Part I

Getting Started

In This Part

Chapter 1: ETL Primer
Chapter 2: Kettle Concepts
Chapter 3: Installation and Configuration
Chapter 4: An Example ETL Solution—Sakila

CHAPTER 1

ETL Primer

The introduction of this book described the need for data integration. This chapter provides a starting point to the wonderful world of data integration and explains the differences and similarities among the three main forms of data integration: ETL, ELT, and EII. To fully understand the reasoning behind using a data warehouse and an ETL solution to load and update data, we start by explaining the differences between a transaction and an analysis database.

OLTP versus Data Warehousing

The first question one might ask is how source data systems differ from *business intelligence* (BI) systems (sometimes still called *decision support systems* or DSS). An individual transaction system, often denoted by the acronym *OLTP* (short for OnLine Transaction Processing), needs to be able to very quickly retrieve a single record of information. When multiple records are needed they are usually tied to a single key that has been retrieved before. Think of an order with the accompanying order lines in an order entry system or a personnel record with all salary and bonus information in an HR system. What's more: this data often needs to be updated as well, usually just one record at a time.

The biggest difference between an OLTP and a BI database (the *data warehouse*, or *DWH*) is the amount of data analyzed in a single transaction. Whereas an OLTP handles many concurrent users and queries touching only a single record or limited groups of records at a time, a data warehouse must have the capability to operate on millions of records to answer a single query. Table 1-1 shows an overview of the major differences between an OLTP and a data warehouse.

Table 1-1: OLTP versus Data Warehouse

CHARACTERISTIC	OLTP	DATA WAREHOUSE
System scope/view	Single business process	Multiple business subjects
Data sources	One	Many
Data model	Static	Dynamic
Dominant query type	Insert/update	Read
Data volume per transaction	Small	Big
Data volume	Small/medium	Large
Data currency	Current timestamp	Seconds to days old
Bulk load/insert/update	No	Yes
Full history available	No	Yes
Response times	< 1 second	< 10 seconds
System availability	24/7	8/5
Typical user	Front office	Staff
Number of users	Large	Small/medium

Of course, it's not as black and white as this table might indicate. The distinctions listed are a rather classic way of looking at the two types of systems. More and more often, business intelligence systems are being used as part of the primary business process. A call center agent might have a screen in front of her with not only customer details such as name and address, but also information about order and payment history retrieved from an *operational data store* (ODS) or a data warehouse. Many CRM systems are already capable of showing a credit or customer score on-the-fly, items that have been pre-calculated in the data warehouse and are available on demand for front office workers. This means that the more the data warehouse is used for operational purposes, the more the same requirements apply as for OLTP systems, especially regarding system availability and data currency.

Probably the most discussed characteristic of the data warehouse is the required response time. Ten years ago, it wasn't a problem when a report query took one or two minutes to retrieve and display its data. Nowadays users expect response times similar to what they're accustomed to when using a search engine. More than ten seconds and users get impatient, start clicking refresh buttons (which will sometimes re-issue the query, making the problem even worse), and eventually avoid using the data warehouse because it's so slow. On the other hand, when the data warehouse is used for data mining purposes, analysts find a response time of several hours totally acceptable, as long as the result to their inquiry is valuable.

What Is ETL?

You know of course that ETL is short for extract, transform, and load; no secrets here. But what exactly do we mean by ETL? A simple definition could be "the set of processes for getting data from OLTP systems into a data warehouse." When we look at the roots of ETL it's probably a viable definition, but for modern ETL solutions it grossly oversimplifies the term. Data is not only coming from OLTP systems but from websites, flat files, e-mail databases, spreadsheets, and personal databases such as Access as well. ETL is not only used to load a single data warehouse but can have many other use cases, like loading data marts, generating spreadsheets, scoring customers using data mining models, or even loading forecasts back into OLTP systems. The main ETL steps, however, can still be grouped into three sections:

1. **Extract:** All processing required to connect to various data sources, extract the data from these data sources, and make the data available to the subsequent processing steps. This may sound trivial but can in fact be one of the main obstacles in getting an ETL solution off the ground.
2. **Transform:** Any function applied to the extracted data between the extraction from sources and loading into targets. These functions can contain (but are not limited to) the following operations:
   - Movement of data
   - Validation of data against data quality rules
   - Modification of the content or structure of the data
   - Integration of the data with data from other sources
   - Calculation of derived or aggregated values based on processed data
3. **Load:** All processing required to load the data in a target system. As we show in Chapter 5, this part of the process consists of a lot more than just bulk loading transformed data into a target table. Parts of the loading process include, for instance, surrogate key management and dimension table management.

The remainder of this section examines how ETL solutions evolved over time and what the main ETL building blocks look like.

The Evolution of ETL Solutions

Data integration needs have existed as long as data has been available in a digital format. In the early computing days, before ETL tools existed, the only way to get data from different sources and integrate it in one way or another was to hand-code scripts in languages such as COBOL, RPG, and later in Perl or PL/SQL. Although this is called the first generation of ETL solutions, it may surprise you that today, about 45 percent of all ETL work is still conducted by using hand-coded programs/scripts. This might have made sense in the days when ETL tools had a six-figure price tag attached to them, but currently there are many open source and other low-cost alternatives available so there's really no point in hand-coding ETL jobs anymore. The main drawbacks of hand-coding are that it is:

- Error prone
- Slow in terms of development time
- Hard to maintain
- Lacking metadata
- Lacking consistent logging/error handling

The second generation of ETL tools (actually the first if we're talking about "tools" rather than the broader "solutions") tried to overcome these weaknesses by generating the required code based on the design of an ETL flow. In the early 1990s, products such as Prism, Carlton, and ETI emerged but most were acquired later by other ETL vendors. ETI is probably the only independent vendor left from those early days that still offers a code-generating solution. The fact that code generators are listed here as second-generation ETL solutions doesn't necessarily mean they are outdated. It's rather the contrary; code generators are alive and kicking, with Oracle's Warehouse Builder arguably being the most well-known product in this category. The popular open source tool Talend is another example of a code-generation solution.

Code generators have their pros and cons; the biggest disadvantage is that most code generators can work with only a limited set of databases for which they can generate code. Soon after the code generators came into use, a third generation of ETL tools emerged. These were based on an engine where all the data processing took place, and a set of metadata that stored all the connection and transformation rules. Because engines have a generic way of working and all the transformation logic is independent from both the source and the target data stores, engine-based ETL tools, in general, are more versatile than code-generating tools. Kettle is a typical example of an engine-based tool; other familiar names in this area are Informatica Powercenter and SQL Server Information Services.

Both code generators and engine-based tools offer some help in discovering the structure of the underlying data sources and the relationships between them, although some tools are more capable in doing this than others. They also require that a target data model is developed either before or during the design of the data transformation steps. After this design phase, the target schema has to be mapped against the source schema(s). This whole process is still very time consuming, and as a result, a new generation of data warehouse tools emerged that are model driven. MDA tools (for *Model Driven Architecture*) try to automate the data warehouse and data mart design process from the ground up by reading the source data model and generating both the target schema and all required data mappings to populate the target tables. There are only a few such tools on the market, with Kalido and BIReady being the most well known. They are no silver bullets, however; MDA tools still require a skilled data warehouse architect to reap the benefits from them. Although they cannot solve every data integration challenge they can be a huge time (and thus money) saver.

DATA WAREHOUSE VERSUS DATA MART

In this book, the terms *data warehouse* and *data mart* are often used as if they are interchangeable items. They're not, and they differ widely in scope, model, and applicability. A data warehouse is meant to be the single, integrated storehouse of (historical) data that can be used for supporting an organization's decision process. As such, it contains data covering a wide range of topics and business processes, for instance finance, logistics, marketing, and customer support. Often, a data warehouse cannot be accessed directly by end user tools. A data mart, in contrast, is meant for direct access by end users and end user tools, and has a limited specific analytical purpose, for instance Retail Sales or Customer Calls.

ETL Building Blocks

The best way to look at an ETL solution is to view it as a business process. A business process has input, output, and one or more units of work, the process steps. These steps in turn also have inputs and outputs, and perform an operation to transform the input into the output. Think, for example, of a claims department at an insurance company. There's a big sign on the door that says Claims Department, which tells the purpose and main process of the department: handling claims. Within the department, each desk or sub-department might have its own specialty: health insurance claims, car insurance claims, travel insurance claims, and so on. When a claim is received at the office, it is checked to find out to which desk it should be sent. The claims officer can then determine whether all required information to handle the claim is available and if not, send it back with further instructions to the submitter. Each day at 9 a.m. this process of handling claims starts, and it runs until 5 p.m.

This example is a lot like an ETL process: data arrives or is retrieved and a validation step determines what kind of data it is. The data is then sent to a specific transformation that is designed to handle that specific data. When the transformation can process the data, it's delivered to the next transformation or a destination table, and in the case of errors, it is transferred to an error handling routine. Each night at 3 a.m., the job is started by a scheduler and it ends when all data is processed.

You might now have a global feeling of how ETL processes are designed. From the preceding examples you can deduce that there must be some mechanism to control the overall process flow, and other more specific parts of the process that do the actual transformation. The first part is called a *job* in Kettle terminology, and the latter part consists of *transformations*. Jobs are the traffic agents of an ETL solution, and transformations are the basic building blocks. Individual transformations can be chained together in a logical order, just like a business process, to form a job that can be scheduled and executed. A transformation in turn can also consist of several steps. A *step* is the third basic building block of a Kettle solution, and the connection between steps and transformation is formed by *hops*. You'll read a lot more about jobs, transformations, steps, and hops in the remainder of this book, but these four building blocks enable you to develop any imaginable ETL solution. Chapter 2 provides a more detailed introduction to these four concepts.

ETL, ELT, and EII

The term *data integration* encompasses more than just ETL. With ETL, data is extracted from one or more source systems and, possibly after one or more transformation steps, physically stored in a target environment, usually a data warehouse. To be able to distinguish between ETL and other forms of data integration, we need a way of classifying and describing these other mechanisms.

Figure 1-1 shows a classic example of a data warehouse architecture. In this figure there are multiple source systems, a staging area where data is extracted to, a central warehouse for storing all historical data, and finally data marts that enable end users to work with the data. Between each of these building blocks a data integration process is used, as shown by the ETL blocks.

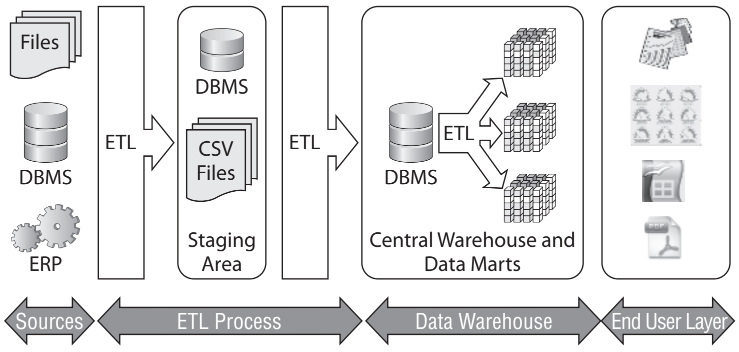

Figure 1-1: Classic data warehouse architecture

This is an architecture that has been used for the past 20 years and has served us well. In fact, many current data warehouse projects still use an architecture similar to the one shown in Figure 1-1. This picture clearly shows that ETL tools are used not only to extract data and load a data warehouse, but also to populate data marts and possibly other databases like an operational data store (not present in the diagram).

Figure 1-1 also shows that there is an intermediate step between the source systems and the data warehouse, called the *staging area*. This part of the overall architecture is merely a drop zone for data; it serves as an intermediate area to get data out of the source systems as quickly as possible. A staging area doesn't necessarily need to be a database system; in many cases, using plain ASCII files to stage data works just as well and is sometimes a faster solution than first inserting the data into a database table.

NON-ETL USE FOR ETL TOOLS

ETL tools are used for more than data warehouse purposes alone. Because they offer a wide range of connectivity and transformation options, another often seen use case is *data migration*. With the help of a tool like Kettle it is fairly easy to connect to database A and migrate all the data to database B. In fact, Kettle has two wizards (Copy Table and Copy Tables) that will handle this for you automatically, including generating the new target tables using the target database SQL syntax.

A third and more complex use case is *data synchronization*, meaning that two (or more) databases are being kept in sync using ETL tools. Although this can be achieved to a certain level, it is not the most common way of using ETL. Usually there are time constraints (changes made in database A need to be available in database B within the shortest achievable amount of time), which make the batch orientation of most ETL tools an unlikely choice for synchronization purposes.

ELT

ELT (short for extract, load, and transform) is a slightly different approach to data integration than ETL. In the case of ELT, the data is first extracted from the source(s), loaded into the target database, and then transformed and integrated into the desired format. All the heavy data processing takes place inside the target database. The advantage of this approach is that in general, a database system is better suited for handling large workloads where hundreds of millions of records need to be integrated. Database systems are also usually optimized for I/O (throughput), which helps to process data faster, too.

There is a big "but" here: In order to benefit from an ELT approach, the ELT tool needs to know how to use the target database platform and the specific SQL dialect being used. This is the reason there aren't a lot of ELT solutions on the market and why a general-purpose ETL tool such as Kettle lacks these capabilities. Nevertheless, most of the traditional closed source ETL vendors augmented their tools with pushdown SQL capabilities, basically resulting in supporting *ETLT* (extract, transform, load, transform) scenarios, where transformation can take place either within the engine (especially for operations not supported by the target database), or after loading inside the database. Leading database vendors such as Microsoft (SQL Server Integration Services) and Oracle (Oracle Warehouse Builder) have a headstart here and have had ETLT capabilities by design because their tools were already tightly integrated with the database. Oracle even bought Sunopsis some time ago, a company that created one of the few specialized ELT solutions on the market (now available as Oracle Data Integrator). Others, like Informatica and Business Objects, have added pushdown SQL capabilities to their products in later releases. An excellent overview of the pros and cons of ETL and ELT can be found in this blog by Dan Linstedt, the inventor of the Data Vault data warehouse modeling technique: `http://www.b-eye-network.com/blogs/linstedt/archives/2006/12/etl_elt_-_chall.php`.

A special product that should be mentioned here is LucidDB. This open source columnar BI database took the ETL and ELT concepts one step further and is capable of handling all the ETL functionality inside the database using extensions to standard ANSI SQL. To do this, LucidDB uses so called *wrappers* around different data sources. After a wrapper is defined for a source (which could be a database, a text file, or even a Web service), the source can be accessed using standard SQL to perform any operation that the SQL language supports. This architecture, of course, makes LucidDB also capable of acting as a lightweight EII solution (Enterprise Information Integration), which we cover in the next section.

EII: Virtual Data Integration

Both ETL and ELT move or copy data physically to another data store, from the OLTP to the data warehouse system. The reasons for using a separate data warehouse and hence, moving the data to that datastore, were explained in the earlier section "OLTP versus Data Warehousing." In more and more cases, however, there is no need to move or copy data. In fact, most users don't even care whether there is an ETL process and a data warehouse complemented with data marts: They just want access to their data! In a way, the data warehouse architecture displayed in Figure 1-1 is like the kitchen of a restaurant. As a customer, I don't really care how my food is prepared—I just want it served in a timely matter and it should taste great. Whatever happens behind those swinging doors is really none of my business. The same applies to a data warehouse: Users don't really care how their data is processed; they just want to access it quickly and easily.

So instead of physically integrating data, it is virtually integrated, making the data accessible in real time when it is needed. This is called *enterprise information integration*, or *EII*; other terms such as *data federation* and *data virtualization* are used as well and have the same meaning. The main advantage of this approach is, of course, the fact that data is always up-to-date. Another advantage is that there is no extra storage layer and no extra data duplication. Some data warehouse environments copy the same data three or four times: once in a staging area, then an operational data store (ODS), the data warehouse itself, and finally the data marts. By using virtual data integration techniques, the data is accessible for an end user as if it were a data mart, but in reality the EII tool takes care of all the translations and transformations in the background.

Although EII sounds like a winning strategy, it does have some drawbacks. Table 1-2 highlights the differences between using a physical and virtual data integration approach.

You can draw some conclusions from Table 1-2. One is that managing large volumes of cleansed, current data using a virtual approach will be challenging, if not impossible. Another conclusion might be that ETL is a tool that typically belongs in the physical integration category, but as you will see in Chapter 22, Pentaho Reporting can be used to invoke Kettle data integration jobs as a data source on an ad-hoc basis, offering some of the advantages of a virtual data warehouse solution combined with all the functionality of a full-fledged ETL tool.

Table 1-2: Virtual versus Physical Data Integration

CHARACTERISTIC	PHYSICAL	VIRTUAL
Data currency	○	●
Query performance/latency	●	⦿
Frequency of access	●	⦿
Diversity of data sources	⦿	●
Diversity of data types	⦿	●
Non-relational data sources	○	●
Transformation and cleansing	●	○
Performance predictability	●	⦿
Multiple interfaces to same data	○	●
Large query/data volume	●	○
Need for history/aggregation	●	○

Legend: ○=Weak, ⦿=Acceptable, ●=Strong

©Mark Madsen, Third Nature, Inc., 2009. All Rights Reserved. Used with Permission.

Data Integration Challenges

Data integration typically poses a number of challenges that need to be addressed and resolved before your solution is up and running. These challenges can be of a political, organizational, functional, or technical nature.

First and foremost, you'll need to find out which data is needed to answer the questions that your organization wants answered and build a solid business case and project plan for delivering that required information. Without a proper business case for starting a business intelligence project, you'll likely fail to get the necessary sponsorship. Technological barriers can be challenging but are in most cases removable; organizational barriers are much harder to take away. Although we won't cover these topics further in this book we just wanted to raise awareness about this important topic.

NOTE For more information, see Ralph Kimball's *Data Warehouse Lifecycle Toolkit* (2nd edition). Chapter 3 addresses gathering business requirements.

A good plan and a business case might get you the necessary support to start a project, but they are not enough to deliver successful solutions. For this, a solid methodology is needed as well, and of course a team of bright and experienced people won't hurt either. For many years IT projects were run using a waterfall approach where a project had its initiation phase, followed by design, development, testing, and moving to production. For business intelligence projects, of which ETL is an important part, this never worked quite well. As you'll see in the following section, a more agile approach fits the typical steps in a BI project much better.

On a more detailed level, you need to face the ETL design challenges, and define how your jobs and transformations will be built, not in a pure technical sense, but in a more functional way. There are many ways in which an ETL tool can be used to solve a specific problem, and no matter which approach is taken, it's mandatory that the same conceptual design is used to tackle similar problems. For instance, if the team decides to stage data to files first, stick to that and don't mix in staging data to a database for some parts of the solution, unless absolutely necessary.

After solving the organizational, project, and design challenges, the first technical issue is finding out where to get the data from, in what format it is available, and what exactly makes up the data you're interested in. Not only might it be a challenge to get access to the data, but connecting to the systems that host the data can be a major issue, too. A lot of the data available in enterprise information systems resides on mainframe computers or other hard-to-access systems such as older proprietary UNIX editions.

Large data volumes are also a challenge. Extracting all the data from the source systems every time you run an ETL job is not feasible in most circumstances. Therefore you need to resolve the issue of identifying what has changed in your source systems to be able to retrieve only the data that has been inserted, updated, or deleted. In some cases, this issue cannot be gracefully resolved and a brute force approach needs to be taken that compares the full source data set to the existing data set in the data warehouse.

Other challenges have to do with the way the data needs to be integrated; suppose there are three different systems where customer data is stored, and the information in these systems is inconsistent or conflicting? Or how do you handle incomplete, inconsistent, or missing data?

Methodology: Agile BI

One of the first challenges in any project is to find a good way to build and deliver the solution, including proper documentation. This holds true for any software package, not only for ETL. Over the years, many project management and software development methodologies have seen the light of day. Maybe you remember the days of the structured analysis and design methodologies, as developed in the 70s by people like Ed Yourdon and Tom DeMarco. These approaches all had a so-called waterfall model in common, meaning that one step in the analysis or design phase needs to be complete before you can move on to the next one. You can find more background information about these methods at `http://en.wikipedia.org/wiki/Structured_Analysis`.

During the 80s and 90s, developers found that these structured, waterfall-based methods weren't always helpful, especially when requirements changed during the project. To cope with these changing requirements, different "agile" development methods emerged, with Scrum arguably being the best-known example. What's so special about agile development? To make that clear, the founders and proponents of agile methodologies came up with the Agile Manifesto, which declares the values of the agile methodology:

- Individuals and interactions over processes and tools

- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The Agile Manifesto (full text available on `http://agilemanifesto.org`) also contains 12 guiding principles that define what Agile is. In short, it's about:

- Early and frequent delivery of working software
- Welcoming changing requirements
- Business and IT working closely together
- Reliance on self-motivated developers and self-organizing teams
- Frequent, face-to-face conversations to discuss issues and progress
- Keeping it simple: maximizing the amount of work not done

NOTE There is an abundant amount of information about agile development and the Scrum methodology available online. A good place to start is `http://en.wikipedia.org/wiki/Scrum_(development)`**.**

Now what has all this to do with business intelligence, and more specifically, ETL? Well, in the case of Pentaho and Kettle: everything! Pentaho has always embraced agile development methods (especially Scrum) to incrementally develop and release new versions of their BI platform and components. Development phases are measured and communicated in Sprints and Milestones, which is also reflected in the download versions you can obtain from the CI (Continuous Integration) repositories. During 2009, Pentaho decided to translate the experience the company had with using agile development methods into an agile BI approach. The intention is not only to support BI developers with a solid methodology, but also to adapt the Pentaho BI suite and all constituent components in such a way that they enforce, enable, and support an agile way of working. The first part of the BI suite that was changed and extended is Kettle, which is the reason we introduced the agile concepts here. You'll see more about the agile capabilities of Kettle later in the book, but as you might already wonder how Kettle supports an agile way of working, here's a look at the basic capabilities Kettle has to offer.

Once you've installed Kettle, it will take only a couple of minutes before you have connected to a data source, read some data, added a transformation, and delivered the data to a destination table. Because Kettle is an engine-based solution, it automatically takes care of a lot of things for you, which helps speed up the process. Kettle also contains a vast (and perhaps at first glance overwhelming) number of standard components and transformation steps. These are prebuilt code blocks that also help in minimizing the development effort and maximizing the speed of solution delivery. Changes in data fields or data types are automatically propagated to subsequent steps in the process, and Kettle can also generate the change scripts needed to alter the final destination table. The integrated modeling and ad-hoc visualization tools enable you to directly show the results of your work to the end user and play with the data in an iterative way. Developers and business users can therefore work closely together using the same toolset. Deviations from plan or from the user's expectations can be taken into account immediately and the jobs and transformations can be changed accordingly.

Because Kettle is tightly linked to the Agile BI initiative by Pentaho, it might be worthwhile to read up on the methodology and how Kettle supports it on the Agile BI wiki. You can find this info at `http://wiki.pentaho.com/display/AGILEBI/Welcome+to+Agile+Business+Intelligence`.

ETL Design

Even when (or perhaps, especially when) an agile approach is taken, your ETL process needs to be designed in one way or another. Because an ETL solution in many respects resembles a workflow or business process, it might make sense to use a flowchart drawing tool to create a high-level design before you start building. Most users will be familiar with flowchart diagrams and can comment and help make your design better. On a more detailed level, it is important to define which parts of the solution are to be reusable and which are not. For instance, creating a date dimension is usually a one-time effort within a data warehouse project. So for an individual project it makes sense to not spend too much time developing a flexible and database-independent solution and just develop a standard script for this. On the other hand, when you're a consultant working for many different customers, it absolutely makes sense to have a generic date dimension generator in your toolbox.

From this discussion, it's easy to see what the most important question is when you start building a data transformation: Should it be reusable in other parts of the solution or not? Depending on the answer, you might spend some extra time in making the transformation generic, for instance by adding extra parameters that enable you to choose a type of database, a date/time format, or other things that change between different solutions.

Data Acquisition

As explained earlier, getting access to and retrieving data from source systems is the first challenge you encounter when an ETL project is started. Don't automatically assume that this is only a technical problem; in many cases not being able to access data directly is caused by internal politics or guidelines. ERP vendors also try to make it difficult or even impossible to access the data in their systems directly. The widely used SAP/R3 system, for instance, has specific clauses in the software license that prohibit direct connections to the underlying database other than by the means provided by SAP. Most financial institutions that run their mission-critical systems on a mainframe also won't let you access these systems directly so you're dependent on data feeds delivered by FTP or via a web service. This isn't necessarily a disadvantage; the SAP system consists of more than 70,000 tables so finding the ones with the data you're interested in might be a very time-consuming task. For situations like this, you need tools that are able to interpret the ERP metadata, which displays a business view of the data. Fortunately, Kettle not only contains a standard data input step for acquiring data from SAP/R3, but also for getting data out of Salesforce.com, arguably the most used and advanced online CRM application. For other standard ERP and CRM solutions such as SugarCRM, OpenERP, ADempiere, or Peoplesoft, you might have to revert to third-party solutions or build your own input step. The capability to read data from a mainframe directly is unfortunately not available so in those cases it's best to have this data delivered from the system in a readable format such as ASCII or UniCode (older mainframe systems still use EBCDIC). More information about accessing ancient Cobol systems and the specific file format challenges involved can be found at `http://jymengant.ifrance.com/jymengant/jurassicfaq.html`.

Beware of Spreadsheets

A major challenge in the field of data acquisition has to do with the way and the format in which the data is delivered. A notorious troublemaker in this area is Excel, so the best advice we can give is to just never accept a data delivery in Excel, unless you can be sure it's system-generated and that it's created on the same machine by a process that's owned by the same user. Excel data problems occur frequently when different internationalization settings are used, causing dates and numeric fields to change from one session to another. As a result, your carefully designed and tested transformation will fail, or at least will generate incorrect results (which is basically the same but harder to track). Most problems, however, are caused by users who will tell you they haven't changed anything but who did, perhaps even unknowingly.

Design for Failure

Even if access to the data isn't a problem and the solution you've built looks rock solid, you always need to make sure that the data source is available before you kick off a process. One basic design principle is that your ETL job needs to be able to fail gracefully when a data availability test fails. Kettle contains many features to do this. You can:

- Test a repository connection.
- Ping a host to check whether it's available.
- Wait for a SQL command to return success/failure based on a row count condition.
- Check for empty folders.
- Check for the existence of a file, table, or column.
- Compare files or folders.
- Set a timeout on FTP and SSH connections.
- Create failure/success outputs on every available job step.

It's also a good idea to add error handling at the job and transformation level. When loading a data warehouse, dependencies often exist between tables. A good example is that a fact table cannot be loaded before all dimension loads have completed. When one of the dimension loads fails, the complete job should fail, too. A good design then lets you correct the errors and enables you to restart the job where only the failed and not-yet-run parts will execute.

Change Data Capture

The first step in an ETL process is the extraction of data from various source systems and passing the data to the next step in the process. A best practice here is the intermediate storage of the extracted data in staging tables or files to make restarts possible without the need of retrieving all data again. This seems like a trivial task, and in the case of initially loading a data warehouse it usually is, apart from challenges incurred from data volumes and slow network connections. But after the initial load, you don't want to repeat the process of completely extracting all data again. This wouldn't be of much use anyway because you already have an almost complete set of data, and it only needs to be refreshed to reflect the current status. All you're interested in is what has changed since the last data load, so you need to identify which records have been inserted, modified, or even deleted. The process of identifying these changes and only retrieving records that are different from what you already loaded in the data warehouse is called *Change Data Capture* or *CDC*.

Basically, there are two main categories of CDC processes, *intrusive* and *non-intrusive*. By intrusive, we mean that a CDC operation has a possible performance impact on the system the data is retrieved from. It is fair to say that any operation that requires executing SQL statements in one form or another is an intrusive technique. The bad news is that most available methods to capture changed data are intrusive, leaving only one non-intrusive option. CDC is covered in depth in Chapter 6.

Data Quality

Much of what is said in the previous section applies here as well: you have to assume that there are quality problems in your data and therefore need to design your transformations to handle these problems. In fact, this isn't entirely true: data quality problems need to be resolved in the source systems, not in the ETL process. However, fixing data quality issues before starting a data warehouse project is a luxury that not many organizations can afford. There's always a pressing need to deliver a solution quickly, and even if serious data quality problems are discovered during the project, they are usually dealt with later or not at all. Hence knowing what's wrong with your data and knowing what to do about it are essential parts of the ETL developer's job description.

Two categories of tools are available to deal with data quality problems. First, you'll need to define a baseline by profiling the data and investigating how good the quality actually is, using a data profiling tool. The purpose of this exercise is twofold: to communicate the results of this exercise back to the data owner (hopefully a business manager with the authority to do something about it), and to serve as input for the data validation steps in the ETL jobs. Second, there are data quality tools that constantly monitor and augment the data based on business and quality rules. In Kettle, the Data Validation step serves as a built-in data quality tool.

Data Profiling

One of the first things to do when starting an ETL project is profile the source data. *Profiling* will tell you how much data there is and what it looks like, both technically and statistically. The most common form of profiling is *column profiling*, where for each column in a table the appropriate statistics are created. Depending on the data type this will give you insight into things like the following:

- Number of NULL or empty values
- Number of distinct values
- Minimum, maximum, and average value (numeric fields)
- Minimum, maximum, and average length (string fields)
- Patterns (for example, ###-###-#### for phone numbers)
- Data distribution

Although most of these operations can be performed using Kettle transformations or just plain SQL, it's better to use a specialized tool such as Data Cleaner from eobjects. Chapter 6 covers data profiling in more detail.

WARNING **Data profiling will only get you so far; logical and/or cross-system quality issues cannot be detected by most data profiling tools, and in order to detect them, a global business glossary and metadata system needs to be in place first. These systems are still very rare.**

Data Validation

Profiling is meant for reporting and setting a baseline, while validation is part of the regular ETL jobs. A simple example is as follows: Some column in a source system can technically contain `NULL` values, but there is a business rule stating that this is a required field. Profiling revealed that there are several records with a `NULL` value in this column. To cope with this situation, a validation step is needed that contains the rule `NOT NULL` for this column, and when the column does contain a `NULL` value, an alternative action should be started. This could be omitting the record and writing it to an error table, replacing the `NULL` value with a default value such as `Unknown`, flagging the record as unreliable, or any other action that is deemed necessary.

ETL Tool Requirements

While this book is specifically about Kettle, it's useful to have an overview of the required features and functionality of an ETL tool in general. This will enable you to decide whether Kettle is the right tool for the job at hand. Each of the following sections first describes the requirement in general and then explains how Kettle provides the required functionality or feature.

Connectivity

Any ETL tool should provide connectivity to a wide range of source systems and data formats. For the most common relational database systems, a native connector (such as OCI for Oracle) should be available. At a minimum, the ETL should be able to do the following:

- Connect to and get data from the most common relational database systems including Oracle, MS SQL Server, IBM DB/2, Ingres, MySQL, or PostgreSQL.
- Read data from ASCII files in a delimited or fixed format.
- Read data from XML files (XML is the lingua franca of data interchange).
- Read data from popular Office formats such as Access databases or Excel spreadsheets.
- Get files from external sites using FTP, SFTP, or SSH (preferably without scripting).

In addition to this, there might be the need to read data using a web service, or to read an RSS feed. In case you need to get data from an ERP system such as Oracle E-Business Suite, SAP/R3, PeopleSoft, or JD/Edwards, the ETL tool should provide connectivity options for these systems as well.

Out of the box, Kettle has input steps for Salesforce.com and SAP/R3. For other ERP or financial systems, an alternative or additional solution might be required. Of course it's always possible to have these systems export a data set to an ASCII file and use that as a source.

Platform Independence

An ETL tool should be able to run on any platform and even a combination of different platforms. Maybe a 32-bit operating system works for the initial development phase, but when data volumes increase and available batch windows decrease, a more powerful solution is required. In other cases, development takes place on a Windows or Mac development PC, but production jobs run on a Linux cluster. You shouldn't have to take special measures to accommodate for this in your ETL solution.

Scalability

Scalability is a big issue; data volumes increase year after year and your systems needs to be able to handle this. Three options should be available for processing large amounts of data:

- **Parallelism:** Enables a transformation to run many streams in parallel, thus utilizing modern multi-core hardware architectures
- **Partitioning:** Enables the ETL tool to take advantage of specific partitioning schemes to distribute the data over the parallel streams
- **Clustering:** Enables the ETL process to divide the workload over more than one machine

This last option especially can be cost prohibitive with proprietary ETL solutions that are licensed per server or per CPU.

Kettle, being a Java-based solution, runs on any computer that has a Java Virtual Machine installed. Any step in a transformation can be started multiple times in parallel to speed up processing. Kettle will then determine how the data is distributed over the different streams. For better control, a partitioning scheme can be used to make sure that each parallel stream contains data with the same characteristics. This resembles how database partitioning works, but Kettle has no specific facilities to work with database partitions. (The benefit of having such a capability is debatable because the database itself is probably better capable of distributing data over the partitions than an ETL tool would be.)

The most advanced scalability feature arguably is the clustering option, which lets Kettle spread the workload over as many machines as deemed necessary. Part IV of this book covers all these scalability features in depth, but a good source to whet your appetite is the white paper written by one of the technical reviewers of this book, Nicholas Goodman of Bayon Technologies. It can be found at `http://www.bayontechnologies.com/bt/ourwork/pdi_scale_out_whitepaper.php`.

Design Flexibility

An ETL tool should provide a developer the freedom to use any desirable flow design and should not limit people's creativity or design requirements by offering only a fixed way of working. ETL tools can be classified as either process- or map-based. A map-based tool offers a fixed set of steps between source and target data, thus severely limiting the freedom to design jobs. Map-based tools are often easy to learn and get you started very quickly, but for more complex tasks, a process-based tool is most likely the better choice. With a process-based tool like Kettle, you can always add additional steps or transformations if needed because of changes in the data or business requirements.

Reuse

Being able to reuse existing parts of your ETL solution is also an indispensable feature. An easy way of doing this is to copy and paste or duplicate existing transformation steps, but that's not really reuse. The term *reuse* refers to the capability to define a step or transformation once and call the same component from different places. Within Kettle this is achieved by the Mapping step, which lets you reuse existing transformations over and over as subcomponents in other transformations. Transformations themselves can be used multiple times in multiple jobs, and the same applies to jobs which can be reused as subjobs in other jobs as well.

Extensibility

There isn't a single ETL tool in the world that offers everything that's needed for every imaginable data transformation task, not even Kettle. This means that it must be possible to extend the basic functionality of the tool in some way or another. Almost all ETL tools offer some kind of scripting option to programmatically perform complex tasks not available in the program itself. Only a few ETL tools, however, offer the option to add standard components yourself by offering an API or other means to extend the toolset. In between these options is a third way that lets you define functions that can be written using a script language and called from other transformations or scripts.

With Kettle, you get it all. Scripting is provided by the Java Script step, and by saving this as a transformation it can be reused in a mapping, resulting in a standard reusable function. In fact, any transformation can be reused in a mapping so creating standard components this way isn't limited to scripting alone. And Kettle is, of course, built with extensibility in mind, offering a plugin-enabled platform. The plugin architecture makes it possible for third parties to develop additional components for the Kettle platform. Several examples of these additional plugins are covered in this book, but it's important to note that all components you find in Kettle, even the ones that are available by default, are actually plugins. The only difference between built-in and third-party plugins could be the available support: If you buy a third-party plugin (for instance a SugarCRM connector), support is provided by the third party, not by Pentaho.

Data Transformations

A good deal of the work involved with an ETL project has something to do with transforming data. Between acquisition and delivery, the data needs to be validated, joined, split, combined, transposed, sorted, merged, cloned, de-duplicated, filtered, deleted, replaced, and whatnot. It's hard to tell what the minimum set of available transformations should be because data transformation requirements differ greatly between organizations, projects, and solutions. Nevertheless, there seems to be a common denominator of basic functions that most of the leading ETL tools (including Kettle) offer:

- Slowly Changing Dimension support
- Lookup values
- Pivot and unpivot
- Conditional split
- Sort, merge, and join
- Aggregate

The only difference between tools is the way these transformations need to be defined. Some tools, for instance, offer a standard SCD (Slowly Changing Dimension) transformation in a single step, while others generate the needed transformations with a wizard. Even Kettle doesn't cover all transformation requirements out of the box. A good example of a missing component is a hierarchy flattener. By *hierarchy* we mean a single table that refers to itself, for instance an employee table where each employee record has an employee ID and a manager ID. The manager ID in the employee record points to an employee ID of another employee who is the manager. Oracle has had a standard "connect by prior" function to cope with this for ages and some ETL tools have a similar feature; in Kettle you'd have to manually handle this issue.

Testing and Debugging

This requirement hardly needs further explanation. Even though an ETL solution is not (at least, we hope not) written in a program language such as Java or C++, it can be looked at as such. This means that what applies to application programming also applies to ETL development: Testing should be an integral part of the project. To be able to test, you need test cases that cover any possible (or at least, the most likely) scenario in a "what if" kind of way. The following are some examples of such scenarios:

- What if we don't get the data delivered on time?
- What if the process breaks halfway through the transformation?
- What if the data in column XYZ contains NULL values?
- What if the total number of rows transformed doesn't match the total number of rows extracted?
- What if the result of this calculation doesn't match the total value retrieved from another system?

Again, the message here is to design for failure. Don't expect that things will work; just assume that things will fail at some point. When designing tests, it's important to differentiate between *black box testing* (also known as *functional testing*) and *white box* testing. In case of the former, the ETL solution is considered a black box where the inner workings are not known to the tester. The only known variables are the inputs and the expected outputs. White box testing (also known as *structural testing*), on the other hand, specifically requires that the tester knows the inner workings of the solution and develops tests to check whether specific transformations behave as expected. Both methods have their advantages and disadvantages, which are covered in Chapter 11.

Debugging is an implicit part of white box testing and enables a developer or tester to run a program step by step to investigate what exactly goes wrong at what point. Not all ETL tools offer extensive debugging functionality where you can step through a transformation row by row, inspecting individual rows and variable allocations. Kettle offers extensive debugging features for both jobs and transformations, as covered in Chapter 11.

Lineage and Impact Analysis

A mandatory feature of any ETL tool is the ability to read the metadata to extract information about the flow of data through the different transformations. Data lineage and impact analysis are two related features that are based on the ability to read this metadata. Lineage is a backward-looking mechanism that will show for any data item where it came from and which transformations were applied to it. This would include calculations and new mappings, such as when price and quantity are used as input fields to calculate revenue. Even if the field's `price` and `quantity` are omitted from further processing, the data lineage function should reveal that the field `revenue` is actually based on `price` and `quantity`.

Impact analysis works the other way around: Based on a source field, the impact on the subsequent transformations and ultimately, destination tables is revealed. You can find in-depth coverage of these subjects in Chapter 14.

Logging and Auditing

The data in the data warehouse needs to be reliable and trustworthy because that's one of the purposes of a data warehouse: provide an organization with a reliable source of information. To guarantee this trustworthiness and have a system of record for all data transformations, the ETL tool should provide facilities for logging and auditing. Logging takes care of recording all the steps that are executed when an ETL job is run, including the exact start and end timestamps for every step. Auditing facilities create a complete trace of the actions performed on the data, including number of rows read, number of rows transformed, and number of rows written. This is a topic where Kettle actually leads the market, as you will see in Chapters 12 and 14.

Summary

This chapter introduced ETL and its history, and explained why data integration is needed. The basic building blocks of a Kettle solution were introduced briefly to give you a feeling for what will be covered in the rest of the book. We also explained the difference and similarities between ETL, ELT, and EII and showed the advantages of each method.

We presented the major challenges you might face when developing ETL solutions:

- Getting business sponsorship and creating a business case
- Choosing a good methodology to guide your work
- Designing ETL solutions
- Data acquisition and the problem with spreadsheets
- Handling data quality issues using profiling and validation

Finally, we highlighted the general requirements of an ETL tool and briefly described how Kettle meets the requirements for the following:

- Connectivity
- Platform independence and scalability
- Design flexibility and component reuse
- Extensibility
- Data transformations
- Testing and debugging
- Lineage and impact analysis
- Logging and auditing

All the topics introduced in this chapter are covered extensively in the rest of this book.