Chapter 11. Real-Time ETL Systems

Building a real-time data warehouse ETL solution demands classifying some often slippery business objectives, understanding a diverse set of technologies, having an awareness of some pragmatic approaches that have been successfully employed by others, and developing engineering flexibility and creativity. This field remains young, with new technologies, emergent methodologies, and new vocabularies. Clearly, this situation can be a recipe for trouble, but real-time data warehousing also offers early adopters great potential to gain a competitive advantage—an intriguing risk versus reward trade-off. This chapter proposes a four-step process to guide the experienced data warehousing professional through the selection of an appropriate real-time ETL technical architecture and methodology:

1. This chapter examines the historical and business contexts of the state of the art real-time data warehouse—providing some *How did we get here*? and *Where are we going*? background.
2. Next, it describes a method for classifying your organization's real-time requirements in a manner that is most useful for selecting design solutions later.
3. The heart of the chapter is an appraisal of several mechanisms for delivering real-time reporting and integration services, the technologies most appropriate for each approach, and their strengths and weaknesses.
4. And finally, a decision matrix is presented; it uses the requirements classifications and approaches previously described, and it guides the ETL team through the selection of a technical approach and methodology.

TIP

It must be stated that this material falls short of a *recipe* for building a real-time ETL solution; as of this writing, such recipes do not exist. As this new technology becomes more popular, you are bound to come up against requirements for which solutions have not yet been perfected. Apply your creativity and the know-how you have gleaned from personal experience in fashioning a solution most appropriate for the specific challenges you face. By doing this, you are doing your part to help advance the progress of real-time data warehousing.

NOTE

PROCESS CHECK

* Planning & Design: This chapter touches every aspect of the ETL system planning and design. We intend this chapter to be read as an *increment* to all the ideas developed in [Chapters 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html) through [10](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html).
* Data Flow: Parts or all of the techniques in this chapter can be added to an existing ETL system framework. But as we emphasize, the conversion from batch-oriented ETL to streaming ETL is a profound end-to-end change.

Why Real-Time ETL?

Not very long ago, engineers vehemently defended the notion that the data warehouse needed to provide an unwavering set of data to business decision makers, providing a reliable information *floor*upon which to stand. For up-to-the-moment reporting against a *twinkling* database, business users were directed to the production applications that run the business. Therefore, users had to go to the data warehouse for a historical picture of what happened in the business as of yesterday and had to look across many OLTP systems for a picture of what was happening today. Business users never fully accepted this divide. Why could they not go to one place to get the business information that they needed?

Well, much has changed, and the data warehouse has now become a victim of its own success. Although the delay between a business transaction and its appearance in the data warehouse is typically less than 24 hours, for many organizations in fast-moving vertical industries, this delay is too much. The data warehouse has become mission critical, too, feeding enriched information back to operational systems that is then used to process transactions, personalize offers, and present up-sell promotions. The push for ever-fresher information is on.

Several other important factors have conspired to force data warehouse practitioners to rethink some earlier positions:

* **Customer relationship management (CRM)**. Modern CRM demands a contemporary, consistent, and complete image of the customer available to all operational systems that directly or indirectly serve the customer—quite a tall order. Despite the marketing claims of leading packaged CRM vendors, this capability cannot be purchased off the shelf; unless all customer-facing systems are retired by the packaged CRM suite, businesses also need to integrate real-time customer information across all of their legacy transactional stovepipe applications. Data warehouses, of course, absolutely need constant customer information streams from operations, but increasingly, operational systems rely on data warehouse enrichment of customer information, too. Therefore, it is predictable that organizations have begun to explore architectural alternatives that can support more generalized integration scenarios—moving operational data between applications and simultaneously into and out of the warehouse—with ever-increasing urgency.
* **The zero-latency enterprise business ideal**. This ideal exhorts the benefits of speed and a single version of the truth. In a real-time, zero-latency enterprise, information is delivered to the right place at the right time for maximum business value. Some people call these *right-time*systems. Just-in-time inventory and supply chains and assemble-to-order/mass customization business models also amplify the need for absolutely current and pervasive information throughout the organization. At present, true zero latency is an unattainable ideal—it takes some time to synchronize information across several production systems and data marts—but the pressure on many modern data warehouses to provide a low-latency view of the health of the business is very real.
* **Globalization and the Web**. Finally, and perhaps most pragmatically, the combined effects of globalization and the Web, which demand round-the-clock operations and access to the data warehouse, in concert with requirements to warehouse ever-broader and deeper sets of data, have severely compressed the time window available to load the data warehouse. The amount of data needing to be warehoused continues to expand, while the window of *business downtime* continues to shrink, challenging the already overworked and under-loved data warehouse's ETL team. Wouldn't it be easier if you could somehow *trickle feed* your data warehouses throughout the day, rather than trying to shoehorn expanding data loads into shrinking windows of acceptable downtime?

These factors have conspired to drive the data warehouse to an increasingly real-time posture.

Defining Real-Time ETL

Real-time ETL is a misnomer for a category of data warehousing services that is neither true real-time nor, in many cases, ETL. Instead, the term refers to software that moves data asynchronously into a data warehouse with some urgency—within minutes of the execution of the business transaction. In many cases, delivering real-time data warehousing demands an approach quite different from the ETL methods used in batch-oriented data warehousing. Simply running conventional ETL batches on an ever-more frequent schedule throughout the day might not be practical, either to the OLTP systems or to the data warehouse. Conversely, including the data warehouse in the OLTP system's transaction commit logic cannot work either. The OLTP system does not have the luxury of waiting for the data warehouse loading transaction to *commit* before it proceeds with its next transaction, nor is any locking or two-phase commit logic practical across systems with different structures and different levels of granularity. Instead, you aspire simply to move the new transactions into a special *real-time partition* (defined later in this chapter) of the data warehouse within some timeframe acceptable to the business, providing analytic support for day-to-day operational decisions. For the time being, this procedure is our practical definition of real-time ETL.

NOTE

This chapter explores some pragmatic approaches to achieving these objectives, using mainstream toolsets familiar to data warehousing engineers. However, real-time data warehousing is a young field, rife with all manner of software-vendor claims and higher risk. The approaches to real-time ETL explored in this chapter attempt to minimize risk through managed expectations and emphasis on mature approaches and execution strategies rather than groundbreaking tool selection. This chapter presents approaches that address the objective of achieving a few minutes latency between business transactions and their availability in the data warehouse.

Challenges and Opportunities of Real-Time Data Warehousing

Real-time data warehousing presents a number of unique challenges and opportunities to the ETL engineer. From a technical architecture perspective, it has the potential to change the big-bang approach needed during the nightly batch ETL load windows to a continuous ETL-like flow throughout the day. System-availability requirements may escalate as the business comes to rely on low-latency availability of business transactions in the data warehouse. If the organization opts for the real-time dimension manager approaches described in this chapter, availability becomes a strategic advantage.

From a data architecture perspective, real-time data warehousing challenges the posture of the data warehouse as system of discrete periodic measurements—a provider of business *snapshots*—advocating instead a system of more comprehensive and continuous temporal information. This shift happens subtly if, for example, the frequency of fact loading increases from once per day to every 15 minutes, but more dramatically if the loading of facts and dimension records occurs continuously. The data warehouse might then capture a record of the business transactions and their dimensional context at all points in time. Slowly changing dimensions become rapidly changing dimensions, and the data warehouse's bearing becomes more operational in nature. In fact, should the real-time data warehouse also support real-time dimension conforming and synchronization, it then evolves into a logical extension of the operational systems themselves.

Real-Time Data Warehousing Review

The real-time approach to data warehousing can trace a clear lineage to what was originally called the ODS. The motivations of the original ODSs were similar to modern real-time data warehouses, but the implementation of real-time data warehouses reflects a new generation of hardware, software, and techniques. The following sections develop these ideas in more detail.

Generation 1—The Operational Data Store

The operational data store, or ODS, is a first-generation data warehousing construct intended to support lower-latency reporting through creation of a distinct architectural construct and application separate from the data warehouse. The ODS is half operational and half decision-support system, attempting to strike a balance between the need to simultaneously support frequent updates and frequent queries. Early ODS architectures depicted it as a place where data was integrated and fed to a downstream data warehouse, thus acting as a kind of extension to the data warehouse ETL layer. Later architectures depict it as a consumer of integrated data from the data warehouse ETL layer and categorize it as a Type 1 through 4 and *internal or external* ODS, depending on where within the overall architecture it resides and the urgency with which it must load data from the operational world.

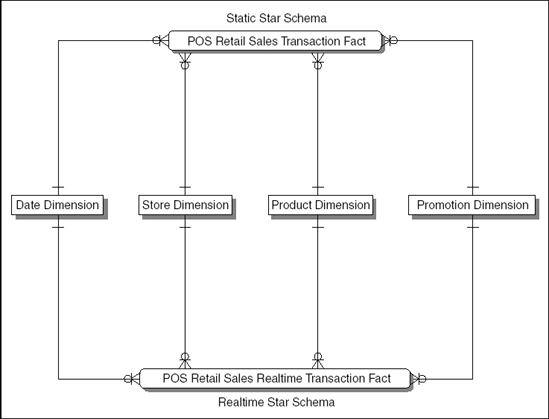
In practice, the ODS has become a catch-all architectural component for data staging, data cleansing, and preparation, as well as operational reporting. By virtue of all these different roles, it is a compromise solution to each of these challenges. A simpler and less compromising alternative exists.

Generation 2—The Real-Time Partition

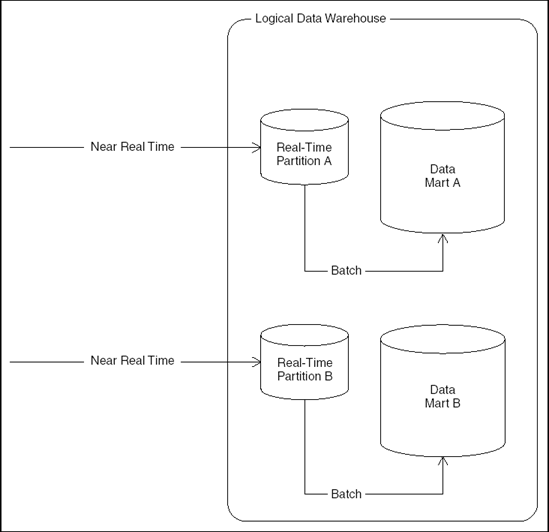
The use of the real-time logical and physical partition, as originally described by Ralph Kimball, is a pragmatic solution available for delivering real-time analytics from a data warehouse. Using this approach, a separate real-time fact table is created whose grain and dimensionality matches that of the corresponding fact table in the static (nightly loaded) data warehouse. This real-time fact table contains only the current day's facts (those not yet loaded into the static data warehouse table).

[Figure 11.1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#the_relationship_between_the_static_and) shows two star schemas associated with a real-time and static retail point-of-sale fact tables, sharing a common set of dimensions.

Each night, the contents of the real-time partition table are written to the static fact table, and the real-time partition is then purged, ready to receive the next day's transactions. [Figure 11.2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#the_logical_relationship_of_the_real-tim) gives an idea of how the process works. In essence, this approach brings the real-time reporting benefits of the ODS into the data warehouse itself, eliminating much ODS architectural overhead in the process.



**Figure 11.1. The relationship between the static and real-time star schemas**



**Figure 11.2. The logical relationship of the real-time partition to its data mart**

Facts are *trickled in* to the real-time fact table(s) throughout the day, and user queries against the real-time table are neither halted nor interrupted by this loading process. Indexing on the real-time fact table is minimal, or nonexistent, to minimize the data-loading effort and its impact on query response times. Performance is achieved by restricting the amount of data in the table (one day only) and by caching the entire real-time fact table in memory. Optionally, a view can be created that combines (Unions) facts in both the real-time and static fact table, providing a virtual star schema to simplify queries that demand views of historical measures that extend to the moment.

If fact records alone are trickle-fed to the real-time partition, some policy is needed to deal with changes to dimensions that occur between the nightly bulk loads. For example, new customer records created during the day for which you have facts might need to be defaulted to a series of generic *new customer* records in the customer dimension to be updated into more descriptive customer records in the evening, when a complete batch load of new and changed customers is loaded into the static customer dimension. Alternatively, the real-time data warehouse can opt to maintain more frequent snapshots of changing dimensional images or to abandon the point-in-time concept altogether and instead capture all dimensional changes that occur.

Later, this chapter describes some of the issues associated with selecting an appropriate policy for dealing with dimensional changes, some pragmatic approaches to trickling data into real-time partition(s) throughout the business day, and the pros and cons of these approaches.

Recent CRM Trends

CRM demands a complete understanding of the organization's history with each customer across all customer *touch points* and insight into the challenges and priorities that the customer faces in their markets. In the past few years, packaged CRM systems have been widely adopted by businesses to support the first of these goals, unifying the simplest and most common customer touch points of the organization. However, while these systems represent an important advance for organizations that had fragmented customer support systems (or no systems support at all), they are not comprehensive. Often, there are older and more specialized systems that support customer interactions that fall outside of the packaged CRM system. These transactions never find their way back to the packaged CRM system. Also, packaged CRM systems typically fall short of equipping the organization with the customer knowledge it needs to be perceived as an intelligent collaborator and partner by its customers because they lack any mechanism for collecting, harvesting, and synchronizing customer and marketplace intelligence across the enterprise. The further splintering of the packaged CRM marketplace into Operational CRM versus Analytic CRM amplifies this divide. Businesses don't have *operational* or *analytic* customers; the same patron must be intelligently served by both operational and decision support systems, working together.

What is needed is a way to bring together with great urgency all of the data about the organization's past and present interactions with the customer, combined with external marketplace information, some mechanism to convert data into customer intelligence, and a means to share this with everyone in the organization. Bringing such things together represents a melding of data warehouse technologies and application integration technologies.

NOTE

CRM vendors are keenly aware of the challenges facing organizations, so some are *bolting on* Business Intelligence capabilities to their operational CRM suites. Too often, the result is rudimentary, simplistic, and difficult to architecturally defend, ultimately failing to provide a differentiating competitive capability.

Generation 2 CRM as we define it in this chapter is not an application that can be purchased and installed; rather, it demands a comprehensive data warehouse of all customer touch points, intelligently selected and utilized marketplace data, a continuous harvesting of customer intelligence from the data warehouse, and a mechanism for sharing and continuously synchronizing customer information across the enterprise. The task of providing such capabilities seems to be landing right in the backyard of the contemporary ETL architect.

The Strategic Role of the Dimension Manager

The glue that binds logically and/or physically separate subject areas (data marts) together in the dimensional data warehouse bus architecture is conformance of dimensions and facts, achieved through the use of dimension manager systems as described in this chapter. Traditionally, the dimension manager has been viewed as a role whose job is the definition, maintenance, and publication of a particular conformed dimension to all data marts that interoperate within the data warehouse bus architecture.

Ultimately, the real-time data warehouse plays a role in the larger objective of providing ready access to the most current and insightful data to all users throughout the enterprise. In addition, to quickly deliver fact records to the data warehouse, tremendous competitive advantage might be found in providing real-time synchronization of key dimensions such as customer or product across all operational systems in the organization. This information-synchronization function can be considered a logical extension of the dimension manager role and is an effective and consistent mechanism for closing the loop between the operational world and that of the data warehouse by providing a means of distribution of data warehouse-derived segmentations and other enrichment information to the operational world.

The customer dimension manager in a strategic real-time data warehouse might not only trickle-feed all data marts with new conformed customer information, but might also cooperate with some mechanism for synchronizing customer information across all interested (subscribing) operational systems. This real-time customer information should include customer intelligence generated by the data warehouse itself.

Clearly, these are ambitious objectives, and as of this writing, no packaged solutions or end-to-end toolsets dramatically simplify the process of building a bidirectional enterprise application integration (EAI)/real-time data warehouse solution. Nonetheless, such systems have been created; the basic building blocks for these systems exist and are getting more mature. The potential for business differentiation provided by such a system is striking, so it is likely that today's early adopters will enjoy marketplace advantages that drive more widespread adoption of such systems in the future. Consider building systems today that, at a minimum, do not impede the organization's ability to evolve to a real-time EAI/data warehousing solution in the future. Organizations under competitive pressures or seeking marketplace differentiation through customer intimacy might need to take the leap now.

Categorizing the Requirement

Clearly, this topic offers a lot to consider from an architectural perspective. Given the rather complex set of strengths and weaknesses associated with the mainstream alternatives for real-time data warehousing, it is important to nail down the scope of your real-time requirements.

Presented in the sections that follow are some litmus test questions that, once answered, help you categorize the set of real-time capabilities needed by your organization and select mainstream methodologies and toolsets appropriate for the task at hand. A matrix appears near the end of the chapter that summarizes this discussion and guides the ETL team in approach and architecture selection.

Data Freshness and Historical Needs

The developmental costs and complexity for reducing latency between OLTP and the data warehouse obey the law of diminishing returns, lowering latency increases complexity and cost in a nonlinear fashion. Therefore, you need to set realistic goals and expectations about the *freshness* of the data needed in the warehouse.

You also need a complete understanding of the set of hard business requirements that cannot be met either through conventional daily data warehouse publication or transactional reports from OLTP systems. Watch for the following red flags as you consider the needs of your would-be real-time data warehouse:

* **Less than five minutes of latency**. Reports with latency this low, as of this writing, cannot be reliably met through mainstream real-time data warehousing. This window of time shrinks continuously, but it always takes some nontrivial amount of processing and time to move, transform, and load information from the OLTP systems to the real-time partition. Organizations that absolutely must have information more than five-minutes fresh should consider running their reports directly against the operational system(s).

NOTE

Enterprise Information Integration (EII) applications do not suffer this latency limitation and can deliver nearly up-to-the-second reports directly from the operational systems. However, they have other characteristics and limitations that must be considered. EII systems and these limitations are discussed later in this chapter.

* **Single data source requirements demanding little or no history**. These reports require none of the integrated and historical data features provided by the data warehouse and are best addressed through the operational system itself. Happily, they should present a very small reporting footprint on the OLTP systems and should not degrade transactional performance significantly. If Web-enabled, these reports can be presented through the business intelligence portal, and they then *feel* to the user community as if they are data-warehouse based.
* **Reports with an entirely different audience from that of the existing data warehouse**. These reports might demand new reporting vocabularies and mechanisms for dissemination, factors that can overly complicate an already complex real-time data warehousing development effort. While not an automatic project-killer, the real-time architect should be aware that business vocabularies and metrics employed by shipping versus marketing management, for example, are likely to be quite different and deeply rooted.
* **No real need for ad-hoc analysis**. If there is little demand for ad-hoc analysis of the low-latency part of data, you may be able to avoid a full-blown streaming ETL system redesign. Perhaps you can simply append a *flash report* of most-recent data from the transaction system to a conventional data warehouse report created with data up through yesterday.
* **Organizations that have not yet successfully implemented a data warehouse**. Attempting a real-time data warehouse as an initial business intelligence development effort, at least for now, is not recommended, simply because it demands mastery of too many simultaneous disciplines. Thankfully, dimensional data warehousing architectures and methods allow the organization to gracefully add real-time reporting capabilities later.

A symptom of one of these red flags should be a report that requires data fresher than last night but is tolerant of at least five minutes of latency. Such a report may also demand continuity in terms of data history, reporting vocabulary, and presentation with the existing non-real-time data warehouse. These red flag reports are appropriate candidates for the real-time data warehousing ETL approaches described.

The next sections discuss some basic requirements for real-time time reporting.

Reporting Only or Integration, Too?

Does the organization need a one-way solution for moving operational data into the data warehouse for reporting purposes only, or are there also requirements for *closing the loop* by moving conformed dimension data between operational applications themselves and/or the data warehouse? For example, is a mechanism needed for moving data warehouse-derived customer segmentations back into the operational systems? This question is perhaps the most influential in selecting a real-time data-warehousing approach.

Certainly, any strategic CRM initiative is likely to require a means of sharing the timeliest and most complete customer information available, which includes both operational customer data (information about recent sales or complaints, for example) and data-warehouse or data mining-derived customer marketing information such as customer segmentation, profiling, and lifetime value. Is the request for real-time reporting a first step in the journey of closing the loop between the operational and decision support systems of the organization (true CRM)?

Just the Facts or Dimension Changes, Too?

Business people and dimensional data warehouse architects describe the world in terms of facts and dimensions, but OLTP systems do not make such crisp distinctions. Nonetheless, as an engineer, you must understand and categorize the OLTP business transactions of interest to your end users and design appropriately.

Are the real-time report requirements focused exclusively on fresh facts, such as recent orders placed, recent telephone calls routed, recent stock trades executed, recent sales calls made, and so on, or are they also concerned with fresh dimension transactions, such as customer or product record updates? If real-time dimensional changes are needed for reporting, are they slowly or rapidly changing? In other words, does the user community need an accurate image of these dimensions as they were at the point in time of the transaction, or can all or some dimensional updates be destructively overwritten when new updates occur? Do reports need to be *repeatable*? Type 1 slowly changing dimensions, in which the changes to a dimension's attributes destructively overwrite the prior values, result in a data warehouse that continuously recasts history, reporting events not as they look at the time of the transaction but as they look in the context of today's dimensions. In this scenario, reports run against the same dimensional elements at different points in time might have slightly or dramatically different results. Type 1 changes are also dangerous because they can invalidate historical aggregates if the Type change is applied to a field used as the basis of an aggregate calculation.

Type 2 and 3 slowly changing dimensions maintain a more granular picture of the dimension images at certain points in time, perhaps daily, but they still do not capture the changes to dimensions that occur between extractions. Real-time dimensional refresh can drive this granularity up to every few minutes or can capture all dimension changes.

The architectural implications are not subtle. By adopting a policy of capturing increasingly frequent dimensional change images, the data warehouse moves away from its earlier posture as a system of periodic measurement (snapshots) and toward a zero-latency decision-support ideal. As data warehousing and application integration technologies begin to commingle, the data warehouse becomes, in effect, a true logical extension of the operational systems that run the enterprise. For the time being, as a practical matter, ETL systems probably need to be designed to provide as near zero latency for measured facts as possible, but allow some or all dimensional attributes to be updated in batches, or microbatches, as developed in this chapter.

Alerts, Continuous Polling, or Nonevents?

Although usually the ETL system has a very well-defined boundary where dimensionally prepared data is handed to the front room, in many cases a real time system cannot have this boundary. The architecture of front-end tools is affected at the same time. There are three data-delivery paradigms that require an end-to-end perspective reaching all the way from the original source to the end user's screen:

* *Alerts*. A data condition at the source forces an update to occur at the user's screen in real time.
* *Continuous polling*. The end user's application continuously probes the source data in order to update the screen in real-time.
* *Nonevent notification*. The end user is notified if a specific event does not occur within a certain time interval or as the result of a specific condition.

In each of these cases, the real-time ETL system is connected all the way to the end user's application, either by sending a notification or by receiving a direct request.

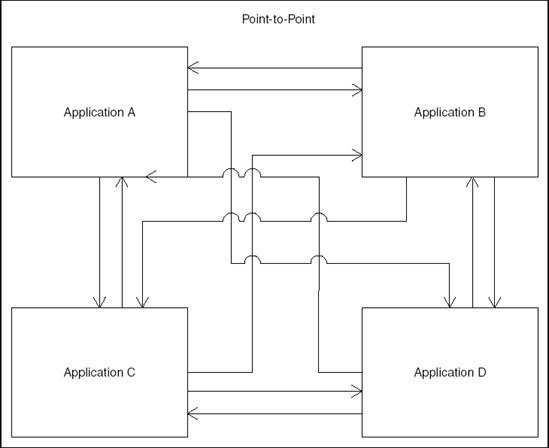
Data Integration or Application Integration?

Assuming that the real-time data warehouse requirement also entails some measure of integration across operational systems, you need to categorize your requirement as either data integration or application integration.

* In general, integration that can be satisfied by simply moving data between databases is called *data integration*. Often, these solutions are point-to-point, executed through (for heterogeneous databases) ASCII file extraction, triggers, database links, and gateways or for homogeneous databases replication services or table snapshots. In essence, data is shared across the back rooms of the participating applications, bypassing application logic entirely. Some higher-end data-integration tools provide centralized administration support for the scheduling and movement of data, supporting a bit more enterprise control and management for point-to-point data-integration chores.
* *Application integration* (sometimes also called functional integration) can be described as building new business solutions by gluing applications together through the use of some common middleware. Middleware is a class of software that is application independent, providing a means to capture, route, and execute transactional messages between applications. In general, connectors or adapters are used to connect the participating applications to the integration network, and brokers are used to route messages according to publication and subscription rules.

Point-to-Point versus Hub-and-Spoke

If your near real-time data warehouse is also supporting some degree of application (or functional) integration, an important factor in selecting an architecture is the number of publishing and subscribing systems that you anticipate supporting in the foreseeable (for example, 24 month) future. This number can help you decide if a relatively simple point-to-point solution will suffice or if a more robust hub-and-spoke architecture will be required.

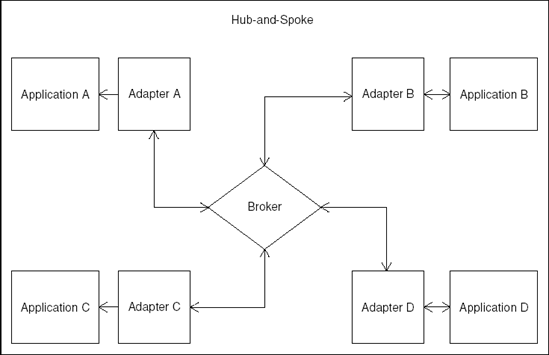


**Figure 11.3. Point-to-point application integration**

[Figure 11.3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#point-to-point_application_integration) shows that, even with a relatively small number of applications exchanging data, point-to-point solutions can demand a very large number of data-exchange interfaces, each of which requires maintenance whenever its source or target applications change.

Adding applications to the integration network also demands new data-exchange interfaces to all publishing and subscribing applications. Nonetheless, organizations that have a short, crisp list of applications demanding conformed dimension integration and that expect this list to remain stable for the foreseeable future might find a point-to-point integration approach quite attractive. It avoids the complexity of creating EAI middleware components and can be partially supported through the use of data-integration technologies such as Capture, Transform, and Flow (CTF) tools described later in the chapter.

In contrast to point-to-point architectures, the number of customer interfaces and cross-system dependencies can be minimized through the use of a hub-and-spoke integration approach (see [Figure 11.4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#hub_and_spoke_application_integration)).



**Figure 11.4. Hub and spoke application integration**

However, the additional burden of building EAI middleware adapters and broker components is not trivial. Each application that participates in the integration network needs an adapter capable of converting specified transactions into generic messages and of interpreting and executing generic messages on the local application. Adapter maintenance is required whenever an associated host application changes or if the set of generic messages changes.

Hard-and-fast rules on the decision boundary between point-to-point or hub-and-spoke architectures do not exist, but organizations anticipating integration across three or more applications or those that expect a growing number of integration-network participants in the foreseeable future can strongly consider hub-and-spoke EAI architectures. This admittedly elastic boundary can also be shaped by the organization's comfort with and commitment to EAI messaging technologies.

Customer Data Cleanup Considerations

If the organization needs real-time cleanup and synchronization of customer data, you need to consider some additional factors in selecting an approach. Does the organization have in place some centralized means of generating new customer keys, one that ensures that no redundant customer records are created? Such systems are quite rare, and it often falls upon the data warehouse customer dimension manager to provide this service for the enterprise.

Assuming that such a system is not in place, it may be appropriate for the real-time customer dimension manager to assume responsibility for matching (deduplicating) customer records. A number of deterministic and probabilistic matching tools available today can help support these requirements, but unfortunately, many of these tools currently run in batch mode only. Customer cleanup utilities that support postal address verification, propensity for fraud segmentation, credit worthiness, or householding might also demand batch processing. It is still possible to approximate real-time performance, however, by building an architecture for moving frequent microbatches, described later in the chapter, into and out of these utilities throughout the day.

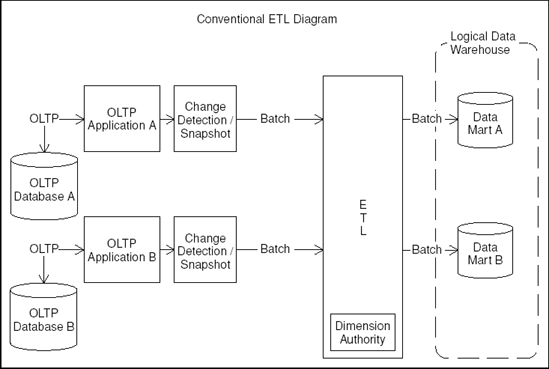
Real-Time ETL Approaches

Through some creative recycling of established ETL technologies and tools, a mature and broad palette of technologies is available to address real-time data warehousing requirements. The sections that follow discuss these technologies.

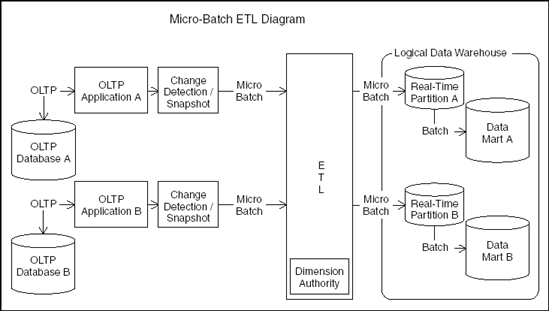
Microbatch ETL

Conventional ETL, the file-based approach described throughout this book, is extremely effective in addressing daily, weekly, and monthly batch-reporting requirements. New or changed transactions (fact records) are moved en masse, and dimensions are captured as point-in-time snapshots for each load. Thus, changes to dimensions that occur between batch processes are not available in the warehouse. ETL, therefore, is not a suitable technique for data or application integration for organizations needing low-latency reporting or for organizations that need more detailed dimensional change capture. But conventional ETL is a simple, direct, and tried-and-true method for organizations that have more casual latency requirements and complex integration challenges. [Figure 11.5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#conventional_etl_diagram) shows the conventional ETL process.

Microbatch ETL is very similar to conventional ETL, except that the frequency of batches is increased, perhaps to as frequently as hourly. These frequent microbatches are run through an otherwise conventional ETL process and directly feed the real-time partitions of the data marts. Once each day, the real-time partitions are copied to the static data marts and are emptied. [Figure 11.6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#microbatch_etl_diagram) shows a diagram of micro-batch ETL.



**Figure 11.5. Conventional ETL diagram**



**Figure 11.6. Microbatch ETL diagram**

The dimension manager systems generate new dimensional images in Type 2 or 3 slowly changing dimensions, but due to the increased run frequency, dimensions that change throughout the day may become rapidly changing and grow deep. An inelegant alternative is to ignore changes to dimensions that occur during the day and instead generate dimension records only for new instances, using default values in all columns. This compromise might suffice for organizations that generate few new records on a given day and are tolerant of dimensional context latency from the previous evening, but it clearly dilutes some of the benefits of real-time reporting. If unavoidable, the only practical solution for dealing with rapidly changing dimensions is the judicious use of a minidimension, where you create separate dimensions for the most frequently changing attributes of a large dimension and thereby reduce the number of new dimensional instances needing to be created by the ETL process.

NOTE

An interesting hybrid alternative is to treat intra-day changes to a dimension as a kind of Type 1, where a special copy of the dimension is associated with the real-time partition exclusively. Changes during the day trigger simple overwrites. At the end of the day, any such changes can be treated as Type 2 as far as the copy of the dimension in the static portion of the data warehouse is concerned. That way, for instance, a credit-worthiness indicator could be set immediately for a customer in the real-time data warehouse.

Microbatch ETL demands a comprehensive job control, scheduling, dependency, and error-mitigation method, one robust enough to run unattended for most of the time and capable of executing data warehouse publication strategies in the face of most common data-loading issues. A number of job-control utilities support this functionality, but custom development work is likely to be needed to make the microbatch ETL data warehouse resemble a *lights out* automated operation.

Microbatch ETL also demands more frequent detection of new and updated transactional records on the OLTP systems, so the load imposed on the operational system must be considered and carefully managed.

Several methods exist for identifying changed record candidates for microbatch ETL load into the real-time data warehouse:

* **Timestamps**. Tried and true, timestamps maintained by the operational system for the creation and update of records can be used by real-time microbatch ETL to differentiate candidate data for extraction. While simple, this method does impose frequent writes of these timestamps on the operational systems for all changes and frequent reads whenever the ETL processes run. Indexing the timestamps improves read performance and reduces read overhead but increases the operational overhead on INSERTs and UPDATEs, sometimes prohibitively so. The ETL engineer must balance these concerns.
* **ETL log tables**. Another approach is to create triggers in the OLTP environment to insert the unique legacy identifiers of new and changed records into a series of special ETL log tables. These specialized tables exist solely to speed ETL processing and are used by the microbatch ETL process to determine which rows have changed since the previous microbatch. The ETL log tables contain the unique identifier of the new or changed dimensional record and perhaps a status value, a timestamp, and a run identifier of the microbatch ETL process that ultimately processes the changed record. The microbatch ETL process joins the ETL log tables to the operational tables where the ETL run identifier is null, extracts the resultant rows, and then deletes (or populates the run identifier of) the ETL Log records extracted. The overhead on the operational system is reduced using this method, because trigger-driven INSERTs do not unduly exercise the OLTP system.
* **Database management system (DBMS) log scrapers**. The DBMS audit log files, created as a byproduct of backup and recovery utilities, can sometimes be utilized to identify new and changed transactions by using specialized utilities called log scrapers. Some of these log-scraping utilities can selectively extract and recreate the SQL statements applied to the database tables of interest since some specified point in time, allowing the ETL to know not just which records have changed since the last extraction, but what elements have changed on these records as well, information that can be utilized by the ETL process in directly applying changes to the target tables in the staging area.
* **Network sniffers**. These utilities monitor some set of interesting traffic on a network and filter and record the traffic that they see. Network sniffers are often used for capturing Web Clickstream traffic because they eliminate the need to stitch together the Web logs from multiple servers in a Web farm, provide sessionizing of Web visits, and improve visibility into the actual Web content delivered from dynamic Web pages. Network sniffers are an ETL alternative wherever there is a stream of traffic requiring data-warehousing analysis, including telecommunication calls routing, manufacturing floor workflow, or EAI messaging traffic.

Microbatch ETL is an excellent choice for data warehouse requirements tolerant of hourly latency without intra-hour dimensional updates and that do not demand bi-directional synchronization of dimensional data between the data warehouse and the operational systems. It is by far the simplest approach for delivering near real-time data warehousing reporting.

Enterprise Application Integration

At the high end of the complexity spectrum lies enterprise application integration (EAI), sometimes called functional integration. EAI describes the set of technologies that support true application integration, allowing individual operational systems to interoperate in new and potentially different ways than they were originally designed.

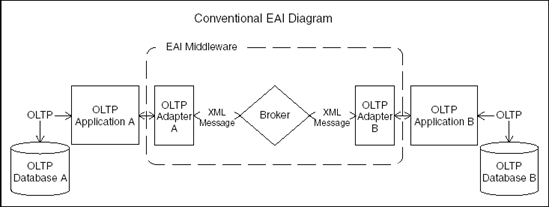
EAI typically entails building a set of adapter and broker components that move business transactions, in the form of *messages*, across the various systems in the integration network, insulating all systems from knowledge or dependencies on other systems in the integration network. Application-specific adapters are responsible for dealing with all of the logic needed to create and execute messages, and brokers are responsible for routing the messages appropriately, based on publication and subscription rules.

Adapters and brokers communicate via application-independent messages, often rendered in XML. When a significant application event occurs such as the update of a customer record, a trigger is fired, and the application's adapter creates a new message. The adapter is also responsible for initiating transactions in its respective application when it receives a message containing information that it has chosen to receive, such as a newly conformed customer record from the customer dimension manager system. Brokers route messages between adapters, based on a set of publication and subscription rules. Messaging queues are often placed between applications and their adapters, and between adapters and brokers, to provide a staging area for asynchronous messaging and to support delivery guarantees and transaction consistency across the integration network.

In the [Figure 11.7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#conventional_eai_diagram), applications A and B operate independently but are able to exchange data and interoperate through EAI messages.

For example, changes to a customer record on application A might fire a trigger detected by application A's adapter, which creates and sends an XML message of the change to a broker. If application B has subscribed to customer-change messages from application A, the broker forwards the message to application B's adapter, which can then apply all or a subset of the customer record change to application B.

Applications A and B do not need to know anything about one another; their respective adapters are responsible for capturing, interpreting, and applying messages for their application. This concept is a powerful one because it allows EAI networks to extend elegantly; introduction of new applications into the integration network requires only the creation of a new adapter and new publication/subscription rules to the broker(s). When we say *only*, we do not imply that the creation of hardened industrial-strength adapters is trivial; quite the opposite is true. Each adapter must be capable of executing potentially complex transactions on its host system and gracefully handling the many concurrency issues that can appear whenever independent applications operate on common logical data. Regardless of the integration approach, certain issues must be dealt with somewhere in the architecture, and EAI adapters do so at the optimal position, as close to the application as is possible.



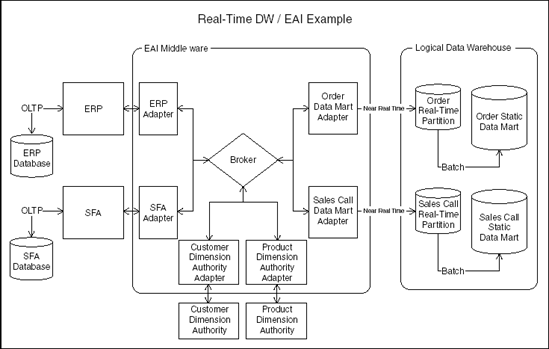
**Figure 11.7. Conventional EAI diagram**

EAI technologies can be powerful enabling tools for the real-time data warehouse because they support the ability to synchronize important data like customer information across applications, and they provide an effective means for distributing data-warehouse-derived information assets, such as new customer segmentation values, across the enterprise.

The real-time EAI data warehouse architecture modularizes the monolithic ETL block by pulling the dimension manager system(s) out as separate architectural components, each with its own adapters, and placing responsibility for most of the transformation and loading chores of the data mart real-time partitions on the data mart adapters. [Figure 11.8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#real-time_data_warehouse_eai_diagram) is a diagram of a real-time EAI data warehouse.

A typical real scenario might involve implementing adapters for a set of OLTP systems such as enterprise resource planning, ERP, and sales-force automation, customer and product dimension manager systems (which perform real-time cleansing and deduplication), and data marts for orders and sales calls.

Any customer or product-change transaction would be captured from the OLTP application by an adapter, sent as a nonconformed dimension message to the broker, which then routes it to whichever systems subscribe to nonconformed dimension messages, typically only the appropriate dimension manager system. The dimension manager system conforms the dimensional record, and its adapter then sends it back as a conformed dimension message to the broker, which then forwards it to all systems that subscribe to conformed dimension data, typically the OLTP systems and data marts.



**Figure 11.8. Real-time data warehouse EAI diagram**

Consider this example. The ERP system updates a customer record; then the ERP adapter detects this change, generates an XML message labeled *Non-Conformed Customer Transaction from ERP*, and sends it to the broker. The broker forwards this message to the customer dimension manager, typically the only system that subscribes to nonconformed customer messages from system ERP. The customer dimension manager receives the message and places the nonconformed customer information in the work queue (or staging area, if micro-batch) of the customer dimension manager. The customer dimension manager works the transaction, and if it results in a change to one or more conformed customer records, the customer dimension manager adapter detects that these changes have occurred, packages these revised customer records into *Conformed Customer Transactions from the Customer Dimension Manager* messages, and sends them to the broker. Assuming that the orders data mart, sales-call data mart, ERP, and SFA systems have all subscribed to *Conformed Customer Transactions from the Customer Dimension Manager* messages, the broker copies and distributes the message to all four of these systems. Each of the four adapters is then responsible for applying the changes to the customer record to their respective applications.

The acceptance of the conformed customer record by the ERP and SFA systems might cause a change to their respective customer records, thereby triggering a new set of EAI transactions.

NOTE

Endless loops, or *race conditions*, must be avoided by devising a selective publication strategy at the edges of the integration network, either at the OLTP systems or the dimension manager systems.

Fact transactions are also captured by the OLTP adapters, sent to the broker as an *Order or Sales Call*fact message, and then routed to all subscribers of these types of messages, typically data marts. The data mart adapter performs all transformations needed and inserts the new transaction directly into the data mart real-time partition.

EAI is a powerful means of synchronizing key business information, both for trickle-feeding data marts and for publishing and distributing data warehouse-derived segmentations to customer-facing OLTP systems. But it can be complex and expensive to implement.

EAI is an excellent approach for organizations whose requirements demand low-reporting latency, who are intolerant of loss of intra-day dimensional updates, or who require bidirectional synchronization of dimension data between the data warehouse and/or the operational systems.

NOTE

Using EAI mechanisms for shoveling high-volume transaction data into the data warehouse may be inefficient if every transaction is separately packaged as an EAI message with significant communications overhead. Before commiting to this design approach, make sure you are anticipating the full volume of message traffic. Also, investigate whether your EAI broker interfaces allow for compact representations of transactional data.

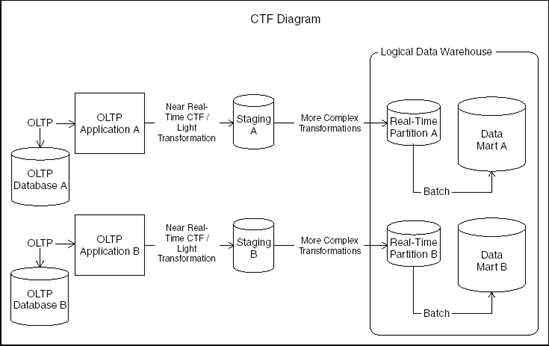
Capture, Transform, and Flow

Capture, Transform, and Flow (CTF) is a relatively new category of data-integration tools designed to simplify the movement of real-time data across heterogeneous database technologies. The application layer of the transactional applications is bypassed. Instead, direct database-to-database exchanges are executed. Transactions, both new facts and dimension changes, can be moved directly from the operational systems to the data warehouse staging tables with low latency, typically a few seconds.

The transformation functionality of CTF tools is typically basic in comparison with today's mature ETL tools, so often real-time data warehouse CTF solutions involve moving data from the operational environment, lightly transforming it using the CTF tool, and then staging it. These light transformation tasks might include standardization of date formats, recasting data types, truncating or expanding field lengths, or applying application-specific code translations into descriptions. Once data is staged, additional transformations beyond the capabilities of the CTF tool are then applied as needed. This subsequent transformation can be invoked either by microbatch ETL or via triggers that fire on INSERT in the staging area. In either transformation scenario, records are then written directly into the real-time partition tables of the data mart. These subsequent transformations might include tasks like data validation, dimensions-record cleansing and matching, surrogate key lookups for dimensional records, and creation of new slowly changing dimensional records as needed. [Figure 11.9](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#ctf_diagram) diagrams CTF.

Some CTF tools can also simplify the batch movement of information from the data warehouse back to the operational systems, such as periodic refreshment of customer data. Because they bypass the application-logic layer, utilizing these features places the burden on the CTF administrator to ensure that the resultant updates do not corrupt operational system transactions or cause transaction loss.

CTF is an excellent approach for organizations whose requirements demand near real-time reporting with some light data-integration needs and for organizations whose core applications enjoy common periods of low activity that can allow minimally disruptive data synchronization to occur. In situations like these, CTF can offer a compelling blend of the some of the benefits of EAI, while avoiding much its complexity.



**Figure 11.9. CTF diagram**

Enterprise Information Integration

Enterprise Information Integration (EII) is another relatively new category of software, built specifically to assist organizations in quickly adding real-time reporting capabilities to their business-intelligence systems. They are, in a sense, a virtual real-time data warehouse, a logical view of the current data in the OLTP systems, presented to the business user in a structure appropriate for analysis, and delivered on the fly via inline ETL transformation.

With conventional ETL, you identify a set of source structures in your OLTP world, a set of target structures in your data warehouse. Then on some schedule, perhaps nightly, a trigger is pulled, and the ETL tool extracts the data, transforms it, and loads it into data warehouse tables. EII operates in a somewhat similar vein, except that instead of a data warehouse, the target might be a report, spreadsheet, or OLE DB or XML object. The EII trigger is pulled by the business analyst whenever he or she needs up-tothe-second operational information. The EII system actually generates a series of queries, typically via SQL, at the moment requested, applies all specified transformations to the resultant data, and delivers the results to the business user.

The capabilities that emerge are quite interesting: a zero-latency reporting engine enhanced with the robust capabilities for data integration associated with mature ETL tools. No history beyond the data available in the OLTP environment is available in EII, so trending reports must still be met by the data warehouse. Of course, the data warehouse itself can be defined as a source of information to the EII system, so the integration of real-time data from the operational world with the historic data from the data warehouse is at least theoretically possible. EII transformational capabilities, while robust, are not without limits. Not all modern ETL and data-cleansing functionality can be supported inline (for example, probabilistic matching), so expectations must be reduced accordingly. Also, because extractions are directly against the OLTP systems, the frequency and complexity of these extractions must be managed in order to manage the size of the footprint on the OLTP technical architecture.

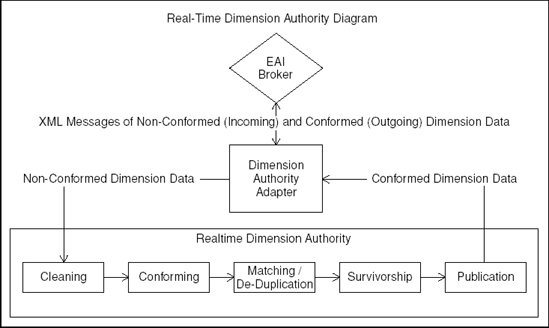
An important strength of EII is its ETL pedigree. EII can be used as an effective data warehouse prototyping device. Organizations that select ETL and EII toolsets from the same vendor might find that successful EII subject areas that need to *evolve* into data warehouse subject areas that can be jump-started by reusing the data-transformation rules developed in the EII tool. EII can also act as a supportive real-time reporting component within an overall data warehouse business intelligence system. Use of conformed dimensions and facts across the data warehouse and EII, as part of the dimensional data warehouse bus architecture, allows these systems to interoperate effectively. Conforming dimensions and facts on the fly, however, is easier said than done. In this scenario, at a minimum, you must place fact constraints on the EII queries to exclude facts that have already been loaded into the data warehouse to avoid double-counting and exercise care in presenting real-time facts associated with new dimension records that have not yet been loaded into the data warehouse.

EII may be a very compelling approach for organizations whose requirements demand near-zero latency real-time reporting of integrated information for a relatively small user base with little historical data. It may also be valuable to organizations that believe that they need to evolve into real-time data warehousing but are unsure of their strategic real-time business requirements or whose business requirements are rapidly changing. And finally, EII may be a compelling choice for organizations in the throes of reorganization or acquisition and need real-time integrated operational reporting as quickly as possible.

The Real-Time Dimension Manager

The real-time dimension manager system, as proposed in this book, used primarily on customer information, converts incoming customer records, which may be incomplete, inaccurate, or redundant, into conformed customer records. *Conformed* does not mean perfect, but it should mean that dimensional records are brought to the best condition that the organization is capable of achieving. In practice, this means that all reasonable measures have been taken to eliminate redundancy, remove untrustworthy data, compile as complete an image as is possible, and assign surrogate data warehouse keys. A general schematic of the real-time dimension manager is presented in [Figure 11.10](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#real-time_dimension_authority_diagram).

The EAI broker is the same EAI middleware component depicted in the other EAI diagrams, and it is responsible for routing messages between adapters, in accordance with its publication and subscription metadata. In the case of the real-time dimension manager, it routes messages associated with nonconformed dimension changes from operational systems to the dimension manager and routes the conformed dimension changes from the dimension manager back to any subscribing operational systems or data marts. The conformed customer messages that come from the real-time dimension manager must also include the set of all legacy keys from the OLTP systems that have been joined together by the conformance process. These keys are then used by the OLTP system adapters in figuring out how to apply the conformed customer message to their respective applications. The resultant changes to the OLTP system can result in the creation of a new record, the update of an existing record, or the *merging* of two or more records that have been deemed redundant by the dimension manager. The legacy keys are also used by the real-time data mart loading software to map the legacy keys that appear on incoming fact records to data warehouse surrogate keys.



**Figure 11.10. Real-Time Dimension manager Diagram**

This continuous exchange of dimensional images is the mechanism for synchronization across all systems that participate in the integration network. The net effect, though, transcends simple synchronization. From the CEO's perspective, the applications now appear to be working together.

Unless planned for, a kind of boomerang effect can develop between systems that participate in the integration network. After a nonconformed dimension record is processed by the dimension manager, the acceptance of the conformed dimensional record by subscribing applications generates messages of dimension change themselves. You must carefully manage this rebound (boomerang) effect and dampen any race conditions, including infinite loops of dimension conformance messaging. The importance of firm EAI architecture is critical to resolving these types of issues; you must establish sound policies governing message publications. Consider the following example policy: Integration network application participants, both OLTP and the dimension manager, might choose to publish messages *only* whenever they update an existing or create a new dimensional record. This policy implies that the dimension manager, for example, does *not* publish conformed messages for all incoming nonconformed messages, just those that result in a change to one or more conformed dimension records. Similarly, an OLTP system receiving a conformed dimension message publishes a new message only if the loading of this conformed image results in a change to its dimension table.

The dimension manager adapter is the EAI middleware component that interacts with the real-time dimension manager system by taking incoming messages of nonconformed data, placing them in a queue for the dimension manager, and listening for publication events triggered by the dimension manager, which it then converts to XML messages and sends on to the broker. The dimension manager adapter insulates the rest of the EAI architecture from any awareness or dependency on the real-time dimension manager system itself.

Referring to [Figure 11.10](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#real-time_dimension_authority_diagram), the business of actually conforming dimensions in real-time is typically modularized into the following subcomponents:

* **Cleaning**. The cleaning component reads incoming nonconformed data and discards dimensional instances that are corrupt or invalid from the job stream. It ensures that required fields are present (which may vary across different sources) and that the data values contained in the attributes are valid—again, from the perspective of the originating system.
* **Conforming**. The conforming component accepts cleaned information from the job stream and performs field-by-field translations of the values from the set of valid values according to the originating system to a conformed domain, a set of enterprise-wide conformed values for each attribute of the dimension. In some cases, the conformed value is arrived at *deterministically*, through lookup tables that map all source system values to conformed values. In other cases, the conformed value is arrived at *probabilistically*, by deriving relationships using fuzzy statistical calculation. Specialized tools for probabilistic standardization are often used to clean up street addresses by putting them in a known format; correcting the spelling of street, city, and state names; and correcting postal zip codes. Some fuzzy tools also correct name misspellings and standardize common names.
* **Matching**. The Matching component accepts cleaned and conformed data from the job stream and attempts to identify and eliminate duplicate records. Again, specialized software tools are often employed to support matching, using deterministic and probabilistic matching techniques. Detailed descriptions of the workings of these tools are beyond the scope of this chapter, but suffice to say that these tools allow developers to define a number of matching scenarios and matching algorithms. The matching engines score the likelihood of a match for each pass and generate a combined score, which is a kind of balanced scorecard for overall likelihood of a match. This final score is then compared to high (certain match) and low (certain no match) thresholds that the dimension manager has defined, and matching group keys are defined. Records that fall between the high and low thresholds are marked for special treatment, typically manual review. The definition of the matching passes and setting of the matching thresholds is part science and part art and is shaped by the organization's tolerance for error, need for low latency, legal and regulatory considerations, and staff available for manual review. Undermatching (tending to err on the side of keeping similar but nonidentical customer records as separate dimensional instances) is generally regarded as more conservative and less intrusive than overmatching and is the norm. Undoing an incorrect match in a real-time EAI environment is not easily accomplished and often means that customer transactions that have been incorrectly consolidated by OLTP systems in response to a merge request from the dimension manager need to be manually split apart and retransmitted.

In a real-time environment, records requiring manual review are typically defaulted to a nonmatch state, and the manual review can then be performed later. For performance reasons, when dealing with large dimensions such as the customer dimension of a retailer, you must restrict the set of candidate records used for deduplication for match performance to meet a reasonable real-time performance obligation. Extracting candidates can speed these matching processes significantly, but it sometimes fails to deliver many candidate records that might have been found to match. Thus, real-time online matching is often a bit of a compromise, and periodic rematching of the entire dimension might be required. This monolithic rematching process can create a large number of conformed dimension messages and operational system updates, which you must managed carefully.

Specialized metadata is needed by the matching component to describe matching pass logic, matching thresholds, and matching override information for records that should never or always be matched. Often, you must develop a specialized user interface to support the manual matching processes and the maintenance of the matching metadata, including pass logic, match thresholds, and candidate extraction.

* **Survivorship**. Once a set of records has been identified as matches for one another, the best image of the dimension must be somehow distilled from the matched records to create a complete and accurate composite image. This distillation process is often referred to as *survivorship* because it ensures that only the best sources of dimensional attributes are survived in the overall dimension conformance process. The survivorship process utilizes business rules that identify, on an attribute-by-attribute basis, the source-system priorities to be applied in surviving the resultant conformed dimension image. Non-null source attribute values are survived into the final integrated dimension record based on a hierarchy of source-system rules captured ideally in metadata. The survivorship component should also support the ability to define groups of attributes that survive from the same source record as a block to avoid strange results when plucking, for example, address line 1 and address line 2 attributes from different source records. The survivorship module also typically handles the generation of distinct point-in-time surrogate keys for dimension records that are slowly changing, while simultaneously maintaining a single key value for all dimensional instances across time. So a customer whose profile has changed ten times must have ten distinct point-in-time surrogate key values, yet each of these should have the same overall customer surrogate key. These two key-handling perspectives are needed because the real-time dimension manager serves two constituencies: the data marts, which must have point-in-time surrogate keys, and the OLTP systems, which need only the most contemporary image of the dimension record.
* **Publication**. Once a dimension image has been fully integrated (cleaned, conformed, matched, and survived), you must determine if the resultant survived record is new or different enough from previous dimensional images to warrant publication to the integration network. The dimension manager typically needs to publish selectively to dampen infinite publication loops. If publication is warranted, the publication component's job is to awaken the dimension manager adapter, which is continuously listening for publication requests, so that it can gather all or a part of the dimensional record, convert it to a conformed dimension XML message, and push it to the EAI broker for distribution throughout the enterprise. Awakening the dimension manager adapter typically takes the form of applying an update to one or more records in a special repository of conformed data, an event which fires a trigger that the adapter is listening for.

Design publication policy to ensure that adequate dimension synchronization is possible throughout the enterprise, while avoiding any endless feedback or race conditions and without delving into application-specific publication and subscription rules, which are best handled by the EAI broker.

The ETL architect designing the real-time dimension manager's responsibilities must carefully dissect business requirements and tread bravely through sometimes difficult political territories. Many managers that seek one version of the truth assume that it will be their version, not the other guy's! Technically, the architect must also decide when and where complete or partial dimension records are passed between applications in the messages, which situations should cause conformed records to be published, how best to deal with possible contention and race conditions, how best to balance the need for application autonomy and conformance, and whether to use straight-through processing with very few disk touchdowns or microbatch processing, discussed in the next section.

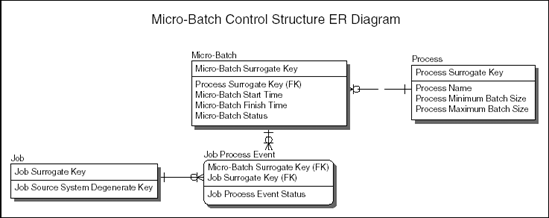
Sound complex? Well, it is. But the real-time dimension manager is truly powerful medicine for the enterprise needing to synchronize enriched customer or other key dimensional information across the enterprise and can provide a competitive advantage to those organizations courageous enough to become early adopters.

Microbatch Processing

You will often face a common dilemma when designing real-time data mart partition or dimensional systems: Should the solution embrace straight-through processing or utilize more frequent microbatches? The current generation of toolsets that support low-latency data movement, such as CTF, often lack some of the transformation capabilities of well-established batch ETL tools. A number of ETL tool vendors have begun to offer real-time versions of their toolsets that process information in a more transactional manner but sometimes with restricted functionality. Designers of the real-time dimension manager systems often face a similar dilemma when selecting deterministic and probabilistic matching tools, some of which operate exclusively operate in batch mode. How can you best coax real-time performance out of tools that operate in batch mode?

One reasonable compromise to the conflicting demands of delivering near real-time performance within constraints imposed by batch-oriented toolsets is to design a solution that processes frequent microbatches, using state transition job control metadata structures.

Consider [Figure 11.11](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#the_microbatch_table_data_model_dot). Each job in the job table represents either a fact or nonconformed dimension transaction record presented for processing to either the real-time data mart or dimension manager. These jobs must pass through several processes, each defined in the process table, such as cleaning and conforming, before they are ready for publication. In the data model in [Figure 11.11](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#the_microbatch_table_data_model_dot), the microbatch table represents the invocation of a small batch of a given process, and each microbatch processes several jobs. The set of jobs processed by each microbatch are captured in the job process event table shown in the figure, which also captures a job event process status attribute of the success or failure of the given job within the microbatch.



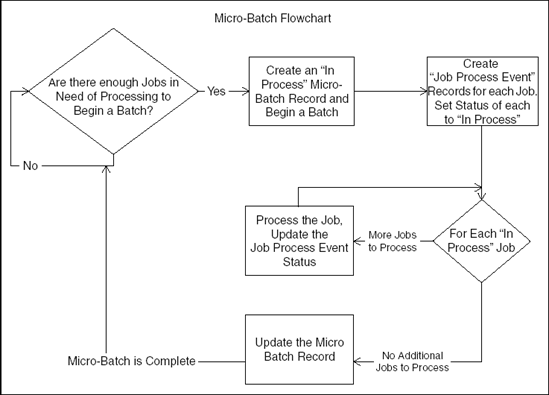
**Figure 11.11. The microbatch table data model**

The processes run continuously, as daemons, looking for jobs that are at the appropriate state for processing. When they find some acceptable minimum number of jobs, they invoke a microbatch, process the jobs, and generate job process event records with appropriate status values; then they continue looking for more jobs to process.

The benefits of building modular process daemons, as opposed to using straight-through processing, is that they can be developed independently, have individually tunable batch-sizing specifications, and be replaced and/or upgraded independently as new toolsets with more features. Also, new processes such as specialized address verification or credit-worthiness scoring can be more easily inserted into the job stream, and jobs that require selective processing are more easily accommodated.

But this flexibility comes at a cost. The microbatch process flow demands that each process have defined and persistent interfaces, typically database tables to pull data from and write to. The additional I/O and complexity imposed by this requirement can be significant. In practice, the complexity can be minimized by designing a single set of work tables used as common sources and targets by all processes, and the I/O can be minimized by caching these tables in the DBMS memory. Nevertheless, the micro-batch approach does not perform as well as a straight-through processing approach. The control records associated with completed jobs should be purged or partitioned frequently to keep them as small as possible.

Each process flow of a typical process in the microbatch scenario is quite simple. As each batch process is invoked, a microbatch record is created, and some appropriate (between minimum and maximum specified batch size from the process table) number of job process event records are created too, one for each job to be processed in the microbatch. This basic technique provides enough information for managing many concurrent microbatches and keeping an adequate audit trail of all batch and job processing. [Figure 11.12](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#microbatch_flowchart_dot) shows how it works.



**Figure 11.12. Microbatch flowchart.**

[Figure 11.12](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#microbatch_flowchart_dot) represents a single process. Each process runs continuously and simultaneously with other processes as daemons, working a mutually exclusive set of jobs to completion, setting the job process event and micro-batch status values accordingly, and then continuing. So, a dimension manager system might have data cleaning, conforming, matching, survivorship, and publication process daemons sometimes working simultaneously on different sets of jobs. A data mart real-time CTF system might have transformation and surrogate key lookup process daemons, and so on. Each daemon is continuously looking for jobs to processes where jobs in this context are defined as job records that have previously been processed to a stage that makes them appropriate candidates for the given process.

As you can see in [Figure 11.12](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#microbatch_flowchart_dot), the status of the job process events is set to *In Process*, and a DBMS transaction begin set point is established. As each job is worked (cleaned, conformed, and so on), the job process event status is updated to either success or failure. Alternatively, to reduce processing overhead, this updating can occur at the end of the batch. Once all jobs have been processed, the batch completes, and the microbatch control table is updated. If it is successful with no fatal failures, and all job process event records have upgraded from *In Process*, a COMMIT is then executed, and the resultant changes are written to the database. If a failure occurs or if an unacceptably high number of job process events are of status Failure, a ROLLBACK is executed and the database returns to the state it was in before the microbatch began.

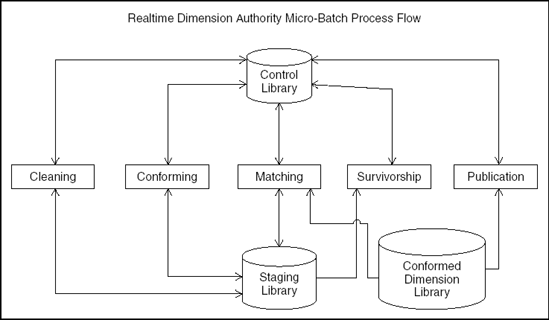
NOTE

ROLLBACK events must not rollback error messages or status values on CONTROL tables. Many DBMSs offer autonomous transaction control options that support this restriction.

Microbatch ETL applied to a real-time dimension manager is shown in [Figure 11.13](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#microbatch_process_flow_for_the_real-tim) as a series of process daemons that read from and update control, staging, and conformed libraries of tables.

Each process updates the status values of the job process event table and modifies and creates data in the staging or conformed dimension libraries.

* Cleaning reads nonvalidated records from the staging library and writes status values only to the control library.
* Conforming reads from cleaned and nonconformed records from staging and writes conformed values back to reserved conformed attributes in staging.
* Matching reads conformed and unmatched records from staging and writes match key values back to staging.



**Figure 11.13. Microbatch process flow for the real-time dimension manager.**

* Survivorship reads matched and nonsurvived records from staging and inserts or updates records in the conformed dimension library.
* Publication reads conformed records from the conformed dimension library and awakens the dimension manager adapter, which then publishes the record to the integration network.

A microbatch ETL system can also be used in concert with CTF for real-time data warehouses that demand more complex data transformations than those supported by CTF tool alone. CTF can be used for near real-time extraction of data from operational systems and light transformation services, dropping data into a staging area. From the staging area, microbatch ETL can be run to handle complex transformations and trickle feed data into the real-time data mart partition tables. From there, real-time reporting is supported, and the normal nightly batch process moves the data to the static data mart tables, emptying the real-time partition tables in the processes, ready for the next day's transactions.

A properly designed microbatch system exhibits good performance, reduced latency, and good scalability due to its high degree of parallel processing. It also supports the ability to logically *insert*jobs at various stages in the job stream by allowing administrators or other processes to create new control records with status values set as needed. This capability can be extremely helpful for dealing with special-case processing that might be needed by the real-time dimension manager: such as injecting manually matched records into the job stream for normal survivorship and publication services. It is a good trick to have in your arsenal for coaxing near real-time-like behavior from batch-processing toolsets.

Choosing an Approach—A Decision Guide

The entire area of real-time data warehousing, at present, can be quite confusing. With so many technologies to choose from, surrounded by so much vendor and analyst hype, and with so few successful case studies from which to draw best practices, selecting an appropriate architecture and approach can be a very daunting task.

The following tables attempt to cut through some of this uncertainty by distilling some of the information discussed in this chapter into guidelines to help you narrow your options. [Table 11.1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#real-time_reporting_decision_guide_matri) is a comparison matrix of the presented approaches for real-time reporting.

[table 11.2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html#dimension_authority_decision_guide) offers a comparison of the approaches presented for real-time dimension manager systems, both those that demand real-time application integration and those that can get by with batch-data integration.

**Table 11.1. Real-Time Reporting Decision Guide Matrix**

|  |  | **EII ONLY** | **EII + STATIC DW** | **ETL** | **CTF** | **CTF-MB-ETL** | **MB-ETL** | **EAI** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **ENTERPRISE INFORMATION INTEGRATION IN PLACE OF REAL-TIME DATA WAREHOUSE** | **ENTERPRISE INFORMATION INTEGRATION IN CONCERT WITH CONVENTIONAL NON-REAL- TIME DATA WAREHOUSE** | **STANDARD ETL PROCESSING** | **CAPTURE, TRANSFORM, FLOW FEEDING REAL-TIME DATA WAREHOUSE** | **CAPTURE, TRANSFORM, FLOW WITH MICRO-BATCH ETL FEEDING REAL-TIME DATA WAREHOUSE** | **MICRO-BATCH ETL FEEDING REAL-TIME DATA WAREHOUSE** | **ENTERPRISE APPLICATION FEEDING INTEGRATION REAL-TIME DATA WAREHOUSE** |
| Historical Data Supported |  | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |  |
| Reporting Data Integration Complexity | Low | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
|  | Moderate | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
|  | High |  |  | ✓ |  | ✓ | ✓ | ✓ |
| Data Freshness/Maximum Latency | 1 Minute | ✓ | ✓ |  |  |  |  | ✓ |
|  | 15 Minutes | ✓ | ✓ |  | ✓ |  |  | ✓ |
|  | 1 Hour | ✓ | ✓ |  | ✓ | ✓ | ✓ | ✓ |
|  | 1 Day | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| This matrix suggests a couple of natural decision boundaries for selecting an approach for real-time reporting, based on reporting latency and integration complexity: | | | | | | | | |
| Organizations facing (or expecting to soon face) significant data-integration challenges should look toward ETL, microbatch ETL, and EAI solutions. These organizations include those with data-integration challenges that demand specialized tools for matching, name and address standardization, postal address verification, householding, and so on. Selecting between an ETL-based approach and an EAI-hybrid approach, in this scenario, should be driven primarily by any anticipated needs for a low-latency real-time dimension manager; EAI technologies are uniquely appropriate for these situations. | | | | | | | | |
| Organizations with, or expecting, low-latency reporting requirements should look toward EII, CTF, and EAI solutions. Again, the jump to the EAI arena is driven primarily by any requirements for the low-latency real-time dimension synchronization features of the real-time dimension manager. | | | | | | | | |

**Table 11.2. Dimension Authority Decision Guide Matrix**

|  |  | **ETL** | **CTF** | **CTF-MB ETL** | **MB ETL** | **EAI** | **EAI—MB-ETL** |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **STANDARD ETL PROCESSING** | **CAPTURE, TRANSFORM, FLOW FEEDING DIMENSION AUTHORITY** | **CAPTURE, TRANSFORM, FLOW WITH MICRO-BATCH ETL FEEDING DIMENSION AUTHORITY** | **MICRO-BATCH ETL FEEDING DIMENSION AUTHORITY** | **ENTERPRISE APPLICATION INTEGRATION FEEDING DIMENSION AUTHORITY** | **ENTERPRISE APPLICATION INTEGRATION AND MICRO-BATCH ETL FEEDING DIMENSION AUTHORITY** |
| Dimension Data Integration Complexity | Low | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
|  | Moderate | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
|  | High | ✓ |  | ✓ | ✓ | ✓ | ✓ |
| Data Freshness / Maximum Latency | 1 Minute |  |  |  |  | ✓ |  |
|  | 15 Minutes |  | ✓ |  |  | ✓ | ✓ |
|  | 1 Hour |  | ✓ | ✓ | ✓ | ✓ | ✓ |
|  | 1Day | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Enterprise Integration | Data Mart Feeds Only | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
|  | Light Data Integration |  | ✓ | ✓ |  | ✓ | ✓ |
|  | Substantial Data Integration |  | ✓ |  |  | ✓ | ✓ |
|  | Application Integration |  |  |  |  | ✓ | ✓ |
| Pure ETL techniques are appropriate for dimension-manager scenarios tolerant of high-latency data integration services to a set of data marts (say, one dimension publication per day). CTF is appropriate for dimension-manager situations that demand low-latency data integration with simpler integration challenges and a small universe of applications to serve. EAI-based approaches are appropriate for dimension manager situations demanding application integration services, such as synchronizing a customer dimension across the enterprise and feeding fresh dimension information to real-time data marts. | | | | | | | |
|  | | | | | | | |

Summary

Real-time ETL is much more than a fad or a new feature. Moving to real-time delivery of data challenges every aspect of the ETL pipeline, both physically and logically. Perhaps the best sound bite for real-time systems is that they replace batch-oriented ETL with streaming ETL. In this chapter, we have presented the state-of-the-art of practical approaches to real-time ETL, and we have pointed out as many of the challenges as we can.

# Chapter 12. Conclusions

Designing and building an ETL system for a data warehouse is an exercise in keeping perspective. This is a typical complex undertaking that demands a comprehensive plan up front. It's easy to start transferring data from a specific source and immediately populate tables that can be queried. Hopefully, end users don't see the results of this prototype because such an effort doesn't scale and can't be managed.

# Deepening the Definition of ETL

We go to considerable lengths in [Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html) to describe the requirements you must surround. These include business needs; compliance requirements; data-profiling results; requirements for such things as security, data integration, data latency, archiving and lineage tracking; and end-user tool delivery. You also must fold in your available skills and your existing legacy licenses. Yes, this is an overconstrained problem.

If you simultaneously keep all these requirements in mind, you must make the BIG decision: Should you buy a comprehensive ETL tool or roll your own with scripts and programs? We've made a serious effort to not bias this book too heavily in either direction, but the bigger the scope and the longer the duration of your project, the more we think a vendor-supplied ETL tool makes sense. Your job is to prepare data, not be a software development manager.

The real value of this book, in our opinion, is the structure we have put on the classic three steps of extract, transform, and load. This book describes a specific set of interwoven techniques that build on each other. This is not a book surveying all possible approaches to building an ETL system! We have expanded the classic three ETL steps into four steps: extract, clean, conform, and deliver. The deliverables of these four steps that uniquely differentiate this book include:

* **Extract:** Methods for choosing the specific original data sources and then combining the logical data map and the data-profiling efforts into a plan for the ETL system. It all begins with the sources. We also suggest specific transformations that take place here rather than in the more traditional cleaning step that follows.
* **Clean:** Schema designs for an error event fact table, an audit dimension, and a series of data-quality screens. We show how these deliverables are usefully integrated into your ETL system.
* **Conform:** Precise definitions for conformed dimensions and conformed facts (with a full discussion of the dimension manager's responsibilities and the replication and publication strategy for dimensions and facts). Conforming is the basis for what is now being called master data management in the industry.
* **Deliver:** Detailed structural specifications for the full range of dimensional models, including slowly changing dimensions, the major fact table types, and bridge tables for multivalued dimensions and hierarchical structures. We show how to build all the dimensional schema variations, and we provide specific detail for managing surrogate keys in each of these situations.

The deliverables in each of these steps provide the foundation for the ETL metadata. Much of the mystery and difficulty of dealing with ETL metadata can be reduced by promoting metadata to the status of real data. The audit dimension described in the cleaning step captures this perspective directly. Since dimensions always describe the context of measurements, we see that the state of the ETL system at the time of delivering a table is just another kind of context. With this in mind, we gracefully attach variations of the audit dimension to all of the data seen by end users through their familiar tools.

In [Chapter 7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html), which covers development, we take you on a tour of many of the specific transformation steps and utilities you need to build an ETL system. If you chose to roll your own, the code snippets we provided are directly relevant. If you have purchased a vendor's ETL tool suite, most of these steps and utilities show up as tangible transformers in the graphical depiction of your ETL data flow. In the second half of [Chapter 7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html), we give you some guidance on DBMS specific techniques for performing high-speed bulk loads, enforcing referential integrity, taking advantage of parallelization, calculating dimensional aggregates, and troubleshooting performance problems.

In [Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html), which covers operations, we start with a comprehensive discussion of scheduling the jobs in your ETL environment, keeping in mind that each environment has its own unique bottlenecks. We then make suggestions for certain control documents to help you manage the ETL system on a day-to-day basis. These include a datamart release document, an ETL performance-tracking document, and a list of usage metrics. We conclude [Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html) with recommendations for security and archiving architectures.

In [Chapter 11](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html), we open the door to the design of real-time data warehouse systems. Real-time is anything too fast for your current ETL. But more to the point, the migration to a real-time perspective almost always requires a jump from batch-oriented ETL to streaming ETL. When making this jump, it is likely that every step of your ETL system and your end-user tools will need to be redesigned. Obviously, this is a step not to be taken lightly. However, nearly all the important steps of batch-oriented ETL must be addressed in a streaming ETL design. You still need to extract, clean, conform, and deliver. For these reasons, we can use the lessons developed in the first ten chapters as the basis for the real-time design.

# The Future of Data Warehousing and ETL in Particular

IT really has only two complementary missions: Get data in, and get data out. Getting the data in, of course, is transaction processing. Over the last 30 years, organizations have spent more than a trillion dollars building progressively more powerful transaction-processing systems whose job is to capture data for operational purposes. But data cannot be a one-way flow: At some point, we must consume data and derive value from it. There is a profound cultural assumption in the business world that if only we could see all of our data, we could manage our businesses more effectively. This cultural assumption is so deeply rooted that we take it for granted. Yet this is the mission of the data warehouse, and this is why the data warehouse is a permanent entity in all of our organizations, even as it morphs and changes its shape. Viewed in this way, it seems reasonable that in the long run, the overall investment in getting data out will rival that of getting data in.

In the last five years, a number of important themes have become the drivers for data warehousing:

* The honeymoon phase for the data warehouse is over. Businesses have lost their patience for technology, and they are insisting that the data warehouse deliver useful business results. The name, at least for now, of this theme is business intelligence (BI). BI is driven by end users, and BI vendors all control the final screens that the users see.
* The data warehouse has become distinctly operational. The old classic distinction between the data warehouse and operational reporting has disappeared. This operational focus gives rise to two huge requirements for the data warehouse. First, the data warehouse must have access to the atomic transactions of the business. If you want to see if a particular order was shipped, you can't look at aggregated data. Every subject area in the data warehouse must have smooth access to the most atomic data at the individual transaction level. Second, many of the operational views of the business need to be available in real-time. Of course, we've developed the definition and the technical responses to this real-time challenge in depth in this book.
* Businesses expect a 360 degree view of their operations. The lightning rod for the 360 degree view is the customer. Every customer-facing process in the business is expected to be available in the data warehouse, and end users want a single view of the customer list across all these processes. This places an enormous burden on the cleaning and conforming steps of the data warehouse, especially if little thought has been given to rationalizing all the views of customer in the operational systems. Although the customer is the most important dimension driving the 360 degree requirement, products, citizens, and vendors present the same challenges in other environments.
* Finally, the explosion of data continues unabated. Technical advances in data capture (especially RFIDs) and data storage are swamping many of our data warehouses, creating the expectation that every data mote be available for analysis.

So, how will these themes change the nature of the ETL task? In our view, the most important reality is the stunning complexity of developing and running an ETL system. As we've stated, this is an over-constrained problem. Read the list of requirements in [Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html) one more time. As the sheer size of data and the number of software and hardware processes mushrooms, it will become less and less feasible to roll your own system. The future will belong to systems that allow you to assemble high-level building blocks of logic.

## Ongoing Evolution of ETL Systems

Other technology areas have gone through similar phases where thresholds of complexity have simply forced the level of tool integration to be much more comprehensive. Integrated circuit designs with millions of components on each chip and software development with millions of lines of code are examples of this evolution. The development of ETL processing pipelines must inevitably respond in the same way if we are to keep up with the increasing volumes of data flowing in.

This means that the ETL designer must be increasingly oriented toward system integration, system monitoring, and system building block assembly, rather than coding. There simply isn't enough time to program very much at a low level.

The theme of analyzing atomic data ever-more precisely will only accelerate. Micromarketing is already descending to the individual customer level, and marketing analysts will want to perform queries that isolate custom subsets of customers based on very complex combinations of attributes and sequential behavior. We see a hint of the challenges of analyzing sequential behavior in [Chapter 6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html) when we place text facts in a positional time series in the customer dimension. Again, we repeat our fundamental belief that the ETL system must be aware of, and participate in, the nature of key analysis modes such as sequential behavior analysis in order to make end-user applications possible. The ETL system is very much like the kitchen of a fine restaurant: The ETL system must arrange the plate before it is brought out of the kitchen.

Sequential behavior analysis will also create much more pressure to query distributed systems. RFID tags go on journeys through doorways. Each doorway is a data-collection device that records the passage of RFID tags. Sequential behavior analysis is possible only when the separate databases at each doorway can be merged into a single data view. Then the journey of an individual RFID tag and whole groups of tags can be analyzed. This is clearly an integration and conforming challenge. The recent mad cow scare was a great example of these issues. The implanted RFID tags in each cow were already in place. But no one could analyze where the specific cow in question had come from or been because the separate RFID generated databases were not accessible or integrated.

Finally, it is appropriate to return to a theme that underlies the whole approach of this book and, indeed, the authors' careers. The gold coin for the data warehouse is being able to respond to the true business needs of the organization. In the final analysis, the most important characteristics of ETL system designers are business-oriented, higher-level system skills that keep the data warehouse aimed in the right direction and succeed most effectively in delivering the data warehouse mission: getting data out.