Real-time Processing Use Cases
Overview

This article describes two data synchronization scenarios that illustrate the benefit of real-time processing. The first use case shows how one business process is more effective after implementing real-time processing. The second use case demonstrates how real-time processing satisfies a common business requirement of message-by-message processing.

In both use cases, ensuring data integrity is very important. An incoming message is processed once and only once, and there is no data loss or data duplication.

Use Case: Data Synchronization

Before Real-time Processing

Company ABC has a custom made, hand-coded application that exists on a relational database. The application is an operational system that is constantly updated. The data warehouse exists on another database. They want to synchronize the changed data between this application and the data warehouse.

Through a system of database triggers, a process loads changed data into a set of tables. It reads and cleanses the data every 10 minutes. It creates an XML document to capture the changes and pushes the XML data to a message-oriented middleware system such as WebSphere MQ. They use the message-oriented middleware system as a staging area for the changed data. Then, another process reads the messages from the message-oriented middleware system. It processes the XML data and finally writes the changes to the data warehouse.

Company ABC requires current data, continuous processing, and an easily managed system.

This scenario does not meet the following requirements of the company:

- **Current data.** The latency is not near real-time. Because the data warehouse is only updated every 10 minutes, the data is not current.
- **Continuous processing.** There are too many steps involved with the data synchronization process, including a message-oriented middleware system in between. The processes involved cannot run as a long-running process. It cannot run 24 hours a day, 7 days a week due to the two stage process of capture and forward.
- **Manageability.** The process requires hand-coding, which can be a very time-consuming and expensive process.

After Real-time Processing

PowerCenter Real Time Edition includes the Change Data Capture (CDC) mechanism, PowerExchange CDC. With this, the changes made by the application are captured in near real time and made available to PowerCenter. PowerCenter extracts the CDC records and applies the changes to the data warehouse.

This scenario meets the following requirements of the company:

- **Current data.** The session can run continuously, 24 hours a day, 7 days a week.
- **Continuous processing.** The latency is near real time. The changes are written to the target data warehouse when the changes are made available by PowerExchange CDC. There are fewer steps involved in the data synchronization process. There is no need for a message-oriented middleware system in between.
- **Manageability.** No hand-coding is required.

For more information about the configuration details for PowerExchange CDC, see PowerExchange Interfaces for PowerCenter.

Use Case: Message-by-Message Processing

Company XYZ needs to process one message at a time. A message must be fully processed and committed to the target before processing the next message in the pipeline so it can be associated with related messages.
For example, company XYZ receives messages that contain sales opportunity information. This information is related in a hierarchical manner. There could be a parent opportunity, child opportunity, and sibling opportunity. Each of these opportunities could appear in a different message. When a message arrives, company XYZ needs the opportunity information to be processed and committed to the target immediately. When the next message with opportunity information arrives, PowerCenter can perform a lookup to the target to consolidate the information from the message with other sibling opportunity information.

To enforce message-by-message processing in a real-time session, configure source-based commit to 1 and configure a larger value for flush latency. The Integration Service commits data to the target using a combination of the commit interval and the flush latency interval. The first condition the Integration Service meets triggers the end of the flush latency period. If you configure a large value for flush latency, the source-based commit configuration triggers the end of the processing and ensures that one message is processed at a time.

**Note:** There will be an impact on performance as you reduce the commit interval or flush latency in a real-time mapping. In this case, throughput is smaller if you configure a higher flush latency or source-based commit, or higher combination of both.

**Authors**

Shankar Subramanian  
Director of Development

Jennifer Posivak  
Senior Technical Writer