19 ETL Subsystems and Techniques

The extract, transformation, and load (ETL) system consumes a disproportionate share of the time and effort required to build a DW/BI environment. Developing the ETL system is challenging because so many outside constraints put pressure on its design: the business requirements, source data realities, budget, processing windows, and skill sets of the available staff. Yet it can be hard to appreciate just why the ETL system is so complex and resource-intensive. Everyone understands the three letters: You get the data out of its original source location (E), you do something to it (T), and then you load it (L) into a final set of tables for the business users to query.

When asked about the best way to design and build the ETL system, many designers say, "Well, that depends." It depends on the source; it depends on limitations of the data; it depends on the scripting languages and ETL tools available; it depends on the staff's skills; and it depends on the BI tools. But the "it depends" response is dangerous because it becomes an excuse to take an unstructured approach to developing an ETL system, which in the worse-case scenario results in an undifferentiated spaghetti-mess of tables, modules, processes, scripts, triggers, alerts, and job schedules. This "creative" design approach should not be tolerated. With the wisdom of hindsight from thousands of successful data warehouses, a set of ETL best practices have emerged. There is no reason to tolerate an unstructured approach.

Careful consideration of these best practices has revealed 34 subsystems are required in almost every dimensional data warehouse back room. No wonder the ETL system takes such a large percentage of the DW/BI development resources!

This chapter is drawn from *The Data Warehouse Lifecycle Toolkit*, *Second Edition* (Wiley, 2008). Throughout the chapter we've sprinkled pointers to resources on the Kimball Group's website for more in-depth coverage of several ETL techniques.

Chapter 19 reviews the following concepts:

- Requirements and constraints to be considered before designing the ETL system
- Three subsystems focused on extracting data from source systems

- Five subsystems to deal with value-added cleaning and conforming, including dimensional structures to monitor quality errors
- Thirteen subsystems to deliver data into now-familiar dimensional structures, such as a subsystem to implement slowly changing dimension techniques
- Thirteen subsystems to help manage the production ETL environment

Round Up the Requirements

Establishing the architecture of an ETL system begins with one of the toughest challenges: rounding up the requirements. By this we mean gathering and understanding all the known requirements, realities, and constraints affecting the ETL system. The list of requirements can be pretty overwhelming, but it's essential to lay them on the table before launching into the development of the ETL system.

The ETL system requirements are mostly constraints you must live with and adapt your system to. Within the framework of these requirements, there are opportunities to make your own decisions, exercise judgment, and leverage creativity, but the requirements dictate the core elements that the ETL system must deliver. The following ten sections describe the major requirements areas that impact the design and development of the ETL system.

Before launching the ETL design and development effort, you should provide a short response for each of the following ten requirements. We have provided a sample checklist (as a note) for each to get you started. The point of this exercise is to ensure you visit each of these topics because any one of them can be a showstopper at some point in the project.

Business Needs

From an ETL designer's view, the business needs are the DW/BI system users' information requirements. We use the term *business needs* somewhat narrowly here to mean the information content that business users need to make informed business decisions. Because the business needs directly drive the choice of data sources and their subsequent transformation in the ETL system, the ETL team must understand and carefully examine the business needs.

NOTE You should maintain a list of the key performance indicators (KPIs) uncovered during the business requirements definition that the project intends to support, as well as the drill-down and drill-across targets required when a business user needs to investigate "why?" a KPI changed.

Compliance

Changing legal and reporting requirements have forced many organizations to seriously tighten their reporting and provide proof that the reported numbers are accurate, complete, and have not been tampered with. Of course, DW/BI systems in regulated businesses, such as telecommunications, have complied with regulatory reporting requirements for years. But certainly the whole tenor of financial reporting has become much more rigorous for everyone.

NOTE In consultation with your legal department or chief compliance officer (if you have one!) and the BI delivery team, you should list all data and final reports subject to compliance restrictions. List those data inputs and data transformation steps for which you must maintain the "chain of custody" showing and proving that final reports were derived from the original data delivered from your data sources. List the data that you must provide proof of security for the copies under your control, both offline and online. List those data copies you must archive, and list the expected usable lifetime of those archives. Good luck with all this. This is why you are paid so well….

Data Quality

Three powerful forces have converged to put data quality concerns near the top of the list for executives. First, the long-term cultural trend that says, "If only I could see the data, then I could manage my business better" continues to grow; today's knowledge workers believe instinctively that data is a crucial requirement for them to function in their jobs. Second, most organizations understand their data sources are profoundly distributed, typically around the world, and that effectively integrating a myriad of disparate data sources is required. And third, the sharply increased demands for compliance mean careless handling of data will not be overlooked or excused.

NOTE You should list those data elements whose quality is known to be unacceptable, and list whether an agreement has been reached with the source systems to correct the data before extraction. List those data elements discovered during data profiling, which will be continuously monitored and flagged as part of the ETL process.

Security

Security awareness has increased significantly in the last few years across IT but often remains an afterthought and an unwelcome burden to most DW/BI teams. The basic rhythms of the data warehouse are at odds with the security mentality; the data warehouse seeks to publish data widely to decision makers, whereas the security interests assume data should be restricted to those with a need to know. Additionally, security must be extended to physical backups. If the media can easily be removed from the backup vault, then security has been compromised as effectively as if the online passwords were compromised.

During the requirements roundup, the DW/BI team should seek clear guidance from senior management as to what aspects of the DW/BI system carry extra security sensitivity. If these issues have never been examined, it is likely the question will be tossed back to the team. That is the moment when an experienced security manager should be invited to join the design team. Compliance requirements are likely to overlap security requirements; it may be wise to combine these two topics during the requirements roundup.

NOTE You should expand the compliance checklist to encompass known security and privacy requirements.

Data Integration

Data integration is a huge topic for IT because, ultimately, it aims to make all systems seamlessly work together. The "360 degree view of the enterprise" is a familiar name for data integration. In many cases, serious data integration must take place among the organization's primary transaction systems before data arrives at the data warehouse's back door. But rarely is that data integration complete, unless the organization has a comprehensive and centralized master data management (MDM) system, and even then it's likely other important operational systems exist outside the primary MDM system.

Data integration usually takes the form of conforming dimensions and conforming facts in the data warehouse. Conforming dimensions means establishing common dimensional attributes across separated databases, so drill-across reports can be generated using these attributes. Conforming facts means making agreements on common business metrics such as key performance indicators (KPIs) across separated databases so these numbers can be compared mathematically by calculating differences and ratios.

NOTE You should use the bus matrix of business processes to generate a priority list for conforming dimensions (columns of the bus matrix). Annotate each row of the bus matrix with whether there is a clear executive demand for the business process to participate in the integration process, and whether the ETL team responsible for that business process has agreed.

Data Latency

Data latency describes how quickly source system data must be delivered to the business users via the DW/BI system. Obviously, data latency requirements have a huge effect on the ETL architecture. Clever processing algorithms, parallelization, and potent hardware can speed up traditional batch-oriented data flows. But at some point, if the data latency requirement is sufficiently urgent, the ETL system's architecture must convert from batch to microbatch or streaming-oriented. This switch isn't a gradual or evolutionary change; it's a major paradigm shift in which almost every step of the data delivery pipeline must be re-implemented.

NOTE You should list all legitimate and well-vetted business demands for data that must be provided on a daily basis, on a many times per day basis, within a few seconds, or instantaneously. Annotate each demand with whether the business community understands the data quality trade-offs associated with their particular choice. Near the end of Chapter 20: ETL System Design and Development Process and Tasks, we discuss data quality compromises caused by low latency requirements.

Archiving and Lineage

Archiving and lineage requirements were hinted at in the previous compliance and security sections. Even without the legal requirements for saving data, every data warehouse needs various copies of old data, either for comparisons with new data to generate change capture records or reprocessing. We recommend staging the data (writing it to disk) after each major activity of the ETL pipeline: after it's been extracted, cleaned and conformed, and delivered.

So when does staging turn into archiving where the data is kept indefinitely on some form of permanent media? Our simple answer is a conservative answer. All staged data should be archived unless a conscious decision is made that specific data sets will never be recovered in the future. It's almost always less problematic to read the data from permanent media than it is to reprocess the data through the ETL system at a later time. And, of course, it may be impossible to reprocess

the data according to the old processing algorithms if enough time has passed or the original extraction cannot be re-created.

And while we are at it, each staged/archived data set should have accompanying metadata describing the origins and processing steps that produced the data. Again, the tracking of this lineage is explicitly required by certain compliance requirements but should be part of every archiving situation.

NOTE You should list the data sources and intermediate data steps that will be archived, together with retention policies, and compliance, security, and privacy constraints.

BI Delivery Interfaces

The final step for the ETL system is the handoff to the BI applications. We take a strong and disciplined position on this handoff. We believe the ETL team, working closely with the modeling team, must take responsibility for the content and structure of the data that makes the BI applications simple and fast. This attitude is more than a vague motherhood statement. We believe it's irresponsible to hand off data to the BI application in such a way as to increase the complexity of the application, slow down the query or report creation, or make the data seem unnecessarily complex to the business users. The most elementary and serious error is to hand across a full-blown, normalized physical model and walk away from the job. This is why we go to such lengths to build dimensional structures that comprise the final handoff.

The ETL team and data modelers need to closely work with the BI application developers to determine the exact requirements for the data handoff. Each BI tool has certain sensitivities that should be avoided and certain features that can be exploited if the physical data is in the right format. The same considerations apply to data prepared for OLAP cubes.

NOTE You should list all fact and dimension tables that will be directly exposed to your BI tools. This should come directly from the dimensional model specification. List all OLAP cubes and special database structures required by BI tools. List all known indexes and aggregations you have agreed to build to support BI performance.

Available Skills

Some ETL system design decisions must be made on the basis of available resources to build and manage the system. You shouldn't build a system that depends on

critical C++ processing modules if those programming skills aren't in-house or can't be reasonably acquired. Likewise, you may be much more confident in building the ETL system around a major vendor's ETL tool if you already have those skills in-house and know how to manage such a project.

Consider the big decision of whether to hand code the ETL system or use a vendor's ETL package. Technical issues and license costs aside, don't go off in a direction that your employees and managers find unfamiliar without seriously considering the decision's long-term implications.

NOTE You should inventory your department's operating system, ETL tool, scripting language, programming language, SQL, DBMS, and OLAP skills so you understand how exposed you are to a shortage or loss of these skills. List those skills required to support your current systems and your likely future systems.

Legacy Licenses

Finally, in many cases, major design decisions will be made implicitly by senior management's insistence that you use existing legacy licenses. In many cases, this requirement is one you can live with because the environmental advantages are clear to everyone. But in a few cases, the use of a legacy license for ETL development is a mistake. This is a difficult position to be in, and if you feel strongly enough, you may need to bet your job. If you must approach senior management and challenge the use of an existing legacy license, be well prepared in making the case, and be willing to accept the final decision or possibly seek employment elsewhere.

NOTE You should list your legacy operating system, ETL tool, scripting language, programming language, SQL, DBMS, and OLAP licenses and whether their exclusive use is mandated or merely recommended.

The 34 Subsystems of ETL

With an understanding of the existing requirements, realities, and constraints, you're ready to learn about the 34 critical subsystems that form the architecture for every ETL system. This chapter describes all 34 subsystems with equal emphasis. The next chapter then describes the practical steps of implementing those subsystems needed for each particular situation. Although we have adopted the industry vernacular, ETL, to describe these steps, the process really has four major components:

- **Extracting**. Gathering raw data from the source systems and usually writing it to disk in the ETL environment before any significant restructuring of the data takes place. Subsystems 1 through 3 support the extracting process.
- **Cleaning and conforming**. Sending source data through a series of processing steps in the ETL system to improve the quality of the data received from the source, and merging data from two or more sources to create and enforce conformed dimensions and conformed metrics. Subsystems 4 through 8 describe the architecture required to support the cleaning and conforming processes.
- **Delivering**. Physically structuring and loading the data into the presentation server's target dimensional models. Subsystems 9 through 21 provide the capabilities for delivering the data to the presentation server.
- **Managing**. Managing the related systems and processes of the ETL environment in a coherent manner. Subsystems 22 through 34 describe the components needed to support the ongoing management of the ETL system.

Extracting: Getting Data into the Data Warehouse

To no surprise, the initial subsystems of the ETL architecture address the issues of understanding your source data, extracting the data, and transferring it to the data warehouse environment where the ETL system can operate on it independent of the operational systems. Although the remaining subsystems focus on the transforming, loading, and system management within the ETL environment, the initial subsystems interface to the source systems for access to the required data.

Subsystem 1: Data Profiling

Data profiling is the technical analysis of data to describe its content, consistency, and structure. In some sense, any time you perform a `SELECT DISTINCT` investigative query on a database field, you are doing data profiling. There are a variety of tools specifically designed to do powerful profiling. It probably pays to invest in a tool rather than roll your own because the tools enable many data relationships to be easily explored with simple user interface gestures. You can be much more productive in the data profiling stages of a project using a tool rather than hand coding all the data content questions.

Data profiling plays two distinct roles: strategic and tactical. As soon as a candidate data source is identified, a light profiling assessment should be made to determine its suitability for inclusion in the data warehouse and provide an early go/no go decision. Ideally, this strategic assessment should occur immediately after

identifying a candidate data source during the business requirements analysis. Early disqualification of a data source is a responsible step that can earn you respect from the rest of the team, even if it is bad news. A late revelation that the data source doesn't support the mission can knock the DW/BI initiative off its tracks (and be a potentially fatal career outcome for you), especially if this revelation occurs months into a project.

After the basic strategic decision is made to include a data source in the project, a lengthy tactical data profiling effort should occur to squeeze out as many problems as possible. Usually, this task begins during the data modeling process and extends into the ETL system design process. Sometimes, the ETL team is expected to include a source with content that hasn't been thoroughly evaluated. Systems may support the needs of the production processes, yet present ETL challenges, because fields that aren't central to production processing may be unreliable and incomplete for analysis purposes. Issues that show up in this subsystem result in detailed specifications that are either 1) sent back to the originator of the data source as requests for improvement or 2) form requirements for the data quality processing described in subsystems 4 through 8.

The profiling step provides the ETL team with guidance as to how much data cleaning machinery to invoke and protects them from missing major project milestones due to the unexpected diversion of building systems to deal with dirty data. Do the data profiling upfront! Use the data profiling results to set the business sponsors' expectations regarding realistic development schedules, limitations in the source data, and the need to invest in better source data capture practices.

Subsystem 2: Change Data Capture System

During the data warehouse's initial historic load, capturing source data content changes is not important because you load all data from a point in time forward. However, many data warehouse tables are so large that they cannot be refreshed during every ETL cycle. You must have a capability to transfer only the relevant changes to the source data since the last update. Isolating the latest source data is called change data capture (CDC). The idea behind CDC is simple enough: Just transfer the data that has changed since the last load. But building a good CDC system is not as easy as it sounds. The key goals for the change data capture subsystem are:

- Isolate the changed source data to allow selective processing rather than a complete refresh.
- Capture all changes (deletions, edits, and insertions) made to the source data, including changes made through nonstandard interfaces.
- Tag changed data with reason codes to distinguish error corrections from true updates.

- Support compliance tracking with additional metadata.
- Perform the CDC step as early as possible, preferably before a bulk data transfer to the data warehouse.

Capturing data changes is far from a trivial task. You must carefully evaluate your strategy for each data source. Determining the appropriate strategy to identify changed data may take some detective work. The data profiling tasks described earlier can help the ETL team make this determination. There are several ways to capture source data changes, each effective in the appropriate situation, including:

Audit Columns

In some cases, the source system includes audit columns that store the date and time a record was added or modified. These columns are usually populated via database triggers that are fired off automatically as records are inserted or updated. Sometimes, for performance reasons, the columns are populated by the source application instead of database triggers. When these fields are loaded by any means other than database triggers, pay special attention to their integrity, analyzing and testing each column to ensure that it's a reliable source to indicate change. If you uncover any NULL values, you must find an alternative approach for detecting change. The most common situation that prevents the ETL system from using audit columns is when the fields are populated by the source application, but the DBA team allows back-end scripts to modify data. If this occurs in your environment, you face a high risk of missing changed data during the incremental loads. Finally, you need to understand what happens when a record is deleted from the source because querying the audit column may not capture this event.

Timed Extracts

With a timed extract, you typically select all rows where the create or modified date fields equal SYSDATE-1, meaning all of yesterday's records. Sounds perfect, right? Wrong. Loading records based purely on time is a common mistake made by inexperienced ETL developers. This process is horribly unreliable. Time-based data selection loads duplicate rows when it is restarted from mid-process failures. This means manual intervention and data cleanup is required if the process fails for any reason. Meanwhile, if the nightly load process fails to run and skips a day, there's a risk that the missed data will never make it into the data warehouse.

Full Diff Compare

A full *diff compare* keeps a full snapshot of yesterday's data, and compares it, record by record, against today's data to find what changed. The good news is this technique is thorough: You are guaranteed to find every change. The obvious bad news is that, in many cases, this technique is very resource-intensive. If a full diff compare

is required, try to do the comparison on the source machine, so you don't have to transfer the entire table or database into the ETL environment. Of course, the source support folks may have an opinion about this. Also, investigate using cyclic redundancy checksum (CRC) algorithms to quickly tell if a complex record has changed without examining each individual field.

Database Log Scraping

Log scraping effectively takes a snapshot of the database redo log at a scheduled point in time (usually midnight) and scours it for transactions affecting the tables of interest for the ETL load. Sniffing involves a polling of the redo log, capturing transactions on-the-fly. Scraping the log for transactions is probably the messiest of all techniques. It's not uncommon for transaction logs to get full and prevent new transactions from processing. When this happens in a production transaction environment, the knee-jerk reaction from the responsible DBA may be to empty the log so that business operations can resume, but when a log is emptied, all transactions within them are lost. If you've exhausted all other techniques and find log scraping is your last resort for finding new or changed records, persuade the DBA to create a special log to meet your specific needs.

Message Queue Monitoring

In a message-based transaction system, the queue is monitored for all transactions against the tables of interest. The contents of the stream are similar to what you get with log sniffing. One benefit of this process is relatively low overhead, assuming the message queue is already in place. However, there may be no replay feature on the message queue. If the connection to the message queue is lost, you lose data.

Subsystem 3: Extract System

Obviously, extracting data from the source systems is a fundamental component of the ETL architecture. If you are extremely lucky, all the source data will be in a single system that can be readily extracted using an ETL tool. In the more common situation, each source might be in a different system, environment, and/or DBMS.

The ETL system might be expected to extract data from a wide variety of systems involving many different types of data and inherent challenges. Organizations needing to extract data from mainframe environments often run into issues involving COBOL copybooks, EBCDIC to ASCII conversions, packed decimals, redefines, OCCURS fields, and multiple and variable record types. Other organizations might need to extract from sources in relational DBMS, flat files, XML sources, web logs, or a complex ERP system. Each presents a variety of possible challenges. Some sources, especially older legacy systems, may require the use of different procedural languages than the ETL tool can support or the team is experienced with. In this

situation, request that the owner of the source system extract the data into a flat file format.

NOTE Although XML-formatted data has many advantages because it is self-describing, you may not want it for large, frequent data transfers. The payload portion of a typical XML formatted file can be less than 10 percent of the total file. The exception to this recommendation could be where the XML payload is a complex deeply hierarchical XML structure, such as an industry standard data exchange. In these cases, the DW/BI team must decide whether to "shred" the XML into a large number of destination tables or persist the XML structure within the data warehouse. Recent advances in RDBMS vendors' support for XML via XPath have made this latter option feasible.

There are two primary methods for getting data from a source system: as a file or a stream. If the source is an aging mainframe system, it is often easier to extract into files and then move those files to the ETL server.

NOTE If the source data is unstructured, semistructured, or even hyperstructured "big data," then rather than loading such data as an un-interpretable RDBMS "blob," it is often more effective to create a MapReduce/Hadoop extract step that behaves as an ETL fact extractor from the source data, directly delivering loadable RDBMS data.

If you use an ETL tool and the source data is in a database (not necessarily an RDBMS), you may set up the extract as a stream where the data flows out of the source system, through the transformation engine, and into the staging database as a single process. By contrast, an extract to file approach consists of three or four discrete steps: Extract to the file, move the file to the ETL server, transform the file contents, and load the transformed data into the staging database.

NOTE Although the stream extract is more appealing, extracts to file have some advantages. They are easy to restart at various points. As long as you save the extract file, you can rerun the load without impacting the source system. You can easily encrypt and compress the data before transferring across the network. Finally, it is easy to verify that all data has moved correctly by comparing file row counts before and after the transfer. Generally, we recommend a data transfer utility such as FTP to move the extracted file.

Data compression is important if large amounts of data need to be transferred over a significant distance or through a public network. In this case, the communications

link is often the bottleneck. If too much time is spent transmitting the data, compression can reduce the transmission time by 30 to 50 percent or more, depending on the nature of the original data file.

Data encryption is important if data is transferred through a public network, or even internally in some situations. If this is the case, it is best to send everything through an encrypted link and not worry about what needs to be secure and what doesn't. Remember to compress before encrypting because encrypted files do not compress very well.

Cleaning and Conforming Data

Cleaning and conforming data are critical ETL system tasks. These are the steps where the ETL system adds value to the data. The other activities, extracting and delivering data, are obviously necessary, but they simply move and load the data. The cleaning and conforming subsystems actually change data and enhance its value to the organization. In addition, these subsystems can be architected to create metadata used to diagnosis what's wrong with the source systems. Such diagnoses can eventually lead to business process reengineering initiatives to address the root causes of dirty data and improve data quality over time.

Improving Data Quality Culture and Processes

It is tempting to blame the original data source for any and all errors that appear downstream. If only the data entry clerks were more careful! We are only slightly more forgiving of keyboard-challenged salespeople who enter customer and product information into their order forms. Perhaps you can fix data quality problem by imposing constraints on the data entry user interfaces. This approach provides a hint about how to think about fixing data quality because a technical solution often avoids the real problem. Suppose Social Security number fields for customers were often blank or filled with garbage on an input screen. Someone comes up with brilliant idea to require input in the 999-99-9999 format, and to cleverly disallow nonsensical entries such as all 9s. What happens? The data entry clerks are forced to supply valid Social Security numbers to progress to the next screen, so when they don't have the customer's number, they type in an artificial number that passes the roadblock.

Michael Hammer, in his revolutionary book *Reengineering the Corporation* (Collins, revised 2003), struck the heart of the data quality problem with a brilliant observation. Paraphrasing Hammer: "Seemingly small data quality issues are, in reality, important indications of broken business processes." Not only does this insight correctly focus your attention on the source of data quality problems, but it also shows you the way to the solution.

Technical attempts to address data quality will not prevail unless they are part of an overall quality culture that must come from the top of an organization. The famous Japanese car manufacturing quality attitude permeates every level of those organizations, and quality is embraced enthusiastically by all levels, from the CEO to the assembly line worker. To cast this in a data context, imagine a company such as a large drugstore chain, where a team of buyers contracts with thousands of suppliers to provide the inventory. The buyers have assistants, whose job it is to enter the detailed descriptions of everything purchased by the buyers. These descriptions contain dozens of attributes. But the problem is the assistants have a deadly job and are judged on how many items they enter per hour. The assistants have almost no awareness of who uses their data. Occasionally, the assistants are scolded for obvious errors. But more insidiously, the data given to the assistants is itself incomplete and unreliable. For example, there are no formal standards for toxicity ratings, so there is significant variation over time and over product categories for this attribute. How does the drugstore improve data quality? Here is a nine-step template, not only for the drugstore, but for any organization addressing data quality:

- Declare a high-level commitment to a data quality culture.
- Drive process reengineering at the executive level.
- Spend money to improve the data entry environment.
- Spend money to improve application integration.
- Spend money to change how processes work.
- Promote end-to-end team awareness.
- Promote interdepartmental cooperation.
- Publicly celebrate data quality excellence.
- Continuously measure and improve data quality.

At the drugstore, money needs to be spent to improve the data entry system, so it provides the content and choices needed by the buyers' assistants. The company's executives need to assure the buyers' assistants that their work is important and affects many decision makers in a positive way. Diligent efforts by the assistants should be publicly praised and rewarded. And end-to-end team awareness and appreciation of the business value derived from quality data is the final goal.

Subsystem 4: Data Cleansing System

The ETL data cleansing process is often expected to fix dirty data, yet at the same time the data warehouse is expected to provide an accurate picture of the data as it was captured by the organization's production systems. Striking the proper balance between these conflicting goals is essential.

One of our goals in describing the cleansing system is to offer a comprehensive architecture for cleansing data, capturing data quality events, as well as measuring

and ultimately controlling data quality in the data warehouse. Some organizations may find this architecture challenging to implement, but we are convinced it is important for the ETL team to make a serious effort to incorporate as many of these capabilities as possible. If you are new to ETL and find this a daunting challenge, you might well wonder, "What's the minimum I should focus on?" The answer is to start by undertaking the best possible data profiling analysis. The results of that effort can help you understand the risks of moving forward with potentially dirty or unreliable data and help you determine how sophisticated your data cleansing system needs to be.

The purpose of the cleansing subsystems is to marshal technology to support data quality. Goals for the subsystem should include:

- Early diagnosis and triage of data quality issues
- Requirements for source systems and integration efforts to supply better data
- Provide specific descriptions of data errors expected to be encountered in ETL
- Framework for capturing all data quality errors and precisely measuring data quality metrics over time
- Attachment of quality confidence metrics to final data

Quality Screens

The heart of the ETL architecture is a set of quality screens that act as diagnostic filters in the data flow pipelines. Each quality screen is a test. If the test against the data is successful, nothing happens and the screen has no side effects. But if the test fails, then it must drop an error event row into the error event schema and choose to either halt the process, send the offending data into suspension, or merely tag the data.

Although all quality screens are architecturally similar, it is convenient to divide them into three types, in ascending order of scope. Jack Olson, in his seminal book *Data Quality*: *The Accuracy Dimension* (Morgan Kaufmann, 2002), classified data quality screens into three categories: column screens, structure screens, and business rule screens.

Column screens test the data within a single column. These are usually simple, somewhat obvious tests, such as testing whether a column contains unexpected null values, if a value falls outside of a prescribed range, or if a value fails to adhere to a required format.

Structure screens test the relationship of data across columns. Two or more attributes may be tested to verify they implement a hierarchy, such as a series of many-to-one relationships. Structure screens also test foreign key/primary key relationships between columns in two tables, and also include testing whole blocks of columns to verify they implement valid postal addresses.

Business rule screens implement more complex tests that do not fit the simpler column or structure screen categories. For example, a customer profile may be tested for a complex time-dependent business rule, such as requiring a lifetime platinum frequent flyer to have been a member for at least five years and have flown more than 2 million miles. Business rule screens also include aggregate threshold data quality checks, such as checking to see if a statistically improbable number of MRI examinations have been ordered for minor diagnoses like a sprained elbow. In this case, the screen throws an error only after a threshold of such MRI exams is reached.

Responding to Quality Events

We have already remarked that each quality screen has to decide what happens when an error is thrown. The choices are: 1) halting the process; 2) sending the offending record(s) to a suspense file for later processing; and 3) merely tagging the data and passing it through to the next step in the pipeline. The third choice is by far the best choice, whenever possible. Halting the process is obviously a pain because it requires manual intervention to diagnose the problem, restart or resume the job, or abort completely. Sending records to a suspense file is often a poor solution because it is not clear when or if these records will be fixed and re-introduced to the pipeline. Until the records are restored to the data flow, the overall integrity of the database is questionable because records are missing. We recommend not using the suspense file for minor data transgressions. The third option of tagging the data with the error condition often works well. Bad fact table data can be tagged with the audit dimension, as described in subsystem 6. Bad dimension data can also be tagged using an audit dimension, or in the case of missing or garbage data can be tagged with unique error values in the attribute itself.

Subsystem 5: Error Event Schema

The error event schema is a centralized dimensional schema whose purpose is to record every error event thrown by a quality screen anywhere in the ETL pipeline. Although we focus on data warehouse ETL processing, this approach can be used in generic data integration (DI) applications where data is being transferred between legacy applications. The error event schema is shown in Figure 19-1.

The main table is the error event fact table. Its grain is every error thrown (produced) by a quality screen anywhere in the ETL system. Remember the grain of a fact table is the physical description of why a fact table row exists. Thus every quality screen error produces exactly one row in this table, and every row in the table corresponds to an observed error.

The dimensions of the error event fact table include the calendar date of the error, the batch job in which the error occurred, and the screen that produced the error. The calendar date is not a minute and second time stamp of the error,

but rather provides a way to constrain and summarize error events by the usual attributes of the calendar, such as weekday or last day of a fiscal period. The error date/time fact is a full relational date/time stamp that specifies precisely when the error occurred. This format is useful for calculating the time interval between error events because you can take the difference between two date/time stamps to get the number of seconds separating events.

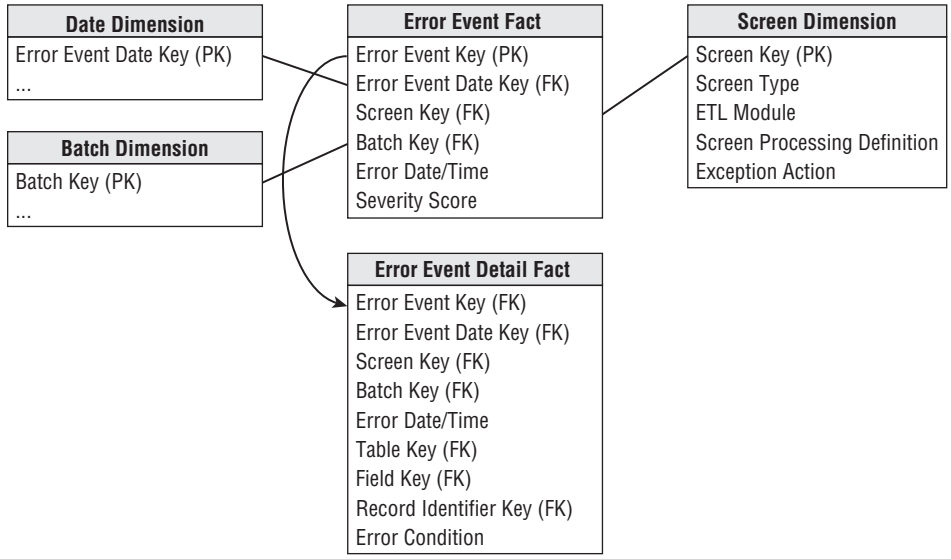

Figure 19-1: Error event schema.

The batch dimension can be generalized to be a processing step in cases in which data is streamed, rather than batched. The screen dimension identifies precisely what the screen criterion is and where the code for the screen resides. It also defines what to do when the screen throws an error. (For example, halt the process, send the record to a suspense file, or tag the data.)

The error event fact table also has a single column primary key, shown as the error event key. This surrogate key, like dimension table primary keys, is a simple integer assigned sequentially as rows are added to the fact table. This key column is necessary in those situations in which an enormous burst of error rows is added to the error event fact table all at once. Hopefully this won't happen to you.

The error event schema includes a second error event detail fact table at a lower grain. Each row in this table identifies an individual field in a specific record that participated in an error. Thus a complex structure or business rule error that triggers a single error event row in the higher level error event fact table may generate many rows in this error event detail fact table. The two tables are tied together by the error event key, which is a foreign key in this lower grain table. The error event detail

table identifies the table, record, field, and precise error condition. Thus a complete description of complex multi-field, multi-record errors is preserved by these tables.

The error event detail table could also contain a precise date/time stamp to provide a full description of aggregate threshold error events where many records generate an error condition over a period of time. You should now appreciate that each quality screen has the responsibility for populating these tables at the time of an error.

Subsystem 6: Audit Dimension Assembler

The audit dimension is a special dimension that is assembled in the back room by the ETL system for each fact table, as we discussed in Chapter 6: Order Management. The audit dimension in Figure 19-2 contains the metadata context at the moment when a specific fact table row is created. You might say we have elevated metadata to real data! To visualize how audit dimension rows are created, imagine this shipments fact table is updated once per day from a batch file. Suppose today you have a perfect run with no errors flagged. In this case, you would generate only one audit dimension row, and it would be attached to every fact row loaded today. All the categories, scores, and version numbers would be the same.

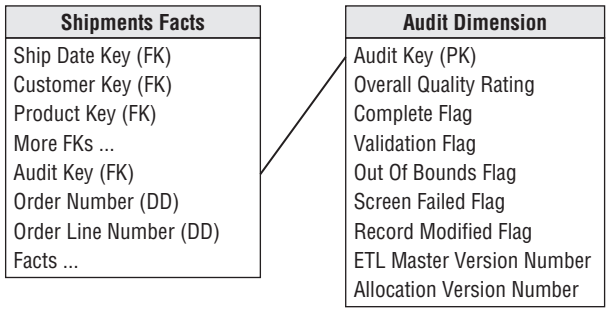

Figure 19-2: Sample audit dimension attached to a fact table.

Now let's relax the strong assumption of a perfect run. If you had some fact rows whose discount dollars triggered an out-of-bounds error, then one more audit dimension row would be needed to flag this condition.

Subsystem 7: Deduplication System

Often dimensions are derived from several sources. This is a common situation for organizations that have many customer-facing source systems that create and manage separate customer master tables. Customer information may need to be merged from several lines of business and outside sources. Sometimes, the data can be matched through identical values in some key column. However, even when a

definitive match occurs, other columns in the data might contradict one another, requiring a decision on which data should survive.

Unfortunately, there is seldom a universal column that makes the merge operation easy. Sometimes, the only clues available are the similarity of several columns. The different sets of data being integrated and the existing dimension table data may need to be evaluated on different fields to attempt a match. Sometimes, a match may be based on fuzzy criteria, such as names and addresses that may nearly match except for minor spelling differences.

Survivorship is the process of combining a set of matched records into a unified image that combines the highest quality columns from the matched records into a conformed row. Survivorship involves establishing clear business rules that define the priority sequence for column values from all possible source systems to enable the creation of a single row with the best-survived attributes. If the dimensional design is fed from multiple systems, you must maintain separate columns with back references, such as natural keys, to all participating source systems used to construct the row.

There are a variety of data integration and data standardization tools to consider if you have difficult deduplicating, matching, and survivorship data issues. These tools are quite mature and in widespread use.

Subsystem 8: Conforming System

Conforming consists of all the steps required to align the content of some or all the columns in a dimension with columns in similar or identical dimensions in other parts of the data warehouse. For instance, in a large organization you may have fact tables capturing invoices and customer service calls that both utilize the customer dimension. It is highly likely the source systems for invoices and customer service have separate customer databases. It is likely there will be little guaranteed consistency between the two sources of customer information. The data from these two customer sources needs to be conformed to make some or all the columns describing customer share the same domains.

NOTE The process of creating conformed dimensions aligns with an agile approach. For two dimensions to be conformed, they must share at least one common attribute with the same name and same contents. You can start with a single conformed attribute such as Customer Category and systematically add this column in a nondisruptive way to customer dimensions in each of the customer-facing processes. As you augment each customer-facing process, you expand the list of processes that are integrated and can participate in drill-across queries. You can also incrementally grow the list of conformed attributes, such as city, state, and country. All this can be staged to align with a more agile implementation approach.

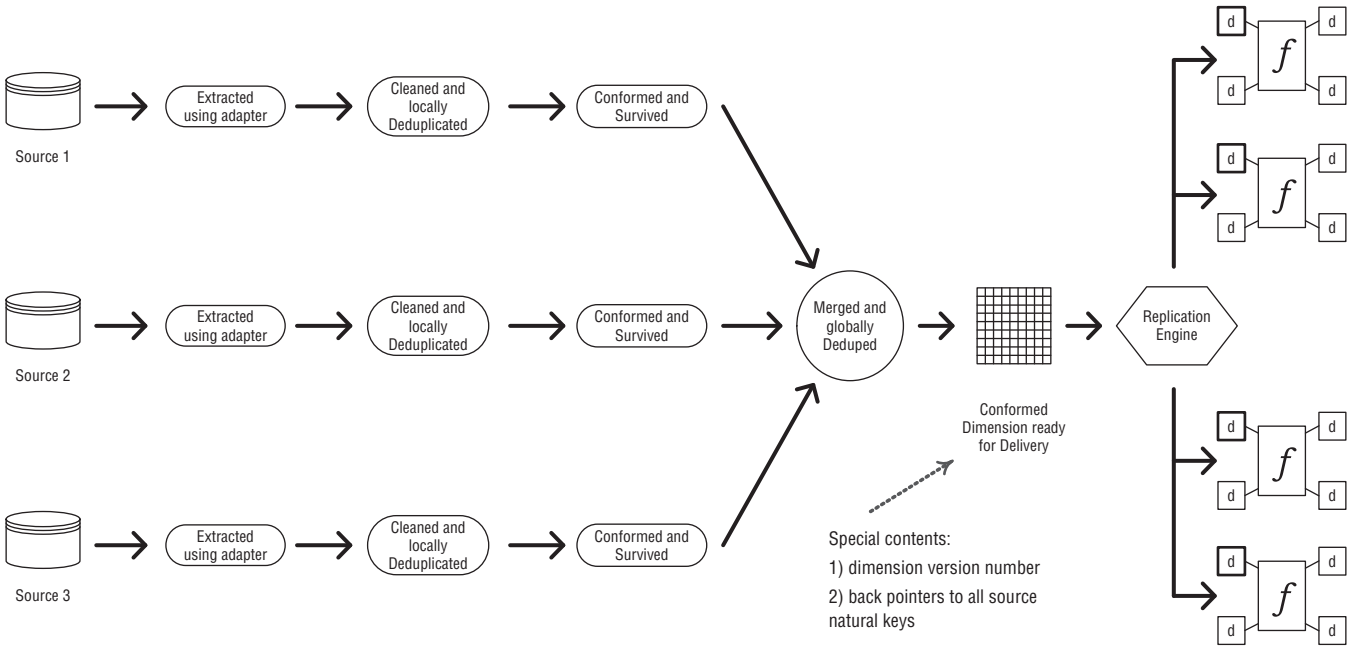

Figure 19-3: Deduplicating and survivorship processing for conformed dimension process.

The conforming subsystem is responsible for creating and maintaining the conformed dimensions and conformed facts described in Chapter 4: Inventory. To accomplish this, incoming data from multiple systems needs to be combined and integrated, so it is structurally identical, deduplicated, filtered of invalid data, and standardized in terms of content rows in a conformed image. A large part of the conforming process is the deduplicating, matching, and survivorship processes previously described. The conforming process flow combining the deduplicating and survivorship processing is shown in Figure 19-3.

The process of defining and delivering conformed dimensions and facts is described later in subsystems 17 (dimension manager) and 18 (fact provider).

Delivering: Prepare for Presentation

The primary mission of the ETL system is the handoff of the dimension and fact tables in the delivery step. For this reason, the delivery subsystems are the most pivotal subsystems in the ETL architecture. Although there is considerable variation in source data structures and cleaning and conforming logic, the delivery processing techniques for preparing the dimensional table structures are more defined and disciplined. Use of these techniques is critical to building a successful dimensional data warehouse that is reliable, scalable, and maintainable.

Many of these subsystems focus on dimension table processing. Dimension tables are the heart of the data warehouse. They provide the context for the fact tables and hence for all the measurements. Although dimension tables are usually smaller than the fact tables, they are critical to the success of the DW/BI system as they provide the entry points into the fact tables. The delivering process begins with the cleaned and conformed data resulting from the subsystems just described. For many dimensions, the basic load plan is relatively simple: You perform basic transformations to the data to build dimension rows for loading into the target presentation table. This typically includes surrogate key assignment, code lookups to provide appropriate descriptions, splitting or combining columns to present the appropriate data values, or joining underlying third normal form table structures into denormalized flat dimensions.

Preparing fact tables is certainly important because fact tables hold the key measurements of the business that the users want to see. Fact tables can be large and time-consuming to load. However, preparing fact tables for presentation is typically more straightforward.

Subsystem 9: Slowly Changing Dimension Manager

One of the more important elements of the ETL architecture is the capability to implement slowly changing dimension (SCD) logic. The ETL system must determine how to handle an attribute value that has changed from the value already stored in the data warehouse. If the revised description is determined to be a legitimate and reliable update to previous information, the appropriate SCD technique must be applied.

As described in Chapter 5: Procurement, when the data warehouse receives notification that an existing row in a dimension has changed, there are three basic responses: type 1 overwrite, type 2 add a new row, and type 3 add a new column. The SCD manager should systematically handle the time variance in the dimensions using these three techniques, as well as the other SCD techniques. In addition, the SCD manager should maintain appropriate housekeeping columns for type 2 changes. Figure 19-4 shows the overall processing flow for handling surrogate key management for processing SCDs.

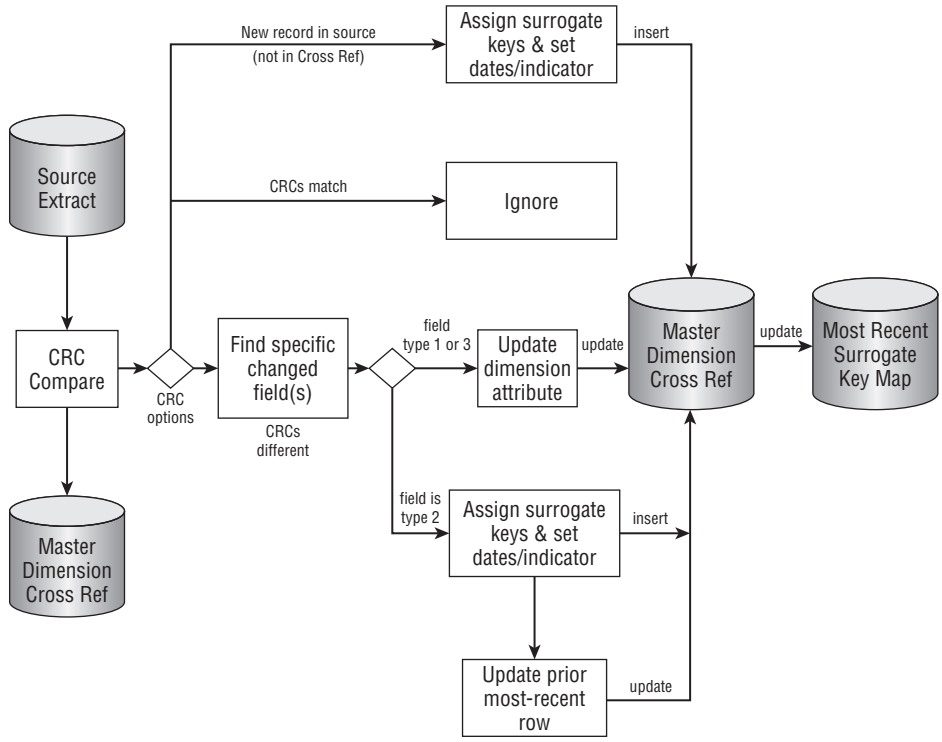

Figure 19-4: Processing flow for SCD surrogate key management.

The change data capture process described in subsystem 2 obviously plays an important role in presenting the changed data to the SCD process. Assuming the change data capture process has effectively delivered appropriate changes, the SCD process can take the appropriate actions.

Type 1: Overwrite

The type 1 technique is a simple overwrite of one or more attributes in an existing dimension row. You take the revised data from the change data capture system and overwrite the dimension table contents. Type 1 is appropriate when correcting data or when there is no business need to keep the history of previous values. For instance, you may receive a corrected customer address. In this case, overwriting is the right choice. Note that if the dimension table includes type 2 change tracking, you should overwrite the affected column in all existing rows for that particular customer. Type 1 updates must be propagated forward from the earliest permanently stored staging tables to all affected staging tables, so if any of them are used to re-create the final load tables, the effect of the overwrite is preserved.

Some ETL tools contain UPDATE else INSERT functionality. This functionality may be convenient for the developer but can be a performance killer. For maximum performance, existing row UPDATEs should be segregated from new row INSERTs. If type 1 updates cause performance problems, consider disabling database logging or use of the DBMS bulk loader.

Type 1 updates invalidate any aggregates built upon the changed column, so the dimension manager (subsystem 17) must notify the affected fact providers (subsystem 18) to drop and rebuild the affected aggregates.

Type 2: Add New Row

The type 2 SCD is the standard technique for accurately tracking changes in dimensions and associating them correctly with fact rows. Supporting type 2 changes requires a strong change data capture system to detect changes as soon as they occur. For type 2 updates, copy the previous version of the dimension row and create a new dimension row with a new surrogate key. If there is not a previous version of the dimension row, create a new one from scratch. Then update this row with the columns that have changed and add any other columns that are needed. This is the main workhorse technique for handling dimension attribute changes that need to be tracked over time.

The type 2 ETL process must also update the most recent surrogate key map table, assuming the ETL tool doesn't automatically handle this. These little two-column

tables are of immense importance when loading fact table data. Subsystem 14, the surrogate key pipeline, supports this process.

Refer to Figure 19-4 to see the lookup and key assignment logic for handling a changed dimension row during the extract process. In this example, the change data capture process (subsystem 2) uses a CRC compare to determine which rows have changed in the source data since the last update. If you are lucky, you already know which dimension records have changed and can omit this CRC compare step. After you identify rows that have changes in type 2 attributes, you can generate a new surrogate key from the key sequence and update the surrogate key map table.

When a new type 2 row is created, you need at least a pair of time stamps, as well as an optional change description attribute. The pair of time stamps defines a span of time from the beginning effective time to the ending effective time when the complete set of dimension attributes is valid. A more sophisticated treatment of a type 2 SCD row involves adding five ETL housekeeping columns. Referring to Figure 19-4, this also requires the type 2 ETL process to find the prior effective row and make appropriate updates to these housekeeping columns:

- Change Date (change date as foreign key to date dimension outrigger)
- Row Effective Date/Time (exact date/time stamp of change)
- Row End Date/Time (exact date/time stamp of next change, defaults to 12/31/9999 for most current dimension row)
- Reason for Change column (optional attribute)
- Current Flag (current/expired)

NOTE It is possible that back-end scripts are run within the transaction database to modify data without updating the respective metadata fields such as the last_modified_date. Using these fields for the dimension time stamps can cause inconsistent results in the data warehouse. Always use the system or as-of date to derive the type 2 effective time stamps.

The type 2 process does not change history as the type 1 process does; thus type 2 changes don't require rebuilding affected aggregate tables as long as the change was made "today" and not backward in time.

NOTE *Kimball Design Tip #80* (available at `www.kimballgroup.com` under the Tools and Utilities tab for this book title) provides in-depth guidance on adding a row change reason code attribute to dimension tables.

Type 3: Add New Attribute

The type 3 technique is designed to support attribute "soft" changes that allow a user to refer either to the old value of the attribute or the new value. For example, if a sales team is assigned to a newly named sales region, there may be a need to track the old region assignment, as well as the new one. The type 3 technique requires the ETL system to alter the dimension table to add a new column to the schema, if this situation was not anticipated. Of course, the DBA assigned to work with the ETL team will in all likelihood be responsible for this change. You then need to push the existing column values into the newly created column and populate the original column with the new values provided to the ETL system. Figure 19-5 shows how a type 3 SCD is implemented.

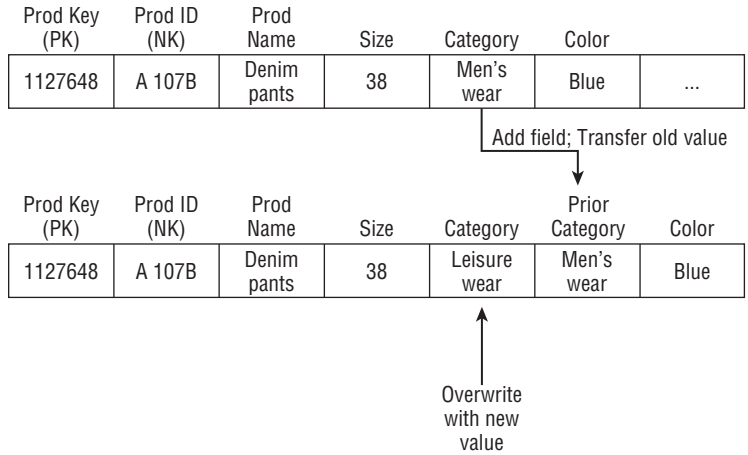

Figure 19-5: Type 3 SCD process.

Similar to the type 1 process, type 3 change updates invalidate any aggregates built upon the changed column; the dimension manager must notify the affected fact providers, so they drop and rebuild the affected aggregates.

Type 4: Add Mini-Dimension

The type 4 technique is used when a group of attributes in a dimension change sufficiently rapidly so that they are split off to a mini-dimension. This situation is sometimes called a rapidly changing monster dimension. Like type 3, this situation calls for a schema change, hopefully done at design time. The mini-dimension requires its own unique primary key, and both the primary key of the main dimension and the primary key of the mini-dimension must appear in the fact table. Figure 19-6 shows how a type 4 SCD is implemented.

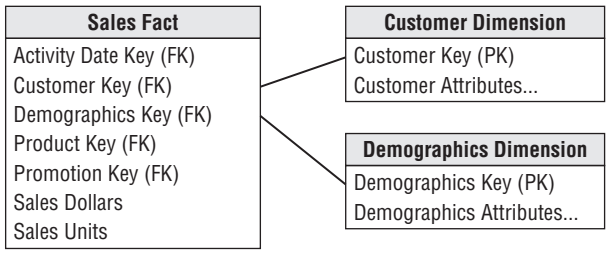

Figure 19-6: Type 4 SCD process.

Type 5: Add Mini-Dimension and Type 1 Outrigger

The type 5 technique builds on the type 4 mini-dimension by also embedding a type 1 reference to the mini-dimension in the primary dimension. This allows accessing the current values in the mini-dimension directly from the base dimension without linking through a fact table. The ETL team must add the type 1 key reference in the base dimension and must overwrite this key reference in all copies of the base dimension whenever the current status of the mini-dimension changes over time. Figure 19-7 shows how a type 5 SCD is implemented.

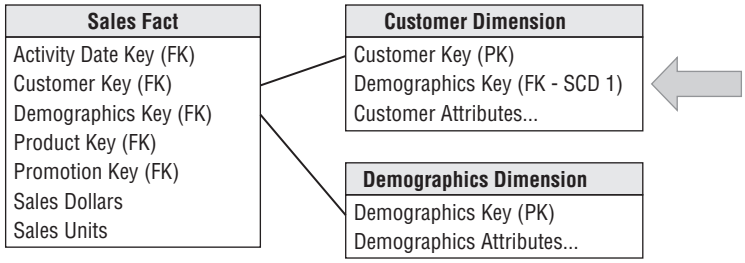

Figure 19-7: Type 5 SCD process.

Type 6: Add Type 1 Attributes to Type 2 Dimension

The type 6 technique has an embedded attribute that is an alternate value of a normal type 2 attribute in the base dimension. Usually such an attribute is simply a type 3 alternative reality, but in this case the attribute is systematically overwritten whenever the attribute is updated. Figure 19-8 shows how a type 6 SCD is implemented.

Type 7: Dual Type 1 and Type 2 Dimensions

The type 7 technique is a normal type 2 dimension paired with a specially constructed fact table that has both a normal foreign key to the dimension for type 2 historical processing, and also a foreign durable key (FDK in Figure 19-9) that is

used alternatively for type 1 current processing, connected to the durable key in the dimension table labeled PDK. The dimension table also contains a current row indicator that indicates whether the particular row is the one to be used for current SCD 1 perspective. The ETL team must augment a normally constructed fact table with this constant value foreign durable key. Figure 19-9 shows how a type 7 SCD is implemented.

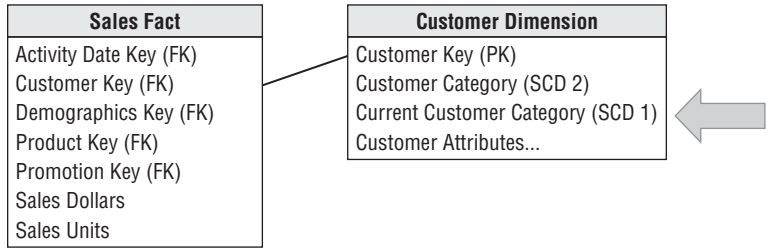

Figure 19-8: Type 6 SCD process.

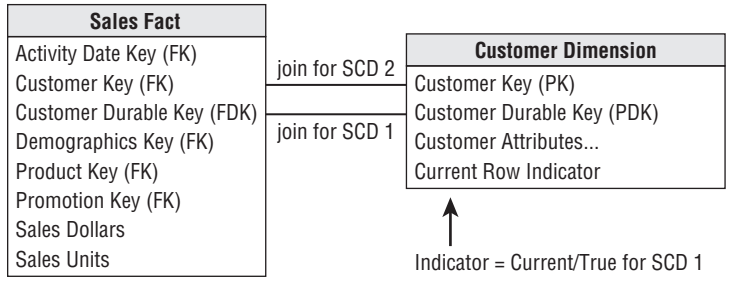

Figure 19-9: Type 7 SCD process.

Subsystem 10: Surrogate Key Generator

As you recall from Chapter 3: Retail Sales, we strongly recommend the use of surrogate keys for all dimension tables. This implies you need a robust mechanism for producing surrogate keys in the ETL system. The surrogate key generator should independently generate surrogate keys for every dimension; it should be independent of database instance and able to serve distributed clients. The goal of the surrogate key generator is to generate a meaningless key, typically an integer, to serve as the primary key for a dimension row.

Although it may be tempting to create surrogate keys via database triggers, this technique may create performance bottlenecks. If the DBMS is used to assign surrogate keys, it is preferable for the ETL process to directly call the database sequence generator. For improved efficiency, consider having the ETL tool generate

and maintain the surrogate keys. Avoid the temptation of concatenating the operational key of the source system and a date/time stamp. Although this approach seems simple, it is fraught with problems and ultimately will not scale.

Subsystem 11: Hierarchy Manager

It is normal for a dimension to have multiple, simultaneous, embedded hierarchical structures. These multiple hierarchies simply coexist in the same dimension as dimension attributes. All that is necessary is that every attribute be single valued in the presence of the dimension's primary key. Hierarchies are either fixed or ragged. A fixed depth hierarchy has a consistent number of levels and is simply modeled and populated as separate dimension attributes for each of the levels. Slightly ragged hierarchies like postal addresses are most often modeled as a fixed hierarchy. Profoundly ragged hierarchies are typically found with organization structures that are unbalanced and of indeterminate depth. The data model and ETL solution required to support these needs require the use of a bridge table containing the organization map.

Snowflakes or normalized data structures are not recommended for the presentation level. However, the use of a normalized design may be appropriate in the ETL staging area to assist in the maintenance of the ETL data flow for populating and maintaining the hierarchy attributes. The ETL system is responsible for enforcing the business rules to assure the hierarchy is populated appropriately in the dimension table.

Subsystem 12: Special Dimensions Manager

The special dimensions manager is a catch-all subsystem: a placeholder in the ETL architecture for supporting an organization's specific dimensional design characteristics. Some organizations' ETL systems require all the capabilities discussed here, whereas others will be concerned with few of these design techniques:

Date/Time Dimensions

The date and time dimensions are unique in that they are completely specified at the beginning of the data warehouse project, and they don't have a conventional source. This is okay! Typically, these dimensions are built in an afternoon with a spreadsheet. But in a global enterprise environment, even this dimension can be challenging when taking into account multiple financial reporting periods or multiple cultural calendars.

Junk Dimensions

Junk dimensions are made up from text and miscellaneous flags left over in the fact table after you remove all the critical attributes. There are two approaches for

creating junk dimensions in the ETL system. If the theoretical number of rows in the dimension is fixed and known, the junk dimension can be created in advance. In other cases, it may be necessary to create newly observed junk dimension rows on-the-fly while processing fact row input. As illustrated in Figure 19-10, this process requires assembling the junk dimension attributes and comparing them to the existing junk dimension rows to see if the row already exists. If not, a new dimension row must be assembled, a surrogate key created, and the row loaded into the junk dimension on-the-fly during the fact table load process.

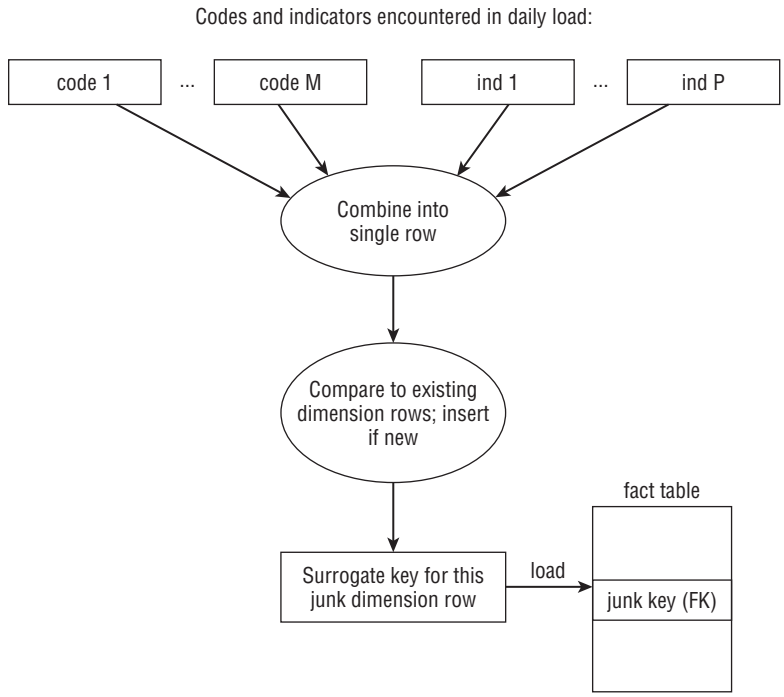

Figure 19-10: Architecture for building junk dimension rows.

NOTE *Kimball Design Tip #113* (available at `www.kimballgroup.com` under the Tools and Utilities tab for this book title) provides more in-depth guidance on building and maintaining junk dimension tables.

Mini-Dimensions

As we just discussed in subsystem 9, mini-dimensions are a technique used to track dimension attribute changes in a large dimension when the type 2 technique is infeasible, such as a customer dimension. From an ETL perspective, creation of

the mini-dimension is similar to the junk dimension process previously described. Again, there are two alternatives: building all valid combinations in advance or recognizing and creating new combinations on-the-fly. Although junk dimensions are usually built from the fact table input, mini-dimensions are built from dimension table inputs. The ETL system is responsible for maintaining a multicolumn surrogate key lookup table to identify the base dimension member and appropriate mini-dimension row to support the surrogate pipeline process described in Subsystem 14, Surrogate Key Pipeline. Keep in mind that very large, complex customer dimensions often require several mini-dimensions.

NOTE *Kimball Design Tip #127* (available at `www.kimballgroup.com` under the Tools and Utilities tab for this book title) provides more in-depth guidance on building and maintaining mini-dimension tables.

Shrunken Subset Dimensions

Shrunken dimensions are conformed dimensions that are a subset of rows and/or columns of one of your base dimensions. The ETL data flow should build conformed shrunken dimensions from the base dimension, rather than independently, to assure conformance. The primary key for the shrunken dimension, however, must be independently generated; if you attempt to use a key from an "example" base dimension row, you will get into trouble if this key is retired or superseded.

NOTE *Kimball Design Tip #137* (available at `www.kimballgroup.com` under the Tools and Utilities tab for this book title) provides more in-depth guidance on building shrunken dimension tables.

Small Static Dimensions

A few dimensions are created entirely by the ETL system without a real outside source. These are usually small lookup dimensions where an operational code is translated into words. In these cases, there is no real ETL processing. The lookup dimension is simply created directly by the ETL team as a relational table in its final form.

User Maintained Dimensions

Often the warehouse requires that totally new "master" dimension tables be created. These dimensions have no formal system of record; rather they are custom descriptions, groupings, and hierarchies created by the business for reporting and analysis

purposes. The ETL team often ends up with stewardship responsibility for these dimensions, but this is typically not successful because the ETL team is not aware of changes that occur to these custom groupings, so the dimensions fall into disrepair and become ineffective. The best-case scenario is to have the appropriate business user department agree to own the maintenance of these attributes. The DW/BI team needs to provide a user interface for this maintenance. Typically, this takes the form of a simple application built using the company's standard visual programming tool. The ETL system should add default attribute values for new rows, which the user owner needs to update. If these rows are loaded into the warehouse before they are changed, they still appear in reports with whatever default description is supplied.

NOTE The ETL process should create a unique default dimension attribute description that shows someone hasn't yet done their data stewardship job. We favor a label that concatenates the phrase Not Yet Assigned with the surrogate key value: "Not Yet Assigned 157." That way, multiple unassigned values do not inadvertently get lumped together in reports and aggregate tables. This also helps identify the row for later correction.

Subsystem 13: Fact Table Builders

Fact tables hold the measurements of an organization. Dimensional models are deliberately built around these numerical measurements. The fact table builder subsystem focuses on the ETL architectural requirements to effectively build the three primary types of fact tables: transaction, periodic snapshot, and accumulating snapshot. An important requirement for loading fact tables is maintaining referential integrity with the associated dimension tables. The surrogate key pipeline (subsystem 14) is designed to help support this need.

Transaction Fact Table Loader

The transaction grain represents a measurement event defined at a particular instant. A line item on an invoice is an example of a transaction event. A scanner event at a cash register is another. In these cases, the time stamp in the fact table is very simple. It's either a single daily grain foreign key or a pair consisting of a daily grain foreign key together with a date/time stamp, depending on what the source system provides and the analyses require. The facts in this transaction table must be true to the grain and should describe only what took place in that instant.

Transaction grain fact tables are the largest and most detailed of the three types of fact tables. The transaction fact table loader receives data from the changed data capture system and loads it with the proper dimensional foreign keys. The pure

addition of the most current records is the easiest case: simply bulk loading new rows into the fact table. In most cases, the target fact table should be partitioned by time to ease the administration and speed the performance of the table. An audit key, sequential ID, or date/time stamp column should be included to allow backup or restart of the load job.

The addition of late arriving data is more difficult, requiring additional processing capabilities described in subsystem 16. In the event it is necessary to update existing rows, this process should be handled in two phases. The first step is to insert the corrected rows without overwriting or deleting the original rows, and then delete the old rows in a second step. Using a sequentially assigned single surrogate key for the fact table makes it possible to perform the two steps of insertion followed by deletion.

Periodic Snapshot Fact Table Loader

The periodic snapshot grain represents a regular repeating measurement or set of measurements, like a bank account monthly statement. This fact table also has a single date column, representing the overall period. The facts in this periodic snapshot table must be true to the grain and should describe only measures appropriate to the timespan defined by the period. Periodic snapshots are a common fact table type and are frequently used for account balances, monthly financial reporting, and inventory balances. The periodicity of a periodic snapshot is typically daily, weekly, or monthly.

Periodic snapshots have similar loading characteristics to those of transaction grain fact tables. The same processing applies for inserts and updates. Assuming data is promptly delivered to the ETL system, all records for each periodic load can cluster in the most recent time partition. Traditionally, periodic snapshots have been loaded en masse at the end of the appropriate period.

For example, a credit card company might load a monthly account snapshot table with the balances in effect at the end of the month. More frequently, organizations will populate a hot rolling periodic snapshot. In addition to the rows loaded at the end of every month, there are special rows loaded with the most current balances in effect as of the previous day. As the month progresses, the current month rows are continually updated with the most current information and continue in this manner rolling through the month. Note that the hot rolling snapshot can sometimes be difficult to implement if the business rules for calculating the balances at the period end are complex. Often these complex calculations are dependent on other periodic processing outside the data warehouse, and there is not enough information available to the ETL system to perform these complex calculations on a more frequent basis.

Accumulating Snapshot Fact Table Loader

The accumulating snapshot grain represents the current evolving status of a process that has a finite beginning and end. Usually, these processes are of short duration and therefore don't lend themselves to the periodic snapshot. Order processing is the classic example of an accumulating snapshot. The order is placed, shipped, and paid for within one reporting period. The transaction grain provides too much detail separated into individual fact table rows, and the periodic snapshot just is the wrong way to report this data.

The design and administration of the accumulating snapshot is quite different from the first two fact table types. All accumulating snapshot fact tables have a set of dates which describe the typical process workflow. For instance, an order might have an order date, actual ship date, delivery date, final payment date, and return date. In this example, these five dates appear as five separate date-valued foreign surrogate keys. When the order row is first created, the first of these dates is well defined, but perhaps none of the others have yet happened. This same fact row is subsequently revisited as the order winds its way through the order pipeline. Each time something happens, the accumulating snapshot fact row is destructively modified. The date foreign keys are overwritten, and various facts are updated. Often the first date remains inviolate because it describes when the row was created, but all the other dates may well be overwritten, sometimes more than once.

Many RDBMSs utilize variable row lengths. Repeated updates to accumulating snapshot fact rows may cause the rows to grow due to these variable row lengths, affecting the residency of disk blocks. It may be worthwhile to occasionally drop and reload rows after the update activity to improve performance.

An accumulating snapshot fact table is an effective way to represent finite processes with well-defined beginnings and endings. However, the accumulating snapshot by definition is the most recent view. Often it makes sense to utilize all three fact table types to meet various needs. Periodic history can be captured with periodic extracts, and all the infinite details involved in the process can be captured in an associated transaction grain fact table. The presence of many situations that violate standard scenarios or involve repeated looping though the process would prohibit the use of an accumulating snapshot.

Subsystem 14: Surrogate Key Pipeline

Every ETL system must include a step for replacing the operational natural keys in the incoming fact table row with the appropriate dimension surrogate keys. Referential integrity (RI) means that for each foreign key in the fact table,

an entry exists in the corresponding dimension table. If there's a row in a sales fact table for product surrogate key 323442, you need to have a row in the product dimension table with the same key, or you won't know what you've sold. You have a sale for what appears to be a nonexistent product. Even worse, without the product key in the dimension, a business user can easily construct a query that will omit this sale without even realizing it.

The key lookup process should result in a match for every incoming natural key or a default value. In the event there is an unresolved referential integrity failure during the lookup process, you need to feed these failures back to the responsible ETL process for resolution, as shown in Figure 19-11. Likewise, the ETL process needs to resolve any key collisions that might be encountered during the key lookup process.

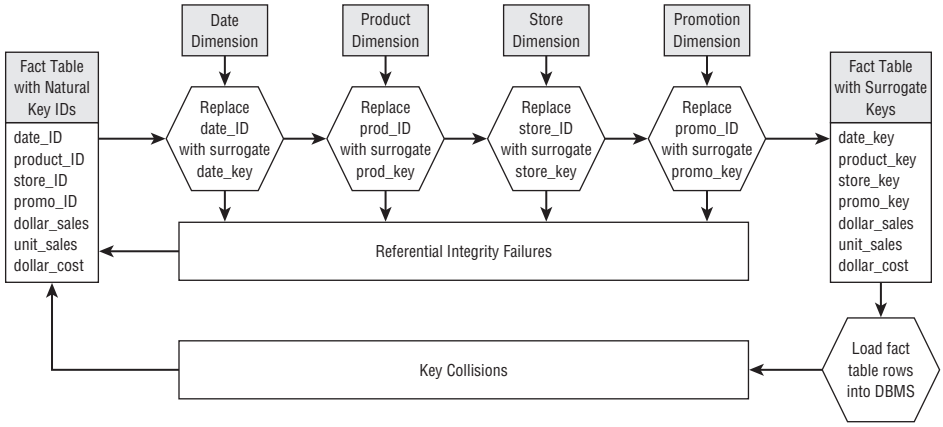

Figure 19-11: Replacing fact record's operational natural keys with dimension surrogate keys.

After the fact table data has been processed and just before loading into the presentation layer, a surrogate key lookup needs to occur to substitute the operational natural keys in the fact table record with the proper current surrogate key. To preserve referential integrity, always complete the updating of the dimension tables first. In that way, the dimension tables are always the legitimate source of primary keys you must replace in the fact table (refer to Figure 19-11).

The most direct approach is to use the actual dimension table as the source for the most current value of the surrogate key corresponding to each natural key. Each time you need the current surrogate key, look up all the rows in the dimension with the natural key equal to the desired value, and then select the surrogate key that aligns with the historical context of the fact row using the current row indicator or begin and end effect dates. Current hardware environments offer nearly unlimited addressable memory, making this approach practical.

During processing, each natural key in the incoming fact record is replaced with the correct current surrogate key. Don't keep the natural key in the fact row—the fact table needs to contain only the surrogate key. Do not write the input data to disk until all fact rows have passed all the processing steps. If possible, all required dimension tables should be pinned in memory, so they can be randomly accessed as each incoming record presents its natural keys.

As illustrated at the bottom of Figure 19-11, the surrogate key pipeline needs to handle key collisions in the event you attempt to load a duplicate row. This is an example of a data quality problem appropriate for a traditional structure data quality screen, as discussed in subsystem 4. In the event a key collision is recognized, the surrogate key pipeline process needs to choose to halt the process, send the offending data into suspension, or apply appropriate business rules to determine if it is possible to correct the problem, load the row, and write an explanatory row into the error event schema.

Note a slightly different process is needed to perform surrogate key lookups if you need to reload history or if you have a lot of late arriving fact rows because you don't want to map the most current value to a historical event. In this case, you need to create logic to find the surrogate key that applied at the time the fact record was generated. This means finding the surrogate key where the fact transaction date is between the key's effective start date and end date.

When the fact table natural keys have been replaced with surrogate keys, the fact row is ready to load. The keys in the fact table row have been chosen to be proper foreign keys, and the fact table is guaranteed to have referential integrity with respect to the dimension tables.

Subsystem 15: Multivalued Dimension Bridge Table Builder

Sometimes a fact table must support a dimension that takes on multiple values at the lowest granularity of the fact table, as described in Chapter 8: Customer Relationship Management. If the grain of the fact table cannot be changed to directly support this dimension, then the multivalued dimension must be linked to the fact table via a bridge table. Bridge tables are common in the healthcare industry, in sales commission environments, and for supporting variable depth hierarchies, as discussed in subsystem 11.

The challenge for the ETL team is building and maintaining the bridge table. As multivalued relationships to the fact row are encountered, the ETL system has the choice of either making each set of observations a unique group or reusing groups when an identical set of observations occurs. Unfortunately, there is no simple answer for the right choice. In the event the multivalued dimension has type 2

attributes, the bridge table must also be time varying, such as a patient's time variant set of diagnoses.

One of the bridge table constructs presented in Chapter 10: Financial Services was the inclusion of a weighting factor to support properly weighted reporting from the bridge table. In many cases, the weighting factor is a familiar allocation factor, but in other cases, the identification of the appropriate weighting factor can be problematic because there may be no rational basis for assigning the weighting factor.

NOTE *Kimball Design Tip #142* (available at `www.kimballgroup.com` under the Tools and Utilities tab for this book title) provides more in-depth guidance on building and maintaining bridge tables.

Subsystem 16: Late Arriving Data Handler

Data warehouses are usually built around the ideal assumption that measured activity (fact records) arrive in the data warehouse at the same time as the context of the activity (dimension records). When you have both the fact records and the correct contemporary dimension rows, you have the luxury of first maintaining the dimension keys and then using these up-to-date keys in the accompanying fact rows. However, for a variety of reasons, the ETL system may need to process late arriving fact or dimension data.

In some environments, there may need to be special modifications to the standard processing procedures to deal with late arriving facts, namely fact records that come into the warehouse very much delayed. This is a messy situation because you have to search back in history to decide which dimension keys were in effect when the activity occurred. In addition, you may need to adjust any semi-additive balances in subsequent fact rows. In a heavily compliant environment, it is also necessary to interface with the compliance subsystem because you are about to change history.

Late arriving dimensions occur when the activity measurement (fact record) arrives at the data warehouse without its full context. In other words, the statuses of the dimensions attached to the activity measurement are ambiguous or unknown for some period of time. If you are living in the conventional batch update cycle of one or more days' latency, you can usually just wait for the dimensions to be reported. For example, the identification of the new customer may come in a separate feed delayed by several hours; you may just be able to wait until the dependency is resolved.

But in many situations, especially real-time environments, this delay is not acceptable. You cannot suspend the rows and wait for the dimension updates to occur; the business requirements demand that you make the fact row visible before

knowing the dimensional context. The ETL system needs additional capabilities to support this requirement. Using customer as the problem dimension, the ETL system needs to support two situations. The first is to support late arriving type 2 dimension updates. In this situation, you need to add the revised customer row to the dimension with a new surrogate key and then go in and destructively modify any subsequent fact rows' foreign key to the customer table. The effective dates for the affected dimension rows also need to be reset. In addition, you need to scan forward in the dimension to see if there have been any subsequent type 2 rows for this customer and change this column in any affected rows.

The second situation occurs when you receive a fact row with what appears to be a valid customer natural key, but you have not yet loaded this customer in the customer dimension. It would be possible to load this row pointing to a default row in the dimension table. This approach has the same unpleasant side effect discussed earlier of requiring destructive updates to the fact rows' foreign keys when the dimension updates are finally processed. Alternatively, if you believe the customer is a valid, but not yet processed customer, you should assign a new customer surrogate key with a set of dummy attribute values in a new customer dimension row. You then return to this dummy dimension row at a later time and make type 1 overwrite changes to its attributes when you get complete information on the new customer. At least this step avoids destructively changing any fact table keys.

There is no way to avoid a brief provisional period in which the dimensions are "not quite right." But these maintenance steps can minimize the impact of the unavoidable updates to the keys and other columns.

Subsystem 17: Dimension Manager System

The dimension manager is a centralized authority who prepares and publishes conformed dimensions to the data warehouse community. A conformed dimension is by necessity a centrally managed resource: Each conformed dimension must have a single, consistent source. It is the dimension manager's responsibility to administer and publish the conformed dimension(s) for which he has responsibility. There may be multiple dimension managers in an organization, each responsible for a dimension. The dimension manager's responsibilities include the following ETL processing:

- Implement the common descriptive labels agreed to by the data stewards and stakeholders during the dimension design.
- Add new rows to the conformed dimension for new source data, generating new surrogate keys.
- Add new rows for type 2 changes to existing dimension entries, generating new surrogate keys.

- Modify rows in place for type 1 changes and type 3 changes, without changing the surrogate keys.
- Update the version number of the dimension if any type 1 or type 3 changes are made.
- Replicate the revised dimension simultaneously to all fact table providers.

It is easier to manage conformed dimensions in a single tablespace DBMS on a single machine because there is only one copy of the dimension table. However, managing conformed dimensions becomes more difficult in multiple tablespace, multiple DMBS, or multimachine distributed environments. In these situations, the dimension manager must carefully manage the simultaneous release of new versions of the dimension to every fact provider. Each conformed dimension should have a version number column in each row that is overwritten in every row whenever the dimension manager releases the dimension. This version number should be utilized to support any drill-across queries to assure that the same release of the dimension is being utilized.

Subsystem 18: Fact Provider System

The fact provider is responsible for receiving conformed dimensions from the dimension managers. The fact provider owns the administration of one or more fact tables and is responsible for their creation, maintenance, and use. If fact tables are used in any drill-across applications, then by definition the fact provider must be using conformed dimensions provided by the dimension manager. The fact provider's responsibilities are more complex and include:

- Receive or download replicated dimension from the dimension manager.
- In an environment in which the dimension cannot simply be replicated but must be locally updated, the fact provider must process dimension records marked as new and current to update current key maps in the surrogate key pipeline and also process any dimension records marked as new but postdated.
- Add all new rows to fact tables after replacing their natural keys with correct surrogate keys.
- Modify rows in all fact tables for error correction, accumulating snapshots, and late arriving dimension changes.
- Remove aggregates that have become invalidated.
- Recalculate affected aggregates. If the new release of a dimension does not change the version number, aggregates have to be extended to handle only newly loaded fact data. If the version number of the dimension has changed, the entire historical aggregate may have to be recalculated.

- Quality ensure all base and aggregate fact tables. Be satisfied the aggregate tables are correctly calculated.
- Bring updated fact and dimension tables online.
- Inform users that the database has been updated. Tell them if major changes have been made, including dimension version changes, postdated records being added, and changes to historical aggregates.

Subsystem 19: Aggregate Builder

Aggregates are the single most dramatic way to affect performance in a large data warehouse environment. Aggregations are like indexes; they are specific data structures created to improve performance. Aggregates can have a significant impact on performance. The ETL system needs to effectively build and use aggregates without causing significant distraction or consuming extraordinary resources and processing cycles.

You should avoid architectures in which aggregate navigation is built into the proprietary query tool. From an ETL viewpoint, the aggregation builder needs to populate and maintain aggregate fact table rows and shrunken dimension tables where needed by aggregate fact tables. The fastest update strategy is incremental, but a major change to a dimension attribute may require dropping and rebuilding the aggregate. In some environments, it may be faster to dump data out of the DBMS and build aggregates with a sort utility rather than building the aggregates inside the DBMS. Additive numeric facts can be aggregated easily at extract time by calculating break rows in one of the sort packages. Aggregates must always be consistent with the atomic base data. The fact provider (subsystem 18) is responsible for taking aggregates off-line when they are not consistent with the base data.

User feedback on the queries that run slowly is critical input to designing aggregations. Although you can depend on informal feedback to some extent, a log of frequently attempted slow-running queries should be captured. You should also try to identify the nonexistent slow-running queries that never made it into the log because they never run to completion, or aren't even attempted due to known performance challenges.

Subsystem 20: OLAP Cube Builder

OLAP servers present dimensional data in an intuitive way, enabling a range of analytic users to slice and dice data. OLAP is a sibling of dimensional star schemas in the relational database, with intelligence about relationships and calculations defined on the server that enable faster query performance and more interesting analytics from a broad range of query tools. Don't think of an OLAP server as a

competitor to a relational data warehouse, but rather an extension. Let the relational database do what it does best: Provide storage and management.

The relational dimensional schema should be viewed as the foundation for OLAP cubes if you elect to include them in your architecture. The process of feeding data from the dimensional schema is an integral part of the ETL system; the relational schemas are the best and preferred source for OLAP cubes. Because many OLAP systems do not directly address referential integrity or data cleaning, the preferred architecture is to load OLAP cubes after the completion of conventional ETL processes. Note that some OLAP tools are more sensitive to hierarchies than relational schemas. It is important to strongly enforce the integrity of hierarchies within dimensions before loading an OLAP cube. Type 2 SCDs fit an OLAP system well because a new surrogate key is just treated as a new member. Type 1 SCDs that restate history do not fit OLAP well. Overwrites to an attribute value can cause all the cubes using that dimension to be reprocessed in the background, become corrupted, or be dropped. Read this last sentence again.

Subsystem 21: Data Propagation Manager

The data propagation manager is responsible for the ETL processes required to present conformed, integrated enterprise data from the data warehouse presentation server to other environments for special purposes. Many organizations need to extract data from the presentation layer to share with business partners, customers, and/or vendors for strategic purposes. Similarly, some organizations are required to submit data to various government organizations for reimbursement purposes, such as healthcare organizations that participate in the Medicare program. Many organizations have acquired package analytic applications. Typically, these applications cannot be pointed directly against the existing data warehouse tables, so data needs to be extracted from the presentation layer and loaded into proprietary data structures required by the analytic applications. Finally, most data mining tools do not run directly against the presentation server. They need data extracted from the data warehouse and fed to the data mining tool in a specific format.

All the situations previously described require extraction from the DW/BI presentation server, possibly some light transformation, and loading into a target format—in other words ETL. Data propagation should be considered a part of the ETL system; ETL tools should be leveraged to provide this capability. What is different in this situation is that the requirements of the target are not negotiable; you *must* provide the data as specified by the target.

Managing the ETL Environment

A DW/BI environment can have a great dimensional model, well-deployed BI applications, and strong management sponsorship. But it cannot be a success until it can be relied upon as a dependable source for business decision making. One of the goals for the DW/BI system is to build a reputation for providing timely, consistent, and reliable data to empower the business. To achieve this goal, the ETL system must constantly work toward fulfilling three criteria:

- **Reliability**. The ETL processes must consistently run. They must run to completion to provide data on a timely basis that is trustworthy at any level of detail.
- **Availability**. The data warehouse must meet its service level agreements (SLAs). The warehouse should be up and available as promised.
- **Manageability.** A successful data warehouse is never done. It constantly grows and changes along with the business. The ETL processes need to gracefully evolve as well.

The ETL management subsystems are the key components of the architecture to help achieve the goals of reliability, availability, and manageability. Operating and maintaining a data warehouse in a professional manner is not much different than any other systems operations: Follow standard best practices, plan for disaster, and practice. Most of the requisite management subsystems that follow might be familiar to you.

Subsystem 22: Job Scheduler

Every enterprise data warehouse should have a robust ETL scheduler. The entire ETL process should be managed, to the extent possible, through a single metadata-driven job control environment. Major ETL tool vendors package scheduling capabilities into their environments. If you elect not to use the scheduler included with the ETL tool, or do not use an ETL tool, you need to utilize existing production scheduling or perhaps manually code the ETL jobs to execute.

Scheduling is much more than just launching jobs on a schedule. The scheduler needs to be aware of and control the relationships and dependencies between ETL jobs. It needs to recognize when a file or table is ready to be processed. If the organization is processing in real time, you need a scheduler that supports your selected real-time architecture. The job control process must also capture metadata regarding the progress and statistics of the ETL process during its execution. Finally, the

scheduler should support a fully automated process, including notifying the problem escalation system in the event of any situation that requires resolution.

The infrastructure to manage this can be as basic (and labor-intensive) as a set of SQL stored procedures, or as sophisticated as an integrated tool designed to manage and orchestrate multiplatform data extract and loading processes. If you use an ETL tool, it should provide this capability. In any case, you need to set up an environment for creating, managing, and monitoring the ETL job stream.

The job control services needed include:

- **Job definition.** The first step in creating an operations process is to have some way to define a series of steps as a job and to specify some relationship among jobs. This is where the execution flow of the ETL process is written. In many cases, if the load of a given table fails, it can impact your ability to load tables that depend on it. For example, if the customer table is not properly updated, loading sales facts for new customers that did not make it into the customer table is risky. In some databases, it is impossible.
- **Job scheduling.** At a minimum, the environment needs to provide standard capabilities, such as time- and event-based scheduling. ETL processes are often based on some upstream system event, such as the successful completion of the general ledger close or the successful application of sales adjustments to yesterday's sales figures. This includes the ability to monitor database flags, check for the existence of files, and compare creation dates.
- **Metadata capture.** No self-respecting systems person would tolerate a black box scheduling system. The folks responsible for running the loads will demand a workflow monitoring system (subsystem 27) to understand what is going on. The job scheduler needs to capture information about what step the load is on, what time it started, and how long it took. In a handcrafted ETL system, this can be accomplished by having each step write to a log file. The ETL tool should capture this data every time an ETL process executes.
- **Logging.** This means collecting information about the entire ETL process, not just what is happening at the moment. Log information supports the recovery and restarting of a process in case of errors during the job execution. Logging to text files is the minimum acceptable level. We prefer a system that logs to a database because the structure makes it easier to create graphs and reports. It also makes it possible to create time series studies to help analyze and optimize the load process.
- **Notification.** After the ETL process has been developed and deployed, it should execute in a hands-off manner. It should run without human intervention, without fail. If a problem does occur, the control system needs to interface to the problem escalation system (subsystem 30).

NOTE Somebody needs to know if anything unforeseen happened during the load, especially if a response is critical to continuing the process.

Subsystem 23: Backup System

The data warehouse is subject to the same risks as any other computer system. Disk drives will fail, power supplies will go out, and sprinkler systems will accidentally turn on. In addition to these risks, the warehouse also has a need to keep more data for longer periods of time than operational systems. Although typically not managed by the ETL team, the backup and recovery process is often designed as part of the ETL system. Its goal is to allow the data warehouse to get back to work after a failure. This includes backing up the intermediate staging data necessary to restart failed ETL jobs. The archive and retrieval process is designed to enable user access to older data that has been moved out of the main warehouse onto a less costly, usually lower-performing media.

Backup

Even if you have a fully redundant system with a universal power supply, fully RAIDed disks, and parallel processors with failover, some system crisis will eventually visit. Even with perfect hardware, someone can always drop the wrong table (or database). At the risk of stating the obvious, it is better to prepare for this than to handle it on-the-fly. A full scale backup system needs to provide the following capabilities:

- **High performance.** The backup needs to fit into the allotted timeframe. This may include online backups that don't impact performance significantly, including real-time partitions.
- **Simple administration.** The administration interface should provide tools that easily allow you to identify objects to back up (including tables, tablespaces, and redo logs), create schedules, and maintain backup verification and logs for subsequent restore.
- **Automated, lights-out operations.** The backup facility must provide storage management services, automated scheduling, media and device handling, reporting, and notification.

The backup for the warehouse is usually a physical backup. This is an image of the database at a certain point in time, including indexes and physical layout information.

Archive and Retrieval

Deciding what to move out of the warehouse is a cost-benefit issue. It costs money to keep the data around—it takes up disk space and slows the load and query

times. On the other hand, the business users just might need this data to do some critical historical analyses. Likewise an auditor may request archived data as part of a compliance procedure. The solution is not to throw the data away but to put it some place that costs less but is still accessible. Archiving is the data security blanket for the warehouse.

As of this writing, the cost of online disk storage is dropping so rapidly that it makes sense to plan many of archiving tasks to simply write to disk. Especially if disk storage is handled by a separate IT resource, the requirement to "migrate and refresh" is replaced by "refresh." You need to make sure that you can interpret the data at various points in the future.

How long it takes the data to get stale depends on the industry, the business, and the particular data in question. In some cases, it is fairly obvious when older data has little value. For example, in an industry with rapid evolution of new products and competitors, history doesn't necessarily help you understand today or predict tomorrow.

After a determination has been made to archive certain data, the issue becomes "what are the long-term implications of archiving data?" Obviously, you need to leverage existing mechanisms to physically move the data from its current media to another media and ensure it can be recovered, along with an audit trail that accounts for the accesses and alterations to the data. But what does it mean to "keep" old data? Given increasing audit and compliance concerns, you may face archival requirements to preserve this data for five, 10, or perhaps even 50 years. What media should you utilize? Will you be able to read that media in future years? Ultimately, you may find yourself implementing a library system capable of archiving and regularly refreshing the data, and then migrating it to more current structures and media.

Finally, if you are archiving data from a system that is no longer going to be used, you may need to "sunset" the data by extracting it from the system and writing it in a vanilla format that is independent of the original application. You might need to do this if the license to use the application will terminate.

Subsystem 24: Recovery and Restart System

After the ETL system is in production, failures can occur for countless reasons beyond the control of the ETL process. Common causes of ETL production failures include:

- Network failure
- Database failure
- Disk failure
- Memory failure
- Data quality failure
- Unannounced system upgrade

To protect yourself from these failures, you need a solid backup system (subsystem 23) and a companion recovery and restart system. You must plan for unrecoverable errors during the load because they will happen. The system should anticipate this and provide crash recovery, stop, and restart capability. First, look for appropriate tools and design processes to minimize the impact of a crash. For example, a load process should commit relatively small sets of records at a time and keep track of what has been committed. The size of the set should be adjustable because the transaction size has performance implications on different DBMSs.

The recovery and restart system is used, of course, for either resuming a job that has halted or for backing out the whole job and restarting it. This system is significantly dependent on the capabilities of the backup system. When a failure occurs, the initial knee-jerk reaction is to attempt to salvage whatever has processed and restart the process from that point. This requires an ETL tool with solid and reliable checkpoint functionality, so it can perfectly determine what has processed and what has not to restart the job at exactly the right point. In many cases, it may be best to back out any rows that have been loaded as part of the process and restart from the beginning.

We often recommend designing fact tables with a single column primary surrogate key. This surrogate key is a simple integer that is assigned in sequence as rows are created to be added to the fact table. With the fact table surrogate key, you can easily resume a load that is halted or back out all the rows in the load by constraining on a range of surrogate keys.

NOTE Fact table surrogate keys have a number of uses in the ETL back room. First, as previously described, they can be used as the basis for backing out or resuming an interrupted load. Second, they provide immediate and unambiguous identification of a single fact row without needing to constrain multiple dimensions to fetch a unique row. Third, updates to fact table rows can be replaced by inserts plus deletes because the fact table surrogate key is now the actual key for the fact table. Thus, a row containing updated columns can be inserted into the fact table without overwriting the row it is to replace. When all such insertions are complete, then the underlying old rows can be deleted in a single step. Fourth, the fact table surrogate key is an ideal parent key to be used in a parent/child design. The fact table surrogate key appears as a foreign key in the child, along with the parent's dimension foreign keys.

The longer an ETL process runs, the more you must be aware of vulnerabilities due to failure. Designing a modular ETL system made up of efficient processes that are resilient against crashes and unexpected terminations can reduce the risk of a failure resulting in a massive recovery effort. Careful consideration of when to physically stage data by writing it to disk, along with carefully crafted points of

recovery and load date/time stamps or sequential fact table surrogate keys enable you to specify appropriate restart logic.

Subsystem 25: Version Control System

The version control system is a "snapshotting" capability for archiving and recovering all the logic and metadata of the ETL pipeline. It controls check-out and check-in processing for all ETL modules and jobs. It should support source comparisons to reveal differences between versions. This system provides a librarian function for saving and restoring the complete ETL context of a single version. In certain highly compliant environments, it will be equally important to archive the complete ETL system context alongside the relevant archived and backup data. Note that master version numbers need to be assigned for the overall ETL system, just like software release version numbers.

NOTE You have a master version number for each part of the ETL system as well as one for the system as a whole, don't you? And you can restore yesterday's complete ETL metadata context if it turns out there is a big mistake in the current release? Thank you for reassuring us.

Subsystem 26: Version Migration System

After the ETL team gets past the difficult process of designing and developing the ETL process and completes the creation of the jobs required to load the data warehouse, the jobs must be bundled and migrated to the next environment—from development to test and on to production—according to the lifecycle adopted by the organization. The version migration system needs to interface to the version control system to control the process and back out a migration if needed. It should provide a single interface for setting connection information for the entire version.

Most organizations isolate the development, testing, and production environments. You need to be able to migrate a complete version of the ETL pipeline from development, into test, and finally into production. Ideally, the test system is identically configured to its corresponding production system. Everything done to the production system should have been designed in development and the deployment script tested on the test environment. Every back room operation should go through rigorous scripting and testing, whether deploying a new schema, adding a column, changing indexes, changing the aggregate design, modifying a database parameter, backing up, or restoring. Centrally managed front room operations such as deploying new BI tools, deploying new corporate reports, and changing security plans should be equally rigorously tested and scripted if the BI tools allow it.

Subsystem 27: Workflow Monitor

Successful data warehouses are consistently and reliably available, as agreed to with the business community. To achieve this goal, the ETL system must be constantly monitored to ensure the ETL processes are operating efficiently and the warehouse is being loaded on a consistently timely basis. The job scheduler (subsystem 22) should capture performance data every time an ETL process is initiated. This data is part of the process metadata captured in the ETL system. The workflow monitor leverages the metadata captured by the job scheduler to provide a dashboard and reporting system taking many aspects of the ETL system into consideration. You'll want to monitor job status for all job runs initiated by the job scheduler including pending, running, completed and suspended jobs, and capture the historical data to support trending performance over time. Key performance measures include the number of records processed, summaries of errors, and actions taken. Most ETL tools capture the metrics for measuring ETL performance. Be sure to trigger alerts whenever an ETL job takes significantly more or less time to complete than indicated by the historical record.

In combination with the job scheduler, the workflow monitor should also track performance and capture measurements of the performance of infrastructure components including CPU usage, memory allocation and contention, disk utilization and contention, buffer pool usage, database performance, and server utilization and contention. Much of this information is process metadata about the ETL system and should be considered as part of the overall metadata strategy (subsystem 34).

The workflow monitor has a more significant strategic role than you might suspect. It is the starting point for the analysis of performance problems across the ETL pipeline. ETL performance bottlenecks can occur in many places, and a good workflow monitor shows where the bottlenecks are occurring. Chapter 20, discusses many ways to improve performance in the ETL pipeline, but this list is more or less ordered starting with the most important bottlenecks:

- Poorly indexed queries against a source system or intermediate table
- SQL syntax causing wrong optimizer choice
- Insufficient random access memory (RAM) causing thrashing
- Sorting in the RDBMS
- Slow transformation steps
- Excessive I/O
- Unnecessary writes followed by reads
- Dropping and rebuilding aggregates from scratch rather than incrementally
- Filtering (change data capture) applied too late in the pipeline
- Untapped opportunities for parallelizing and pipelining

- Unnecessary transaction logging especially if doing updates
- Network traffic and file transfer overhead

Subsystem 28: Sorting System

Certain common ETL processes call for data to be sorted in a particular order, such as aggregating and joining flat file sources. Because sorting is such a fundamental ETL processing capability, it is called out as a separate subsystem to ensure it receives proper attention as a component of the ETL architecture. There are a variety of technologies available to provide sorting capabilities. An ETL tool can undoubtedly provide a sort function, the DBMS can provide sorting via the SQL SORT clause, and there are a number of sort utilities available.

Sorting simple delimited text files with a dedicated sort package is awesomely fast. These packages typically allow a single read operation to produce up to eight different sorted outputs. Sorting can produce aggregates where each break row of a given sort is a row for the aggregate table, and sorting plus counting is often a good way to diagnose data quality issues.

The key is to choose the most efficient sort resource to support the requirements within your infrastructure. The easy answer for most organizations is to simply utilize the ETL tool's sort function. However, in some situations it may be more efficient to use a dedicated sort package; although ETL and DBMS vendors claim to have made up much of the performance differences.

Subsystem 29: Lineage and Dependency Analyzer

Two increasingly important elements being requested of the ETL system are the ability to track both the lineage and dependencies of data in the DW/BI system:

- **Lineage.** Beginning with a specific data element in an intermediate table or BI report, identify the source of that data element, other upstream intermediate tables containing that data element and its sources, and all transformations that data element and its sources have undergone.
- **Dependency.** Beginning with a specific data element in a source table or an intermediate table, identify all downstream intermediate tables and final BI reports containing that data element or its derivations and all transformations applied to that data element and its derivations.

Lineage analysis is often an important component in a highly compliant environment where you must explain the complete processing flow that changed any data result. This means the ETL system must display the ultimate physical sources and all subsequent transformations of any selected data element, chosen either from the

middle of the ETL pipeline or on a final delivered report. Dependency analysis is important when assessing changes to a source system and the downstream impacts on the data warehouse and ETL system. This implies the ability to display all affected downstream data elements and final report fields affected by a potential change in any selected data element, chosen either in the middle of the ETL pipeline or an original source (dependency).

Subsystem 30: Problem Escalation System

Typically, the ETL team develops the ETL processes and the quality assurance team tests them thoroughly before they are turned over to the group responsible for day-to-day systems operations. To make this work, the ETL architecture needs to include a proactively designed problem escalation system similar to what is in place for other production systems.

After the ETL processes have been developed and tested, the first level of operational support for the ETL system should be a group dedicated to monitoring production applications. The ETL development team becomes involved only if the operational support team cannot resolve a production problem.

Ideally, you have developed ETL processes, wrapped them into an automated scheduler, and have robust workflow monitoring capabilities peering into the ETL processes as they execute. The execution of the ETL system should be a hands-off operation. It should run like clockwork without human intervention and without fail. If a problem does occur, the ETL process should automatically notify the problem escalation system of any situation that needs attention or resolution. This automatic feed may take the form of simple error logs, operator notification messages, supervisor notification messages, and system developer messages. The ETL system may notify an individual or a group depending on the severity of the situation or the processes involved. ETL tools can support a variety of messaging capabilities including e-mail alerts, operator messages, and notifications to mobile devices.

Each notification event should be written to a database used to understand the types of problems that arise, their status, and resolution. This data forms part of the process metadata captured by the ETL system (subsystem 34). You need to ensure that organizational procedures are in place for proper escalation, so every problem is resolved appropriately.

In general, the support structure for the ETL system should follow a fairly standard support structure. First, level support is typically a help desk that is the first point of contact when a user notices an error. The help desk is responsible for resolution whenever feasible. If the help desk cannot resolve the issue, the second level support is notified. This is typically a systems administrator or DBA on

the production control technical staff capable of supporting general infrastructure failures. The ETL manager is the third level support and should be knowledgeable to support most issues that arise in the ETL production process. Finally, when all else fails, the ETL developer should be called in to analyze the situation and assist with resolution.

Subsystem 31: Parallelizing/Pipelining System

The goal of the ETL system, in addition to providing high quality data, is to load the data warehouse within the allocated processing window. In large organizations with huge data volumes and a large portfolio of dimensions and facts, loading the data within these constraints can be a challenge. The paralleling/pipelining system provides capabilities to enable the ETL system to deliver within these time constraints. The goal of this system is to take advantage of multiple processors or grid computing resources commonly available. It is highly desirable, and in many cases necessary, that parallelizing and pipelining be automatically invoked for every ETL process unless specific conditions preclude it from processing in such a manner, such as waiting on a condition in the middle of the process.

Parallelizing is a powerful performance technique at every stage of the ETL pipeline. For example, the extraction process can be parallelized by logically partitioning on ranges of an attribute. Verify that the source DBMS handles parallelism correctly and doesn't spawn conflicting processes. If possible, choose an ETL tool that handles parallelizing of intermediate transformation processes automatically. In some tools it is necessary to hand create parallel processes. This is fine until you add additional processors, and the ETL system then can't take advantage of the greater parallelization opportunities unless you modify the ETL modules by hand to increase the number of parallel flows.

Subsystem 32: Security System

Security is an important consideration for the ETL system. A serious security breach is much more likely to come from within the organization than from someone hacking in from the outside. Although we don't like to think it, the folks on the ETL team present as much a potential threat as any group inside the organization. We recommend administering role-based security on all data and metadata in the ETL system. To support compliance requirements, you may need to prove that a version of an ETL module hasn't been changed or show who made changes to a module. You should enforce comprehensive authorized access to all ETL data and metadata by individual and role. In addition, you'll want to maintain a historical record of all accesses to ETL data and metadata by individual and role. Another issue to be careful of is the bulk data movement process. If you move data across the network,

even if it is within the company firewall, it pays to be careful. Make sure to use data encryption or a file transfer utility that uses a secure transfer protocol.

Another back room security issue to consider is administrator access to the production warehouse server and software. We've seen situations where no one on the team had security privileges; in other cases, everyone had access to everything. Obviously, many members of the team should have privileged access to the development environment, but the production warehouse should be strictly controlled. On the other hand, someone from the DW/BI team needs to be able to reset the warehouse machine if something goes wrong. Finally, the backup media should be guarded. The backup media should have as much security surrounding them as the online systems.

Subsystem 33: Compliance Manager

In highly compliant environments, supporting compliance requirements is a significant new requirement for the ETL team. Compliance in the data warehouse involves "maintaining the chain of custody" of the data. In the same way a police department must carefully maintain the chain of custody of evidence to argue that the evidence has not been changed or tampered with, the data warehouse must also carefully guard the compliance-sensitive data entrusted to it from the moment it arrives. Furthermore, the data warehouse must always show the exact condition and content of such data at any point in time that it may have been under the control of the data warehouse. The data warehouse must also track who had authorized access to the data. Finally, when the suspicious auditor looks over your shoulder, you need to link back to an archived and time-stamped version of the data as it was originally received, which you have stored remotely with a trusted third party. If the data warehouse is prepared to meet all these compliance requirements, then the stress of being audited by a hostile government agency or lawyer armed with a subpoena should be greatly reduced.

The compliance requirements may mean you cannot actually change any data, for any reason. If data must be altered, then a new version of the altered records must be inserted into the database. Each row in each table therefore must have begin and end time stamps that accurately represents the span of time when the record was the "current truth." The big impact of these compliance requirements on the data warehouse can be expressed in simple dimensional modeling terms. Type 1 and type 3 changes are dead. In other words, all changes become inserts. No more deletes or overwrites.

Figure 19-12 shows how a fact table can be augmented so that overwrite changes are converted into a fact table equivalent of a type 2 change. The original fact table consisted of the lower seven columns starting with activity date and ending with

net dollars. The original fact table allowed overwrites. For example, perhaps there is a business rule that updates the discount and net dollar amounts after the row is originally created. In the original version of the table, history is lost when the overwrite change takes place, and the chain of custody is broken.

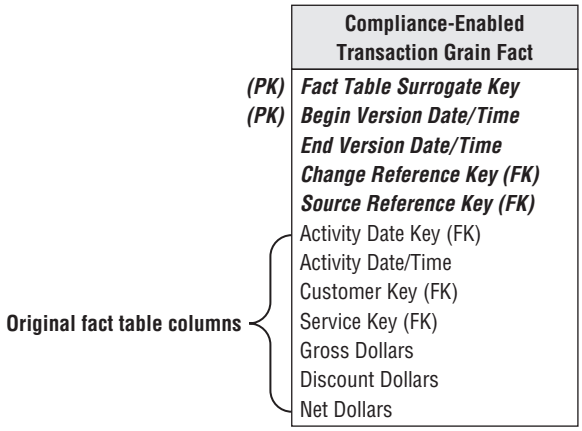

Figure 19-12: Compliance-enabled transaction fact table.

To convert the fact table to be compliance-enabled, five columns are added, as shown in bold. A fact table surrogate key is created for each original unmodified fact table row. This surrogate key, like a dimension table surrogate key, is just a unique integer that is assigned as each original fact table row is created. The begin version date/time stamp is the exact time of creation of the fact table row. Initially, the end version date/time is set to a fictitious date/time in the future. The change reference is set to "original," and the source reference is set to the operational source.

When an overwrite change is needed, a new row is added to the fact table with the same fact table surrogate key, and the appropriate regular columns changed, such as discount dollars and net dollars. The begin version date/time column is set to the exact date/time when the change in the database takes place. The end version date/time is set to a fictitious date/time in the future. The end version date/time of the original fact row is now set to the exact date/time when the change in the database takes place. The change reference now provides an explanation for the change, and the source reference provides the source of the revised columns.

Referring to the design in Figure 19-12, a specific moment in time can be selected and the fact table constrained to show exactly what the rows contained at that moment. The alterations to a given row can be examined by constraining to a specific fact table surrogate key and sorting by begin version date/time.

The compliance machinery is a significant addition to a normal fact table (refer to Figure 19-12). If the compliance-enabled table is actually used for only demonstrating

compliance, then a normal version of the fact table with just the original columns can remain as the main operational table, with the compliance-enabled table existing only in the background. The compliance-enabled table doesn't need to be indexed for performance because it will not be used in a conventional BI environment.

For heaven's sake, don't assume that all data is now subject to draconian compliance restrictions. It is essential you receive firm guidelines from the chief compliance officer before taking any drastic steps.

The foundation of a compliance system is the interaction of several of the subsystems already described married to a few key technologies and capabilities:

- **Lineage analysis.** Show where a final piece of data came from to prove the original source data plus the transformations including stored procedures and manual changes. This requires full documentation of all the transforms and the technical ability to rerun the transforms against the original data.
- **Dependency analysis.** Show where an original source data element was ever used.
- **Version control**. It may be necessary to rerun the source data through the ETL system in effect at the time, requiring the exact version of the ETL system for any given data source.
- **Backup and restore.** Of course, the requested data may have been archived years ago and need to be restored for audit purposes. Hopefully, you archived the proper version of the ETL system alongside the data, so both the data and the system can be restored. It may be necessary to prove the archived data hasn't been altered. During the archival process, the data can be hash-coded and the hash and data separated. Have the hash codes archived separately by a trusted third party. Then, when demanded, restore the original data, hash code it again, and then compare to the hash codes retrieved from the trusted third party to prove the authenticity of the data.
- **Security.** Show who has accessed or modified the data and transforms. Be prepared to show roles and privileges for users. Guarantee the security log can't be altered by using a write once media.
- **Audit dimension.** The audit dimension ties runtime metadata context directly with the data to capture quality events at the time of the load.

Subsystem 34: Metadata Repository Manager

The ETL system is responsible for the use and creation of much of the metadata involved in the DW/BI environment. Part of the overall metadata strategy should be to specifically capture ETL metadata, including the process metadata, technical metadata, and business metadata. Develop a balanced strategy between doing nothing and doing too much. Make sure there's time in the ETL development tasks to

capture and manage metadata. And finally, make sure someone on the DW/BI team is assigned the role of metadata manager and owns the responsibility for creating and implementing the metadata strategy.

Summary

In this chapter we have introduced the key building blocks of the ETL system. As you may now better appreciate, building an ETL system is unusually challenging; the ETL system must address a number of demanding requirements. This chapter identified and reviewed the 34 subsystems of ETL and gathered these subsystems into four key areas that represent the ETL process: extracting, cleaning and conforming, delivering, and managing. Careful consideration of all the elements of the ETL architecture is the key to success. You must understand the full breadth of requirements and then set an appropriate and effective architecture in place. ETL is more than simply extract, transform, and load; it's a host of complex and important tasks. In the next chapter we will describe the processes and tasks for building the ETL system.