Moving Toward Real-Time Data Warehousing

John P. McKenna is an independent consultant specializing in business intelligence and data warehouse technology. He has over 25 years of IT experience providing businesses with custom Oracle-based solutions.
mckennaj@optonline.net

John P. McKenna

Abstract

Business decisions must be made with speed and accuracy if organizations are to remain competitive. Because information that supports business decisions must be made available quickly, data warehousing professionals must move toward architectures that provide near-real-time information to the business intelligence (BI) and analytic systems. Moving toward real-time data warehousing presents many challenges but can yield significant value and competitive advantage to organizations that invest in them. This article will examine the who, what, where, when, and why of real-time data warehousing. We will also review some architectural changes that will move our data warehouses toward real-time information flows.

The Need for Real-Time Data Warehousing

Today, competition is fierce and business decisions must be made quickly and with accurate data. Decision makers cannot rely on days- or weeks-old data. Information must flow continuously from the operational transaction systems to the data warehouses; we must enhance our data warehouse architectures to provide near-real-time data flows.

For the last 30 years, the typical data warehouse has been built using batch interfaces that provide information accurate as of the last interface cutoff. Typically these interfaces are daily, weekly, or monthly and yield information that is days or weeks out of date. Basing key business decisions on such data can lead to poor outcomes. Imagine day-trading on the stock market with yesterday's price information—it would be quite a challenge to trade effectively.

Not all information needs to be as current as stock trading data, and the value of near-real-time data and analytics—and the tolerable level of data latency—

varies greatly by the type of data processed and how an organization uses it. However, building data warehouses with the most current data possible will provide enhanced business value. With a few incremental changes to the data warehouse architecture, we can move closer to real-time information at little or no additional cost. We must move toward building highly scalable information architectures that can provide near-real-time analytics and decision support, and we can do this by eliminating all possible latency from information flows.

Why not simply use source systems for analytics? There are several factors to consider: Does a single source system contain *all* the information required for analytics? In my experience, usually not. Source system performance must also be considered. Will running analytics *and* reporting on the source system inhibit OLTP performance? If either of these factors is an issue, then a real-time data warehouse effort should be considered.

True real-time data and analytics is impossible, of course. Even a financial trading house that builds a "real-time" trading platform will experience a degree of latency, even if it is hundredths or thousandths of a second. The goal is to minimize the latency between committing a source-system transaction and having the data available for analysis and reporting.

With a few incremental changes to the data warehouse architecture, we can move closer to real-time information at little or no additional cost.

Getting to Near-Real-Time

To reduce the latency of data arriving in the data warehouse, we must examine each component of the data warehouse architecture for all possible latency. We will begin by (1) reviewing the overall architecture to examine what impact various architectural styles may have on building a near-real-time environment and (2) examining the ETL architecture where the data movement occurs. The ETL architecture has the greatest impact on data latency, so we will identify several design patterns that can be used for near-real-time information flows. Specifically, we will examine several aspects of ETL processing, including:

- Source data acquisition (extracts)
- Data transformations
- Data loading
- Error handling
- Job synchronization

After examining the near-real-time design patterns for ETL processing, we will review some topics that require special consideration in a near-real-time environment:

- Data quality
- Summary data
- Analytics and reporting
- Business impact

Data Architecture

A key decision in any data warehouse implementation is what overall architectural style to employ for the warehouse. Should we follow Bill Inmon's conventions to build a corporate information factory (or DW 2.0), or is Ralph Kimball's bus architecture more appropriate? Should an operational data store (ODS) be utilized for the near-real-time data? The right answer will depend on many factors, including the current state of data warehouse development, organizational readiness, and the scope of the project being undertaken.

The good news is the near-real-time design patterns we will discuss can be incorporated into each of these architectural styles. They are not dependent on the architectural style, level of normalization, or existence of a specific warehouse component (i.e., ODS). The design patterns require that all data streams flow with minimal latency and techniques be applied to eliminate serialization of interfaces.

ETL Architecture

Designing an ETL architecture for near-real-time data requires different design constructs than are typically found in many data warehouse environments. Data must be captured from transaction streams or extracted into small, incremental batches (microbatches) that are fed immediately to the target platform in data-loading routines. Data validation must occur within the initial load routine to eliminate the latency associated with staging the data into a temporary load file or table. Validation errors must be written to an error file or table so that the remaining records can be loaded immediately and not suspended with the invalid records. The following sections will examine these ETL steps in greater detail.

Source Data Acquisition (Data Extracts)

All information flows within the ETL architecture must be designed to minimize the "batching" of data and move toward the "streaming" of data. This process begins with the source data acquisition process or data extracts. There are several design patterns for obtaining near-real-time data. They include real-time data replication, message bus transports, and microbatch architectures.

All information flows within the ETL architecture must be designed to minimize the "batching" of data and move toward the "streaming" of data.

Real-Time Data Replication

Many tools can obtain real-time data streams from source databases. These products typically obtain source transaction data from mining the source database transaction logs and microbatching or streaming the transactions to the target environment. These transaction streams are then applied to the target database and a replicated table is produced with minimal latency. Replication products are readily available for most mainstream database platforms and provide proven frameworks for acquiring a real-time data stream.

The main disadvantage of this approach is that many of these tools cannot transform the data during the replication process. These tools are most suitable for an ODS that is a replica of the source system table and replicated with minimal latency.

Message Buses/Transaction Systems

In addition to real-time data replication technology, more organizations are utilizing message bus technology (e.g., Java Message Service) for transaction management. The advantage of this architecture is that it is a publish/subscribe environment and is very convenient for publishing transaction data to multiple downstream consumer applications. If your source transaction system utilizes this type of infrastructure, you could create a subscriber module to tap into the data stream and feed it to the data warehouse.

This architecture will only provide a near-real-time data stream if the data on the message bus is published in near-real-time. This architecture is also convenient for publishing data out to one or more marts from the enterprise data warehouse.

Microbatch Architecture

A third alternative for obtaining near-real-time data is to develop a microbatch architecture in which the source platform extracts data in small, incremental batches that are passed to the data warehouse environment. I have used this design approach several times with parameterized extract and load routines so that the size of the microbatches can be adjusted based on latency requirements and optimal microbatch sizing. This architecture can be incorporated into an existing batch environment by enhancing the extract and load routines to handle incremental extracts instead of daily or weekly extracts.

Mixed Architectures

The previous three data acquisition architectures can also be combined to form mixed architecture environments. For example, you could publish microbatch extracts onto a message bus for downstream data mart loading, or employ a microbatch architecture for transaction data feeds while utilizing real-time replication for master/reference data feeds.

Data Transformations

Data transformation processing does not normally add significant latency to data warehouse processing. As long as the data transformation rules are automated and do not require user intervention, no changes should be required to this aspect of the architecture. The only factor to keep in mind is that all data transformation rules should be data driven with history retained. In the event a near-real-time feed needs to be reprocessed, ensure the transformation logic matches the original run (unless running against current mappings is desired).

Data Loading

To minimize the latency of data in the warehouse, data should be loaded as soon as it arrives. This can be performed with a variety of techniques but must match the data extract method employed. For example, if the data acquisition strategy is to use real-time data replication, then a compatible data-load method must be utilized. I have frequently utilized microbatch architecture with data-load routines that immediately queue up the next microbatch for loading. This type of architecture enables you to tailor batch sizes to optimize array loading mechanisms and incur minimal latency.

The use of staging/landing tables should be avoided for several reasons. Using staging/landing tables requires an additional write for every record processed. Because disk writes are the most expensive operation on a system, they should be minimized. Furthermore, staging tables add an additional layer of latency to the information flow—the latency we are trying to eliminate.

To build a reliable information system, minimize the complexity and potential points of failure within it. Staging tables add another potential point of failure where data may be suspended and need to be reconciled. There are situations where staging tables may be advantageous; the most common reason is to reduce the complexity of the ETL code when high-level ETL tools are used.

Error Handling

Error handling in data loads must be done differently when building a near-real-time data warehouse. If incoming data fails a data validation check, we cannot suspend the entire incoming data stream. We must suspend only the invalid record and alert the data steward that there are rejected records to be reviewed. This allows the data loads to continue without interruption and permits the valid data to be loaded without additional latency. Error monitoring and resolution procedures need to be prepared so that invalid records are not allowed to accumulate and affect the accuracy of the warehouse.

These data review procedures need communication mechanisms with the user community so that any material differences caused by suspended transactions do not lead to poor business decisions. The data steward must be diligent in reviewing and resolving all suspended records.

Error monitoring and resolution procedures need to be prepared so that invalid records are not allowed to accumulate and affect the accuracy of the warehouse.

Job Synchronization

Job schedules can add latency to data arriving in the warehouse. Data warehouse loads are frequently designed to load all reference/master file data first and then process the transaction/fact data. This job sequencing is convenient because it ensures that the reference/master data used for transaction validation is up to date. However, making the transaction data loads wait for the reference data loads to complete serializes data loads and adds latency to the data availability in the warehouse.

The design pattern that eliminates this dependency is commonly referred to as *stubbing*. This design pattern enables transaction data loads to occur independently from the reference/master file maintenance. When a new key value for the reference/master data is encountered in the transaction data, a "stub" record for the reference/master record is created. When the reference/master file mainte-

nance is performed, the initial "stub" record is updated to include the missing attributes.

Data Quality

Data quality control mechanisms for near-real-time warehouses require special considerations. In the near-real-time environment, data is continuously changing, making balancing to the source system a moving target. The simplest approach is to apply latency to the quality control routines so that balancing can occur on a static snapshot at some point in history.

If there is zero tolerance for minor data deviations due to suspended transactions or other processing errors, you should not be embarking on a near-real-time data warehouse project.

Whether this approach will be acceptable depends on the sensitivity of the data. For example, running automated balance validation routines with a 10- to 30-minute lag to ensure there are no in-flight transactions is a reasonable approach for many environments. If applying a minimal lag time to the balance validation routines is not acceptable, there will be some heavy lifting necessary to develop dynamic balancing routines to ensure an accurate data environment.

A major consideration for the design of the data validation routines is the amount of time it takes to identify a potential data variance. If users are working with the data in near-real-time, any delay in the identification of data issues is a window of opportunity for inaccurate results to be used for management decisions. This is dangerous and must be discussed with the project stakeholders before embarking on a near-real-time data warehouse project. If there is zero tolerance for minor data deviations due to suspended transactions or other processing errors, you should not be embarking on a near-real-time data warehouse project. Even if minor deviations are acceptable, you need to make provisions for additional data quality checks.

As with error handling, your organization must notify users when a data-quality check fails. Open communication here is a double-edged sword. If the users are constantly being informed about slight variances in the data, they will lose trust in the system and not rely on it for important business decisions. On the other hand, if management is acting on information that is inaccurate, this could lead to devastating results. Speak openly and frankly in the project planning stages and establish defined thresholds for user notification. A problem escalation procedure that provides early notification to key stakeholders and broad notification for high-severity issues is a critical component to ensure a quality environment. The goal is to insulate users from business-as-usual issues of minor severity, but provide prompt notification for severe issues that compromise the integrity of the analytics.

Summary Data

Summary data presents a challenge for the near-real-time data warehouse. Using the normal approach to computing summary data via a nightly, weekly, or monthly process does not provide near-real-time metrics. Although it is easy enough to rely on detail transaction data for the near-real-time metrics, this may present performance problems if the data streams are large.

An alternative is to design data summarization processes similar to an OLTP environment, where summary records are created/updated at the point when the detail transaction is committed. In an OLTP environment that requires real-time account balances, the account balance (summary) record is updated when the detail transaction is saved. This type of transaction is common in an OLTP environment but is not normally found in a data warehouse environment.

Restraint and great care are required if you start down this road of keeping near-real-time balance records. The records will not be as current as the data in the OLTP system and can get out of sync with the detail if the

proper error-handling and control mechanisms are not in place. You must also be careful not to replicate the OLTP processing and logic in the data warehouse, as this would be a costly and dangerous endeavor. Having two systems (OLTP and DW) yielding the same metrics with different values because of latency is a sure path to disaster.

Analytics and Reporting

To provide true business value, data must be available in near-real-time to the user access layer of the data warehouse. This may seem obvious, but there are a few items to consider in the user-access layer. Many analytics and reporting tools contain caching mechanisms that store prior query results to improve performance. If the caching mechanism in the tool does not detect the changes from the near-real-time data and refresh the cache, then users will not have real-time analytics and reporting.

Another consideration for the user-access layer is whether all dashboards, reports, and analytics need to be near-real-time. For example, it would be more efficient to build weekly or monthly reports from summary tables and not place an undue burden on the near-real-time data streams.

Business Impact

Before we conclude our discussion of moving toward real-time data warehousing, we should consider the impact to the business consumers of data. Working with data that is continually changing is a paradigm shift for many business users. Individuals who are used to getting daily or monthly reports from operational systems will need to be "reconditioned" to accept data that is continuously changing. When key metrics are fluid, changing from minute to minute, there is a tendency to think something is wrong. This is another area where early discussions during project planning will be beneficial. Setting client expectations early and often will lead to a much smoother transition at implementation.

Conclusion

In the highly competitive marketplace of the 21st century, we need to provide data solutions that are timely and accurate to support rapid business decisions. By enhancing the way we architect and build our data warehouse infrastructure, we can move toward more real-time data availability. Making this shift will provide our business communities more timely information for making rapid business decisions. Care must be taken to incorporate adequate monitoring and control structures to ensure our near-real-time data stores are current and accurate. With proper planning, we can provide enhanced value and empower data warehousing to be a greater strategic asset to our organizations. ■