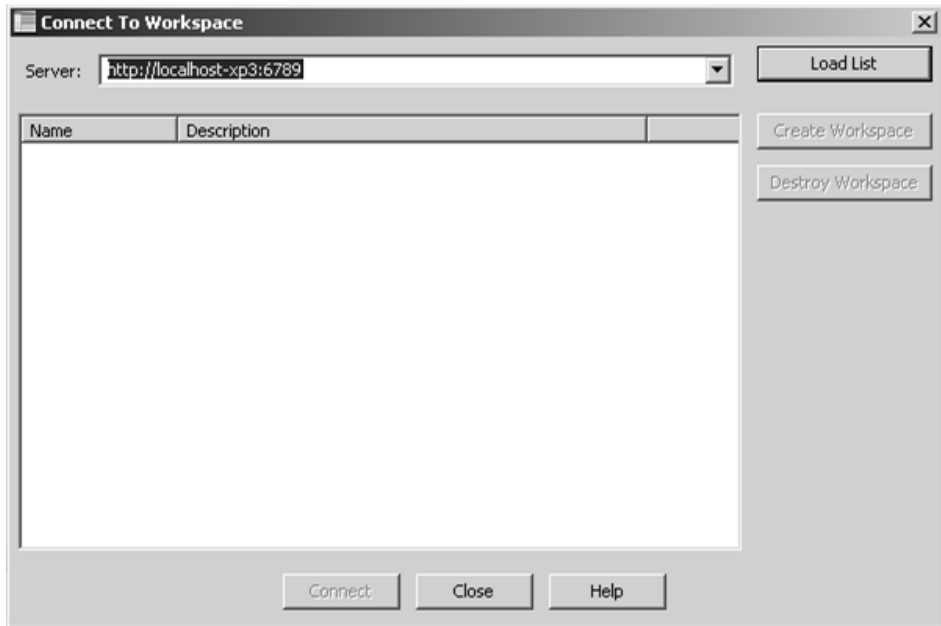


Examine the different sections, called views, of the main Sybase CEP Studio window:

- The Explorer view allows you to create, examine, and manipulate the components of your application.
- The Properties view allows you to examine and modify details specific to a particular component.
- The Output view displays operational messages.

The Explorer view shows that you are working in the Sybase CEP environment. An environment is a mechanism within Sybase CEP Studio that allows you to view and work with a subset of your applications. The default environment is Sybase CEP, but you can create additional environments to suit your needs. Note that environments are specific to Sybase CEP Studio - Sybase CEP Server has no knowledge of environments, nor does the Sybase CEP Eclipse plug-in.

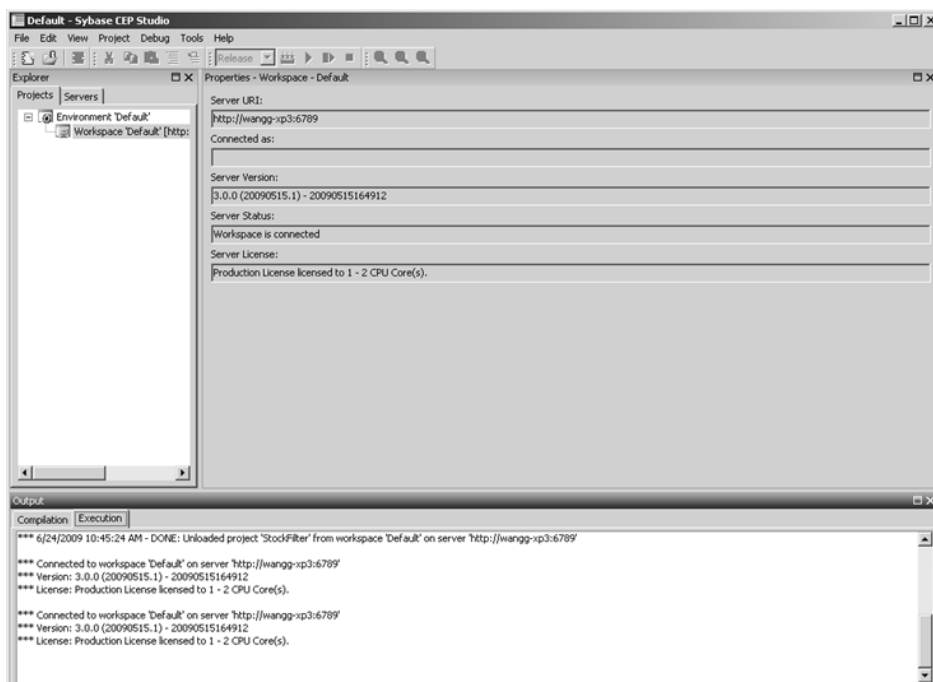
2. On the File menu, click **Connect to Workspace**:



You use a workspace to partition groups of projects into independent working areas. The default is named Default, but you can create additional workspaces to suit your needs. Note that, unlike environments, workspaces do exist within Sybase CEP Server and are common to all Sybase CEP development tools.

3. Click Default in the list of workspaces and then click the Connect button:





Notice how the views have changed:

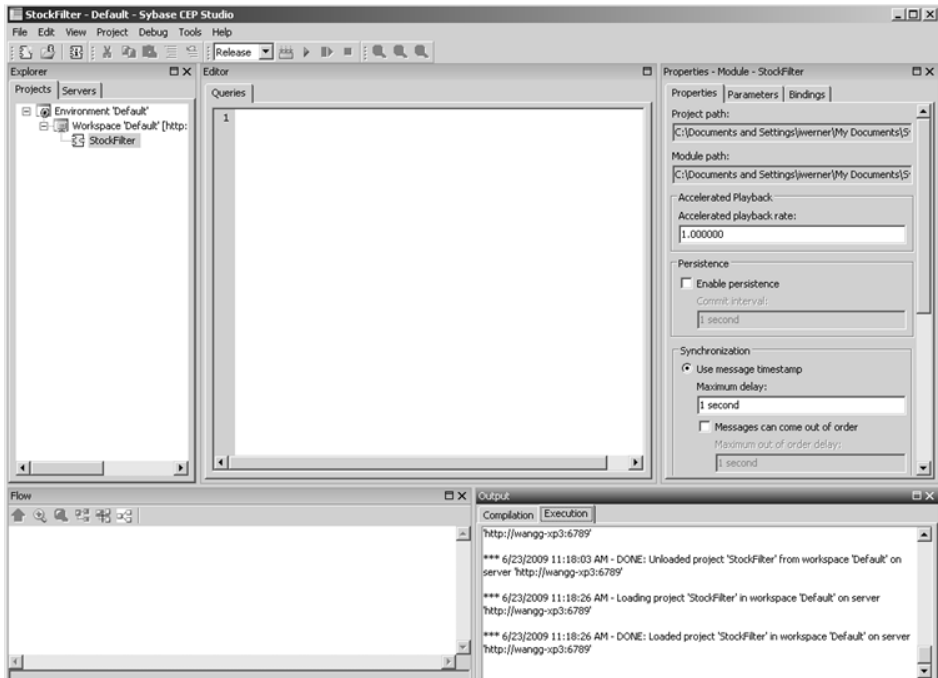
- The Explorer view shows your connection to the default workspace.
- The Properties view shows information specific to that workspace.
- The Output view shows messages about making the connection to the workspace.

## Creating a Project

Use the New project function to create a new project.

A Sybase CEP project contains application components, including groups of executable statements and connections to data input and output. Follow these steps to create a new project:

1. On the File menu, click New Project.
2. Change the directory location for your project by typing a new path into the Location text box or by clicking the Browse button and then selecting or creating a directory.
3. Type "StockFilter" into the Name text box and then click OK.:



The window now displays two additional views:

- The Editor view, where the application code is entered.
- The Flow view, which displays a flow diagram of your project and all of its components. If your project is empty, the Flow view will display nothing.

The contents of the other views have changed:

- The Explorer view now shows your new project.
- The Properties view now shows properties specific to the new project.
- The Execution tab in the Output view displays messages about loading the project. Projects are loaded in order to be edited and started.

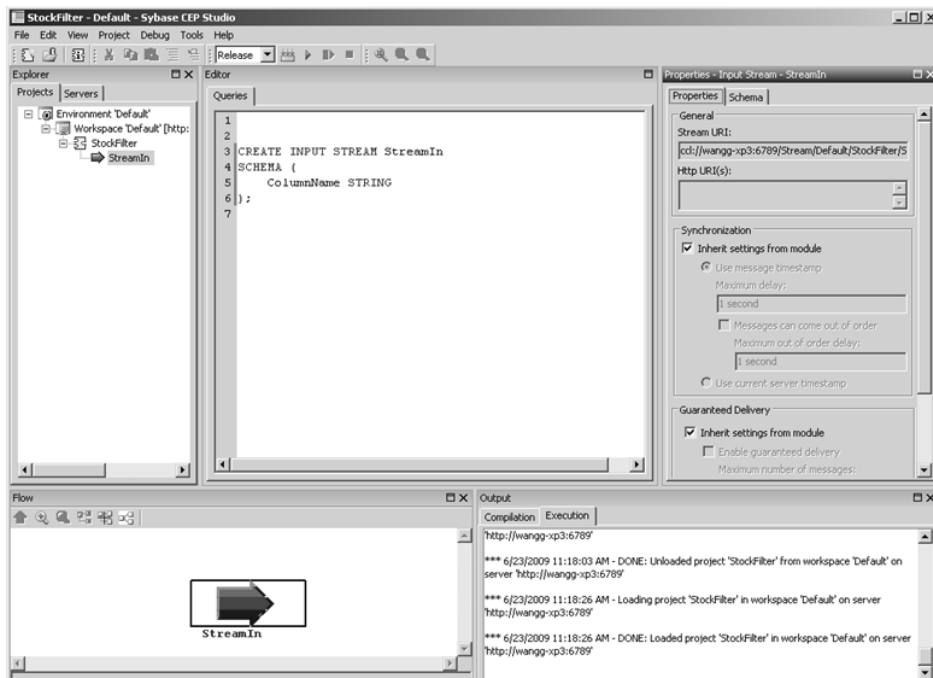
## Adding an Input Stream

Add an input stream to filter income data from particular values.

With Sybase CEP Engine, continuous data flows in data streams (usually just called streams). The project will filter incoming data for particular values, so you must identify how this data is flowing into the project by adding an input stream.

Follow these steps to add an input stream to your project:

1. Click StockFilter in the Explorer view.
2. On the Project menu, click Add Input Stream.:



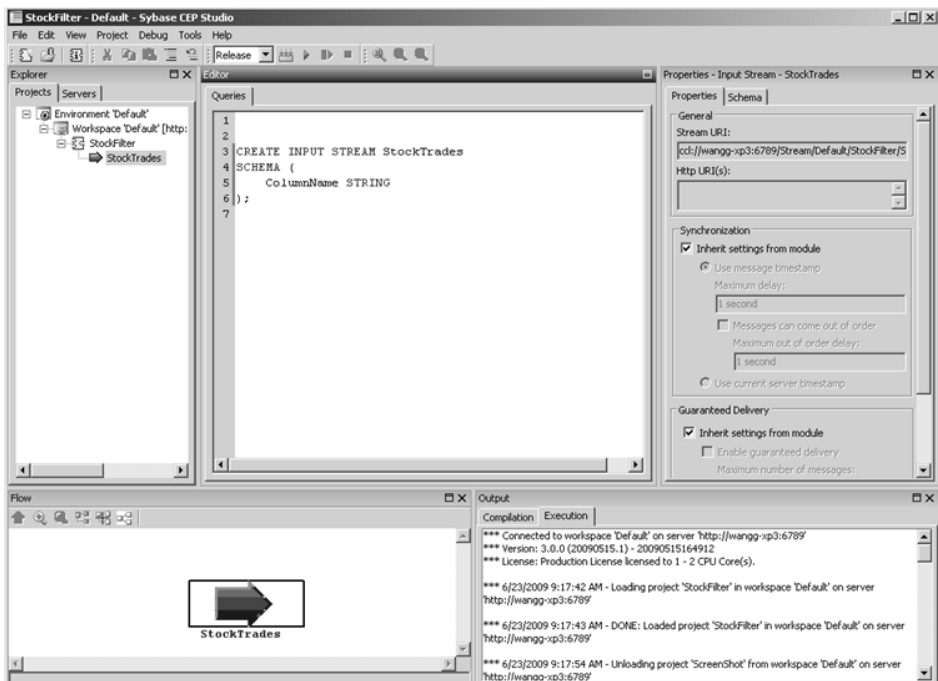
The contents of the views include:

- The Explorer view, which shows the new input stream, identified with a blue arrow and named StreamIn.
- The Properties view, which shows properties specific to the new stream.
- The Flow view, which displays an icon for the stream.
- The Editor view, which displays the statement that creates the new stream.

Note that right-clicking a project name in the Explorer view opens a shortcut menu containing project-specific commands. You can access similar shortcut menus for any other component displayed in the Explorer view and even for objects displayed in the Flow view. Throughout this tutorial, when you are instructed to click a menu command, you can use the shortcut menu instead if you prefer.

3. Examine the lines in the Queries tab:
  - Sybase CEP applications are written in Sybase CEP's Continuous Computation Language (CCL), a language similar to SQL. When changes are made to your project using the Sybase CEP Studio user interface, Sybase CEP Studio may insert CCL code for you.

- CCL reserved words are capitalized by convention to make them easier to differentiate, but CCL is actually case-insensitive. Note that Sybase CEP Studio color-codes different parts of the syntax, displaying reserved words in blue.
  - This statement creates an input stream named StreamIn with a default schema containing a single column.
  - The specific formatting of these lines is mainly for readability. You only need a single space to separate the components of a CCL statement, but CCL treats single spaces, multiple spaces, and new lines as equivalent. A semi-colon ends each CCL statement.
4. In the Queries tab, replace "StreamIn" with "StockTrades" and then wait a few seconds for Sybase CEP Studio to update the window.:



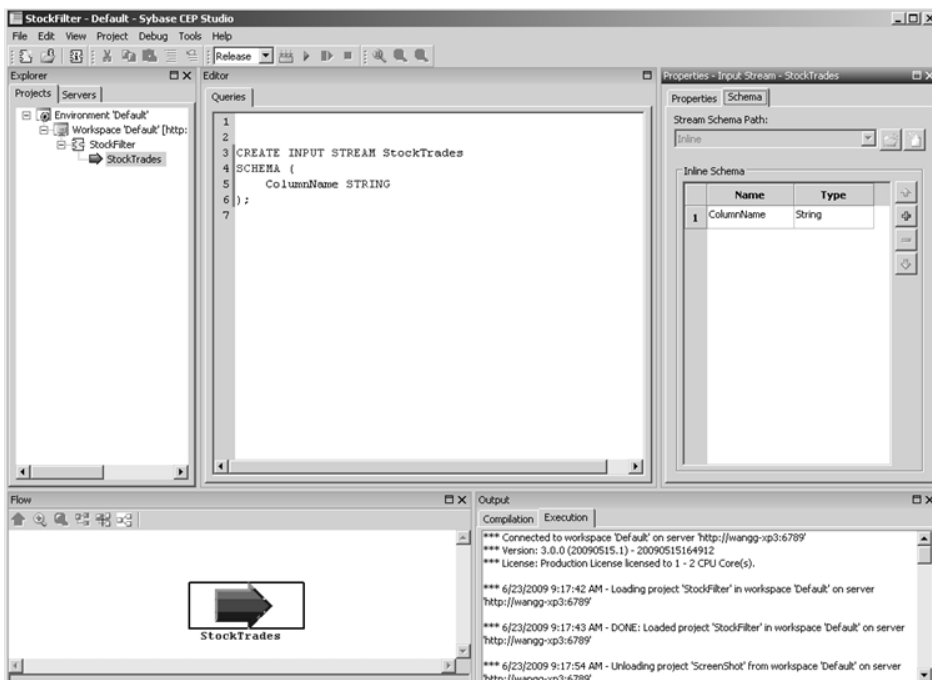
## Defining a Stream Schema

Define your input schema in order to specify the format of the data in it.

Data in a Sybase CEP stream is formatted in rows and columns, similar to database tables. Each stream requires a schema definition, which specifies the format of the data in it.

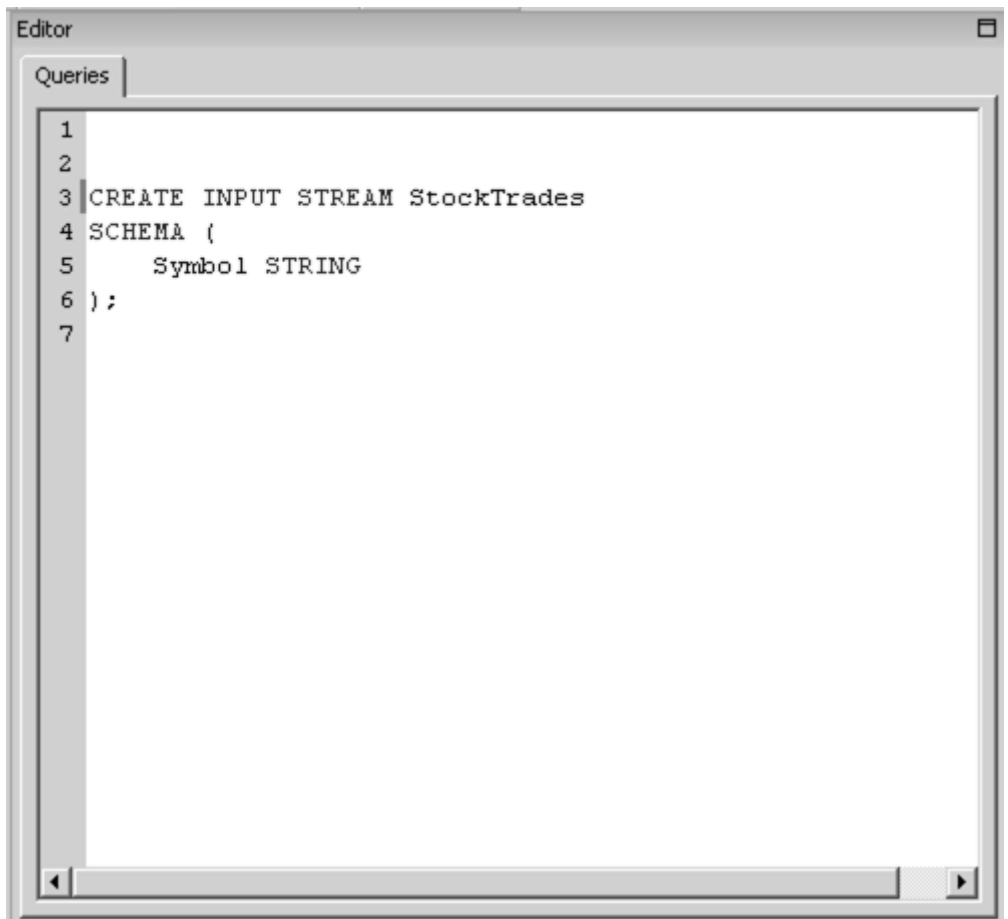
Follow these steps to modify the default schema of your input stream:

1. Click StockTrades in the Explorer view and then click the Schema tab in the Properties view:



The Schema tab shows the default schema containing a single column of type String.

2. Replace "ColumnName" with "Symbol" in the Name text box, and then click somewhere outside that box. Sybase CEP Studio updates the Queries tab to match:



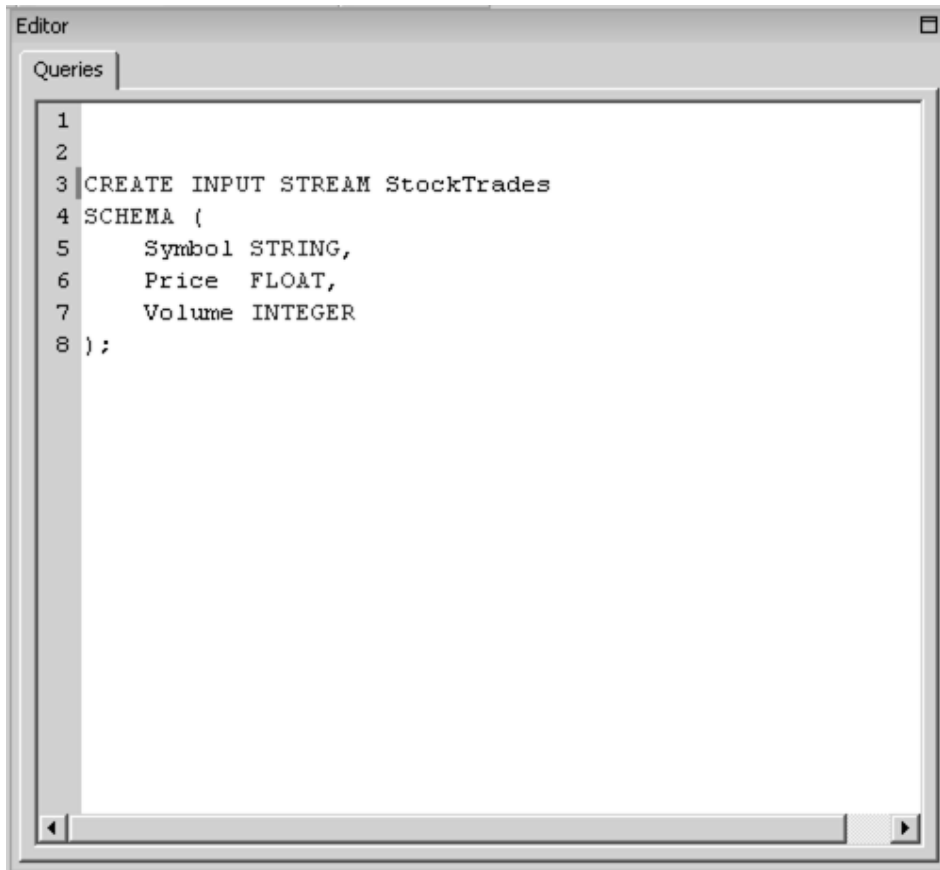
The screenshot shows a window titled "Editor" with a tab labeled "Queries". The text area contains the following SQL code:

```
1  
2  
3 CREATE INPUT STREAM StockTrades  
4 SCHEMA (  
5     Symbol STRING  
6 );  
7
```

3. Click the Insert column button (labeled with a plus sign) in the Schema tab.
4. Type "Price" into the Name text box and then click Float in the Type list.
5. Add a third column named Volume of type Integer. Your schema definition should look like this:



And the Queries tab should look like this:



The UI controls in the Schema tab are for convenience, and you can simply enter the CCL text into the Queries tab directly if you prefer.

Note: Sybase CEP Studio automatically saves changes as they are made, eliminating the need to save your project explicitly.

## Attaching an Input Adapter

---

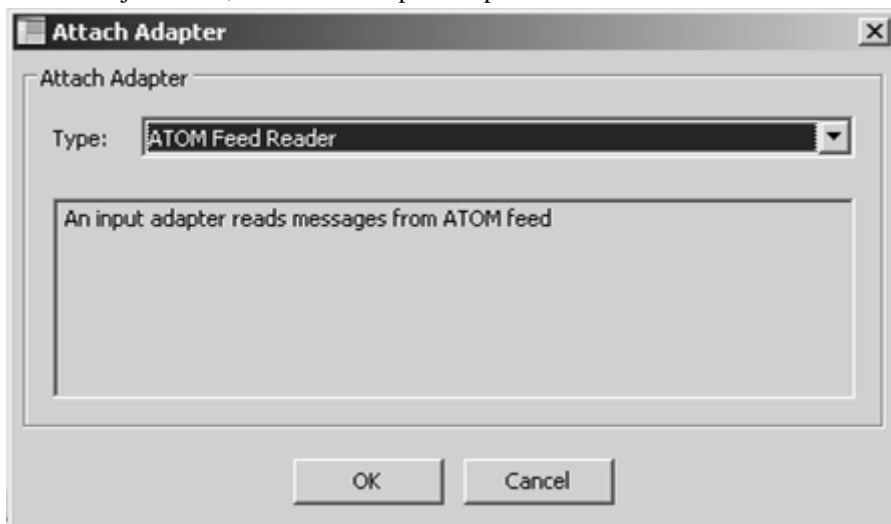
Attach an input adapter to your input stream. This adapter converts data from an external source.

After creating an input stream and specifying the format for the data on that stream, create a mechanism for feeding or publishing data to the stream. Input adapters convert data from an external source into a Sybase CEP-compatible format. Input adapters are attached to input streams.

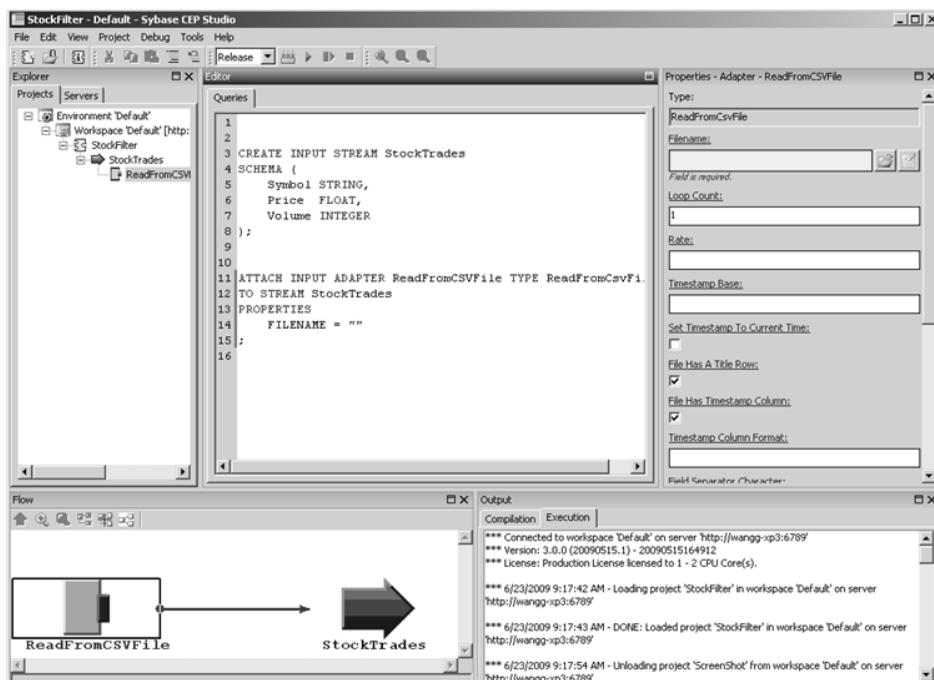
Follow these steps to attach an input adapter to your data stream:



1. Click StockTrades in the Explorer view.
2. On the Project menu, click Attach Input Adapter:



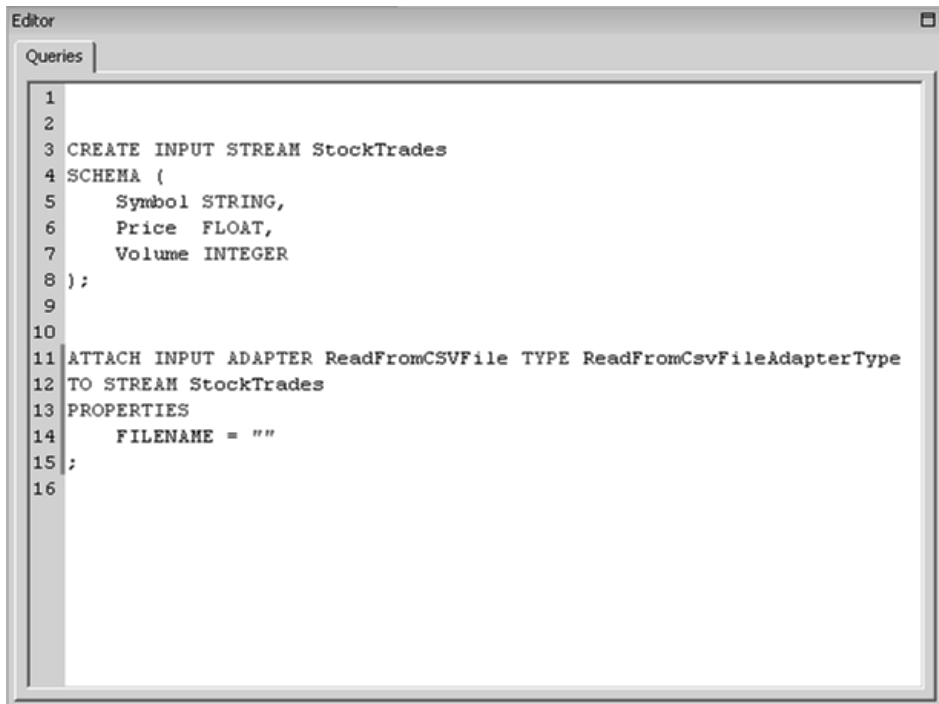
3. In the Type list, click Read from CSV File and then click OK:



The Queries tab now includes the CCL statement that attaches the input adapter and the Properties view now displays properties specific to this adapter. To see an explanation of a

particular property, click its name. Clicking the name again closes the explanation. The Flow view also displays the input adapter and shows the connection to the input stream.

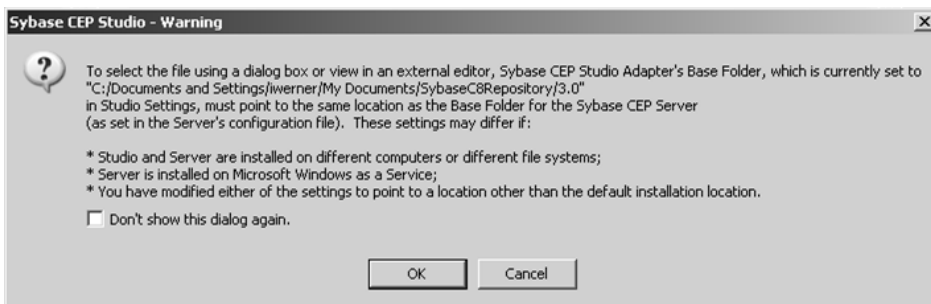
4. Examine the new statement in the Queries tab:



```
Editor
Queries
1
2
3 CREATE INPUT STREAM StockTrades
4 SCHEMA (
5     Symbol STRING,
6     Price  FLOAT,
7     Volume INTEGER
8 );
9
10
11 ATTACH INPUT ADAPTER ReadFromCSVFile TYPE ReadFromCsvFileAdapterType
12 TO STREAM StockTrades
13 PROPERTIES
14     FILENAME = ""
15 ;
16
```

An Attach Adapter statement specifies the kind of adapter (INPUT), the name of the adapter (ReadFromCSVFile), the type of adapter (ReadFromCSVFileAdapterType), and the name of the stream to which it is attached (StockTrades). Finally, a list of PROPERTIES modify the behavior of the adapter. Each type of adapter uses different properties. This adapter requires the name of a file. For more information about the Attach Adapter statement, see "ATTACH ADAPTER Statement" in the *Sybase CEP CCL Reference Guide*. For more information about adapters and adapter properties, see "Adapters Supplied by Sybase CEP" in the *Sybase CEP Integration Guide*.

5. In the Queries tab, replace "ReadFromCSVFile" with "StockTradesReader" to change the name of the adapter.
6. In the Properties view, click the Select an existing file button (labeled with an open folder), to the right of the Filename text box:



The warning that appears explains that Sybase CEP Studio expects that the files you use with input adapters are kept in the SybaseC8Repository. Sybase CEP Server expects input files to be in the adapters base folder. If you install both applications on the same computer, then both Sybase CEP Studio and Sybase CEP Server use the same folder. If you install the applications on separate computers, or if someone modifies the default settings for either Sybase CEP Studio or Sybase CEP Server, then you will not be able to use this method to identify input adapter data files.

7. Click OK.
8. Locate the SybaseC8Repository (in Windows, the default location is under My Documents), and navigate to version/examples/GettingStarted/DataFiles (where version is the version number of Sybase CEP Engine) and then open the file stock-trades.csv .
9. In the Properties view, type "2" into the Rate text box to indicate that the adapter should read two rows per second from the file. This slow rate allows you to see the behavior of the project more easily when you run it. Without this rate, the entire data file is processed in less than a second.
10. Clear the File Has Timestamp Column check box and then check the Set Timestamp To Current Time check box to indicate that the data file does not include a timestamp and that Sybase CEP Engine should assign the current time to each row as it arrives. The Properties tab should look like this:

## Filtering Data

Properties - Adapter - StockTradesReader

Type:  
ReadFromCsvFile

Filename:  
\$BaseFolder\examples\GettingStarted\DataFiles\stock-trades.csv

Loop Count:  
1

Rate:  
2

Timestamp Base:

Set Timestamp To Current Time:

File Has A Title Row:

File Has Timestamp Column:

Timestamp Column Format:

Field Separator Character:  
,

And the Queries tab should look like this:

```

1
2
3 CREATE INPUT STREAM StockTrades
4 SCHEMA (
5     Symbol STRING,
6     Price FLOAT,
7     Volume INTEGER
8 );
9
10
11 ATTACH INPUT ADAPTER StockTradesReader TYPE ReadFromCsvFileAdapterType
12 TO STREAM StockTrades
13 PROPERTIES
14     FILENAME           = "$BaseFolder\examples\GettingStarted\DataFiles\stock-trades.csv",
15     TIMESTAMPCOLUMN    = "false",
16     USECURRENTTIMESTAMP = "true",
17     RATE                = "2"
18 ;

```

Again, you can choose to set adapter properties either with the Sybase CEP Studio UI elements or by typing the CCL text into the Queries tab directly. The order of the lines specifying the properties is not significant.

## Adding an Output Stream

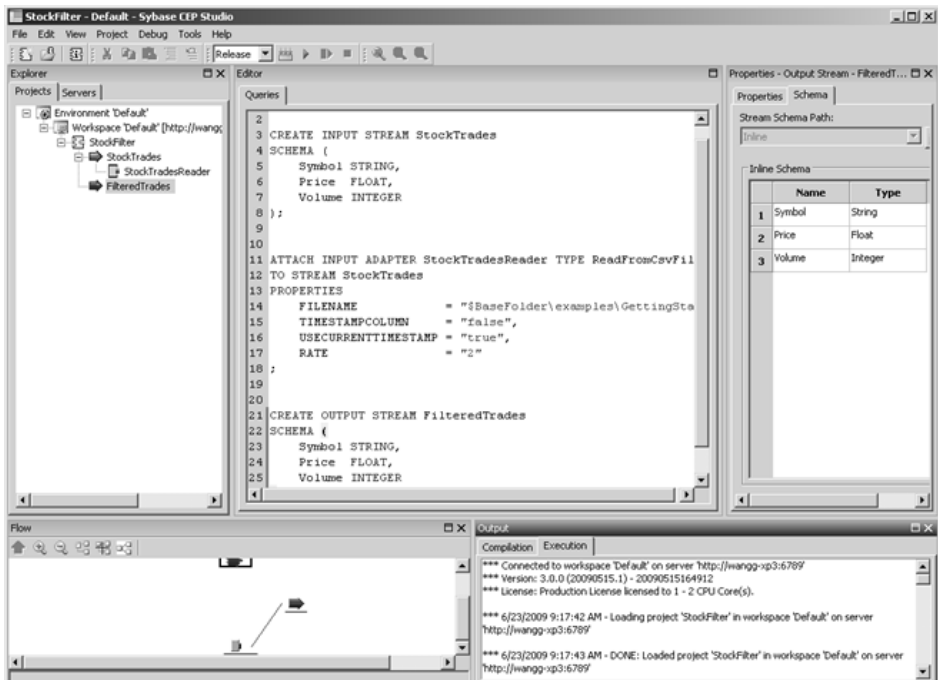
---

Create an output stream for your data. The output stream converts data for external use.

Once output has been provided for your project, specify where the output will be sent by adding an output stream.

Follow these steps to add an output stream to your project:

1. On the Project menu, click Add Output Stream.
2. Replace "StreamOut" with "FilteredTrades" in the new CREATE OUTPUT STREAM statement in the Queries tab.
3. Copy the three-line schema definition from the CREATE INPUT STREAM statement at the top of the Queries tab and then paste it over the schema definition for the output stream.:



For the purposes of this tutorial, you're just going to be looking at the data that flows through the output stream. In a real application, you would attach the output stream to some kind of adapter to convert the data for use by an external application.

## Writing a Query

Write application code which filters incoming data.

Now, with your project components in place, you're ready to write the application code that will filter incoming data for rows containing a specific stock symbol.

Follow these steps to write a query:

1. Type the following text into the Queries tab after the existing statements, starting at about line 25. As you type, notice the autocompletion feature. When the autocompletion list appears, click the word you want to use and then press the TAB key:

```
INSERT INTO FilteredTrades
```

Autocompletion works not only for CCL reserved words, like INSERT and INTO, but also for stream and column names that you've defined yourself. This first line of code specifies that the results of the query be published to the FilteredTrades output stream.

2. Enter the rest of the query:

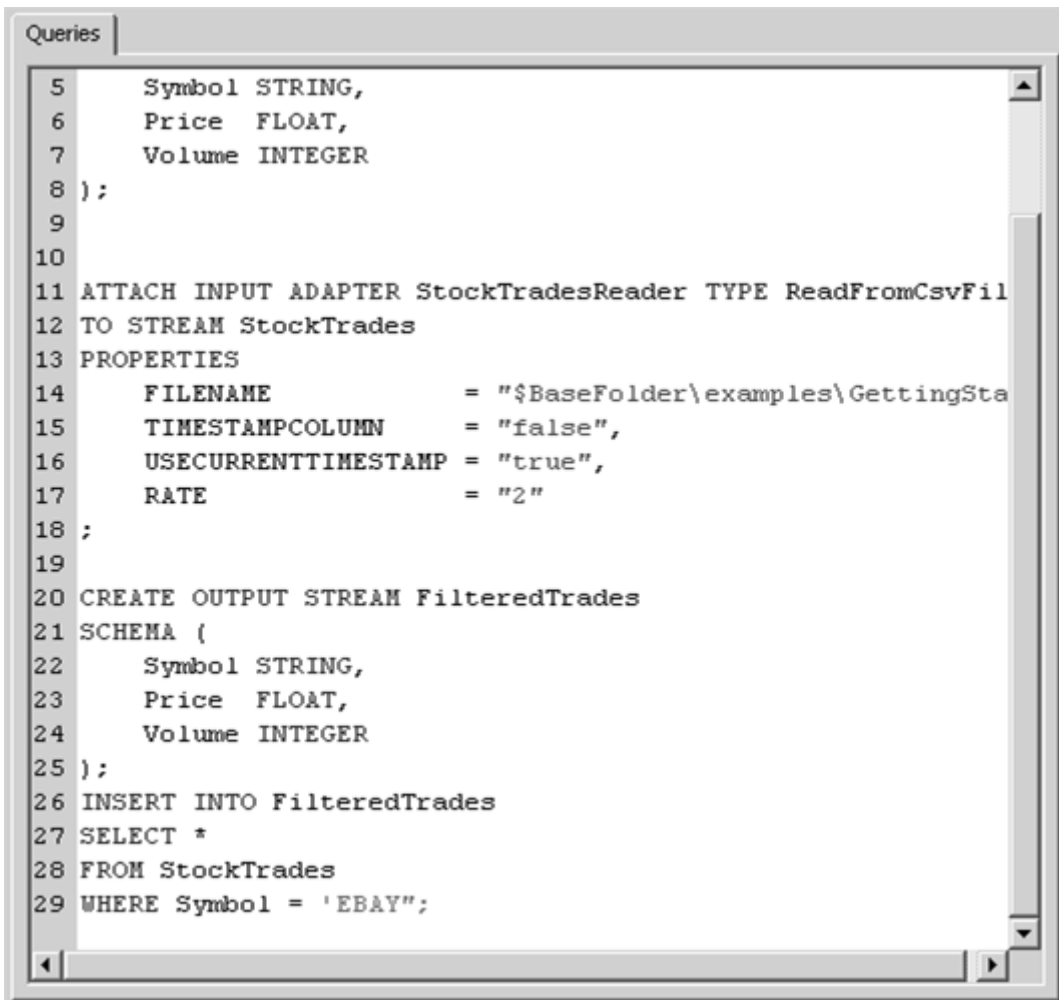
```

SELECT *
FROM StockTrades
WHERE Symbol = 'EBAY';

```

These lines of code tell Sybase CEP Engine to take all fields from the input stream (StockTrades), but only from rows where the value of the Symbol column is "EBAY". A semi-colon ends the Query statement.

Your Queries tab should look like this:



```

5      Symbol STRING,
6      Price  FLOAT,
7      Volume INTEGER
8 );
9
10
11 ATTACH INPUT ADAPTER StockTradesReader TYPE ReadFromCsvFil
12 TO STREAM StockTrades
13 PROPERTIES
14     FILENAME           = "$BaseFolder\examples\GettingSta
15     TIMESTAMPCOLUMN    = "false",
16     USECURRENTTIMESTAMP = "true",
17     RATE               = "2"
18 ;
19
20 CREATE OUTPUT STREAM FilteredTrades
21 SCHEMA (
22     Symbol STRING,
23     Price  FLOAT,
24     Volume INTEGER
25 );
26 INSERT INTO FilteredTrades
27 SELECT *
28 FROM StockTrades
29 WHERE Symbol = 'EBAY';

```

Notice how Sybase CEP Studio color-codes different parts of the syntax for easy identification: reserved words are blue; column names, stream names, and numeric values

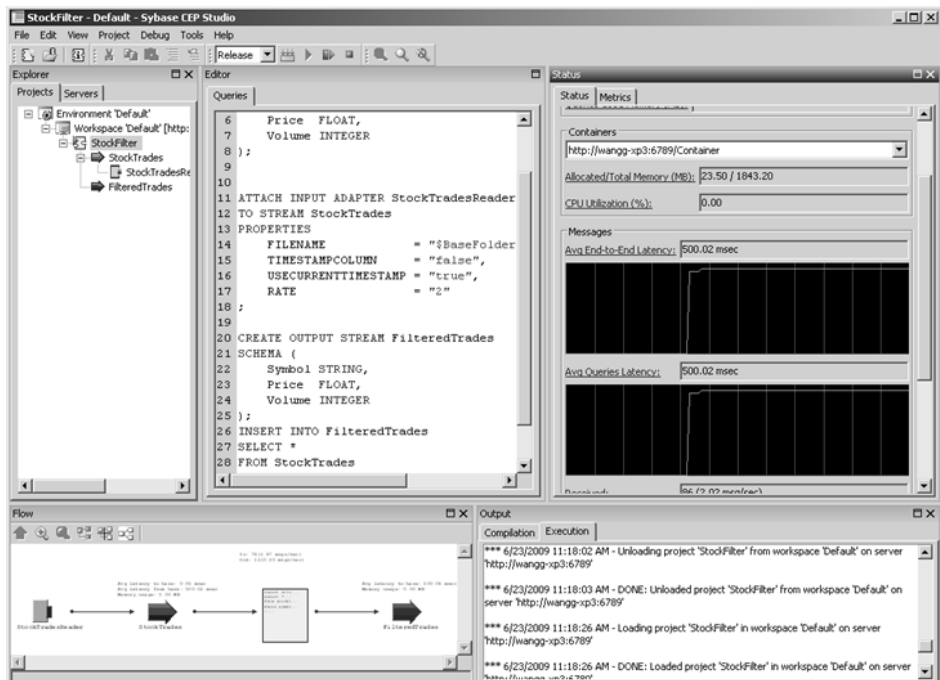
are black; string literals enclosed in double quotation marks are red, and string literals enclosed in single quotation marks are magenta.

## Starting a Project

Create a new project and run the new project.

Finally, you're ready to compile and run your project. Follow these steps to start your project and examine the outcome:

1. On the Debug menu, click View All Streams in Module 'StockFilter' (or press F9). Two windows appear which will display the data flowing through the streams when you run your project. Position them on your screen so that you can see the contents of both at the same time (or click Arrange All Viewers on the Debug menu).
2. On the Debug menu, click Start (or click the Start button on the toolbar, which looks like a green arrowhead):



Clicking Start compiles and runs the project. The Output view shows the results of the compilation. Any errors will appear here. The Properties view is replaced with a Status view, showing information about the running project, including memory and CPU usage.

3. Examine the StockTrades stream viewer:



StockTrades (Default/StockFilter/StockTrades)

Filter Expression:

Timestamp	Symbol	Price	Volume
2009/06/23 13:41:56.746230	AAPL	35.1500000000...	2000
2009/06/23 13:41:57.246230	SWY	21.5500000000...	400
2009/06/23 13:41:57.746230	EBAY	30.4400000000...	100
2009/06/23 13:41:58.246230	FILE	6.4900000000...	100
2009/06/23 13:41:58.746230	IBM	75.0800000000...	300
2009/06/23 13:41:59.246230	IBM	75.1500000000...	200
2009/06/23 13:41:59.746230	SWY	21.7000000000...	500
2009/06/23 13:42:00.246230	EBAY	30.5000000000...	400
2009/06/23 13:42:00.746230	AAPL	35.0000000000...	100
2009/06/23 13:42:01.246230	EBAY	29.9500000000...	300
2009/06/23 13:42:01.746230	MSFT	24.9700000000...	200
2009/06/23 13:42:02.246230	MSFT	24.9700000000...	4000

This viewer shows the data in the input stream, as it is read from the file by the input adapter. Notice the rate and the number of messages displayed at the bottom of the window. Remember that you set the message rate as a property of the input adapter. Also note the timestamp column, which is not read from the file but added by Sybase CEP Engine - another property of the input adapter.

4. Now examine the FilteredTrades stream viewer.:

FilteredTrades (Default/StockFilter/FilteredTrades)

Filter Expression:

Timestamp	Symbol	Price	Volume
2009/06/23 13:44:05.746230	EBAY	30.5000000000...	800
2009/06/23 13:44:06.746230	EBAY	29.9500000000...	800
2009/06/23 13:44:12.746230	EBAY	30.5000000000...	800
2009/06/23 13:44:13.746230	EBAY	29.9500000000...	900
2009/06/23 13:44:18.246230	EBAY	30.5000000000...	200
2009/06/23 13:44:19.746230	EBAY	30.5000000000...	100
2009/06/23 13:44:20.746230	EBAY	29.9500000000...	400
2009/06/23 13:44:25.246230	EBAY	30.5000000000...	800
2009/06/23 13:44:26.246230	EBAY	29.9500000000...	800
2009/06/23 13:44:29.246230	EBAY	30.4400000000...	800
2009/06/23 13:44:32.246230	EBAY	30.5000000000...	800
2009/06/23 13:44:33.246230	EBAY	29.9500000000...	1700

Show only:

Rate: 0.40 msg/sec Total: 95 msg

This viewer shows the data in the output stream after being processed by your query. Notice the rate and number of messages in this window, both of which are lower than in the input stream viewer. As you would expect from your query, all of the output rows have "EBAY" in the Symbol column.

5. On the Debug menu, click Stop (or click the Stop button on the toolbar, shaped like a red square) to halt execution.

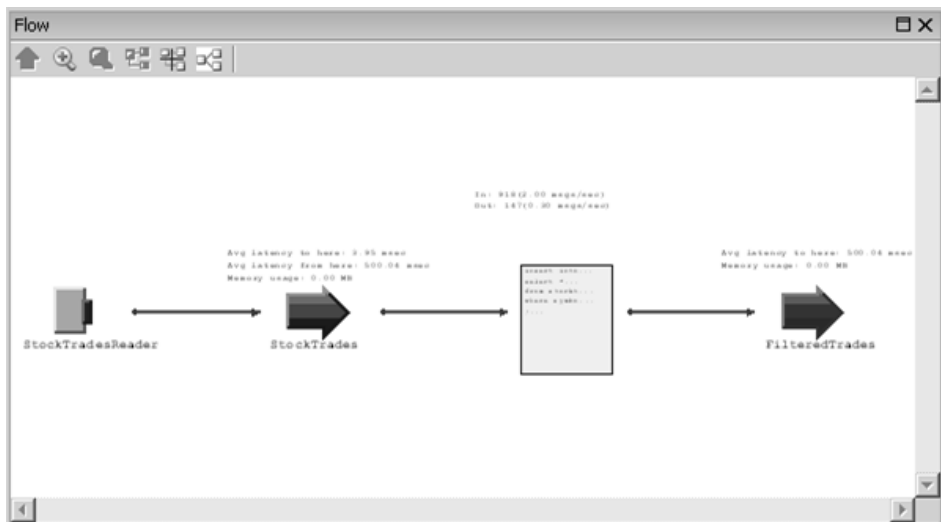
## Using the Flow View

Use the flow view to see a graphical representation of projects.

The Flow view presents a graphical representation of your project. It can help you visualize how project components interact and even help you locate logic errors.

Follow these steps to explore the Flow view:

1. Click the Maximize button in the Flow view title bar:



2. Move the pointer over each of the components in turn, pausing to notice the related CCL code that appears.
3. Right-click each component to display the shortcut menu commands available for each.
4. Start the project, maximize the Flow view again, and then examine the information about latency, memory usage, and message traffic displayed in the Flow view.
5. Stop the project.
6. In the Flow view, click each component in turn while watching the Queries tab. Notice that the relevant CCL code is indicated when you click a component.
7. Drag the title bar of the Flow view and experiment with placing the view in other locations, including completely undocked. You can move the other Sybase CEP Studio views, with the exception of the Editor view, in the same way.
8. Click Restore Default Layout on the View menu.

## Examining Project Files

---

Identify different plain text files which contain information about your project.

Sybase CEP Studio stores information about your project in a collection of plain text files which you can open in a text editor to examine.

Follow these steps to identify the different files:

1. Navigate to the directory where your project is stored and examine the files and subdirectories:

Name ▲	Size	Type	Date Modified
data		File Folder	10/1/2008 9:45 AM
modules		File Folder	10/1/2008 10:07 AM
StockFilter.ccp	1 KB	CCP File	10/1/2008 8:42 AM

Files named with a .ccp extension contain project information.

2. Examine the contents of the modules directory:

Name ▲	Size	Type	Date Modified
StockFilter.ccl	1 KB	CCL File	10/1/2008 10:07 AM

Files named with a .ccl extension hold query modules, made up of CCL statements. When you start a project, the query modules are compiled and sent to Sybase CEP Server for execution. Sybase CEP Server stores compiled query modules in files named with a .ccx extension.

You may need to know the location of different project files for use in some CCL statements or to facilitate working in a shared development environment.

## Modifying the Query: Exercise

---

Explore additional filter capabilities with Sybase CEP Engine.

The project as it stands now only filters based on a single value of a single column. However, much more sophisticated filtering can be accomplished with Sybase CEP Engine.

## Filtering Data

Complete the following exercises to explore additional capabilities. See *Solutions* on page 25 for possible solutions to these exercises.

1. Insert "OR Symbol = 'SWY'" before the semi-colon at the end of the Query statement so that the last line looks like this:

```
WHERE Symbol = 'EBAY' OR Symbol = 'SWY' ;
```

What do you expect the output of this query to be? Run the modified query and verify your expectations.

2. Modify the query to filter for each of the following combinations. Run the project after each modification and note the results. (For more information about operators, see "Operators" in the *Sybase CEP CCL Reference Guide*.)
  - Rows with "MRK" in the Symbol column, Volume greater than 400, and Price less than 34.
  - Rows with "EBAY" or "MRK" in the Symbol column and Volume greater than 1000 but less than 3000.
  - Rows with the product of Price and Volume no more than 3000.

### Next

#### Questions

Answer the following questions to reinforce the information presented in this tutorial. See *Solutions* on page 25 for answers to these questions.

1. What is the purpose of Sybase CEP Environments and Workspaces? What is one significant difference between the two?
2. What Sybase CEP component translates data from an external format to one that Sybase CEP Engine can process (or vice versa)? How do you associate such a component with a stream?

## Wrap-up

---

Summary of how to filter data with Sybase CEP Engine.

Filtering is one of the most common tasks people perform with Sybase CEP Engine. In this tutorial, you filtered stock trade data based on combinations of stock symbol, volume, and price. Filtering is critical in a variety of applications. For example, you might filter sensor readings at a manufacturing plant for a particular RFID tag, or watch for visits to your Web site from a particular IP address.

As you built your project with Sybase CEP Studio, you explored workspaces, data streams, adapters, and simple Continuous Computation Language statements:

Now you are ready to explore other common tasks you can accomplish with Sybase CEP Engine, starting with maintaining state.

### *Further Reading*

For more information about topics covered in this tutorial, see:

- "Sybase CEP Studio" in the *Sybase CEP Studio Guide* .
- "Operators" and "Query Statements" in the *Sybase CEP CCL Reference Guide* .
- "Sybase CEP Adapters" in the *Sybase CEP Integration Guide* .

## **Solutions**

---

Solutions to modifying the query exercise.

### **Modifying the Query: Solution**

1. The modified query filters for rows with either "EBAY" or "SWY" in the Symbol column.
2. The following WHERE clauses produce the requested output:
  - WHERE Symbol = "MRK" AND Volume > 400 AND Price < 34
  - WHERE Symbol = "MRK" OR Symbol = "EBAY" AND Volume > 1000 AND Volume < 3000
  - WHERE Price \* Volume <= 3000

### *Questions*

1. Sybase CEP Workspaces are grouping mechanisms you can use to organize your projects, while Sybase CEP Environments are only available in Sybase CEP Studio, and allow you to view and work with different subsets of your workspaces.
2. An input adapter translates data from an external format to one that Sybase CEP Engine can process, while an output adapter translates data from Sybase CEP Engine to an external format. You attach an adapter to a stream.



# Maintaining State

Sybase CEP Engine will retain rows in order to perform calculations or comparisons across these rows.

## Getting Started: Maintaining State

---

Maintaining state enables you to perform aggregate calculations. There are four ways to maintain the state in the Sybase CEP Engine.

### Maintaining State: Overview

In the *Filtering Data* on page 1 tutorial, you worked with queries that examined rows individually. In those queries, rows arrive, are processed, and are gone. Most applications require that Sybase CEP Engine maintain state, that is, retain multiple rows in order to perform calculations or comparisons across those rows. In this tutorial, you will work with a query that calculates the average sales price over several transactions for specific stock symbols. Maintaining state and performing aggregate calculations are critical tasks for a variety of applications, such as calculating the average temperature over several thermometer readings, determining the total spending per visitor to a Web site over a given time period, or counting the number of times an object passes the same sensor in a manufacturing plant.

#### Objectives

After completing this tutorial, you will be able to:

- Compose CCL code to retain multiple rows.
- Describe four different ways that you can maintain state with Sybase CEP Engine.
- List the CCL components that calculate averages, sums, and counts.

#### Before You Begin

Before you begin the activities in this tutorial, be sure you have access to the following:

- An instance of Sybase CEP Server running on your computer. If you don't have such an instance, follow the directions in *Installing and Starting Sybase CEP Server*. Note that attempting to complete the tutorial while connected to a remote instance of Sybase CEP Server will not work because of the data files used in the tutorial.
- Sybase CEP Studio installed on your computer. If you have not installed Sybase CEP Studio, follow the directions in *Installing Sybase CEP Studio*.
- The completed StockFilter project from the *Filtering Data* on page 1 tutorial.

## Maintaining State

---

Perform calculations or comparisons across rows. Replace the CCL statements that filter rows with statements that create a window.

A Sybase CEP window maintains rows in memory to allow you to perform calculations or comparisons across those rows.

In this activity, you will be modifying the StockFilter project you created in the *Filtering Data* on page 1 tutorial:

1. Open the StockFilter project. If you are continuing immediately after completing that tutorial, you can skip these steps:
  - a) Start Sybase CEP Studio.
  - b) On the File menu, click Load Project.
  - c) Navigate to the directory where you stored the StockFilter project for the *Filtering Data* on page 1 tutorial and then open the file StockFilter.ccp. You can also find a copy of the completed project in SybaseC8Repository/version/examples/GettingStarted/CompletedProjects/AfterFilteringData.
2. Replace the query statement in the Queries tab beginning with "INSERT" with the following text:

```
-- Create a window
CREATE WINDOW FilteredTradesWindow
SCHEMA (Symbol STRING, Price FLOAT,
        Volume INTEGER)
KEEP 3 ROWS;

-- Insert rows into the window
INSERT INTO FilteredTradesWindow
SELECT *
FROM StockTrades
WHERE Symbol = 'EBAY';
```

Note: You can also copy the text from the file MaintainingStateCCL.txt in the directory SybaseC8Repository/version/examples/GettingStarted/CCLFiles (where version is the version number of Sybase CEP Engine), between the lines "-- Maintaining State Code Block 1" and "-- Maintaining State Code Block 2."

The following points describe the CCL code:

- The first statement creates a window named FilteredTradesWindow with a schema identical to that of the StockTrades input stream. Note that you can define a schema directly in CCL, not just through the Sybase CEP user interface as you did for the data streams. The KEEP clause specifies that this window maintain three rows. This code block is a single CCL statement, which ends with a semi-colon.



- The second statement simply filters the input stream for rows with "EBAY" in the Symbol column and inserts those rows into FilteredTradesWindow.
  - Notice the italicized green text in the Queries tab. These are comments, each beginning with two hyphens (you can also begin a comment with two slashes or enclose comments between /\* and \*/).
3. Locate the drop-down list in the toolbar and change it from Release to Debug. This allows you to examine the contents of the window, which is not possible when Sybase CEP Studio is running in release mode.
  4. On the Debug menu, click View All Streams in Module 'StockFilter' to see the rows as they are published to the input stream (a viewer opens for the FilteredTrades output stream as well, but the query doesn't use that stream yet).
  5. On the Debug menu, click View All Named Windows in Module 'StockFilter' to view the contents of the window.
  6. Position the viewers in order to see both the StockTrades stream and the FilteredTradesWindow window.
  7. Click the Start button (the green arrowhead on the toolbar) to compile and run the project.
  8. Examine the stream viewer for the input stream (use the Disconnect All Viewers button, marked with two vertical bars, to halt the display of all viewers, and the Reconnect All Viewers button, which appears as a green arrowhead in each viewer window, to start the display again):

Timestamp	Symbol	Price	Volume
2009/06/24 09:09:42.080358	AAPL	35.1500000000...	300
2009/06/24 09:09:42.580358	SWY	21.5500000000...	200
2009/06/24 09:09:43.080358	JPM	36.3500000000...	500
2009/06/24 09:09:43.580358	FILE	6.4000000000...	400
2009/06/24 09:09:44.080358	MRK	34.0000000000...	100
2009/06/24 09:09:44.580358	SWY	21.8000000000...	100
2009/06/24 09:09:45.080358	MRK	34.1000000000...	3000
2009/06/24 09:09:45.580358	MRK	34.1000000000...	100
2009/06/24 09:09:46.080358	MRK	33.9000000000...	200
2009/06/24 09:09:46.580358	MRK	34.0500000000...	100
2009/06/24 09:09:47.080358	IBM	75.2500000000...	300
2009/06/24 09:09:47.580358	PFE	28.3000000000...	20000
2009/06/24 09:09:48.080358	EBAY	30.6000000000...	20000

Notice the rows that appear in red. Both the latest row arriving in the stream and a row matching the filter in the Query statement appear in red.

9. Now examine the window viewer:

Timestamp	Symbol	Price	Volume
2009/06/24 09:09:59.580358	EBAY	30.5000000000...	300
2009/06/24 09:10:00.580358	EBAY	29.9500000000...	1000
<del>2009/06/24 09:10:07.580358</del>	<del>EBAY</del>	<del>29.9500000000...</del>	<del>20000</del>
2009/06/24 09:10:07.580358	EBAY	30.6000000000...	400

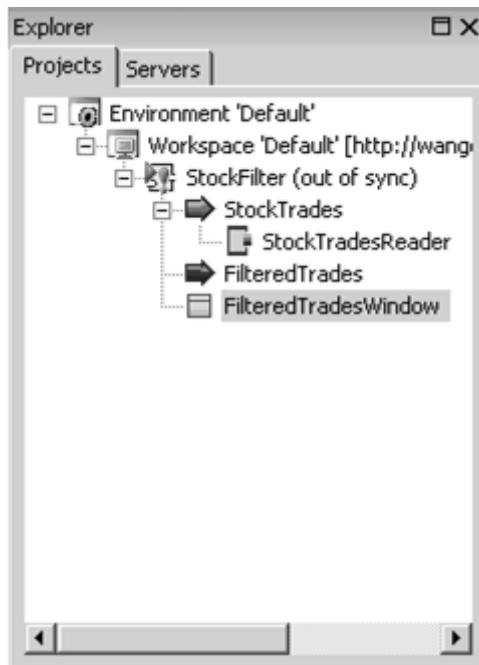
Rate: 0.54 msgs/sec      Window size: 3 msgs

In the window viewer, the latest row being added to the window is shown in red. The row being removed is shown in red with a line through it. Notice that the timestamp of the row being removed changes to reflect the time that it is removed.

This type of window is called a count-based window, because it maintains a specific number of rows. Once that number of rows is reached, a new row arriving on the stream displaces the oldest row in the window.

10. Make sure that the viewers are actively displaying rows and then click the Stop button to stop the project.

Notice that the window you created with CCL now appears in the Explorer view:



## Maintaining State per Column Value

---

Modify the count-based window to maintain the rows within the window.

The count-based window you just created maintains a total of three rows. Now modify the window to maintain three rows for each stock symbol.

Follow these steps to maintain rows for each stock symbol:

1. Insert "PER Symbol" before the semi-colon at the end of the Create Window statement and then remove the WHERE clause from the Query statement. The results should look like this:

```
-- Create a window
CREATE WINDOW FilteredTradesWindow
SCHEMA (Symbol STRING, Price FLOAT,
        Volume INTEGER)
KEEP 3 ROWS PER Symbol;

-- Insert rows into the window
INSERT INTO FilteredTradesWindow
SELECT *
FROM StockTrades;
```

Adding the PER clause specifies that the window maintain three rows for each unique value in the Symbol column, rather than three rows total. Note the green bar to the left of

the statement in the Queries tab, which is an indicator of the relationship between this CCL statement and the associated component in the Explorer view and the Flow view.

2. Start the project again, and then examine the window viewer:

Timestamp	Symbol	Price	Volume
2009/06/24 09:37:33.272727	SWY	21.7000000000...	1300
2009/06/24 09:37:33.772727	MW	37.0800000000...	300
2009/06/24 09:37:34.272727	EBAY	30.5000000000...	3000
2009/06/24 09:37:34.772727	AAPL	35.0000000000...	100
2009/06/24 09:37:35.272727	EBAY	29.9500000000...	200
2009/06/24 09:37:35.772727	MSFT	24.9700000000...	100
2009/06/24 09:37:36.272727	MSFT	24.9700000000...	100
2009/06/24 09:37:36.772727	JPM	36.2500000000...	200
2009/06/24 09:37:37.272727	JPM	36.3500000000...	1000
2009/06/24 09:37:37.772727	FILE	6.4000000000...	450
2009/06/24 09:37:38.772727	SWY	21.8000000000...	100
2009/06/24 09:37:39.772727	MRK	34.1000000000...	400
2009/06/24 09:37:40.272727	MRK	33.9000000000...	500
2009/06/24 09:37:40.772727	MRK	34.1000000000...	100
2009/06/24 09:37:40.772727	MRK	34.0500000000...	3000

Rate: 3.11 msgs/sec      Window size: 27 msgs

Notice that rows entering and leaving the window are marked just as before.

3. Click the Disconnect All Viewers button to examine the rows more carefully. Scroll up and down in the viewer to see that Sybase CEP Engine removes a row when the window reaches the limit of three for a specific value of the Symbol column.
4. Reconnect the viewers and then stop the project.

## Aggregating Values

Perform a common action on multiple rows within a window.

You create a window because you want to perform some action on multiple rows. A common action is to calculate an aggregated value for a set of rows. In this activity, you'll calculate the average sales price over the contents of the window for each stock symbol.

Follow these steps to calculate the average sales price:

1. Replace the two statements beginning with "-- Create a window" in the Queries tab with the following text (or copy the text from MaintainingStateCCL.txt between the lines "-- Maintaining State Code Block 2" and "-- Maintaining State Code Block 3"):

```
-- Create a window
CREATE WINDOW FilteredTradesWindow
SCHEMA (Symbol STRING, Price FLOAT,
        Volume INTEGER)
KEEP 3 ROWS PER Symbol;
```

```

-- Insert rows into the window
INSERT INTO FilteredTradesWindow
SELECT *
FROM StockTrades;

-- Calculate an average and insert into the output stream
INSERT INTO FilteredTrades
SELECT *, AVG(Price)
FROM FilteredTradesWindow
WHERE Symbol = 'EBAY' OR Symbol = 'IBM'
GROUP BY Symbol;

```

The first two statements are exactly the same as before. The third statement sends rows to an output stream by connecting the output of one query to the input of another. Here's a brief explanation of the new code:

- The new statement takes all of the columns from the row arriving from the input stream and the calculated average of the values in the Price column of the rows in the window (using one of the Sybase CEP predefined aggregate functions) and sends the resulting row to the output stream FilteredTrades.
- The WHERE clause limits the output to rows with either "EBAY" or "IBM" in the Symbol column.
- The GROUP BY clause causes the aggregate function to calculate the average for groups of rows in the window based on the value of the Symbol column. Without this clause, the aggregate function would perform the calculation over all rows in the window.

The output stream must have a new column to receive the average price.

2. Modify the schema defined in the CREATE OUTPUT STREAM statement by adding a comma after "INTEGER" followed by "AvgPrice FLOAT". The statement should look like this:

```

CREATE OUTPUT STREAM FilteredTrades
SCHEMA (
    Symbol STRING,
    Price FLOAT,
    Volume INTEGER,
    AvgPrice FLOAT
);

```

3. Run the project again and examine the viewer for the output stream:

Timestamp	Symbol	Price	Volume	AvgPrice
2009/06/24 09:55:16.312811	EBAY	29.9500000000...	200	30.2966666666...
2009/06/24 09:55:22.312811	IBM	75.2500000000...	1100	75.1600000000...
2009/06/24 09:55:23.312811	EBAY	30.6000000000...	100	30.3500000000...
2009/06/24 09:55:28.812811	IBM	75.2500000000...	300	75.2166666666...
2009/06/24 09:55:29.812811	EBAY	30.6000000000...	20000	30.3833333333...
2009/06/24 09:55:31.312811	EBAY	30.4400000000...	400	30.5466666666...
2009/06/24 09:55:32.312811	IBM	75.0800000000...	9800	75.1933333333...
2009/06/24 09:55:32.812811	IBM	75.1500000000...	2500	75.1600000000...
2009/06/24 09:55:34.312811	EBAY	30.5000000000...	300	30.5133333333...
2009/06/24 09:55:35.312811	EBAY	29.9500000000...	20000	30.2966666666...
2009/06/24 09:55:39.312811	IBM	75.0800000000...	250	75.1033333333...
2009/06/24 09:55:39.812811	IBM	75.1500000000...	2500	75.1266666666...
2009/06/24 09:55:41.312811	EBAY	30.5000000000...	300	30.3166666666...

Notice that Sybase CEP Engine publishes a row to the output stream as soon as a row meeting the filter condition arrives in the window. A row arriving from the input stream triggers both the aggregation calculation and the output row.

4. Stop the project.

## Exercises

Complete the following exercises to explore additional aggregate functions. See *Solutions* on page 40 for possible solutions to these exercises.

1. Replace "AVG(Price)" in the query with "SUM(Volume)". Start the project and view the results. What other change would you make in a real project to accommodate the new calculation?
2. Modify the query to calculate the volume-weighted average price (defined as the sum of [Volume times Price], divided by the sum of Volume) over the contents of the window.
3. Modify the query to determine the maximum price of the last ten transactions per symbol. See *Sybase CEP Functions* in the *Sybase CEP CCL Reference Guide e* for more information about available aggregate functions.

## Consolidating the Code

Create window statement and a single query statement.

At this point, the query module uses one Create Window statement and two Query statements. You can achieve the same result with a single Query statement.

1. Replace the last three statements in the Queries tab, beginning with "-- Create a window", with the following text (or copy the text from MaintainingStateCCL.txt between the lines "-- Maintaining State Code Block 3" and "-- Maintaining State Code Block 4"):

```
-- Create an unnamed window to track average price by stock symbol
INSERT INTO FilteredTrades
SELECT *, AVG(Price)
FROM StockTrades
KEEP 3 ROWS
WHERE Symbol = 'EBAY' OR Symbol = 'IBM'
GROUP BY Symbol;
```

This single statement produces the same results as the three statements introduced in *Aggregating Values* on page 32. In this case, the KEEP clause by itself creates an implicit or unnamed window (as opposed to the named window you defined before). When you use the GROUP BY clause with an unnamed window, it groups the aggregate calculations and also groups the contents of the window, just like the PER clause. In this case, the window behaves as if you had specified "KEEP 3 ROWS PER Symbol".

2. Run the project and verify that the results are as expected.

Note that the WHERE clause in this query restricts the output and aggregate calculation, not the contents of the unnamed window - the window holds three rows for every value of the Symbol column, not just for EBAY and IBM.

You cannot view unnamed windows with Sybase CEP Studio.

3. Stop the project.

## Exploring Window Types

---

Use different window types when maintaining a specific number of rows situations.

Maintaining a specific number of rows is useful in many situations, but sometimes you may want to keep all rows for a certain amount of time.

1. Edit the query to replace "KEEP 3 ROWS" with "KEEP 2 SECONDS":

```
-- Create an unnamed window to track average price by stock symbol
INSERT INTO FilteredTrades
SELECT *, AVG(Price)
FROM StockTrades
KEEP 2 SECONDS
WHERE Symbol = 'EBAY' OR Symbol = 'IBM'
GROUP BY Symbol;
```

Now the window maintains a variable number of rows, keeping each row for two seconds. The part of the KEEP clause that specifies how rows are maintained is called the window policy.

2. Run the project and examine the output stream:

Timestamp	Symbol	Price	Volume	AvgPrice
2009/06/24 10:18:37.381928	IBM	75.2500000000...	20000	75.2500000000...
2009/06/24 10:18:38.381928	EBAY	30.6000000000...	20000	30.6000000000...
2009/06/24 10:18:39.381928	IBM	75.2500000000...	20000	Null
2009/06/24 10:18:39.881928	EBAY	30.4400000000...	2200	30.5200000000...
2009/06/24 10:18:40.381928	EBAY	30.6000000000...	20000	30.4400000000...
2009/06/24 10:18:40.881928	IBM	75.0800000000...	100	75.0800000000...
2009/06/24 10:18:41.381928	IBM	75.1500000000...	1300	75.1150000000...
2009/06/24 10:18:41.881928	EBAY	30.4400000000...	2200	Null
2009/06/24 10:18:42.881928	IBM	75.0800000000...	100	75.1500000000...
2009/06/24 10:18:42.881928	EBAY	30.5000000000...	3000	30.5000000000...
2009/06/24 10:18:43.381928	IBM	75.1500000000...	1300	Null
2009/06/24 10:18:43.881928	EBAY	29.9500000000...	200	30.2250000000...
2009/06/24 10:18:44.881928	EBAY	30.5000000000...	3000	29.9500000000...
2009/06/24 10:18:45.881928	EBAY	29.9500000000...	200	Null

Notice that the AvgPrice column sometimes contains Null. The query produces an output row (and therefore calculates the average) not only when a row arrives on the input stream, but also when a row is removed from the window, because either event affects the value of any aggregate functions. If the sub-window for a specific Symbol value is empty when a row is removed from the window (which happens when a row arrives with a particular Symbol value but no additional row with that value arrives in the next two seconds), the calculation results in Null.

This type of window is called a time-based window. It is also called a sliding window, because Sybase CEP Engine continuously removes rows from the window (based on the policy) as new rows arrive.

3. Stop the project.
4. Insert the word "EVERY" after the word "KEEP" in the query:

```
-- Create an unnamed window to track average price by stock symbol
INSERT INTO FilteredTrades
SELECT *, AVG(Price)
FROM StockTrades
KEEP EVERY 2 SECONDS
WHERE Symbol = 'EBAY' OR Symbol = 'IBM'
GROUP BY Symbol;
```

Adding EVERY to the KEEP clause specifies that the window maintain all rows that arrive during the specified interval. At the end of each interval, the window empties.

5. Run the project and observe the output stream:



FilteredTrades (Default/StockFilter/FilteredTrades)

Filter Expression:

Timestamp	Symbol	Price	Volume	AvgPrice
2009/06/24 10:20:54.631005	EBAY	30.5000000000...	1000	30.5000000000...
2009/06/24 10:20:55.631005	EBAY	29.9500000000...	500	30.2250000000...
2009/06/24 10:21:01.631005	IBM	75.2500000000...	20000	75.2500000000...
2009/06/24 10:21:02.631005	EBAY	30.6000000000...	20000	30.6000000000...
2009/06/24 10:21:04.131005	EBAY	30.4400000000...	2200	30.4400000000...
2009/06/24 10:21:05.131005	IBM	75.0800000000...	100	75.0800000000...
2009/06/24 10:21:05.631005	IBM	75.1500000000...	1300	75.1150000000...
2009/06/24 10:21:07.131005	EBAY	30.5000000000...	3000	30.5000000000...
2009/06/24 10:21:08.131005	EBAY	29.9500000000...	200	29.9500000000...
2009/06/24 10:21:14.131005	IBM	75.2500000000...	1100	75.2500000000...
2009/06/24 10:21:15.131005	EBAY	30.6000000000...	100	30.6000000000...
2009/06/24 10:21:20.631005	IBM	75.2500000000...	300	75.2500000000...
2009/06/24 10:21:21.631005	EBAY	30.6000000000...	20000	30.6000000000...
2009/06/24 10:21:23.131005	EBAY	30.4400000000...	400	30.4400000000...

Show only:

Rate: 0.54 msgs/sec      Total: 17 msgs

Now the window doesn't remove individual rows based on age, but instead removes all rows at the end of each two-second interval (the interval timing is based on the epoch rather than the timestamp of the first row in the stream). This is called a jumping window, as opposed to sliding.

Because of this, the query produces output when a row arrives from the input stream that meets the filter condition in the `WHERE` clause, but not when rows are removed from the window when the interval expires. If no rows arrive during a particular interval, the query produces no output. The calculation never results in `NULL` because it is never performed when a group is empty.

- Stop the project.

## Exercise

Complete the following exercise to explore an additional type of window. See *Solutions* on page 40 for possible solutions to this exercise.

- Modify the `KEEP` clause to read `"KEEP EVERY 4 ROWS"`. Run the query and observe the results. What do you think this type of window is called?

## Maintaining Rows Based on Rank

Windows are also defined based on the rank of a value of a column.

So far, all of the windows you've defined have retained the newest rows (rows that arrived most recently), whether the window was count-based or time-based. You can also define a window policy that retains rows based on the rank of a value of a column, so that the window doesn't necessarily retain the most recent rows.

## Maintaining State

Follow these steps to create a window that maintains rows with the largest value in a specific column:

1. Replace the last statement in the Queries tab with the following text (or copy the text from MaintainingStateCCL.txt between the line "-- Maintaining State Code Block 4" and the end of the file):

```
-- Create an unnamed window to track the sum of
-- the two largest volumes by stock symbol
INSERT INTO FilteredTrades
SELECT *, SUM(Volume)
FROM StockTrades
KEEP 2 LARGEST ROWS BY Volume
WHERE Symbol = 'EBAY'
GROUP BY Symbol;
```

The SELECT clause in this statement includes an aggregate function to calculate the sum of the value in the Volume column for all rows in the implicit window. Note that this calculation appears in the output stream in the column named AvgPrice.

The KEEP clause specifies that the window maintain the two rows with the largest values in the Volume column (the GROUP BY clause specifies that the window keep two rows for each unique value in the Symbol column). When a row arrives on the input stream, Sybase CEP Engine compares the value in the Volume column to the values in the same column of the window for rows with a matching value in the Symbol column. If the value in the new row is higher than either of the values in the window, the new row is added to the window, and the row with the lowest value in the Volume column is removed.

The WHERE clause restricts the output to rows with "EBAY" in the Symbol column.

2. Make sure that viewers are open and visible for both the input and output stream, and then run the project.
3. Observe the output stream:

FilteredTrades (Default/StockFilter/FilteredTrades)

Filter Expression:

Timestamp	Symbol	Price	Volume	AvgPrice
2009/06/24 10:24:53.832579	EBAY	30.4400000000...	100	100.000000000...
2009/06/24 10:24:56.832579	EBAY	30.5000000000...	1000	1100.000000000...
2009/06/24 10:24:57.832579	EBAY	29.9500000000...	500	1500.000000000...
2009/06/24 10:25:04.832579	EBAY	30.6000000000...	20000	21000.000000000...
2009/06/24 10:25:06.332579	EBAY	30.4400000000...	2200	22200.000000000...
2009/06/24 10:25:09.332579	EBAY	30.5000000000...	3000	23000.000000000...
2009/06/24 10:25:10.332579	EBAY	29.9500000000...	200	23000.000000000...
2009/06/24 10:25:17.332579	EBAY	30.6000000000...	100	23000.000000000...

Show only:

Rate: 0.32 msgs/sec      Total: 8 msgs

If you examine the value in the AvgPrice column (which is actually the sum of the values in the Volume column in the window), you see that the value never decreases. Remember that this viewer displays the rows as they flow through the output stream, not rows in the window. Sybase CEP Engine produces an output row and calculates the aggregate sum whenever a row arrives on the input stream that matches the filter condition ("EBAY" in the Symbol column). The output row contains the aggregate calculation as well as columns from the input row, not from a row in the window. Sybase CEP Engine calculates the aggregate across the contents of the window, so the sum is based on the two largest values that have arrived since the project started, and does not necessarily include the value in the Volume column of the current row.

4. Stop the project.

## Exercises

Complete the following exercises to explore additional window capabilities. See *Solutions* on page 40 for possible solutions to these exercises.

1. Modify the KEEP clause to read "KEEP UNTIL 'time'", replacing time with a time that is a minute or two in the future, in the form hour:minute (using a 24-hour clock). The quotes around the time specification are required. Run the query and observe the results. What do you think KEEP UNTIL does?
2. Modify the project to output the average of the three lowest prices per symbol. See the "KEEP Clause" section of the Sybase CEP *CCL Reference Guide* for more information.

## Wrap-up

---

Summary of maintaining rows in a memory containing data.

In this tutorial, you defined count-based and time-based windows, both sliding and jumping, to maintain rows in memory containing stock trade information based on a variety of window policies. You also performed calculations across the rows in the window, including sums and averages, and explored the difference between a named and an unnamed window.

You can use these techniques in a broad range of applications, such as counting the number of users who click each link on a specific Web site page, or tracking the applications using the most memory on a particular server, or validating travel time of a particular part through the manufacturing process.

The next tutorial in the series allows you to correlate data from multiple sources.

### *Further Reading*

For more information about topics covered in this tutorial, see *Create Window Statement* and *Sybase CEP Functions* in the *Sybase CEP CCL Reference Guide*.

## Solutions

---

Solutions to maintaining state exercise.

### Aggregating Values

#### **remove sub-heading**

1. You should also change the schema of FilteredTrades, replacing AvgPrice with a column named something like TotalVolume of type Integer.
2. Change the SELECT clause in the third code block to this:

```
SELECT *, SUM(Volume * Price) / SUM(Volume)
```

3. Replace 2 with 10 in the KEEP statement of the first code block, and change the SELECT clause in the third code block to this:

```
SELECT *, MAX(Price)
```

## Exploring Window Types

### *Exercise*

- A window that maintains rows until a specific number is reached and then empties is called a jumping count-based window.

## Maintaining Rows Based on Value

### remove sub-heading

1. Code similar to the following clears the window at the specified time:

```
INSERT INTO FilteredTrades
SELECT *, SUM(Volume)
FROM StockTrades
KEEP UNTIL '13:08'
WHERE Symbol = 'EBAY'
GROUP BY Symbol;
```

KEEP UNTIL retains rows in the window until the specified time, and then empties the window. In this example, the value in the AvgPrice column increases for each row until the specified time, then drops to the value of the column in the first row arriving after the specified time and begins increasing again. If you specify a general time, the window empties at the same time every day. If you identify the day of week, it empties at the same time of the same day every week, and so on. You can also specify a list of times. See the "KEEP Clause" section of the Sybase CEP *CCL Reference Guide* for more information.

2. The following CCL code performs the requested task, only producing output for rows with "EBAY" in the Symbol column:

```
INSERT INTO FilteredTrades
SELECT *, AVG(Price)
FROM StockTrades
KEEP 3 SMALLEST ROWS BY Price
WHERE Symbol = 'EBAY'
GROUP BY Symbol;
```



# Joins

Used to combine data from multiple sources.

In earlier tutorials, you worked with queries that process data from a single data stream or window. In many situations, however, you may need to correlate information from multiple data sources with a single CCL statement. For example, if you are writing an application that works with weather patterns, you might need to create a query to correlate temperature readings with wind data and to publish the combined results to a single destination. In this tutorial, you will work with queries that combine requests for information about a particular stock with information about recent stock trades.

## **Combining Data from Multiple Sources**

This section provides a tutorial on combining data from multiple sources.

After completing this tutorial, you will be able to:

- Write a Query statement that combines information in a data stream with information in a window.
- Describe the equivalent syntax choices available for combining stream and window data.
- List and compare three different ways of combining stream and window data.

## **Before Beginning the Combining Data Tutorial**

Before you begin the activities in this tutorial, be sure you have access to the following:

- An instance of Sybase CEP Server running on your computer. If you don't have such an instance, follow the directions in "Installing Sybase CEP Engine" in the Sybase CEP *Installation Guide* . Note that attempting to complete the tutorial while connected to a remote instance of Sybase CEP Server will not work because of the data files used in the tutorial.
- Sybase CEP Studio installed on your computer. If you have not installed Sybase CEP Studio, follow the directions in "Installing Sybase CEP Studio" for your platform in the *Sybase CEP Installation Guide* .
- The completed Sybase CEP project from the *State Persistence* on page 27 tutorial.

## **Joining a Stream to a Window**

In this activity you will be combining a stream that represents a request for information with a window containing trade information in order to provide the price and time of the most recent trade for the specified stock symbol.

You will be modifying the StockFilter project you worked with in the *State Persistence* on page 27 tutorial.

Follow these steps to open the StockFilter project and to remove the stream that you no longer need for this tutorial (if you are continuing immediately after completing the previous tutorial, you can skip the first three steps):

1. Start Sybase CEP Studio.
2. On the File menu, click Load Project.
3. Navigate to the directory where you stored the StockFilter project for the *State Persistence* on page 27 tutorial and then open StockFilter.ccp. You can also find a copy of the completed project in SybaseC8Repository/version/examples/GettingStarted/CompletedProjects/AfterMaintainingState.
4. Click FilteredTrades in the Explorer view.
5. On the Project menu, click Delete Stream and then click Yes in the dialog box to confirm the deletion. Note that the CCL statement creating the stream is removed from the Queries tab.

### **Creating a Join Query**

Follow these steps to create a query that uses data from two sources:

1. Replace the Query statement beginning with "-- Create an unnamed window" in the Queries tab with the following text:

```
-- Create a named window

CREATE WINDOW TradesWindow
SCHEMA (Symbol STRING, Price FLOAT, Volume INTEGER)
KEEP LAST ROW PER Symbol;

-- Create an input stream

CREATE INPUT STREAM TradeInquiries
SCHEMA (Symbol STRING);

-- Create an output stream

CREATE OUTPUT STREAM TradeResults
SCHEMA (Symbol STRING, Price FLOAT, TimeOfTrade TIMESTAMP);

-- Populate TradesWindow with rows from
-- the StockTrades stream

INSERT INTO TradesWindow
SELECT *
FROM StockTrades;
```

Note: You can also copy the text from the file CombiningSourcesCCL.txt in the directory SybaseC8Repository/version/examples/GettingStarted/CCLFiles (where version is the version number of Sybase CEP Engine), between the lines "-- Combining Sources Code Block 1" and "-- Combining Sources Code Block 2."



The first three statements create a window and two additional streams. Note the policy of the window: **KEEP LAST ROW** maintains the most recent row in the window, just like **KEEP 1 ROW**. The last statement populates `TradesWindow` with rows arriving in the `StrockTrades` stream.

2. Copy the following text and then append it to the end of the contents in the Queries tab (or copy the text from `CombiningSourcesCCL.txt` between the lines "-- Combining Sources Code Block 2" and "-- Combining Sources Code Block 3"):

```
-- Create the join

INSERT INTO TradeResults
SELECT TradeInquiries.Symbol,
       TradesWindow.Price,
       GETTIMESTAMP(TradesWindow)
FROM TradeInquiries JOIN TradesWindow
ON TradeInquiries.Symbol = TradesWindow.Symbol;
```

A query that combines data from more than one data source is called a join. This statement joins a data stream carrying values that represent stock ticker symbols to a window that keeps the latest price and volume information for each symbol it receives:

- The **SELECT** clause refers to columns in both data sources. It publishes the symbol from `TradeInquiries`, the price from `TradesWindow`, and the timestamp of the `TradesWindow` row (notice the `GETTIMESTAMP` function). Because both `TradeInquiries` and `TradesWindow` have a `Symbol` column, you must qualify the column reference with the name of the data source.
- The **FROM** clause lists the query's two data sources, `TradeInquiries` and `TradesWindow`, linked by the word **JOIN**.
- The **ON** clause establishes the join condition. In this case, the condition is that the value of the `Symbol` column in the `TradesWindow` row must match the value of the `Symbol` column in the incoming `TradeInquiries` row. Sybase CEP Engine combines the incoming `TradeInquiries` row with each `TradesWindow` row that meets the join condition and then publishes a separate row containing the columns specified in the **SELECT** clause for each combination. Sybase CEP Engine ignores rows in `TradesWindow` that do not meet the join condition.

### Next

In terms of the example, `TradesWindow` maintains information about the most recent trade for each stock. A row arriving on `TradeInquiries` is a request for information about the most recent trade of a specific stock. Rows published to `TradeResults` contain the symbol of the stock, the price of that stock for the most recent trade, and a timestamp of when the row representing that trade arrived in the window.

## Executing the Join

Examine the results of the join.

Follow these steps to run the project and then examine the results of the join:

1. Switch the project from Release mode to Debug mode, if it isn't in Debug mode already.
2. Click TradeInquiries in the Explorer view and then click the View Stream button (which looks like a magnifying glass) in the toolbar to open a stream viewer. Repeat these steps for TradeResults and TradesWindow and then position all the viewers so that they are visible.
3. Start the project.
4. Click TradeInquiries in the Explorer view, and then click Send Rows on the Debug menu. The Send Rows dialog box opens:

The screenshot shows a dialog box titled "Send Rows - Project StockFilter". It features the following elements:

- Input Stream:** A dropdown menu showing the path "ccl://wangg-xp3:6789/Stream/Default/StockFilter/TradeInquiries".
- Row Timestamp:** A text input field containing "2009/06/24 15:45:11.791302".
- Column Values:** A section containing a "Symbol:" label, an empty text input field, and the text "(String)".
- Mode Selection:** Two radio buttons at the bottom: "Single-row Mode" (which is selected) and "Multi-row Mode".
- Buttons:** "Clear" and "Send" buttons located at the bottom center.

In a real-world application, inquiries for a specific stock would probably arrive from some external source through an input adapter. In this activity, you will insert rows into the stream one at a time using the Sybase CEP Studio Send Rows command, which allows you to send rows into a stream immediately. Sending rows into TradeInquiries in this way will allow you to see the join results more clearly. The Send Rows dialog box includes the following elements:

- **Input Stream:** Shows the stream URL of the TradeInquiries data stream into which the row will be sent. The URL identifies the stream by the host name of the computer on which the project is running, the port that Sybase CEP Engine is using, and the stream's path within the project.
- **Row Timestamp:** Shows the timestamp for the row you are about to send. Since the TradeInquiries stream assigns row timestamps based on the current Sybase CEP Server time, this field displays the current Sybase CEP Server time and cannot be edited.

- Column Values: Shows all the columns for the stream you selected (in this case, TradeInquiries only has one column) and allows you to enter values into these columns and then send them as a row into the stream.
5. Enter "EBAY" into the Symbol box and then click Send to send a row with the value EBAY in the Symbol column into the TradeInquiries stream.
  6. Examine the stream and window viewers. (To see the results of the query more clearly, click the Disconnect All Viewers button in one of the viewers.)

The TradeInquiries stream viewer shows the row that you sent:

Timestamp	Symbol
2009/06/25 08:57:10.473706	EBAY

The TradeResults stream viewer shows the results of the join. Note that the Timestamp column shows the timestamp of the row in the output stream, while TimeOfTrade shows the timestamp of the row in TradesWindow that met the join condition:

Timestamp	Symbol	Price	TimeOfTrade
2009/06/25 08:57:10.473706	EBAY	29.9500000000...	2009/06/25 08:57:10.221783

The join will never produce more than one output row because the window policy only maintains a single row for each symbol, so only one row in the window can possibly match the row in the data stream. Notice that the query only publishes output when a row arrives in TradeInquiries, not when a row arrives in TradesWindow. A query containing a join cannot include more than one data stream, and the query executes only when a row arrives in that stream. Since data streams do not keep state, a row in a data stream exists only at the moment when Sybase CEP Engine processes it. Furthermore, Sybase CEP Engine processes rows sequentially. Even when two rows appear to have identical timestamps,

Sybase CEP Engine always processes one of the rows first and the other one second. Therefore, rows in two data streams cannot be joined to each other, because the two data streams never contain rows at exactly the same time. Likewise, stream-to-window joins cannot execute when a row arrives in the window, because the data stream to which the window is joined is always empty at the moment when the window receives the row.

7. Look in the TradesWindow window viewer for a row where the Symbol column has a value of EBAY and the Timestamp column has the same value as the TimeOfTrade column in the TradeResults stream viewer. This is the window row that was joined to the row you sent into the TradeInquiries stream.
8. Examine the CCL code in the Queries tab again and note this additional information:
  - A join created with a single JOIN in the FROM clause is called an inner join. This type of join does not publish any rows that do not meet the join condition.
  - The ON clause of an inner join can be any expression that evaluates to true or false.
9. Click the Reconnect All Viewers button to reconnect the stream and window viewers, and then click Send in the Send Rows dialog box to send another row to the input stream.
10. Examine the viewers to confirm that the results are as you expect and then stop the project.

## **Exploring Join Variations: Exercise**

Explore how the output of the join query changes when you change the join condition or alter other statements in your project.

### **Exercises**

Complete the following exercises to explore additional join variations. See *Solutions* on page 54 for possible solutions to these exercises.

1. Start the project and then send a row with a value of "YHOO" in the Symbol column into TradeInquiries. Examine the output in TradeResults. What results do you see? Why did the query produce these results? Stop the project.
2. In the KEEP clause, change "LAST ROW" to "3 ROWS". Start the project. Wait approximately 10 seconds to allow TradesWindow to accumulate rows and then send a row with a value of "EBAY" in the Symbol column into TradeInquiries. Examine the output in TradeResults. How many rows of output did the project produce? Why? Stop the project. In the KEEP clause, change "3 ROWS" back to "LAST ROW".
3. Delete the ON clause from the Query statement, but leave the semicolon at the end of the statement. Start the project and then send a row with a value of "IBM" in the Symbol column into TradeInquiries. Examine the output in TradeResults. How many rows did the project produce? Compare the values in the Price column of the output to the values in the Price column of TradesWindow. Why do you think the Query statement produced these results? Stop the project.
4. Insert the following text immediately before the semicolon at the end of the Query statement:

```
WHERE TradeInquiries.Symbol = TradesWindow.Symbol
```

Start the project and then send a row with a value of "EBAY" in the Symbol column into TradeInquiries. Examine the output in TradeResults. Can you detect any differences between the results of this query and the results of the earlier query that contained an ON clause instead of the WHERE clause? Stop the project.

## **Creating an inner join: Exercise**

Explore different syntax for an inner join.

In previous activities and exercises you worked with an inner join using the JOIN keyword. In this activity you will explore a different syntax for an inner join.

1. Replace the word JOIN in the join Query statement with a comma (,) and then eliminate extra spaces. The Query statement should now look like this:

```
-- Create the join

INSERT INTO TradeResults
SELECT TradeInquiries.Symbol,
       TradesWindow.Price,
       GETTIMESTAMP(TradesWindow)
FROM TradeInquiries, TradesWindow
WHERE TradeInquiries.Symbol = TradesWindow.Symbol;
```

2. Outside of Sybase CEP Studio, navigate to SybaseC8Repository/version/examples/GettingStarted/DataFiles (where version is the version number of Sybase CEP Engine) and then copy the file inquiries.csv to the directory data in the StockFilter project directory.
3. Add the following text to the end of the contents in the Queries tab (or copy the text from CombiningSourcesCCL.txt between the lines "-- Combining Sources Code Block 3" and "-- Combining Sources Code Block 4"):

```
-- Create an input adapter, which provides
-- data to the TradeInquiries stream

ATTACH INPUT ADAPTER TradeInquiriesReader
TYPE ReadFromCsvFileAdapterType
TO STREAM TradeInquiries
PROPERTIES
  FILENAME = "$ProjectFolder/data/inquiries.csv",
  LOOPCOUNT = "5",
  RATE = ".5",
  USECURRENTTIMESTAMP = "YES",
  TITLEROW = "YES",
  TIMESTAMPCOLUMN = "NO"
```

This statement attaches an input adapter to the TradeInquiries stream and sets some of the adapter's properties.

4. Run the project and then examine the output in TradeResults:

Timestamp	Symbol	Price	TimeOfTrade
2009/06/25 09:49:15.470660	JPM	36.3500000000...	2009/06/25 09:49:11.1707!
2009/06/25 09:49:30.970759	EBAY	29.9500000000...	2009/06/25 09:49:26.6707!
2009/06/25 09:49:30.970759	JPM	36.3500000000...	2009/06/25 09:49:28.6707!
2009/06/25 09:49:30.970759	MSFT	24.9700000000...	2009/06/25 09:49:27.6707!
2009/06/25 09:49:45.236475	EBAY	29.9500000000...	2009/06/25 09:49:44.6707!
2009/06/25 09:49:45.236475	JPM	36.3500000000...	2009/06/25 09:49:28.6707!
2009/06/25 09:49:45.236475	MSFT	24.9700000000...	2009/06/25 09:49:45.1707!
2009/06/25 09:49:45.236475	MRK	34.0500000000...	2009/06/25 09:49:38.1707!

Notice that the output is similar to the output you saw in the last exercise. Because of the Attach Adapter statement, multiple rows with different values in the Symbol column arrive on TradeInquiries, which in turn produces more output rows.

5. Examine the Query statement in the Queries tab again:

- This statement uses a comma instead of the word JOIN to create the inner join but produces the same results.
- A comma-separated join is always an inner join and always uses a WHERE clause to create the join condition instead of an ON clause.

6. Stop the project.

### Exercise

Complete the following exercises to reinforce the join syntax. See *Solutions* on page 54 for a possible solution to this exercise.

- Rewrite the following query to use comma-separated join syntax:

```
INSERT INTO FrequentWinners
SELECT PlayerScore.ID, PlayerScore.WinningNumber
FROM PlayerScore JOIN DailyRecord
ON PlayerScore.ID = DailyRecord.ID
   AND PlayerScore.Win = TRUE
   AND DailyRecord.DailyWins > 3;
```

## Creating an outer join: Exercise

How to create output joins even when the join condition isn't met.

So far, you have worked with inner joins - joins that produce output only when rows meet the join condition. For some applications you may want to produce output even if the join condition isn't met. For example, what if you want to know when an inquiry about a stock arrives even if it doesn't match any of the available trade information?

Follow these steps to produce output even when the join condition isn't met:

1. Replace the Query statement in the Queries tab with the following text (or copy the text from CombiningSourcesCCL.txt between the lines "-- Combining Sources Code Block 4" and "-- Combining Sources Code Block 5"):

```
-- Create the join

INSERT INTO TradesResults
SELECT Ti.Symbol,
       Tw.Price,
       GETTIMESTAMP(Tw)
FROM TradeInquiries AS Ti LEFT OUTER JOIN TradesWindow AS Tw
ON Ti.Symbol = Tw.Symbol;
```

The new FROM clause changes the inner join to a left outer join, which publishes a row for every row in the left data source (TradeInquiries) even if the join condition isn't met.

Notice that the FROM clause contains AS clauses, which define an alias for each of the data sources. This example defines an alias for every data source, but you can define aliases for just a subset, if you want. Every reference to the data sources in the rest of the statement now use the alias. Once you define an alias for a data source, you must refer to that data source with the alias throughout the statement.

2. Start the project and then examine the viewer for TradeInquiries:

Timestamp	Symbol
2009/01/27 11:19:48.923313	IBM
2009/01/27 11:19:50.923313	AAPL
2009/01/27 11:19:52.923313	YHOO
2009/01/27 11:19:54.923313	FILE
2009/01/27 11:19:56.923313	MSFT
2009/01/27 11:19:58.923313	MRK
2009/01/27 11:20:00.923313	GOOG
2009/01/27 11:20:02.923313	SWY
2009/01/27 11:20:04.923313	EBAY
2009/01/27 11:20:06.923313	JPM
2009/01/27 11:20:08.923313	IBM
2009/01/27 11:20:10.923313	AAPL
2009/01/27 11:20:12.923313	YHOO
2009/01/27 11:20:14.923313	FILE

3. Compare those rows to the output in TradeResults:

TradeInquiries (iwerner/StockFilter/TradeInquiries)

Filter Expression:

Timestamp	Symbol	Price	TimeOfTrade
2009/01/27 11:20:36.923313	MSFT	24.9700000000...	2009/01/27 11:20:32.423313
2009/01/27 11:20:38.923313	MRK	34.0500000000...	2009/01/27 11:20:36.923313
2009/01/27 11:20:40.923313	GOOG	Null	Null
2009/01/27 11:20:42.923313	SWY	21.7000000000...	2009/01/27 11:20:41.923313
2009/01/27 11:20:44.923313	EBAY	29.9500000000...	2009/01/27 11:20:43.923313
2009/01/27 11:20:46.923313	JPM	36.2500000000...	2009/01/27 11:20:45.423313
2009/01/27 11:20:48.923313	IBM	75.1500000000...	2009/01/27 11:20:46.423313
2009/01/27 11:20:50.923313	AAPL	35.0000000000...	2009/01/27 11:20:48.423313
2009/01/27 11:20:52.923313	VHQQ	Null	Null
2009/01/27 11:20:54.923313	FILE	6.4000000000...	2009/01/27 11:20:51.423313
2009/01/27 11:20:56.923313	MSFT	24.9700000000...	2009/01/27 11:20:56.923313
2009/01/27 11:20:58.923313	MRK	34.0000000000...	2009/01/27 11:20:51.923313
2009/01/27 11:21:00.923313	GOOG	Null	Null
2009/01/27 11:21:02.923313	SWY	21.7000000000...	2009/01/27 11:21:00.923313

Show only:

Rate: 0.00 msgs/sec      Total: 0 msgs

Notice that some of the output rows contain Nulls. When a row arrives in TradeInquiries for which there are no rows in TradesWindow that meet the join condition, the columns in the output row that would otherwise contain data from TradesWindow contain Nulls.

4. Examine the Query statement syntax again. The FROM clause of a left outer join between a stream and a window must list the stream on the left side of the join.
5. Stop the project.
6. Replace the Query statement in the Queries tab with the following text (or copy the text from CombiningSourcesCCL.txt between the line "-- Combining Sources Code Block 5" and the end of the file):

```
-- Create the join

INSERT INTO TradeResults
SELECT Ti.Symbol,
       Tw.Price,
       GETTIMESTAMP(Tw)
FROM TradeInquiries AS Ti LEFT OUTER JOIN TradesWindow AS Tw
ON Ti.Symbol = Tw.Symbol
WHERE Tw.Volume > 1000;
```

This variation simply adds a WHERE clause to filter the output.

7. Start the project and then compare the output to the previous results:



The screenshot shows a window titled "TradeResults (iwerner/StockFilter/TradeResults)". The address bar contains "cd://localhost:6789/Stream/iwerner/StockFilter/TradeResults". Below the address bar is a "Filter Expression:" field. The main area contains a table with the following data:

Timestamp	Symbol	Price	TimeOfTrade
2009/01/27 11:22:56.347485	MSFT	24.9700000000...	2009/01/27 11:22:53.347485
2009/01/27 11:22:58.347485	MRK	34.0500000000...	2009/01/27 11:22:57.847485
2009/01/27 11:23:04.347485	EBAY	30.5000000000...	2009/01/27 11:23:03.847485
2009/01/27 11:23:08.347485	IBM	75.1500000000...	2009/01/27 11:23:02.347485
2009/01/27 11:23:22.347485	SWY	21.7000000000...	2009/01/27 11:23:21.847485
2009/01/27 11:23:24.347485	EBAY	29.9500000000...	2009/01/27 11:23:23.847485
2009/01/27 11:23:26.347485	JPM	36.3500000000...	2009/01/27 11:23:25.847485
2009/01/27 11:23:28.347485	IBM	75.1500000000...	2009/01/27 11:23:28.347485

At the bottom of the window, there is a "Show only:" field, a "Rate: 0.00 msg/sec" indicator, and a "Total: 0 msg" indicator.

Notice that the output no longer contains Nulls:

- In a left outer join, the join condition in the ON clause is processed first. A left outer join requires an ON clause and uses a more restrictive form of the ON clause than an inner join: it must consist only of one or more equality comparisons, each of which compares a column in the left data source with a column in the right data source. If you use more than one equality comparison, you must separate them with the keyword AND.
- If your query includes a WHERE clause, as in this case, Sybase CEP Engine applies the filter to the results of the join. Because the filter in this WHERE clause is based on a column in TradesWindow, Sybase CEP Engine will only produce output rows when the join condition is met. Any rows from the join that don't include values from TradesWindow cannot pass the WHERE filter.

#### 8. Delete the query's WHERE clause and then stop the project.

### Exercises

Complete the following exercises to explore different types of joins. See *Solutions* on page 54 for possible solutions to these exercises.

#### 1. Identify the syntax errors in the following query:

```
INSERT INTO TradeResults
SELECT Ti.Symbol,
       Tw.Price,
       GETTIMESTAMP(Tw)
FROM TradesWindow AS Tw LEFT OUTER JOIN TradeInquiries AS Ti
ON Ti.Symbol = Tw.Symbol AND Tw.Price > 25
WHERE Tw.Volume > 1000;
```

Rewrite the query using correct syntax.

2. Change the FROM clause of the Query statement to "FROM TradesWindow AS Tw RIGHT OUTER JOIN TradeInquiries AS Ti". Start the project and then examine the output in TradeResults. Can you detect any differences in this output from the output of the left outer join? This query uses a right outer join. Based on what you know about left outer joins, provide a definition for a right outer join. Why does the FROM clause in a right outer join switch the order of the data sources compared to a left outer join? Stop the project.

## **Combining data tutorial: Conclusion**

Summary of the several kinds of inner and outer join queries.

In this tutorial, you worked with several kinds of inner and outer join queries, which combined rows from a data stream with rows representing stock trade information contained in a window. You also explored different kinds join conditions and data-filtering mechanisms. These join queries allowed you to determine the prices and times of sale for stocks, based on incoming inquiries about particular stock symbols. You can use joins in a wide variety of other applications, such as correlating login information in one data source with data about security alerts in another, or analyzing clickstream activity by joining a stream that delivers information about visits to a particular Web site with a window containing information about recent patterns of activity on the site.

The next tutorial in this series teaches you how to detect patterns of row arrival in one or more data streams.

### *Further Reading*

For more information about topics covered in this tutorial, see "FROM Clause: Join Syntax " and "ON Clause: Join Syntax" in the *Sybase CEP CCL Reference Guide* .

## **Solutions**

Solutions to the join exercises.

### **Exploring Join Variations: Solution**

1. The query doesn't publish any rows to TradeResults, because TradesWindow doesn't contain any rows with YHOO in the Symbol column.
2. The query published three rows of output to TradeResults, because TradesWindow contained three rows with EBAY in the Symbol column. The SELECT clause of the query processed the combination of the incoming TradeInquiries row with each of these three TradesWindow rows.
3. The output stream contains as many rows as are in TradesWindow when you send the row into TradeInquiries. Because the query has no ON clause join condition, Sybase CEP Engine paired the incoming TradeInquiries row with each row contained in TradesWindow, regardless of the values in the various columns.

In real-world applications, joins without a join condition are rare, as such queries tend to have limited usefulness and generate a large number of output rows.

- The output is the same as for a query using "ON TradeInquiries.Symbol = TradesWindow.Symbol". It is customary to use an ON clause in a query that uses a JOIN keyword in its FROM clause, but for an inner join you can use a WHERE clause to set the condition instead, with the same result.

### **Creating an inner join: Solution**

- Here is the query, rewritten to use comma-separated join syntax:

```
INSERT INTO FrequentWinners
SELECT PlayerScore.ID, PlayerScore.WinningNumber
FROM PlayerScore, DailyRecord
WHERE PlayerScore.Win = TRUE
      AND PlayerScore.ID = DailyRecord.ID
      AND DailyRecord.DailyWins > 3;
```

Notice that the SELECT clause of this query refers to only one of the query's data sources.

### **Creating an outer join: Solution**

- This query contains two errors:
  - The FROM clause creates a left outer join with a window on the left side of the join and a data stream on the right side. A left outer join between a data stream and a window must list the stream on the left.
  - The ON clause of an outer join can only include equality comparisons, so "Tw.Price > 25" is not valid.

The following syntax is valid:

```
INSERT INTO TradeResults
SELECT Ti.Symbol,
      Tw.Price,
      GETTIMESTAMP(Tw)
FROM TradeInquiries AS Ti LEFT OUTER JOIN TradesWindow AS Tw
ON Ti.Symbol = Tw.Symbol
WHERE Tw.Volume > 1000 AND Tw.Price > 25;
```

- This right outer join query produced identical results to the left outer join query you used in the first activity. The SELECT clause of a right outer join processes any row that arrives in the query's right data source, whether or not there are any rows in the left data source that meet the join condition. If no rows in the left data source meet the join condition, items that refer to columns in the left data source publish NULLs in the output row. Because a stream-to-window join executes only when a row arrives in the stream, a right outer join between a stream and a window must list the stream on the right.

## **Creating an additional join filter: Exercise**

In addition to the join condition in the ON clause, you can add further filters to a join query.

For example, say you want to publish only prices for trades of more than a 1000 shares.

**Exercise**

Add a **WHERE** clause to your join query that filters out all rows of less than 1000 from the `TradesWindow Volume` column. The **WHERE** clause comes after the **ON** clause and before the semi-colon (;) at the end of the statement. Restart the project and look at the output. What happens? Notice whether or not rows that do not match the join criteria are published. What does this tell you about the order in which the **WHERE** clause processes rows in relation to the **ON** clause and the **SELECT** clause? Stop the project.

## Predicting and Controlling Join Output

---

In the previous activity your join generated one row: the result of a combination of the incoming row in the stream and the one row in the window that met the join condition.

Let's see what happens if we send a row into `TradesInquiries` that has no corresponding trade record in `TradesWindow`.

1. Restart project.
2. Go back to the insert row box and enter the value `YHOO` in the box and click send.
3. Look at the viewers again. Notice that `TradeResults` does not have any output for this. Since the **ON** clause didn't find a match in the window for the incoming `YHOO` symbol, nothing was published.
4. Stop the project.

**Exercise**

Change the **KEEP** clause in the `Create Window` to `KEEP 3 ROWS PER Symbol`. If you manually send the same value into the stream as you did in the previous exercise, how many lines of output might it generate in the results stream? Restart the project and try sending the row into the stream. What rows in the window was it joined with? Stop the project and change the **KEEP** clause back to `KEEP LAST ROW PER Symbol` before proceeding.

## Seeing the Join in Action: Exercise

---

Now follow these steps to see this join you created in action

1. Locate the project from to **Debug**, if it is in **Release** mode by selecting the **Debug** option from the drop-down list, as described in .
2. Select your project and click the plus sign.
3. Click on `TradeInquiries` and click the viewer button, etc.
4. Similarly select and open viewers for `TradesWindow` and `TradeResults` and position all the viewers so they are clearly visible.
5. Start project

## Join Example

In the real world trade inquiries for a specific stock would probably arrive from some external source. In this example, you will use a debugging feature of Sybase CEP Studio to feed streams in manually one at a time. This will allow you to see the join results more clearly.

1. Open insert row box.
2. Select TradeInquiries stream in box. The dialog box now shows all the columns for the stream you selected and allows you to manually enter values into these columns (in this case only one column -- Symbol) and send them as a row into the stream. The dialog box also shows a **Row Timestamp** column. Since the **TradeInquiries** stream assigns Sybase CEP Server timestamps based on the current Sybase CEP Server time, this field displays the current Sybase CEP Server time and cannot be set manually.
3. Enter the **EBAY** in Symbol and click **Send** to send a row with the value **EBAY** in the **Symbol** column into the TradeInquiries stream.
4. Now look at your stream and window viewers and click on the pause button to stop the scrolling of rows and see query results better.

TradeInquiries viewer shows the row you manually sent to it.

The TradesWindow viewer shows the match in the window -- the most recent EBAY trade -- which is joined to the incoming row in TradeInquiries.

The TradeResults stream shows the output -- the latest price for EBAY and the time at which the trade was concluded.

5. Use the step forward and back buttons to look at the relationships between the incoming rows.
6. Look at the output again. Notice the join doesn't produce any output when nothing comes in in the stream. A join can have only one stream and executes only when a row arrives in the stream.
7. When a row does arrive in the stream, it is matched up separately with all rows in the window that meet the ON clause join condition -- in this case all rows that have the same value in the Symbol column as the incoming row in TradeInquiries. Each of these row combinations is then processed as a separate row by the SELECT clause and each generates its own row in the TradeResults stream. (When there is no ON condition, all combinations of the row arriving in the stream and each of the rows contained in the window are passed to the SELECT clause.)
8. Unpause the stream and window viewers.

Now let's see what happens if we send a row into TradesInquiries that has no corresponding trade record in TradesWindow.

1. Restart project.
2. Go back to the insert row box and enter the value YHOO in the box and click send.
3. Look at the viewers again. Notice that TradeResults does not have any output for this. The type of join you created is called an inner join. An inner join publishes only row combinations that meet the join condition in the ON clause (if there is an ON clause). In this case, that means that if a row arrives in the TradeInquiries stream and causes the query to execute, rows are only made available to the SELECT clause if the value in the Symbol column of the TradeInquiries stream has a corresponding value in the TradesWindow Symbol column. Since the ON clause didn't find a match in the window for the incoming YHOO symbol, nothing was published.
4. Stop the project.

# Event Pattern Matching

Event pattern matching occurs when a CCL application detects or matches event patterns

In the previous tutorials you worked with Query statements that took input rows and manipulated the contents of those rows to produce output rows. A different goal for a Sybase CEP application can be to recognize when rows arrive in a specific sequence or pattern. For example, frequent repeated connection attempts from the same IP address may indicate a network security issue. A product passing the RFID sensors at a manufacturing facility in the wrong order may be the result of a process failure. Under certain circumstances, brokers placing a stock order for themselves just before placing an order for their customer may be illegal. This tutorial uses the last example to demonstrate how to write a CCL application to detect or match event patterns.

## Event pattern matching tutorial overview

---

This section provides a tutorial on event pattern matching in the Sybase CEP Server.

After completing this tutorial, you will be able to:

- Define a Sybase CEP event.
- Write a Query statement that detects a pattern of events.
- Identify the Sybase CEP function that allows you to publish information about the events that match a pattern.

**Before you begin the activities in this tutorial, be sure you have access to the following:**

- An instance of Sybase CEP Server running on your computer. If you don't have such an instance, follow the directions in "Installing Sybase CEP Engine" in the Sybase CEP *Installation Guide*. Note that attempting to complete the tutorial while connected to a remote instance of Sybase CEP Server will not work because of the data files used in the tutorial.
- Sybase CEP Studio installed on your computer. If you have not installed Sybase CEP Studio, follow the directions in "Installing Sybase CEP Studio" for your platform in the *Sybase CEP Installation Guide*.

## Detecting a sequence of events: Exercise

---

Detect patterns of an arriving row.

With Sybase CEP Engine, an event is an arriving row. In this tutorial, you will use a project to monitor for fraud by detecting patterns of rows representing incoming stock trade requests from customers and the resulting orders placed by brokers. The first goal is to detect "front

running," defined as a broker placing an order for him or her self for the same stock as a customer, immediately before placing the order for the customer.

Follow these steps to open a project that searches for front running:

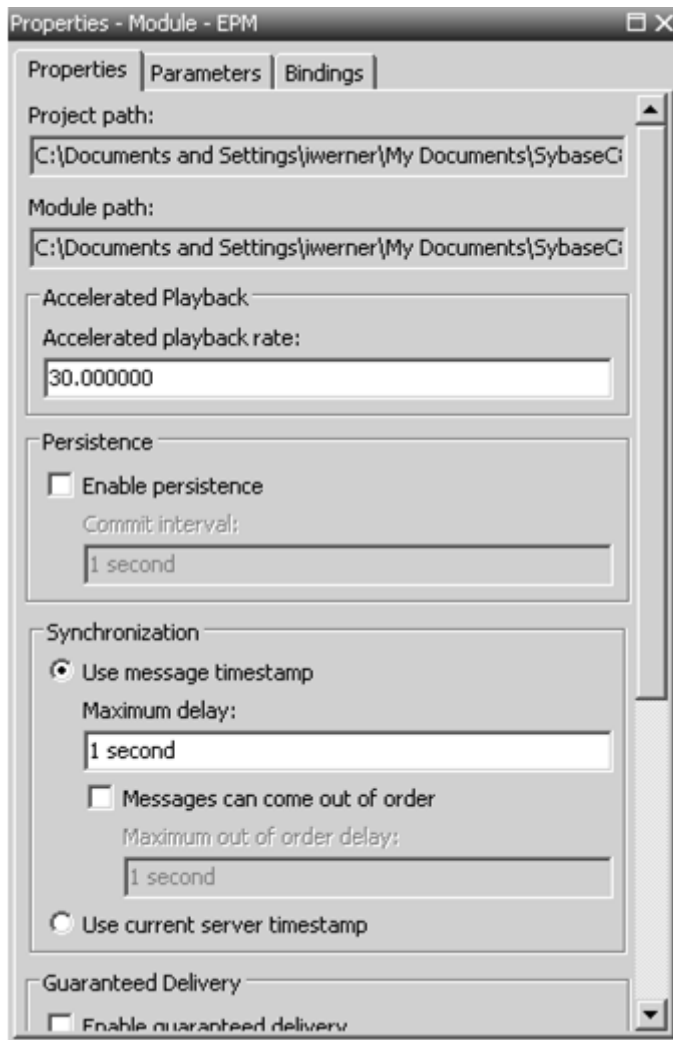
1. Outside of Sybase CEP Studio, locate the directory SybaseC8Repository/version/examples/GettingStarted/Projects (where version is the version number of Sybase CEP Engine). Copy the directory EventPatternMatching into another location under SybaseC8Repository/version.
2. Start Sybase CEP Studio.
3. If the Explorer view shows any projects loaded into the workspace, unload them by clicking each in turn and then clicking Unload Project on the File menu (this is just to keep things uncluttered; other projects in Sybase CEP Studio won't interfere with the new project).
4. Click Load Project on the File menu. Navigate to your copy of EventPatternMatching and then open the project file.

### **Examine and Run the Project**

Follow these steps to examine and run the project:

1. Use the Explorer view and the Properties view to examine the project components:
  - The first statement defines a schema named OrdersSchema.
  - The input streams StreamPhoneOrders, StreamPlacedOrders, and StreamWebOrders all use OrdersSchema and are each attached to a ReadFromCSVFile adapter. These streams carry stock trade information for an imaginary brokerage.
  - The properties for each input adapter include the specification File Has Timestamp Column and a Timestamp Column Format string. For this project, Sybase CEP Engine reads the timestamp for each row from the file rather than setting the timestamp to the current time. As a result, the Synchronization properties for the project as a whole specify Use message timestamp rather than Use current server timestamp.
  - The output stream StreamFraudAlerts carries rows identifying fraudulent orders.
2. Click the project name in the Explorer view and then enter "30" into the Accelerated playback rate box on the Properties tab.:





When a project uses message timestamp, Sybase CEP Engine processes rows according to the timestamp contained in each row. For example, if the timestamps in two rows are 12 minutes apart, Sybase CEP Engine processes the second row 12 minutes after processing the first row. Accelerated playback speeds up this processing by the specified multiplier. For this project, the CSV files contain about 30 minutes of data, so accelerating the playback 30 times means it will take Sybase CEP Engine approximately one minute to process all of the rows. Accelerated playback can be useful when you're analyzing historical data when you're testing a project in development before deploying to production.

3. Copy the following text into the Queries tab below the other statements (or copy the text from PatternMatchingCCL.txt between the lines "-- Pattern Matching Code Block 1" and "-- Pattern Matching Code Block 2"):

```
-- Match a pattern indicating front running:
--   Row with customer order
--   Row with placed order for broker
--   Row with placed order for customer

INSERT INTO StreamFraudAlerts
SELECT "Front Running", Broker.OrderId,
      Broker.Symbol, Broker.Customer
FROM StreamPhoneOrders Phone, StreamPlacedOrders Broker,
      StreamPlacedOrders Cust
MATCHING [10 SECONDS: Phone, Broker, Cust]
ON Phone.Broker = Broker.Customer = Cust.Broker
  AND Phone.Symbol = Broker.Symbol = Cust.Symbol
WHERE Phone.OrderId = Cust.OrderId;
```

This Query statement includes the following clauses:

- The SELECT clause specifies a literal string to be inserted into each output row.
- The FROM clause joins three data sources from two input streams. Note that the clause assigns an alias to each data source without using the word AS.
- The MATCHING clause looks for a row arriving on Phone, Broker, and Cust, in that order, within 10 seconds. Broker and Cust are two different aliases for the same input stream, so the sequence being matched is a row arriving on StreamPhoneOrders followed by two rows arriving on StreamPlacedOrders within 10 seconds.
- The ON clause defines two conditions for the join, specifying that the values in the indicated columns be equal. When you define an ON clause with a MATCHING clause, you must define one or more equalities across all the data sources specified in the FROM clause.
- The WHERE clause places a further condition on the join, limiting the search to a sequence when the values in the OrderId column of Phone and Cust are equal.

As a result, the query detects the following pattern of events:

- A row arrives on StreamPhoneOrders.
  - A row arrives on StreamPlacedOrders with the same value in the Symbol column and with a value in the customer ID column (Broker.Customer) that matches the value in the broker ID column in the first row (Phone.Broker). This indicates that the broker has placed an order for him or her self for the same stock that the customer ordered.
  - A row arrives on StreamPlacedOrders with the same value in the Symbol column as the first two rows, the same value in the broker ID column as the first row and the customer ID column in the second row, and with the same value in the OrderId column as the first row. This is the final event, indicating that the broker placed the order for the customer.
4. Open viewers for StreamPhoneOrders, StreamPlacedOrders, and StreamFraudAlerts. You can open a viewer for a specific stream by clicking the name in the Explorer view and then clicking the View Stream button (labeled with a magnifying glass) in the toolbar, or by

clicking View Stream in the shortcut menu associated with the stream name. Note that the Query statement does not use StreamWebOrders, so a viewer for that stream will not display anything.

5. Start the project and allow it to run for about a minute, until you see order 1005 in the StreamPlacedOrders viewer:

Timestamp	OrderId	Symbol	Volume	Br
2009/05/06 08:23:02.000000	1002	FILE	100	14
2009/05/06 08:24:31.000000	1312	SWY	10	14
2009/05/06 08:24:35.000000	1102	SWY	100	14
2009/05/06 08:28:32.000000	1103	EBAY	310	12
2009/05/06 08:33:34.000000	1003	IBM	1000	14
2009/05/06 08:41:33.000000	1314	FILE	10	12
2009/05/06 08:41:39.000000	1104	FILE	33	12
2009/05/06 08:45:02.000000	1214	IBM	1100	12
2009/05/06 08:45:08.000000	1004	IBM	9800	12
2009/05/06 08:46:33.000000	1315	IBM	30	12
2009/05/06 08:52:02.000000	1243	SWY	100	14
2009/05/06 08:53:34.000000	1005	SWY	400	14

Rate: 0.25 msgs/sec      Total: 15 msgs

Notice that, at the bottom of the viewer, there's a pair of rows that would match the pattern, except that the rows are more than 10 seconds apart.

6. Examine the StreamFraudAlerts viewer:

Timestamp	Alert	OrderId	Symbol
2009/05/06 08:23:02.000000	Front Running	1212	FILE
2009/05/06 08:45:08.000000	Front Running	1214	IBM

Disconnected

The rows in the output stream include the value of the OrderId column from Broker, which is the row containing the fraudulent order, not the original customer request. Look through the rows in the other viewers to ensure that you can identify how the rows in the streams match the pattern in the Query statement.

7. Stop the project.

## Detecting One of Several Events

Match one or two events, followed by other events in order.

The query in the last activity detected a sequence of events in a specific order.

Follow these steps to examine a project that detects front running with the initial order coming from either of two sources:

1. Replace the Query statement in the Queries tab with the following text (or copy the text from PatternMatchingCCL.txt between the lines "-- Pattern Matching Code Block 2" and "-- Pattern Matching Code Block 3"):

```
-- Match a pattern indicating front running:
--   Row with customer order
--   (from either of two streams)
--   Row with placed order for broker
--   Row with placed order for customer

INSERT INTO StreamFraudAlerts
SELECT "Front Running", Broker.OrderId,
      Broker.Symbol, Broker.Customer
FROM StreamPhoneOrders Phone, StreamWebOrders Web,
      StreamPlacedOrders Broker, StreamPlacedOrders Cust
MATCHING [10 SECONDS: Phone || Web, Broker, Cust]
ON Phone.Broker = Web.Broker = Broker.Customer = Cust.Broker
   AND Phone.Symbol = Web.Symbol = Broker.Symbol = Cust.Symbol
WHERE Phone.OrderId = Cust.OrderId
   OR Web.OrderId = Cust.OrderId;
```

The FROM clause in this Query statement uses four data sources from three streams. The MATCHING clause uses the "or" operator (||) to look for a row arriving on either StreamPhoneOrders or StreamWebOrders, followed by two rows arriving on StreamPlacedOrders. The ON clause conditions now include equality across all four data sources. The WHERE clause condition now matches against either StreamPhoneOrders or StreamWebOrders.

2. Open a viewer for StreamWebOrders, and then run the project, again allowing it to run for about a minute.
3. Examine the viewer for StreamFraudAlerts:

The screenshot shows a window titled "StreamFraudAlerts (iwerner/EventPatternMatching/StreamFraudAlerts)". The address bar shows the path "cd://localhost:6789/Stream/iwerner/EventPatternMatching/StreamFraudAlert". Below the address bar is a "Filter Expression:" field. The main area contains a table with the following data:

Timestamp	Alert	OrderId	Symbol
2009/05/06 08:23:02.000000	Front Running	1212	FILE
2009/05/06 08:24:35.000000	Front Running	1312	SWY
2009/05/06 08:41:39.000000	Front Running	1314	FILE
2009/05/06 08:45:08.000000	Front Running	1214	IBM

At the bottom of the window, there is a "Show only:" field and a status bar showing "Rate: 0.14 msgs/sec" and "Total: 4 msgs".

The output stream now includes rows identifying front running for orders originating in either the StreamPhoneOrders or StreamWebOrders stream. Examine the other viewers again to ensure that you can explain how the different rows matched the pattern in the Query statement.

#### 4. Stop the project.

#### Exercise

Complete the following exercise to explore additional pattern matching capabilities. See *Solutions* on page 70 for a possible solution to this exercise.

- Insert an exclamation point before "Cust" in the MATCHING clause so that it looks like this:

```
MATCHING [10 SECONDS: Phone || Web, Broker, !Cust]
```

What do you expect the results of this change will be? Run the project and verify your expectation.

## Subdividing a Sequence: Exercise

Set a subset of events to occur within a specific amount of time independent of the overall timing.

When you define a pattern-matching query, you may need a subset of the events to occur within a specific amount of time independent of the overall timing. For example, perhaps front running is defined in your organization as a broker order being placed no more than two seconds before a valid order for the same stock, which must occur no more than 10 seconds after the customer order arrives.

## Event Pattern Matching

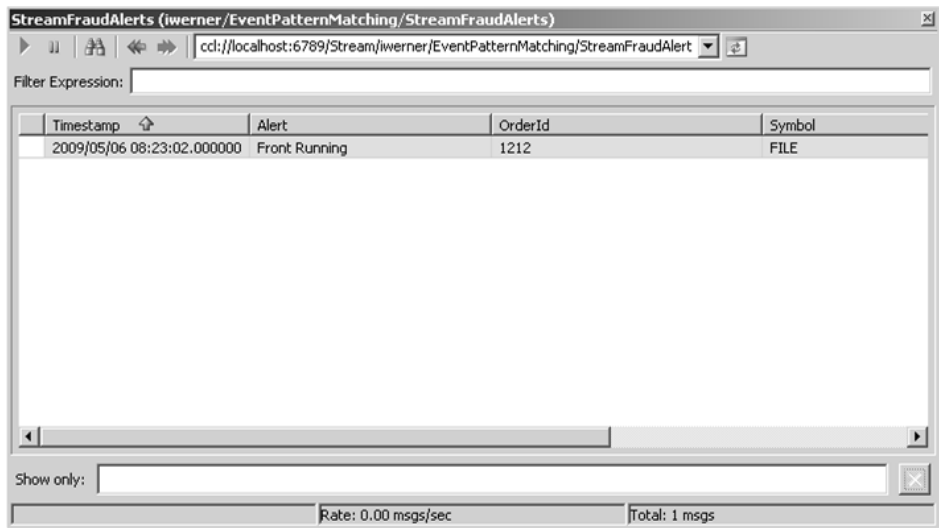
1. Replace the Query statement in the Queries tab with the following text (or copy the text from PatternMatchingCCL.txt between the lines "-- Pattern Matching Code Block 3" and "-- Pattern Matching Code Block 4"):

```
/* Match a pattern indicating front running:
   Row with customer order
   (from either of two streams)
   Row with placed order for broker
   Row with placed order for customer
   (last two within 2 secs) */

INSERT INTO StreamFraudAlerts
SELECT "Front Running", Broker.OrderId, Broker.Symbol,
Broker.Customer
From StreamPhoneOrders Phone, StreamWebOrders Web,
   StreamPlacedOrders Broker, StreamPlacedOrders Cust
MATCHING [10 SECONDS: Phone || Web, [2 SECONDS: Broker, Cust]]
ON Phone.Broker = Web.Broker = Broker.Customer = Cust.Broker
   AND Phone.Symbol = Web.Symbol = Broker.Symbol = Cust.Symbol
WHERE Phone.OrderId = Cust.OrderId
   OR Web.OrderId = Cust.OrderId;
```

The only change to this Query statement from the previous activity is in the MATCHING clause. The entire pattern must still happen within 10 seconds, but now the row arriving on Cust must also arrive within two seconds of the row arriving on Broker.

2. Start the project and let it run to completion.
3. Examine the StreamFraudAlerts stream viewer:



The screenshot shows a window titled "StreamFraudAlerts (iwerner/EventPatternMatching/StreamFraudAlerts)". The window contains a table with the following data:

Timestamp	Alert	OrderId	Symbol
2009/05/06 08:23:02.000000	Front Running	1212	FILE

At the bottom of the window, there is a status bar showing "Rate: 0.00 msgs/sec" and "Total: 1 msgs".

Now only a single sequence of rows matches the pattern and produces an output row. Look through the other viewers to ensure that you can identify the rows that matched the pattern.

#### 4. Stop the project.

#### Exercise

Complete the following exercise to explore additional pattern matching capabilities. See *Solutions* on page 70 for a possible solution to this exercise.

- Modify the query so that the pattern requires a row on Broker and on Cust, but in either order. See "MATCHING Clause" in the for information about the conjunction operator. Which row, identified by order number, is now included in the output stream that didn't match the pattern before this modification?

## Publishing the Matched Pattern

---

Examine a query that produces output rows that include the pattern that caused the output to be published.

Depending on how complex a pattern-matching query is, it may be difficult to determine exactly what sequence of events caused an output row. For example, you may need to know if the front running alert was related to an order originating on StreamPhoneOrders or on StreamWebOrders.

Follow these steps to examine a query that produces output rows that include the pattern that caused the output to be published:

1. Replace the Query statement in the Queries tab with the following text (or copy the text from PatternMatchingCCL.txt between the line "-- Pattern Matching Code Block 4" and the end of the file):

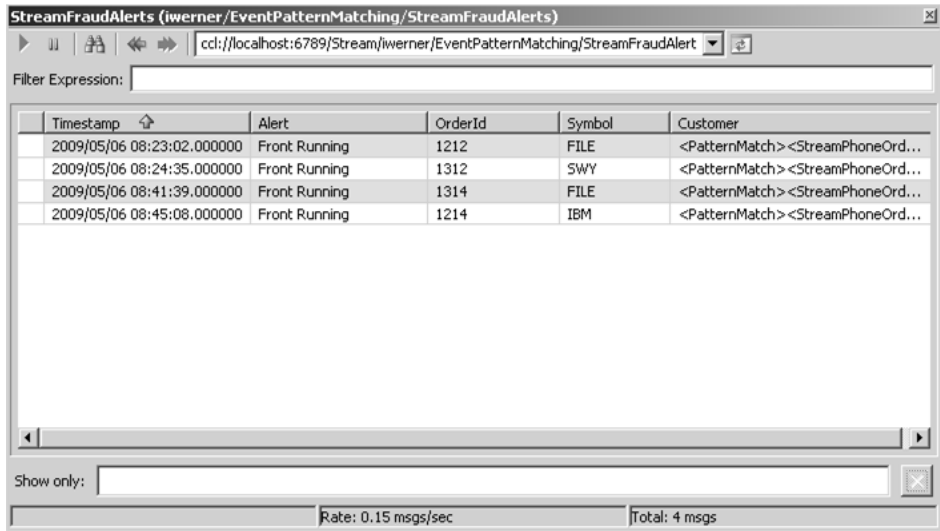
```
/* Match a pattern indicating front running
   and publish the pattern */

INSERT INTO StreamFraudAlerts
SELECT "Front Running", Broker.OrderID,
      Broker.Symbol, XMLPATTERNMATCH()
FROM StreamPhoneOrders Phone, StreamWebOrders Web,
      StreamPlacedOrders Broker, StreamPlacedOrders Cust
MATCHING [10 SECONDS: Phone || Web, Broker, Cust]
ON Phone.Broker = Web.Broker = Broker.Customer = Cust.Broker
   AND Phone.Symbol = Web.Symbol = Broker.Symbol = Cust.Symbol
WHERE Phone.OrderID = Cust.OrderID
   OR Web.OrderID = Cust.OrderID;
```

The SELECT clause includes the results of the XMLPATTERNMATCH function, which generates an XML tree containing all of the events that matched the pattern. Note that this function is only valid when used within a Query statement with a MATCHING clause.

## Event Pattern Matching

2. Locate the schema definition for the output stream StreamFraudAlerts and change the data type of the Customer column from String to XML, since it will now hold XML elements.
3. Run the project, again allowing it to run for about a minute.
4. Examine the FraudAlerts stream viewer:



The screenshot shows a window titled "StreamFraudAlerts (iwerner/EventPatternMatching/StreamFraudAlerts)". The address bar shows the path "cd://localhost:6789/Stream/iwerner/EventPatternMatching/StreamFraudAlert". Below the address bar is a "Filter Expression:" field. The main area is a table with the following data:

Timestamp	Alert	OrderId	Symbol	Customer
2009/05/06 08:23:02.000000	Front Running	1212	FILE	<PatternMatch><StreamPhoneOrd...
2009/05/06 08:24:35.000000	Front Running	1312	SWY	<PatternMatch><StreamPhoneOrd...
2009/05/06 08:41:39.000000	Front Running	1314	FILE	<PatternMatch><StreamPhoneOrd...
2009/05/06 08:45:08.000000	Front Running	1214	IBM	<PatternMatch><StreamPhoneOrd...

At the bottom of the window, there is a "Show only:" field and a status bar showing "Rate: 0.15 msg/sec" and "Total: 4 msg".

The final column, still labeled Customer, contains the XML tree. The tree is a very long string, so it is difficult to see in the stream viewer.

5. Right-click a row in the stream viewer and then click Copy in the shortcut menu.
6. Open a text editor or word processing application and then paste the text into a window:

```
Timestamp,Alert,OrderId,Symbol,Customer
2009/05/06 08:24:35.000000,Front Running,
1312,SWY,<PatternMatch>...
```

The first line is a list of the column names. The second line is the row itself, showing the values in the columns. What follows is an example of the Id column that holds the results of the XMLPATTERNMATCH function, formatted for readability:

```
<PatternMatch>
  <StreamPhoneOrders>
    <C8_Timestamp/>
    <OrderId/>
    <Symbol/>
    <Volume/>
    <Broker/>
    <Id/>
  </StreamPhoneOrders>
  <StreamWebOrders>
    <C8_Timestamp>2009-05-06 08:24:30.000000</C8_Timestamp>
```



```

<OrderId>1102</OrderId>
<Symbol>SWY</Symbol>
<Volume>100</Volume>
<Broker>14</Broker>
<Id>551</Id>
</StreamWebOrders>
<StreamPlacedOrders>
  <C8_Timestamp>2009-05-06 08:24:31.000000</C8_Timestamp>
  <OrderId>1312</OrderId>
  <Symbol>SWY</Symbol>
  <Volume>10</Volume>
  <Broker>14</Broker>
  <Id>14</Id>
</StreamPlacedOrders>
<StreamPlacedOrders>
  <C8_Timestamp>2009-05-06 08:24:35.000000</C8_Timestamp>
  <OrderId>1102</OrderId>
  <Symbol>SWY</Symbol>
  <Volume>100</Volume>
  <Broker>14</Broker>
  <Id>551</Id>
</StreamPlacedOrders>
</PatternMatch>

```

This example shows that the fraud alert resulted from a row that arrived on StreamWebOrders.

7. Stop the project.

## Event pattern matching tutorial conclusion

---

Summary of event pattern matching.

In this tutorial, you worked with queries that detected patterns of events - rows arriving, or not arriving, in a specified sequence within a given time period. Event pattern matching can be applied to a variety of situations: detecting network security threats, sending notifications when temperature readings in a building repeatedly fall outside an optimal range, or tracking employee activity to identify potential fraud.

This is the final tutorial. From here, you can begin exploring Sybase CEP Engine on your own, using the Sybase CEP documentation as your guide.

### *Further Reading*

For more information about topics covered in this tutorial, see "MATCHING Clause" in the *Sybase CEP CCL Reference Guide*.

## Solutions

---

Solutions to event pattern matching tutorial.

### *Detecting a sequence of events: Solution*

- The not operator (!) specifies the absence of an event. In this case, the pattern matches only if no row arrives on Cust within 10 seconds of the row arriving on Phone. In other words, the broker doesn't place the order for the customer within the 10-second limit.

### *Subdividing a Sequence: Solution*

- The following MATCHING clause accomplishes the specified task:

```
MATCHING [10 SECONDS: Phone || Web, [2 SECONDS: Broker && Cust]]
```

The row containing order ID 1314 matches the new pattern, but not previous patterns.