Chapter 7. Development

"By failing to prepare, you are preparing to fail." **Benjamin Franklin**

If you have been reading this book in sequence, you now have a detailed model of the challenges you face building your ETL system. We have described the data structures you need ([Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html)), the range of sources you must connect to ([Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html)), a comprehensive architecture for cleaning and conforming the data ([Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html)), and all the target dimension tables and fact tables that constitute your final delivery ([Chapters 5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch05.html) and [6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html)). We certainly hope that you can pick and choose a subset of all this for your ETL system!

Hopefully, you are at the point where you can draw a process-flow diagram for your proposed ETL system that clearly identifies at a reasonable level of detail the extracting, cleaning, conforming, and delivering modules.

Now it's time to decide what your ETL system development platform is and how to go about the development. If you have the luxury of starting fresh, you have a big fork in the road: Either purchase a professional ETL tool suite, or plan on rolling your own with a combination of programming and scripting languages. We tried to give you an even handed assessment of this choice in [Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html). Maybe you should go back and read that again.

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow : *Extract* → Clean → Conform → Deliver

In the next section, we give you a brief listing of the main ETL tool suites, data-proofing systems, data-cleansing systems, and scripting languages available as of this writing, but in writing a book intended to have a useful shelf life of several years, please understand that we intend this only as a general guide. We invite you to perform an Internet search for each of these vendors and scripting languages to get their latest offerings.

The first half of this chapter is a spirited and we hope entertaining tour through a number of basic low-level transforms you must implement. We have chosen to illustrate these with simple UNIX utilities like ftp, sort, gawk, and grep, keeping in mind that the professional ETL tool suites would have proprietary data-flow modules that would replace these examples.

The second half of this chapter focuses on DBMS specific techniques for performing high-speed bulk loads, enforcing referential integrity, taking advantage of parallelization, and troubleshooting performance problems.

Current Marketplace ETL Tool Suite Offerings

In [Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html), we discussed the pros and cons of purchasing a vendor's ETL tool suite or rolling your own ETL system with hand-coding. From the data warehouse point of view, the ETL marketplace has three categories: mainline ETL tool, data profiling, and data cleansing.

In alphabetical order, the main ETL tool suite vendors as of this writing, with product names where the company has products in other categories, are:

* Ab Initio
* Ascential DataStage
* BusinessObjects Data Integrator
* Cognos DecisionStream
* Computer Associates Advantage Data Transformation
* CrossAccess eXadas
* Data Junction Integration Studio (acquired by Pervasive)
* DataHabitat ZeroCode ETL
* DataMirror Transformation Server
* Embarcadero DT/Studio
* ETI (Evolutionary Technologies International)
* Hummingbird ETL
* IBM DB2 Data Warehouse Manager
* Informatica (PowerCenter and SuperGlue)
* Information Builders iWay
* Mercator Inside Integrator (acquired by Ascential)
* Microsoft SQL Server DTS (Data Transformation Services)
* Oracle9i Warehouse Builder
* Sagent Data Flow Server (acquired by Group 1)
* SAS Enterprise ETL Server
* Sunopsis

Most of the names in these lists are copyright, their owners. The main data-profiling vendors at the time of this writing are:

* Ascential (ProfileStage)
* Evoke Software
* SAS
* Trillium/Harte Hanks (with the Avelino acquisition)

The main data cleansing vendors at the time of this writing are:

* Ascential (acquisition of Vality)
* First Logic
* Group 1
* SAS DataFlux
* Search Software America
* Trillium (acquired Harte Hanks)

If you perform an Internet search for each of these products, you will get a wealth of information and their current statuses.

ETL tool suites typically package their functionality as a set of *transforms*. Each performs a specific data manipulation. The inputs and outputs of these transforms are compatible so that the transforms can easily be strung together, usually with a graphical interface. Typical categories of transforms that come built with dozens of examples in each category include:

* Aggregators
* General expressions
* Filters
* Joiners
* Lookups
* Normalizers
* Rankers
* Sequence generators
* Sorters
* Source readers (adapters)
* Stored procedures
* Updaters
* XML inputers and outputers
* Extensive facilities for writing your own transforms in a variety of languages

Current Scripting Languages

Interesting scripting languages available on a variety of platforms (typically UNIX, Linux, Windows, and, in some cases, IBM mainframes) include:

* JavaScript
* Perl
* PHP
* Python
* Tcl

All of these scripting languages excel at reading and writing text files and invoking sort routines including native OS sorting packages as well as commercial packages like SyncSort and CoSort. Several have good interfaces to commercial DBMSs as well.

Of course, one can always drop down to C or C++ and *do anything*. While all the ETL alternatives eventually allow escapes into C or C++, it would be unusual to build the entire ETL system at such a low level.

Time Is of the Essence

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow: *Extract* → Clean → Conform → Deliver

Throughout the ETL system, time or, more precisely, throughput, is the primary concern. Mainly, this translates to devising processing tasks that ultimately enable the fastest loading of data into the presentation tables and then the fastest end user response times from those tables. Occasionally, throughput rears its head when cleaning up unwieldy or dirty data.

Push Me or Pull Me

In every data warehouse, there inevitably is data that originates from flatfile systems. The first step to incorporating this data into the data warehouse is moving it from its host server to the ETL server. Flat files can either be pushed from the source host systems or pulled by the ETL server.

Which approach works best? The honest answer to this question is, well, *both*. However, the more important question to ask is *when?*—as in "When is the source file available to be moved?"

In many cases, the source files that must be moved are from operational business systems, and the files are often not available to be moved to the ETL server until after the operational systems nightly batch processes are completed. If the ETL server attempts to *pull* the file, it risks attempting to start the file transfer before the file is ready, in which case the data loaded into the warehouse might be incorrect or incomplete. In these situations, having the host system *push* the source files has the following advantages:

* The FTP step to push the source file can be embedded into the operational system's batch process so that the file is pushed as soon as it is prepared by the host system, thereby starting the ETL process at the earliest possible time and minimizing idle time during the load window.
* Errors in the process of preparing the source file can prevent the file transfer from being initiated, thereby preventing incorrect or incomplete data from being loaded into the data warehouse.

The ETL server must have an FTP host service running in order to support pushing source files from the host systems.

NOTE

In many cases, an interrupted FTP process must be restarted. The larger the download file and the tighter the batch window, the riskier relying on *simple* FTP becomes. If this is important to you, you should try to find a resumable FTP utility and/or verify the capabilities of your ETL tool suite to resume an interrupted transfer. Also, when looking at these added-value, higher-end, FTP-like capabilities, you may be able to get compression and encryption at the same time.

It is equally likely that some of the files needed by the ETL process are available at any time the ETL process needs them. In these cases, the ETL server can *pull* the files when it needs them. The ETL server must establish an FTP connection to the host file server. Running FTP from a Unix shell script or Windows batch file is quite simple. On both platforms, the filetransfer commands can be passed to FTP via an external command file, as in the following example.

NOTE

In the following pages, we describe many low-level data-manipulation commands including sorting, extracting subsets, invoking bulk loaders, and creating aggregates. For clarity, we show command-line versions of each of these commands. Obviously, in a commercial ETL tool suite, all of these commands would be invoked graphically by clicking with the mouse. Keep that in mind!

You can embed the following command in a Windows batch file:

**ftp -n -v -s:transfer.ftp**

-n options turns off login prompting

-v turns off remote messages

-s: specifies the command file, In this case "transfer.ftp"

The content of command file transfer.ftp might be something like:

open hostname

user userid password

cd /data/source

lcd /etl/source

ascii

get source\_1.dat

get source\_2.dat

get source\_3.dat

... ... ...

get source\_n.dat

bye

NOTE

On a UNIX system, the commands are the same, but the syntax for passing a command file is slightly different.

Ensuring Transfers with Sentinels

Whether you are pushing or pulling your flat-file sources, you need to be sure that your file transfer completes without any errors. Processing a partially transferred file can lead to corrupt or incomplete data being loaded into the data warehouse.

An easy way to ensure your transfers are complete is to use *sentinel* (or signal) files. The sentinel file has no meaningful content, but its mere existence signifies the readiness of the file(s) to which it relates.

* In the push approach, a sentinel file is sent after the last true source file is pushed. When the ETL receives the sentinel file, it signifies that all of the source files have been completely received and that the ETL process can now safely use the source files. If the transfer of the source files is interrupted, the sentinel file is not sent, and the ETL process suspends until the error is corrected and the source files are resent.
* Sentinel files can also be used in a pull environment. In this case, the source host sends a sentinel file only to notify the ETL server that the source files are available. Once the ETL server receives the sentinel, it can initiate the FTP process to pull the source files to begin the ETL process.

Either way, the ETL process must include a method to *poll* the local file system to check for the existence of the sentinel file. Most dedicated ETL tools include this capability. If you are manually developing the ETL process, a Windows NT/2000 server scheduled task or Unix cron job can accomplish this task.

Sorting Data during Preload

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow: Extract → Clean → Conform → Deliver

Certain common ETL processes call for source data to be sorted in a particular order to achieve the desired outcome. Such ETL processes as aggregating and joining flat-file sources require the data to be presorted. Some ETL tools can handle these tasks in memory; however, aggregating or joining unsorted data is significantly more resource-intensive and time-consuming than doing so on sorted data.

When the source data is contained in a database, sorting is easily accomplished by including an order by clause in the SQL that retrieves the data from the database. But if the source data is from flat files, you need to use a sort utility program to arrange the data in the correct order prior to the ETL process.

Sorting on Mainframe Systems

Every mainframe system includes either IBM's DFSORT or SyncSort's SORT utility program. Syncsort and DFSORT commands are virtually identical and are quite simple. With few exceptions, mainframe data files are formatted in fixed widths, so most sorts are accomplished by simply specifying the positions and lengths of the data elements on which the data are to be sorted. We use the sample sales file that follows to show how mainframe sorts are accomplished.

20000405026Discount Electronics 00014SUBWOOFER

^^^^0000015000019800000297000

20000406005City Electronics 00008AMPLIFIER

^^^^0000035000026100000913500

20000407029USA Audio and Video 00017KARAOKE MACHINE

^^^^0000020000008820000176400

20000410010Computer Audio and Video 00017KARAOKE MACHINE

^^^^0000010000008820000088200

20000411002Computer Audio and Video 00017KARAOKE MACHINE

^^^^0000035000008820000308700

20000411011Computer Audio and Video 00008AMPLIFIER

^^^^0000010000026100000261000

20000415019Computer Discount 00018CASSETTE PLAYER/RECORDER

^^^^0000020000006840000136800

20000418013Wolfe''s Discount 00014SUBWOOFER

^^^^0000025000019800000495000

20000418022USA Audio and Video 00008AMPLIFIER

^^^^0000015000026100000391500

20000419010Computer Audio and Video 00023MP3 PLAYER

^^^^0000010000017640000176400

20000419014Edgewood Audio and Video 00006CD/DVD PLAYER

^^^^0000020000044100000882000

20000419016Computer Audio and Video 00014SUBWOOFER

^^^^0000030000019800000594000

20000419021Computer Audio and Video 00014SUBWOOFER

^^^^0000035000019800000693000

20000419028Bayshore Electronics 00020CD WALKMAN

^^^^0000015000004140000062100

The COBOL copybook for this file would be:

01 SALES-RECORD.

05 SALE-DATE PIC 9(8).

05 CUSTOMER-ID PIC X(3).

05 CUSTOMER-NAME PIC X(27).

05 PRODUCT-ID PIC X(5).

05 PRODUCT-NAME PIC X(28).

05 UNIT-COST PIC 9(4)V99 COMP-3.

(this takes up only 4 physical bytes)

05 UNITS PIC 9(7).

05 UNIT-PRICE PIC 9(7)V99.

05 SALE-AMOUNT PIC 9(7)V99.

The basic structure of the SORT command is:

SORT FIELDS=(st,len,dt,ad)

where st denotes the starting position, len denotes the length, dt denotes the data type, and addenotes the sort order (ascending or descending). So, sorting our sales file by customer-id is coded as follows:

SORT FIELDS=(9,3,BI,A)

meaning, sort on positions 9 to 11 in ascending order, treating the data as *binary*. To perform sorts on multiple fields, simply supply the st,len,dt,ad parameters for each additional sort field.

For example, suppose your ETL task is to aggregate this sale data by year, product, and customer. The source data is at a daily grain, and its natural order is also by day. Aggregating data from its natural order would be a quite complex task, requiring creating, managing, and navigating arrays of memory variables to hold the aggregates until the last input record is processed and again navigating the memory arrays to load the aggregates into the warehouse. But by presorting the source data by the aggregate key (year + product-id + customer-id), the ETL task to aggregate the data becomes fairly simple. The command for sorting the data by the aggregate key is as follows:

SORT FIELDS=(1,4,BI,A,39,5,BI,A,9,3,BI,A)

Once sorted in this way, the ETL process to aggregate the data can be made extremely efficient. As source records are read, the values of the key fields year, product-id, and customer-id are compared to the key values of the preceding record, which are held in memory variables. As long as the keys are the same, the units and sales amounts are added to cumulative memory variables. When the keys change, the aggregate values for the preceding key are loaded to the warehouse from the memory variables, and the memory variables are reset to begin accumulating the aggregates for the new keys.

As discussed in earlier chapters, mainframe data often poses certain challenges unique to mainframes. The SORT utility has a rich set of data types to help you handle these challenges. While using BI (binary) as the data type works in many situations, there are a number of alternate data types that handle special situations, including those listed in [Table 7.1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html#alternate_mainframe_data_types).

**Table 7.1. Alternate Mainframe Data Types**

| **DATA TYPE** | **USAGE** |
| --- | --- |
| PD | Use to properly sort numeric values stored in *packed decimal* (or COMP-3) format. |
| ZD | Use to properly sort numeric values stored in *zoned decimal* format. |
| AC | Use to properly sort data by the ASCII codes associated with the data, rather than by the mainframe native EBCDIC codes. Use this format for mixed (alphanumeric) fields when the data is transferred from a mainframe to an ETL process on a Unix or Windows system. |
| *dates* | Believe it or not, you will likely encounter legacy system files with dates still in pre-Y2K formats (that is, without explicit centuries). SORT has a rich set of data types for handling such dates and assigning them to the proper century. |

This table represents just a small subset of the available data types. Many others are available for the multitude of numeric and other data formats you might encounter.

You can also mix data formats on a compound index. So, for example, sorting sales file by year and descending unit-cost uses the following command:

SORT FIELDS = (1,4,BI,A,72,4,PD,D)

Sorting on Unix and Windows Systems

Flat files on Unix and Windows systems, which are ASCII character-based, are not plagued by the antiquated data formats (packed-decimal, and so on) concocted on the mainframe systems of old to save disk space. But these systems present challenges of their own.

Among the most common sorting challenge you'll face is sorting delimited or other unstructured data files. In this context, *unstructured* refers to the fact that the data is not arranged in neat columns of equal width on every record. As such, unlike the mainframe examples, you can't specify the sort keys positionally.

Instead, the sort utility must be able to parse the records using the delimiters. (Of course, the mainframe utilities SyncSort and CoSort are available on Unix and Windows platforms, too.) The following extract shows the same sales data used earlier in the chapter now formatted as a comma delimited file.

04/05/2000,026,Discount Electronics,

00014,SUBWOOFER,124.74,15,198.00,2970.00

04/06/2000,005,City Electronics,00008,AMPLIFIER,164.43,35,261.00,9135.00

04/07/2000,029,USA Audio and Video,00017,KARAOKE MACHINE,

55.57,20,88.20,1764.00

04/10/2000,010,Computer Audio and Video,00017,KARAOKE MACHINE,

55.57,10,88.20,882.00,

04/11/2000,002,Computer Audio and Video,00017,KARAOKE MACHINE,

55.57,35,88.20,3087.00,

04/11/2000,011,Computer Audio and

Video,00008,AMPLIFIER,164.43,10,261.00,2610.00

04/15/2000,019,Computer Discount,00018,CASSETTE

PLAYER/RECORDER,43.09,20,68.40,1368.00

04/18/2000,013,Wolfe''s Discount,

00014,SUBWOOFER,124.74,25,198.00,4950.00

04/18/2000,022,USA Audio and

Video,00008,AMPLIFIER,164.43,15,261.00,3915.00

04/19/2000,010,Computer Audio and Video,00023,MP3

PLAYER,111.13,10,176.40,1764.00

04/19/2000,014,Edgewood Audio and Video,00006,CD/DVD

PLAYER,277.83,20,441.00,8820.00

04/19/2000,016,Computer Audio and

Video,00014,SUBWOOFER,124.74,30,198.00,5940.00

04/19/2000,021,Computer Audio and

Video,00014,SUBWOOFER,124.74,35,198.00,6930.00

04/19/2000,028,Bayshore Electronics,00020,CD WALKMAN,

16.08,15,21.40,621.00

The basic syntax for the Unix sort command is as follows:

sort +start\_field\_number -stop\_field\_number file

Fields are numbered beginning from zero, so in the sales file, the field numbers are as follows:

(0) SALE-DATE

(1) CUSTOMER-ID

(2) CUSTOMER-NAME

(3) PRODUCT-ID

(4) PRODUCT-NAME

(5) UNIT-COST

(6) UNITS

(7) UNIT-PRICE

(8) SALE-AMOUNT

The default delimiter for the sort program is white space. The --t option allows you to specify an alternate delimiter. To replicate the first example again, sorting by customer-id, the sort command would be:

sort -t, +1 −2 sales.txt > sorted \_sales.txt

which means, begin sorting on column 1 (customer-id) and stop sorting on column 2 (customer-name). The sort output is currently directed to standard output (terminal), so you redirect the output to a new file: sorted\_sales.txt.

If the stop column is not specified, sort sorts on a compound key consisting of every column beginning with the start column specified.

Look at how to perform the aggregate key sort (year + product-id + customer-id) used earlier in the chapter to prepare the data for an aggregation ETL process. Year is last part of the field 0, product-id is field 3, and customer-id is field 1. The main challenge is limiting the sort to use only the year portion of the date. Here's how:

sort -t, +0.6 −1 +3 −4 +1 −2 sales.txt > sorted\_sales.txt

To sort on the year portion of the date (field 0), you specify the starting byte within the field following a period. As with the field numbers, byte numbers start with 0, so +0.6 means to sort on the seventh byte of the date.

Up to this point, these Unix sort examples have used alphabetic sorts. However, alphabetic sorts won't yield the desired results on quantitative numeric fields. For example, sorting the sales-file unit cost alphabetically would yield incorrect results—the CD WALKMAN, with a cost of 16.08, would be placed after the SUBWOOFER, with a cost of 124.74. To solve this problem, you need to specify that the unit-cost field is numeric, as follows:

sort -t, +5n −6 sales.txt > sorted\_sales.txt

To change the sort order from ascending to descending, use the --r (reverse) option. For example, sorting the sale data by descending year + unit cost is specified by the following:

sort -t, +0.6r −1 +5n −6 sales.txt > sorted\_sales.txt

Other useful sort options are listed in [Table 7.2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html#switches_for_the_unix_sort_command).

**Table 7.2. Switches for the UNIX sort command**

| **DATA TYPE** | **USAGE** |
| --- | --- |
| -f | Ignore case in alphabetic sort fields |
| -b | Ignore leading blanks in sort fields |
| -d | Ignore punctuation characters |
| -i | Ignore nonprintable characters |
| -M | Sort 3-letter month abbreviations (for example, JAN precedes FEB, and so on) |

NOTE

A rich set of Unix utility commands, including *sort, grep*, and *gawk* to name few key utilities, have been *ported* to the Windows operating system and are available as freeware. There are many Web sites from which you can obtain these utilities. We found a relatively complete set in a zipped file at http://unxutils.sourceforge.net/.

Trimming the Fat (Filtering)

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow: Extract → Clean → Conform → Deliver

Source files often contain loads of data not pertinent to the data warehouse. In some cases, only a small subset of records from the source file is needed to populate the warehouse. Other times, only a few data elements from a wide record are needed. One sure way to speed up the ETL process is to eliminate unwanted data as early in the process as possible. Creating extract files on the source host system provides the greatest performance gain, because, in addition to the improvement in the ETL process itself, the time spent on file transfers is reduced in proportion to the reduction in file size. Whether you shrink a source file by picking, say, half the records in the file or half the fields on each record, you save time transferring the data to the ETL server and save I/O time and memory processing the smaller file during the ETL process.

The easiest extract files to create are those where only a subset of the source file records is needed. This type of extract can generally be created using utility programs, which are also typically the most efficient running programs on the system.

The following sections discuss creating extracts on mainframe systems and Windows and Unix systems.

Extracting a Subset of the Source File Records on Mainframe Systems

On mainframe systems, the SORT utility happens to be perhaps the fastest and easiest way to create extract files without writing COBOL or fourth-generation language (SAS, FOCUS, and so on) programs.

The simplest case is to create an extract in which only a subset of the records is needed. SORT allows to you specify source records to either include or omit from the extract file.

INCLUDE COND=(st, len, test, dt, val)

OMIT COND=(st, len, test, dt, val)

The st indicates the start position of the input field, len is its length, test is the Boolean test to perform, dt is the data type of the input field, and val is the value to compare against. For this example, we use the sample sales file from the prior sort examples in the chapter. Here's how to extract only records for sales from the year 2000 and higher.

SORT FIELDS=COPY

INCLUDE COND=(1,4,GE,CH,C'2000')

This could also be coded with an EXCLUDE as:

SORT FIELDS=COPY

OMIT COND=(1,4,LT,CH,C'2000')

Compound conditions are created by joining condition sets with AND or OR clauses. To select only records from the year 2000 and later for customer-ids over 010, use the following:

SORT FIELDS=COPY

INCLUDE COND=(1,4,GE,CH,C'2000', AND,9,3,GT,CH,C'010')

More complex conditions can be created using a combination of ANDs, Orss, and parentheses to control the order of execution. You can even search for fields containing a certain value. For example, to choose customers with the word *Discount* in their names, code the INCLUDE statement as follows:

SORT FIELDS=COPY

INCLUDE COND=(12,27,EQ,CH,C'Discount')

Because the 27-byte input field specified is longer than the constant Discount, SORT searches through the entire input field for an occurrence of the constant. (This is equivalent to coding a SQL where clause of LIKE '%Discount%')

Extracting a Subset of the Source File Fields

Creating an extract file containing only the fields necessary for the data warehouse ETL process can have an enormous impact on the size of the ETL source files. It is not at all uncommon to have source files with dozens and dozens of data elements of which only a small handful are needed for the ETL process. The impact of extracting only the required fields can have an enormous impact on file size even when the source file record is relatively small. Considering that some source files have millions of records, extracting only the required fields can shave tens or hundreds of megabytes off the data to be transferred to and processed by the ETL server.

Lo and behold, SORT can also handle the task of selecting a subset of the fields in a source file to shrink the amount of data that must be transferred to the ETL server using the OUTFIL OUTREC statement. The sales file we have used in the examples thus far has a record length of 100 bytes. Suppose your ETL process required only the sale-date, customer-id, product-id, unit-cost, units, and sale-amount fields, which total 36 bytes. An extract with only these fields would be about one-third the size of the full source file. To shrink it further, you can choose only records from the year 2000 and later.

SORT FIELDS=COPY

INCLUDE COND=(1,4,CH,GE,C'2000')

OUTFIL OUTREC=(1,8,9,3,39,5,72,4,76,7,92,9)

In this simplest form, the OUTREC clause comprises simply pairs of starting positions and lengths of the fields to copy to the extract file. However, this still leaves you with some undesirable remnants of mainframe days. The unit-cost, units, and sale-amount are still in their mainframe storage formats. These fields are not usable in these native mainframe formats when transferred to the ETL server. To be usable on the Unix or Windows ETL server, you must reformat these numeric fields to *display*format.

SORT FIELDS=COPY

INCLUDE COND=(1,4,CH,GE,C'2000')

OUTFIL OUTREC=(1,8,9,3,39,5,

72,4,PD,EDIT=IT.TT,LENGTH=7,

76,7,ZD,EDIT=IT,LENGTH=7,

92,9,ZD,EDIT=IT.TT,LENGTH=10)

In this format, the unit-cost, which is stored in *packed numeric* format on the source file, is exploded to a 7-byte display field taking the form *9999.99*. Likewise, the units and sale-amount are reformatted to display as *9999999* and *9999999.99*, respectively.

Clearly, the mainframe SORT utility is a powerful ally in your pursuit of mainframe data. The techniques cited in the preceding examples demonstrate just a subset of its rich functionality. Proficiency with SORT can be your ticket to self-sufficiency when you need to acquire mainframe data.

Extracting a Subset of the Source File Records on Unix and Windows Systems

Now let's look at how to accomplish these same extract tasks on Unix and Windows systems. Again, we use a Unix utility, gawk, that has been ported to Windows. Gawk is the GNU version of the programming language awk. The basic function of awk is to search files for lines (or other units of text) that contain certain patterns.

The syntax we use for gawk is as follows:

gawk -f cmdfile infile > outfile

The --f option specifies the file containing the gawk commands to execute. The infile specifies the input source file, and *outfile* specifies the file to which the gawk output is redirected.

The first extract task is to select only the sales from 2000 and later. The gawk command would be something like the following:

gawk -fextract.gawk sales.txt > sales\_extract.txt

The extract.gawk command file contains the following:

BEGIN {

FS=",";

OFS=","}

substr($1,7,4) >= 2000 {print $0}

The BEGIN {..} section contains commands to be executed before the first source record is processed. In this example, FS="," and OFS="," stipulate that fields in the input and output files are delimited by commas. If not specified, the default delimiters are spaces.

The extract logic is contained in the statement substr($1,7,4) >= 2000. It says to select records where the seventh through tenth bytes of the first ($1) field are greater than or equal to 2000.

The {print $0} statement says to output the entire source record ($0) for the selected records.

Compound conditions are created by joining condition sets with && (and) or || (or) clauses. To select only records from the year 2000 and later for customer-ids over 010, use the following:

BEGIN {

FS=",";

OFS=","}

substr($1,7,4) >= 2000 && $1 > "010" {print $0}

And to find the records where the customer-name contains *Discount*:

BEGIN {

FS=",";

OFS=","}

$3~/Discount/ {print $0}

As you can see, with a bit of knowledge of the gawk command, you can very easily create these time-saving and space-saving extracts.

Extracting a Subset of the Source File Fields

You can also use gawk to create field extracts. As before, suppose you want to create an extract containing only the sale-date, customer-id, product-id, unit-cost, units, and sale-amount fields. Again, you want only to extract records from 2000 and later. Here's how:

BEGIN {

FS=",";

OFS=","}

substr($1,7,4) >= 2000 {print $1,$2,$4,$6,$7,$9}

The print statement specifies the columns to include in the output.

NOTE

Take note that in gawk, the fields are numbered starting with $1, and $0 refers to the entire input record. This differs from the sort command, where $0connotes the first field.

Now suppose you want to create an extract file that has fixed field widths rather than a delimited file. Well, gawk can do that as well. Here's how you make the prior extract into a fixed-width file:

BEGIN {

FS=",";

OFS=","}

{substr($1,7,4) >= 2000

{printf "%-11s%-4s%-6s%07.2f%08d%010.2f\ n", $1,$2,$4,$6,$7,$9}}

Here, the printf command (formatted print) takes the place of the regular print command. printfis followed by a format string. The % sign denotes the beginning of a format, and the number of formats must match the number of output fields. The formats you commonly use are:

* %n**s:** For text strings, n is the minimum output length
* %0n**d:** For decimal numbers, n is the minimum output length
* %0n.m**f:** For floating point numbers, n is the total number of digits, m the decimal digits.

By default, data is right-justified. To left-justify data (as you would for most text fields), precede the field length with a dash as in the preceding example. To left-pad numeric formats with zeros, precede the length with a zero, (for example, %07.2f) The newline indicator \n at the end of the format string tells gawk to put each output record on a new line.

Creating Aggregated Extracts on Mainframe Systems

Suppose you want to summarize the sample sales files by month, customer, and product, capturing the aggregate units and sales-amounts. Here's the mainframe SORT commands to accomplish this:

INREC FIELDS=(1,4,5,2,9,3,39,5,3Z,76,7,3Z,91,9)

SORT FIELDS=(1,14,CH,A)

SUM FIELDS=(15,10,ZD,25,12,ZD)

You can see two new commands here—INREC and SUM. INREC work the same way as OUTREC, except that it operates on the input records before any sort operations are processed. In this case, the input records are reformatted to include only the fields needed for the aggregate—year, month, customer-id, product-id, units, and sale-amount. Take note as well of the two 3Z entries. These left-pad units and sale-amount with zeros prevent arithmetic overflows from occurring from the SUM operation. The effect is to increase the size of these fields by three bytes each.

Next, the SORT command specifies the fields used to sort the file. The SORT FIELDS act as the key for the SUM operation, essentially acting like a SQL GROUP BY clause. But note that the SORT FIELDS use the reformatted record layout—so the SORT FIELDS can be simply defined as positions 1 through 14, which now contain year, month, customer-id, and product-id.

Finally, the SUM command specifies the quantitative fields to be summed (or aggregated). Again, the reformatted field positions (and lengths, in this case) are used. So whereas units occupied positions 76–82 (a total of 7 bytes) in the source file, in the reformatted records, units occupies positions 15–24 (10 bytes).

Using this technique, the output file is only 36 bytes wide (versus the original 100 byte source records) and contains only one record per combination of year, month, customer-id, and product-id in the file. The network and ETL server will thank you for shrinking the size of the source file in this way.

Note, however, that transaction systems are often configured with minimal temp space available. This may affect your decision to compress data at the source during extraction. See the discussion on this topic later in this chapter under "Using Aggregates and Group Bys."

Creating Aggregated Extracts on UNIX and Windows Systems

Accomplishing the same aggregation using the UNIX/Windows utilities is a bit more complex but not much. You need to use both the sort and gawk utilities together. The sort output is *piped* to the gawk command using the |pipe character. Here's the command:

sort -t, +0.6 −1 +0.0 −0.2 +1 −2 +3 −4 | gawk -fagg.gawk > agg.txt

First, review the sort.

+0.6 −1 = year

+0.0 −0.2 = month

+1 −2 = customer-id

+3 −4 = product-id

The agg.gawk command file follows. We've added comments (preceded by #) to explain how it works:

# set delimiters

BEGIN {

FS=",";

OFS=","}

#initialize variables for each record

{inrec += 1}

{next\_year=substr($1,7,4) substr($1,1,2)}

{next\_cust=$2}

{next\_product=$4}

#after a new year and month record

#write out the accumulated total\_units and total\_sales for the

prior year

inrec > 1 && ( \

next\_year != prev\_year \

|| next\_cust != prev\_cust \

|| next\_product != prev\_product) \

{print prev\_year,prev\_cust,prev\_product,total\_units,total\_sales}

#accumulate the total\_sales sales and count of records

{total\_units += $7}

{total\_sales += $9}

#if the year changed reinitialize the aggregates

next\_year != prev\_year {

total\_units = $7;

total\_sales = $9}

#store the year (key) of the record just processed

{prev\_year = next\_year}

{prev\_cust = next\_cust}

{prev\_product = next\_product}

#after the last record, print the aggregate for the last year

END {print prev\_year,prev\_cust,prev\_product,total\_units,total\_sales}

It's a bit more complex than the mainframe sort but not very much. It is simply a matter of keeping track of the key values as records are processed and writing out records as the keys change. Note also the END command.

This command ensures that the last aggregate record is output. Once you get familiar with the operators and learn how the flow of control works, you'll be aggregating in no time.

Using Database Bulk Loader Utilities to Speed Inserts

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow: Extract → Clean → Conform → Deliver

If after using sorting, extracting, and aggregating techniques to get your source data to the ETL server as quickly as possible, you still face a daunting amount of data that needs to be loaded into the data warehouse regularly, it's time to master the bulk-load functionality of your database management system. Bulk loaders can interact with the database system in a more efficient manner than plain old SQL can and give your ETL a tremendous performance boost. We use Oracle's SQL\*LOADER utility to discuss the benefits of bulk loaders. You can find similar bulk-load functionalities in most other database management systems.

NOTE

One important caveat is that many bulk loaders are limited to handling inserts into the database. As such, they can provide a real benefit for inserting large volumes of data, but if your process involves updating existing records, you may be out of luck. Depending on the number of rows you need to insert and update, you may find that with careful preprocessing of your input data, you can separate the updates from the inserts so that at least the inserts can be run in pure *bulk-loader mode*. Note that IBM's Red Brick system supports UPDATE else INSERT logic as part of its bulk loader.

In its basic *conventional path* method, SQL\*LOADER uses INSERT statements to add data to tables, and the database operates in the same manner as if the inserts were part of a regular SQL procedure. All indexes are maintained; primary key, referential integrity, and all other constraints are enforced; and insert triggers are fired. The main benefit to using SQL\*LOADER in this mode is that it provides a simple way to load data from a flat file with minimal coding.

A variety of syntax styles for invoking SQL\*LOADER exist. Here's an example for loading the sales file we used in the previous examples into an Oracle table with SQL\*LOADER.

sqlldr userid=joe/etl control=sales.ctl data=sales.txt log=sales.log

bad=sales.bad rows=1000

The control file sales.ctl would contain the following:

LOAD DATA

APPEND INTO TABLE SALES

FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'

(SALE\_DATE DATE(20) "MM/DD/YYYY",

CUSTOMER\_ID,

CUSTOMER\_NAME,

PRODUCT\_ID,

PRODUCT\_NAME,

UNIT\_COST,

UNITS,

UNIT\_PRICE,

SALE\_AMOUNT)

Of course, the target table SALES would have to already exist in Oracle. Again, this simple example uses conventional SQL INSERT functionality to load data into the database. Next, we want to look at ways to improve performance.

The second, performance-enhancing mode for SQL\*LOADER is *direct* mode. Changing the prior load to use direct mode is achieved by simply adding the direct=true clause to the sqlldr command, as shown in the following.

sqlldr userid=joe/etl control=sales.ctl data=sales.txt log=sales.log

bad=sales.bad rows=1000 direct=true

Here's how direct mode increases performance:

1. SQL\*LOADER places an exclusive lock on the table, preventing all other activity.
2. Database constraints (primary and unique key constraints, foreign key constraints, and so on) are not enforced during direct loads. If violations occur, the associated indices are left in an unstable state and require manual clean up before rebuilding the indices.
3. Foreign key constraints are disabled by the direct load and must be re-enabled after the load. All rows are checked for compliance with the constraint, not just the new rows.
4. Insert triggers do not fire on rows inserted by direct loads, so a separate process must be developed to perform the actions normally handled by the triggers if necessary.

So the efficiencies of direct load don't come free. We'd be particularly wary of using direct loads if you expect dirty data that will prevent your primary and foreign key constraints from being re-enabled after the load. But if you have robust processes for ensuring that the data to be loaded is clean, direct loads of large source files is the way to go.

Whether you are using conventional or *direct path* mode, SQL\*LOADER has a fairly rich set of functionality beyond the simple example shown previously. Some key functions include the following:

* Handles fixed-width, delimited, and multiline input
* Accepts input from multiple files
* Loads to multiple target table
* Loads partitioned tables
* Manages and updates indexes efficiently

A number of other, more programmatic features allow conditional processing, value assignments, and so on. However, these features should be avoided, since they generally operate on each input row and can considerably degrade the performance of the bulk load. After all, performance is what using the bulk loader is all about.

Preparing for Bulk Load

Many of the ETL tools on the market today can stream data directly from their tool through the database bulk-load utility into the database table. But not all of the tools utilize the bulk-load utilities the same way. Some are more efficient than others, and some require extra plug-ins to make them compatible with bulk loaders. Regardless of how you pass data, as an ETL developer, it is important that you understand how to prepare your data to be processed by a bulk-load utility.

Bulk loading is the most efficient way to get data into your data warehouse. A bulk loader is a utility program that sits outside of the database and exists for the sole purposes of getting large amounts of data into the database very quickly. Each database management system has a different, proprietary, bulk-load utility program. The popular ones are listed in [Table 7.3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html#bulk_load_utilities).

Generally speaking, the various bulk-load utilities work in the same way. For the purpose of illustrating the functionality of a bulk-load utility, we'll discuss Oracle's SQL\*Loader; at the time of this writing, we believe it is the common denominator of bulk loaders in the domain of data warehouses.

**Table 7.3. Bulk Load Utilities**

| **DBMS** | **BULK LOAD UTILITY NAME** | **COMMENTS** |
| --- | --- | --- |
| Oracle | SQL\*Loader | Requires a control file that describes the data file layout.  Two important parameters for optimal performance are:  DIRECT={TRUE | FALSE } PARALLEL={TRUE | FALSE} |
| Microsoft SQL Server | Bulk Copy Program (BCP) | Microsoft also offers BULK INSERT that can be faster than BCP. It saves a significant amount of time because it doesn't need to utilize the Microsoft NetLib API. |
| IBM DB2 | DB2 Load Utility | DB2 accepts Oracle Control and Data files as input sources. |
| Sybase | Bulk Copy Program (BCP) | Also supports DBLOAD with the parameter BULKCOPY = 'Y'. |

Once you understand the general concepts of bulk loading, the similarities among the loaders makes learning each specific utility a breeze.

Bulk loaders typically need two files to function properly:

* **Data file**. The data file contains the actual data to be loaded into the data warehouse. Data can be in various file formats and layouts, including a variety of delimiters. All of these parameters are defined in the control file.
* **Control file**. The control file contains the metadata for the data file. The list of the various parameters is extensive. Following is a list of the basic elements of SQL\*Loader control file.
  + The location of the source file
  + Column and field layout specifications
  + Data-type specifications
  + The data mapping from the source to the target
  + Any constraints on the source data
  + Default specifications for missing data
  + Instructions for trimming blanks and tabs
  + Names and locations of related files (for example, event log, reject, and discarded record files)

For a comprehensive guide to SQL\*Loader command syntax and usage, refer to *Oracle SQL\*Loader: The Definitive Guide*, by Jonathan Gennick and Sanjay Mishra (O'Reilly & Associates, 2001).

NOTE

Even if you must pay the penalty for the I/O of writing data to a physical file before it is bulk loaded into the data warehouse, it is still likely to be faster than accessing the database directly and loading the data with SQL INSERT statements.

Many ETL tools can *pipe* data directly into the database via the bulk-load utility without having to place data on disk until it hits its final destination—the data warehouse fact table. Others can create the required control and data files on your files system. From there, you need to write a command-line script to invoke the bulk loader and load the data into the target data warehouse.

The main purpose of purchasing an ETL tool is to minimize hand-coding any routines, whether extracting, transforming, or loading data. But no tool on the market can solve every technical situation completely. You'll find that seamlessly pipelining data through bulk loaders or any third-party load utilities will be a bit of a challenge. Experiment with tool plug-ins and other application extenders such as *named pipes*, and exhaust all options before you determine bulk loading is not feasible.

If you have not yet purchased your ETL tool, make sure to test potential products for their compatibility with your DBMS bulk-load utility during your proof-of-concept. If you already own an ETL tool and it cannot prepare your data for bulk loading, do not throw it away just yet. You need to prepare the data manually. Configure your ETL tool to output your data to a flat file, preferably comma delimited. Then, create a control file based on the specifications of the output file and required load parameters. The control file should need to be changed only when physical attributes change within the source or target, like when new columns are added or data types are modified.

Managing Database Features to Improve Performance

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow:Extract → Clean → Conform → Deliver

As we all know, there's more to the database than tables and the data contained therein. Powerful features like indexes, views, triggers, primary and foreign key constraints, and column constraints are what separate a database management system from a flat-file system. Managing these features can consume significant amounts of system resources as your database grows and, as a result, can drag down the performance of the ETL load process.

With this in mind, the first thing to do is review the database design and remove any unnecessary indexes, constraints, and triggers. Then consider the following options to improve load performance:

1. Disable foreign key (referential integrity) constraints before loading data. When foreign key constraints are enabled, for each row loaded the database system compares the data in foreign key columns to the primary key values in the parent table. Performance can be enhanced considerably by disabling foreign key constraints on fact tables having several foreign key constraints.

Remember, though, that the database validates every row in the table (not just new ones) when you enable foreign key constraints after the load. Make sure your foreign key columns are indexed to ensure that the re-enabling the constraints does not become a bottleneck in itself.

1. Keep database statistics up to date. Database statistics managed by the database management system track the overall sizes of tables, the sizes and number of unique values in indexes, and other facts about the efficiency of how data is stored in the database. When an SQL SELECT statement is submitted to the database management system, it uses these statistics to determine the fastest access path to supply the requested data. Optimally, you should update the statistics after each load. However, if your load process is frequent (daily) and the daily percentage change in the size of the database is relatively small, updating statistics weekly or monthly should be sufficient to keep performance levels high. Partitioning large tables decreases the time it takes to update statistics, since the statistics need not be refreshed on the static (or near-static) partitions but only on the *current* partition.
2. Reorganize fragmented data in the database. Tables become fragmented when rows are frequently updated and/or deleted, and response time degrades as a result.

When dealing with large fact tables, one way to minimize the occurrence of such fragmentation is to create *partitioned* tables. Partitioned tables are typically organized by time period (for example, a sales table with separate partitions for each year). Once a year is complete, the partition containing the sales data for that year will in all likelihood remain static and thus no longer be susceptible to fragmentation. ETL tools have the ability to automatically streamline loads based on the partition scheme specified in the DBMS dictionary.

Data for the current year, though, is constantly being deleted and reloaded, and so the current partition becomes fragmented. Reorganizing a fragmented table rewrites the data in the table in contiguous storage blocks and eliminates dead space that arises when rows are updated and deleted. If your load process performs updates and deletes on large (fact) tables, consider reorganizing the table every month or so or more frequently if warranted. The reorganization can be set to run each time data is loaded, if significant fragmentation occurs with each load.

Again, partitioning reduces the time it takes to reorganize tables. The older, static partitions will rarely, if ever, need to be reorganized. Only the current partition will need reorganizing.

The Order of Things

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow:Extract → Clean → Conform → Deliver

The ordinal position of jobs within a batch is crucial when you are loading a data warehouse, primarily because the ETL needs to enforce referential integrity in the data warehouse. Referential integrity (RI) means that a primary key must exist for every foreign key. Therefore, every foreign key, which is known as the child in a referential relationship, must have a parent primary key. Foreign keys with no associated parents are called orphans. It is the job of the ETL process to prevent the creation of orphans in the data warehouse.

In transaction systems, RI is usually enforced within the database management system. Database-level RI enforcement is required in a transaction environment because humans enter data one row at a time—leaving a lot of room for error. Errors or actions that create RI violations cause data to become corrupt and of no use to the business. Users find amazing ways to unintentionally corrupt data during data entry. Once data is corrupt, it is worthless—a cost that cannot be overturned.

Enforcing Referential Integrity

Unlike transaction systems vulnerable to volatile data-entry activity, the data warehouse has its data loaded in bulk via a controlled process—the ETL system. The ETL process is tested and validated before it ever actually loads production data. The entry of data into the data warehouse is in a controlled and managed environment. It's common practice in the data warehouse to have RI constraints turned off at the database level, because it depends on the ETL to enforce its integrity.

Another reason RI is typically disabled in the DBMS is to minimize overhead at the database level to increase load performance. When RI is turned on within the database, every row loaded is tested for RI—meaning every foreign key has a parent in the table that it references—before it is allowed to be inserted.

RI in the data warehouse environment is much simpler than in transaction systems. In transaction systems, any table can essentially be related to any other table, causing a tangled web of interrelated tables. In a dimensional data warehouse, the rules are simple:

* Every foreign key in a fact table must have an associated primary key in a dimension.
* Every primary key in a dimension does not need an associated foreign key on a fact table.

Those trained in normalization know this is called a zero-to-many relationship. If you already have a dimensional data warehouse implemented, or have read any of the *Toolkit* books, you know that not all dimensional models are that straightforward. In reality, a fact can be associated to many records in a dimension (with a bridge table as described in [Chapters 5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch05.html) and [6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html)) and dimensions can be snowflaked. In addition to facts and dimensions, the ETL must contend with outriggers and hierarchy tables. The ETL team must understand the purpose and functions of each of the types of tables in the dimensional data model to effectively load the data warehouse. Review [Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html) for more information on the different types of tables found in a dimensional model.

The following list is offered as a guide to the ordinal position of load processes for a given data mart.

1. Subdimensions (outriggers)
2. Dimensions
3. Bridge tables
4. Fact tables
5. Hierarchy mappings
6. Aggregate (shrunken) dimensions
7. Aggregate fact tables

**Subdimensions**

A subdimension, as discussed in [Chapter 5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch05.html), is simply a dimension attached to another dimension, when the design is *permissibly snowflaked*. A subdimension may play the role of a primary dimension in some situations. The calendar date dimension is a good example of an entity that is frequently a primary dimension as well as a subdimension.

Subdimensions are usually the first to be loaded in the data warehouse because the chain of dependency starts with the outermost tables, namely the subdimensions. Facts depend on dimensions, and dimensions depend on subdimensions. Therefore, subdimensions must be loaded, and their keys defined, before any other table downstream in the structure can be populated. The caveat is that depending on business requirements and your particular environment, it's possible that some subdimensions are rarely used and not considered mission critical. That means if a failure occurs to prevent the subdimension from loading successfully, it may be acceptable to continue with the load process of its associated dimension anyway.

**Dimensions**

Once the subdimensions are loaded, you can load the dimensions. Dimensions that have subdimensions need to lookup the surrogate key in the subdimension so it can be inserted into the dimension during the load process. Naturally, dimensions that do not have subdimensions can be loaded at once, without waiting for anything else to complete.

NOTE

Smaller dimensions without dependencies should be loaded concurrently and utilize parallel processing. Larger dimensions can also be loaded in this fashion, but test their performance for optimal results before you commit to this strategy. Sometimes, it is faster to load large dimensions individually to alleviate contention for resources. Unfortunately, trial and error is the best rule for action in these cases.

Dimension loads must complete successfully before the process continues. If a dimension load fails, the scheduler must halt the load process from that point forward to prevent the rest of the jobs from loading. If the process continues to load without the dimension information populated, the data warehouse will be incomplete and viewed as corrupt and unreliable. Enforcing the dependencies between jobs is crucial for the data warehouse to maintain a respectable reputation.

**Bridge Tables**

A bridge table sits between a dimension and a fact table when a single fact record can be associated to many dimension records. Bridge tables are also used between a dimension and a multivalued subdimension. For example, a bridge table is needed when a fact is at the grain of a patient treatment event in a medical billing database and many patient diagnoses are valid at the moment of the treatment. After the patient diagnosis dimension is loaded, the treatment transaction table is scanned to determine which diagnoses occur together. Then the bridge table is loaded with a surrogate key to assemble the diagnoses ordered together into groups.

Not all data marts contain bridge tables, but when they do, the tables must be loaded immediately after the dimensions but before the fact table load starts. If a fact table depends on a bridge table, the bridge table load must complete successfully before the fact table load can be executed. If you attempt to load the fact table with the bridge table partially loaded, groups will be missing from the table, and data from the fact table will become suppressed when it is joined to the bridge table.

NOTE

CROSS-REFERENCE Information on loading bridge tables can be found in [Chapter 5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch05.html); techniques for using a bridge table to find the groups while loading facts are found in [Chapter 6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html).

**Fact Tables**

Fact tables are dependent on virtually all other tables in the dimensional data model and are usually loaded last. Once the subdimensions, dimensions, and bridge tables are loaded, the fact table has all of the look-ups it needs and is ready to be loaded. Remember, RI is enforced here, so you must ensure that every foreign key in the fact table has an associated primary key in its relative dimension or bridge table.

Fact tables typically take longest of all the different types of tables in the data warehouse to load; you should begin the fact table load process as soon as all of its related tables are loaded. Do not wait for all of the dimensions in the data warehouse to load before kicking off the fact load. Only the dimensions and bridge tables directly related to the fact table need to complete before the associated fact table load can begin.

Because of the extreme volume of data usually stored in fact tables, it's a good idea to process their loads in parallel. The scheduler should spawn the ETL process into multiple threads that can run concurrently and take advantage of parallel processing. The next chapter discusses more about optimizing your fact table loads.

**Hierarchy Mapping Tables**

Hierarchy mapping tables are specially designed to traverse a hierarchy that lives within a dimension. See [Chapter 5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch05.html). Hierarchy mapping tables are not dependent on facts or bridge tables (unless, of course, the fact table itself contains the hierarchy). Technically, hierarchy tables can be loaded immediately following their relative dimension load, but we recommend loading them at the end of the data-mart process to enable the long-running fact table loads to begin, and finish, sooner.

Regardless of where the hierarchy is physically placed in a batch, its success or failure should have no bearing on the other processes in the batch. Don't kill the launch of a fact table process because of a failure in a hierarchy mapping table. The mapping table can be restarted independently of any fact table load.

The Effect of Aggregates and Group Bys on Performance

Aggregate functions and the Group By clause require databases to utilize a tremendous amount of *temp space*. Temp space is a special area managed by the DBMS to store working tables required to resolve certain queries that involve sorting. Most DBMSs attempt to perform all sorting in memory and then continue the process by writing the data to the temp space after the allocated memory is full. If you attempt to build aggregates for the data warehouse with SQL, you have a few issues to address.

SQL is processed on the server where it is executed. That means that if you attempt to aggregate data in your extract query, you will likely blowout the allocated temp space in the source transaction system. By design, transaction systems keep their temp space very small compared to the space allocated on data warehouses. When you need to build aggregate tables, it's good practice to utilize the ETL engine or a third-party tool specifically dedicated to sorting data at lightning-fast speeds.

You should adjust your aggregates incrementally with a dedicated tool that supports incremental updates to aggregates.

NOTE

Do not attempt to execute aggregating SQL with a **Group By** clause in your data extraction query. The **Group By** clause creates an implicit sort on all of the columns in the clause. Transaction systems are typically not configured to handle large sort routines, and that type of query can crash the source database. Extract the necessary atomic-level data and aggregate later in the ETL pipeline utilizing the ETL engine or a dedicated sort program.

Performance Impact of Using Scalar Functions

Scalar functions return a single value as output for a single input value. Scalar functions usually have one or more parameters. As a rule, functions add overhead to query performance, especially those that must evaluate values character by character. The following functions are known performance inhibitors:

* SUBSTR()
* CONCAT()
* TRIM()
* ASCII()
* TO\_CHAR()

This list is not exhaustive. It is served as an example to get you thinking about the different types of functions available in your database. For example, TO\_CHAR() is a data-type conversion function. If TO\_CHAR() inhibits performance, you can imagine that TO\_DATE() and TO\_NUMBER() also do. Try to substitute database functions with operators. For example, in Oracle, the CONCAT() function can be replaced with the double pipe || to concatenate two strings.

NOTE

Databases are getting better at handling functions. Oracle has introduced function-based indexes that speed up response time for function-based constraints on queries. Look for more advanced functionality from the database vendors as they integrate the ETL with their base products.

Avoiding Triggers

Database triggers are stored procedures executed by the occurrence of an event in the database. Events such as deleting, inserting, or updating data are common events related to database triggers. The problem is that each event is the occurrence of a record trying to get into the database, and the database must fire off the stored procedure between each record. Triggers are notorious for slowing down transactions as well.

If you should need event-based execution of a process, use the ETL engine to accomplish the task, especially for performing such tasks as appending audit metadata to records or enforcing business rules. ETL engines can perform such tasks in memory without requiring I/O.

Overcoming ODBC the Bottleneck

[Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html) offers insight into the layers within the Open Database Connectivity (ODBC) manager, but it's worth mentioning again here that ODBC is usually an unnecessary layer in your communication between the ETL engine and the database that can—and should—be avoided. ODBC adds layers of code to each SQL statement. It is equivalent to using a translator while teaching a class. The message eventually gets across but is a much slower process. And at times, things do get lost in translation.

Try to obtain native drivers to communicate between the ETL engine and the databases in that participate in process. Remember, just as a chain is only as strong as its weakest link, the ETL is only as fast as its slowest component. If you include ODBC in your ETL solution, you will not achieve optimal performance.

Benefiting from Parallel Processing

Processing the ETL in parallel is probably the most powerful way to increase performance. Each time you add another process, the throughput proportionally increases. This section does not discuss the technical architecture options (SMP, MPP, NUMA, and so on). Instead, we offer the benefits of processing the ETL in parallel versus sequential processing.

*Parallel processing*, in its simplest definition, means that more than one operation is processed at a time. As you can imagine, three major operations exist in any given ETL process—extract, transform, and load. You can, and should, take advantage of parallel processing in as many of them as possible.

Parallelizing Extraction Queries

The effective way to parallelize extraction queries is to logically partition the data set into subsets of equal size. We say *logically partition* because partitioning data is usually a physical database function. In this case, you divide the data based on ranges of an attribute. For example, you can divide the effective date by year. Therefore, if you have ten years of data, you have ten logical partitions. Each partition is retrieved by a separate SQL statement and executed concurrently. The potential problem with this approach is that the database identifies each SQL statement as a separate process and attempts to maximize the memory allocated to each. Therefore, if you have very memory-intensive extraction queries, you can bring the server to its knees by replicating and executing such intensive processes.

Fortunately, most DBMSs have the capability to process a query in parallel, realizing it is the same process and managing memory accordingly. Optimal parallel solutions usually combine the two techniques—spawn several extract queries, each with a different range of values, and then parallelize each of those processes with database-specific parallel query techniques.

Each database—those that support it—has its own syntax for executing queries in parallel. In Oracle, you enable parallelization by setting the *degree* parameter when you create a table, or you can alter the table after it's created to enable parallelized queries. Run the following query to check to see what the parallel parameters for a table are:

Select table\_name, degree, instances from all\_tables where

table\_name = '<TABLE\_NAME>'

The preceding query returns three columns:

* **Table Name**. The name of the table being checked for parallelism
* **Degree**. The number of concurrent threads that would be used on each instance to resolve a query
* **Instances**. The number of database instances that the query can span to resolve a query

NOTE

You do not need Oracle Parallel Server to run parallel processes. As long as you have the parallel degree set greater than 1, the query runs in as many processes as are indicated. However, to span instances, you must have multiple instances active and have Oracle Parallel Server running.

Unfortunately, most transaction tables have the parallel degree set to 1 by default. And as you have probably found out, the source system DBA is not about to alter tables for the data warehouse team. Luckily, you don't need them to. Since the extraction query is a static, reusable SQL statement, it is permissible to insert a *hint* to override the physical parallel degree to tell the DBMS to parallelize the query on the fly! Dynamic parallelization is a robust mechanism invaluable for speeding up extract queries.

To dynamically parallelize a query, insert a hint that specifies the number of threads that you want to run concurrently and the number of instances you want to span.

select /\*+ full(products) parallel(products,4,1) \*/

product\_number, product\_name, sku, unit\_price from products

where product\_status = 'Active'

The hint in the query is marked by a proceeding /\*+ and is terminated with \*/. Notice that the hint made the query execute on four different threads on a single instance dynamically. By quadrupling the execution threads, you can usually come awfully close to quadrupling the total throughput for the process. Obviously, other variables, such as memory and the physical attributes on the source system and tables, which the ETL team has no control over, also affect performance. So, don't expect performance increases to be 100-percent proportional to the number of parallel degrees specified. Refer to your DBMS user's manual for the calculation to determine the optimal parallel degree setting for your specific situation.

Parallelizing Transformations

If you are using SQL for your transformation logic, you can use the hint offered in the last section for any SQL DML statement. However, if you are using a dedicated ETL tool, and by now you probably are, you have two options to parallelize your transformations:

1. Purchase a tool that can natively parallelize an operation.
2. Manually replicate a process, partition the input data, and execute the processes in parallel.

Obviously, you want to strive for the first option. However, some tools do not natively support parallelism within jobs. If you have very large data sets, parallelism is not a nice option but a requirement. Luckily, the ETL vendors realize that data volumes are growing at a rapid pace, and they are quickly adding parallelization functionality to their tool sets.

If you have a tool (or an add-on to a tool) that enables transformations to be processed in parallel, simply follow the guidelines set by the vendor to achieve optimal results.

On the other hand, if you need to replicate processes manually, you should take the following steps:

1. Analyze the source system to determine the best way to partition data. If the source table is partitioned, use the column that the partition is based on. If it is not partitioned, examine the date fields, that is, effective date, add date, and so on. Usually, partitioning by date makes a nice, even distribution of volume across partitions. Often, in cases such as Orders, the volume can increase across partitions over time (a sign that business is good). In those cases, consider range partitioning the primary key or creating a hash partition, perhaps doing MODs on the primary key, which is a simple way to split data evenly.
2. The next step is to replicate the ETL process as many times as you want parallel threads to run concurrently. Look for a tool that minimizes the amount of redundant code. Remember, if you have four copies of an ETL process, all four copies need to be maintained. It's better to utilize a tool that can execute the same job with different batches that feed the job different data sets.
3. Finally, set up several batch jobs, one for each process, to collect and feed the appropriate data sets based on the ranges of values determined in step one. If you have an extremely volatile source system, we recommend that you run a preprocess that scans the source data and determines the best ranges to evenly distribute the data sets across the replicated ETL jobs. Those ranges (start value and end value) should be passed to the ETL jobs as parameters to make the process a completed automated solution.

NOTE

If you have a substantial amount of data being fed into your data warehouse, processing all of your ETL operations sequentially will not suffice. Insist on an ETL tool that can natively process multiple operations in parallel to achieve optimal throughput (where parallelization is built directly into the transformation engine, not implemented as *parallel extenders*).

Parallelizing the Final Load

In the earlier section discussing parallelizing extraction queries, we assume that you do not have control over the structures in the database and that you need to add a database hint to have your query spawn multiple threads that run concurrently. However, in the target, the presentation area of the data warehouse, you do—or at least should—have some say in how the structures are built. It's in the best interest of the data warehouse team to architect the tables to have multiple degrees of parallelization when they are created.

Earlier in this chapter, we recommend that you minimize SQL inserts, updates, and deletes and utilize the bulk-load utility. Furthermore, when using Oracle's SQL Loader, you should make sure to set the DIRECT parameter to TRUE to prevent unnecessary logging.

Now we want to introduce one more technique to extend the extract and transform parallel processing: Spawn multiple processes of SQL Loader—one for each partition—and run them in parallel. When you run many SQL Loader processes concurrently, you must set the PARALLEL parameter to TRUE. No faster way exists—at least at the time of this writing—to load a data warehouse than following these three rules:

1. Utilize the bulk loader.
2. Disable logging.
3. Load in parallel.

More information about using bulk loaders can be found in [Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html). For an exhaustive reference for the Oracle SQL Loader utility, read *Oracle* *SQL\*Loader: The Definitive Guide* by Jonathan Gennick and Sanjay Mishra (O'Reilly & Associates 2001).

Troubleshooting Performance Problems

No matter how efficient you make your ETL system, you still stand a chance of having performance issues. However, as Robin Williams says so eloquently in the film *Good Will Hunting*, "It's not your fault." When you are dealing with very large data sets, sometimes they decide to make their own rules. On more than one occasion, we've come across a situation where everything is configured correctly, but for some unexplainable reason, it just doesn't work!

When a job catches you by surprise and performs with lackluster results, don't fight it. Simply take a pragmatic approach to find the operation within the process causing the bottleneck and address that specific operation. Monitor areas such as CPU, memory, I/O, and network traffic to determine any high-level bottleneck.

If no substantial bottlenecks are detected outside of the actual ETL process, you need to dive inside the code. Use the process of elimination to narrow down potential bottlenecks. To eliminate operations, you must have the ability to isolate each operation and test it separately. Code isolation tends to be quite difficult if you are hand-coding the entire process in SQL or another procedural language. Virtually all of the ETL tools provide a mechanism to isolate components of a process to determine undesired bottlenecking.

The best strategy is to start with the extraction process; then work your way through each calculation, look-up, aggregation, reformatting, filtering, or any other component of the transformation process; and then finally test the I/O of the actual data load into the data warehouse.

To begin the isolation process for detecting bottlenecks, copy the ETL job and modify the copy of the job to include or exclude appropriate components as needed. As you step through the process, you will likely need to delete the copy and recopy the job to restore changes made to test preceding components. Follow these steps to isolate components of the ETL process to identify bottlenecks:

1. **Isolate and execute the extract query**. Usually, the extraction query is the first operation in the process and passes the data directly into the next transformation in the pipeline. To isolate the query, temporarily eliminate all transformations and any interaction with databases downstream from the extract query and write the result of the query directly to a flat file. Hopefully, the ETL tool can provide the duration of the query. If not, use an external monitoring tool, or, in Oracle, use the SET TIMING ON command before you execute the process. That setting automatically displays the elapsed time of the query once it completes. If the extract query does not return the rows substantially faster than when the whole process is enabled, you've found your bottleneck, and you need to tune your SQL; otherwise, move on to Step 2.

NOTE

In our experience, badly tuned SQL is by FAR the most common reason for slowness.

1. **Disable filters**. Believe it or not, sometimes feeding data in an ETL job and then filtering the data within the job can cause a bottleneck. To test this hypothesis, temporarily disable or remove any ETL filters downstream from the extract query. When you run the process, watch the *throughput*. Keep in mind that the process might take longer, but its how much data is processed during that time that's important. If the throughput is substantially faster without the filter, consider applying a constraint in the extract query to filter unwanted data.
2. **Eliminate look-ups**. Depending on your product, reference data is cached into memory before it is used by the ETL process. If you retrieve a lot of data in your look-ups, the caching process can take an inordinate amount of time to feed all of the data into memory (or to disk). Disable each look-up, one at a time, and run the process. If you notice an improvement in throughput with one or more look-ups disabled, you have to minimize the rows and columns being retrieved into cache. Note that even if you are not caching your look-up, you may still need to minimize the amount of data that the look-up query returns. Keep in mind that you need only the column being referenced and the column being selected in your look-ups (in most cases, the natural key and surrogate key of a dimension). Any other data is usually just unnecessary I/O and should be eliminated.
3. **Watch out for sorters and aggregators**. Sorters and aggregators tend to hog resources. Sorters are especially bad because they need the whole dataset in memory to do their job. Disable or remove any resource-intensive transformations such as sorters and aggregators and run the process. If you notice a substantial improvement without the components, move those operations to the operating system. Quite often, it's much faster to sort or presort for aggregates outside of the database and ETL tool.
4. **Isolate and analyze each calculation or transformation**. Sometimes the most innocent transformations can be the culprit that causes ETL performance woes. Remove each remaining transformation, one at a time, and run the process. Look for things such as implicit defaults or data-type conversions. These seemingly harmless operations can have substantial impact on the ETL process. Address each operation independently for the best bottlenecking detection and remedy.
5. **Eliminate any update strategies**. As a general rule, the update strategies that come packaged in ETL tools are notoriously slow and are not recommended for high-volume data loads. The tools are getting better, so test this process before removing it. If the update strategy is causing a bottleneck, you must segregate the inserts, updates, and deletes and run them in dedicated streams.
6. **Test database I/O**. If your extraction query and the rest of the transformations in your ETL pipeline are running efficiently, it's time to test the target database. This is a simple test. Redirect the target to load to a flat file instead of a database. If you see a noticeable improvement, you must better prepare your database for the load. Remember to disable all constraints, drop all indexes, and utilize the bulk loader. If you still cannot achieve desired performance, introduce a parallel process strategy for the data-load portion of the ETL.

Increasing ETL Throughput

This section is a summary of sorts. It can be used as a quick reference and a guideline for building new processes. The ETL development team is expected to create ETL jobs that obtain the maximum possible throughput. We recommend the following ten rules, which are applicable for hand-coded solutions as well as for various ETL tools for boosting throughput to its highest level:

1. **Reduce I/O**. Minimize the use of staging tables. Pipeline the ETL to keep the data in memory from the time it is extracted to the time it is loaded.
2. **Eliminate database reads/writes**. When staging tables are necessary, use flat files instead of database tables when you must touch the data down to disk.
3. **Filter as soon as possible**. Reduce the number of rows processed as far upstream in the process as you can. Avoid transforming data that never makes its way to the target data warehouse table.
4. **Partition and parallelize**. The best way to increase throughput is to have multiple processes process the data in parallel.
   * Parallelize the source system query with parallel DML.
   * Pipeline and parallelize transformations and staging.
   * Partition and load target tables in parallel.
5. **Update aggregates incrementally**. Rebuilding aggregates from scratch is a process-intensive effort that must be avoided. You should process deltas only and add those records to existing aggregates.
6. **Take only what you need (columns and rows)**. Similar to the filtering recommendation, do not retrieve rows unessential to the process. Likewise, do not select unessential columns.
7. **Bulk load/eliminate logging**.
   * Utilize database bulk-load utility.
   * Minimize updates; delete and insert instead.
   * Turn off logging.
   * Set DIRECT=TRUE.
8. **Drop database constraints and indexes**. Foreign key (FK) constraints are unnecessary overhead; they should be dropped—permanently (unless they are required by your aggregate navigator). If FKs are required, disable them before the ETL process and enable them as a post-process. Leave indexes for updates and deletes to support WHERE clauses only. Drop all remaining indexes for inserts. Rebuild all indexes as a post-process.
9. **Eliminate network traffic**. Keep working files on local disk drives. Also, place the ETL engine on the data warehouse server.
10. **Let the ETL system do the work**. Minimize any dependency on DBMS functionality. Avoid stored procedures, functions, database key generators, and triggers; determine duplicates.

Many of the top ten rules have already been discussed in previous chapters in this book. For a more comprehensive examination, you'll find that [Chapters 5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch05.html), [6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html), and [7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html) offer especially great details on optimal design strategies.

Each of the top ten rules for boosting ETL productivity is discussed briefly in the following sections.

TIP

Rebuilding indexes can take a lot of time. It's recommended that you partition high-volume target tables. Not only can you truncate and reload the data in a partition, while leaving the rest of the table intact, but indexes local to a partition can be dropped and rebuilt, regardless of how data is maintained. Rebuilding the subset of the index can save a substantial amount of time during the post-load process.

Reducing Input/Output Contention

Because databases and operating systems each interact with input and output so differently, we don't attempt to explain the technical operations of I/O in this section. However, we do maintain the stance that I/O must be reduced to an absolute minimum. Obviously, you may need to touch down data for various reasons. The number-one permissible reason to touch down data is when you need to minimize access to the source system or if your source system allows only one-shot to retrieve the data you need from it. In those cases, it's good practice to write the extraction result set to disk as soon as it is retrieved. That way, in case of failure, you can always reprocess data from the saved copy instead of penetrating the source system again.

Excessive I/O is a remarkably common offender. In most cases, intermediary tables or files can be omitted without any loss of functionality while their respective processes benefit from increased throughput. If you find yourself creating staging tables and many jobs to read and write to them, stop! Step back and analyze the total solution. By eliminating staging tables, you not only reduce I/O—the biggest performance hit—but also you reduce the number of jobs that need to be maintained and simplify the batch and scheduling strategy.

Eliminating Database Reads/Writes

The ETL process often requires data to be touched down to disk for various reasons. It can be to sort, aggregate, or hold intermediate calculations or just retain for safekeeping. The ETL developer has a choice of using a database for these purposes of flat files. Databases require much more overhead than simply dumping data into a flat file. And ETL tools can manipulate data from a flat file just as easily as database data. Therefore, it's a preferred practice to utilize sequential or flat files in the data-staging area whenever possible.

The irony of this recommendation is that the ultimate goal of the data warehouse is to present data in a way that it has optimal query response time and that the solution is a relational database management system. However, even though the ETL may need to read intermediary data, it does not *query* data in the same sense end users do in the data warehouse's presentation layer. ETL staging processes are static and repeated, whereas the data warehouse must support unpredictable, ad-hoc queries.

Dramatic performance improvements can be obtained by simply redirecting the staging database tables to flat files. The downside to eliminating the database in the staging area is that the associated metadata that comes for *free* by the nature of the database is lost. By choosing to use flat files, you must maintain any metadata related to the files manually (unless your ETL tool can capture the metadata).

Filtering as Soon as Possible

This tip addresses what is possibly the most common mistake in ETL design. Whenever we conduct design reviews of existing ETL processes, one of the first things we look for is the placement of filters. A filter is a component that exists in most ETL products that applies constraints to the data after it's been retrieved. Filters are extremely useful because in many cases you need to constrain on fields from the source system that are not indexed. If you were to apply the constraint in the extraction SQL on a nonindexed field, the source database would need to perform a full table scan, a horrendously slow process. Conversely, if the source system indexes the field you want to constrain, this would be the preferred place for filtering because you eliminate extracting unwanted records.

We often notice that filters are placed downstream of very complex calculations or process-intensive and I/O-intensive data look-ups. Granted, at times you must perform certain calculations before filters are applied. For example, when you have to figure the dwell time of a Web page, you must calculate the difference between the current page hit and the next before you dispose of the unwanted pages.

As a general rule, you should keep and apply ETL filters as far upstream in the process as requirements permit. Typically, filtering advantages are best achieved when they are placed immediately following the initial SQL statement that extracts data from the source system and before any calculations or look-ups occur. Precious processing is wasted if you transform data and then throw it away.

NOTE

Apply ETL filters to reduce the number of rows to process instead of applying constraints to the extraction SQL only if the source system database does not have the appropriate indexes to support your constraints. Because other factors such as table size, SQL complexity, network configuration, and so on play a role in data-retrieval performance, it makes sense to test both strategies before deciding on the optimal solution.

Partitioning and Parallelizing

Partitioning and parallelizing your ETL process is more than a design issue; it requires specific hardware and software and software solutions as well. One can partition data without executing its ETL in parallel and visa versa. But if you attempt to parallelize without partitioning, you can incur bottlenecking. An effective partition and parallelization strategy for unpredictable source data is to create hash partitions on your target tables and apply that same partition logic to the ETL process. Be careful though—hash partitions may not be an optimal solution for the data warehouse ad-hoc queries. Work closely with the data warehouse architect to implement the most appropriate partition strategy. Refer to earlier in this chapter for techniques and advantages concerning parallel processing.

Updating Aggregates Incrementally

Aggregates are summary tables that exist in the data warehouse specifically designed to reduce query time. Aggregates make a dramatic performance gain in the data warehouse because queries that had to scan through hundreds of millions of rows now can achieve the same results by scanning a few hundred rows. This drastic reduction in rows is attributable to the ETL process combining additive facts in a mechanical rollup process. More complex summaries that depend on complex business rules are not what we call aggregates in the dimensional world. Remember that an aggregate is used in conjunction with a query rewrite capability that applies a fairly simple rule to judge whether the aggregate can be used rather than a dynamic aggregation of atomic data at query time.

Aggregates are computed in several different ways in a mature data warehouse environment:

* **Calculating aggregate records that depend only on the most recent data load**. Product rollups and geographic rollups (for instance) generated entirely from the most recent data load should be calculated by sorting and summarizing the data outside the DBMS. In other words, don't use the DBMS's sort routines when native OS sorts are much faster. Remember that the computation of aggregates is merely a process of sorting and summarizing (creating *break rows*).
* **Modifying an existing aggregate in place by adding or subtracting data**. This option is called *tweaking* the aggregate. An existing aggregate spanning an extended period of time may be modified when that period of time includes the current load. Or an existing aggregate may be modified when the criteria for the aggregate are changed. This can happen, for example, if the definition of a product category is modified and an aggregate exists at the category level, or a rollup above the category level. If the tweak to the category is sufficiently complicated, a quality-assurance check needs to be run, explicitly checking the aggregate against the underlying atomic data.
* **Calculating an aggregate entirely from atomic data**. This option, called a *full rollup*, is used when a new aggregate has been defined or when the first two options are too complex to administer.

Taking Only What You Need

It doesn't make much sense to retrieve hundreds of thousands or millions (or even billions) of rows if only a few hundred of the records are new or have been modified since the last incremental ETL process. You must select a mechanism for retrieving deltas from the source system only. There are several ways to approach change-data capture depending on what's available in the source transaction system. Refer to [Chapter 6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html) for a display of the many different techniques for capturing changed data in the source system and techniques for determining the most appropriate for your particular situation.

Once you have the rows trimmed down to a manageable size for your incremental loads, you must next ensure that you don't return more columns than necessary. Returning excessive columns is commonly encountered in look-ups in ETL tools. Some ETL tools automatically select all of the columns in a table whether they are needed or not when it is used for a look-up. Pay special attention to explicitly *unselect*columns that are not vital to the process. When you are looking up surrogate keys, you typically need only the natural and surrogate keys from a dimension. Any other column in a dimension is superfluous during a surrogate key look-up process.

Bulk Loading/Eliminating Logging

Bulk loading is the alternative to inserting data into the data warehouse one row at a time, as if it were a transaction system. The biggest advantage of utilizing a bulk loader is that you can disable the database logging and load in parallel. Writing to the rollback log consumes overhead as well as I/O and is unnecessary in the data warehouse. Specific bulk-load techniques and advantages to bulk loading are offered throughout this book.

Dropping Databases Constraints and Indexes

Another certain way to have a positive impact on loading your data warehouse is to drop all of the constraints and indexes from the target of the ETL process. Remember, the data warehouse is not transactional. All data is entered via a controlled, managed mechanism—ETL. All RI should be enforced by the ETL process, making RI at the database level redundant and unnecessary. After a table is loaded, the ETL must run a post-process to rebuild any dropped indexes.

Eliminating Network Traffic

Whenever you have to move data across *wires*, the process is vulnerable to bottlenecking and performance degradation.

Depending on your infrastructure, it sometimes makes sense to run the ETL engine on the data warehouse server to eliminate network traffic. Furthermore, benefits can also be achieved by storing all of the staging data on internal disk drives, rather than by having the data travel over the network just to touch down during the ETL process.

This recommendation can be a Catch-22, meaning that in your situation, putting the ETL engine on your data warehouse database server might actually make your performance worse, not better. Work with your ETL, database, and hardware vendor to achieve the best solution for your specific requirements.

Letting the ETL Engine Do the Work

ETL products are specifically designed to extract, transform, and load massive amounts like no other nondata warehousing solution. With minimal exception, most databases are designed to support transactional and operational applications. Database-procedural programming is good for supporting data-entry applications but is not optimal for processing large data sets at once. The use of cursors—where each record is analyzed individually before moving on to the next—is notoriously slow and usually results in unacceptable performance while processing very large data sets. Instead of using procedures stored within the database, it's beneficial to utilize the ETL engine for manipulating and managing the data.

Summary

This chapter has provided an overview and some examples of the technologies you need to choose to develop your ETL system. You must start by choosing a development environment: either a dedicated ETL tool suite from one of the vendors we have listed or a development environment based on operating system commands driven by scripting languages, with occasional escapes into a low-level programming language.

In the second half of this chapter, we have given you some guidance on DBMS-specific techniques for performing high-speed bulk loads, enforcingRI, taking advantage of parallelization, calculating dimensional aggregates, and troubleshooting performance problems.

Now we're ready to *get operational* in [Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html) and manage this wonderful technology suite we have built.

Chapter 8. Operations

"Overall system speed is governed by the slowest component."

**—Gene Amdahl**

Developing ETL processes that load the data warehouse is just part of the ETL development lifecycle. The remainder of the lifecycle is dedicated to precisely executing those processes. The timing, order, and circumstances of the jobs are crucial while loading the data warehouse, whether your jobs are executed real-time or in batch. Moreover, as new jobs are built, their execution must integrate seamlessly with existing ETL processes. This chapter assumes that your ETL jobs are already built and concentrates on the operations strategy of the ETL.

In this chapter, we discuss how to build an ETL operations strategy that supports the data warehouse to make its data reliably on time. In the first half of this chapter, we discuss ETL schedulers as well as tips and techniques for supporting ETL operations once the system has been designed.

The second half of this chapter discusses the many ways in which you can measure and control ETL system performance at the job or system level. (We discuss database software performance in [Chapter 7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html).) You have more than a dozen *knobs* for controlling performance, and we give you a balanced perspective on which are most important in your environment.

At the end of this chapter, we recommend a simple but effective approach to ETL system security at the database, development environment, QA-environment, production-environment, and basic file-system levels.

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Test/Release

Data Flow: Extract → Clean → Conform → Deliver

This chapter describes best practices for running your ETL operations. Operations includes initial data loads, execution and monitoring the daily flow of data, capacity planning, performance monitoring, maintenance of the metadata repository, and controlling access to the back room databases.

Scheduling and Support

The ETL execution strategy falls into two major categories:

* **Scheduling**. ETL scheduling is a comprehensive application that does much more than arrange for jobs to execute at a given time. In reality, the time of day that a job executes is almost insignificant. Instead, an effective scheduler involves the designation of relationships and dependencies between ETL jobs and acts as a reliable mechanism to manage the physical implementation of the execution strategy.
* **Support**. Once the data warehouse is deployed, it invariably becomes a mission-critical application. Users, as well as other downstream applications, depend on the data warehouse to provide them with the information they need to function properly. If the data warehouse is not loaded consistently, it is deemed a failure. To make certain that the ETL process runs and completes, it must be actively monitored and supported by a production-support team.

Reliability, Availability, Manageability Analysis for ETL

A data warehouse can have the best dimensional data model, a best-of-breed business-intelligence tool, and sponsorship from the highest executives. But it is not a proven solution until it is considered a dependable source for corporate analytical information.

The goal of a new data warehouse is to build a reputation for being a consistent, reliable data source to support corporate data analysis to empower the business. To be a success, the ETL and the data warehouse teams must fulfill three key criteria:

* **Reliability**. The ETL process must run consistently, without fail. The data within must be trustworthy at any level of granularity.
* **Availability**. The data warehouse must be up, running, and available for use as promised by the data warehouse manager during initial kick-off meetings with the sponsors and users. ETL jobs must execute and complete within the allocated load window.
* **Manageability**. Remember that the data warehouse is never finished. It must have the capability to change and expand as your company grows. The ETL processes must evolve gracefully with the data warehouse. To achieve extensibility, keep processes as simple as possible; break down complex routines into smaller, simpler components. At the same time, avoid an upsurge of jobs to carry out processes. Moreover, a crucial part of designing the execution strategy is ensuring the ability to support the ETL. The ETL team must provide metadata for all components of the ETL and document recovery procedures for every failure scenario. If you are hand-coding your system, make sure you have the management skills and perspectives to control a long-term software development environment.

The ETL manager must appraise each phase of the data warehouse by using the Reliability, Availability, and Manageability (RAM) criteria to score the project. The jobs and scheduling approach must pass each of the three criteria to get a perfect score and earn the right to deploy. If no metadata or recovery documentation exists, points are deducted and the processes must be revisited and enhanced or corrected. Jobs that are overly complex making them virtually impossible to maintain must be streamlined to progress to the next stage of the lifecycle. Each deployment of the data warehouse must achieve a perfect RAM score before it is rolled into production.

Etl Scheduling 101

Scheduling ETL processes is an obvious necessity to get them to run, so why write nearly a whole chapter about it? This chapter explains not just execution but execution *strategy*. A strategy is an elaborate and systematic plan of action. Anyone can execute a program, but developing an execution strategy requires skill.

For example, during a data warehouse and ETL design review, a user was complaining that the data warehouse was not available until 11:00 a.m. With this information, we immediately started to review ETL jobs to find where the bottleneck was so we could recommend a remedy. We shortly discovered the jobs were efficient and should not have taken three full hours to process from execution to completion. "That's correct!" claimed an ETL developer on the project. "I kick them off as soon as I arrive at work, around 8:00 a.m., and they complete in three hours—by 11 o'clock." In disbelief, we interrogated the developer about automation—and the lack of it in his implementation. He claimed he was never trained in the ETL tool's scheduler, so he had to kick the jobs off manually.

Even though you execute your programs, it is imperative that you do so systematically. It is crucial that the ETL team understand the tools in your environment and have the ability to properly schedule and automate the ETL process to consistently load the data warehouse.

Scheduling Tools

Any enterprise data warehouse must have a robust enterprise ETL scheduler. Major ETL vendors package schedulers with their core ETL engine offerings. Some offer little more than a way to execute your ETL jobs depending on the time of the day, while others offer comprehensive ETL execution solutions that can trigger ETL jobs based on a variety of vital criteria.

If you are not satisfied with the packaged scheduler bundled with your ETL product or you opted to venture the ETL without a dedicated product, you have a few alternatives. Regardless of whether you buy a dedicated ETL scheduler, use your existing production-scheduling system, or manually code your ETL jobs to execute, a production ETL scheduler should meet certain criteria to be a viable enterprise solution.

**Required Functionality of an ETL Scheduler**

The next sections examine some of the options available to automate the ETL process. Many options are available, and each varies in cost and ease of use. Certain functionality is required in production ETL environments. When you select (or build) your ETL scheduling solution, make sure it contains the functionality discussed in the following sections.

**Token Aware**

Often, the data warehouse requires data acquired from an external source. External data providers are common, and your ETL solution must be able to accommodate their data. External data sources are usually provided as a flat file or in XML format. Reading and processing this data is by and large quite simple; the challenge is to make the ETL process aware of data's existence. Unlike database sources, where you can look in tables' audit columns to recognize new rows, external sources typically dump data files into a directory on the file system via FTP. As long as the format is correct each time, the ETL process can handle the data. But how does the ETL system know when an externally sourced data file has arrived and should begin its process? The ETL system must be able to recognize that the file has appeared in the file system and execute automatically. This process is called *token aware*.

Tokens are files created in the file system to trigger an ETL event. Applications that are token aware can poll a directory (or database table) for the arrival of a token file (or a row). When you handle flat files, Web logs, or external sourced data, you must avoid processing the same file repeatedly and also ensure that you don't miss running the ETL process if the file arrives late. The token file is considered a token because it is not necessarily the actual file processed; it can be an indicator file that tells a process to execute merely by its arrival.

**Intra-Day Execution**

Daily processing is becoming less acceptable in today's society, where expectations for immediate action are set so high. ETL processes must have the ability to run multiple times throughout the day and even on demand. Where monthly or daily incremental loads used to suffice, there are now calls for 12-hour, six-hour and four-hour increments; even hourly updates are becoming more common where real-time technology does not exist. These aggressive requirements mean that not only must your ETL jobs be efficient, but your scheduling system must be steadfast to manage the exorbitant number of processes that run throughout the day.

Moreover, your process must be able to span over the stroke of midnight—and restart outside of its allocated window. The practice of hard-coding *SYSDATE-1* to look for *yesterday's* data is not adequate for launching and selecting data from your source systems. The ETL system must be able to capture new data from source systems, regardless of when it was created or when the process is executed.

**Real-Time Capability**

Real-time execution is a reality of data warehousing that cannot be ignored. It is so important that we dedicate an entire chapter to the subject. [Chapter 11](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html) discusses several techniques for achieving real-time ETL execution. Real-time ETL is becoming more commonplace in most enterprises. More and more users now expect the data warehouse to be continuously updated and are growing impatient with *stale* data. Soon, real-time ETL will not be a luxury but a standing requirement.

Furthermore, as the data warehouse evolves, its value is realized by the most unexpected users. Because it offers clean, consistent, and reliable data, the data warehouse is becoming a source system itself. Transaction applications are increasingly becoming dependent on the data warehouse to be a standardized source for common reference data elements. To fulfill this so-called *closed-loop*movement, the data warehouse must update continuously to support operational applications.

**Command-Line Execution**

ETL products have dedicated so much energy toward creating slick graphical user interfaces (GUI) for their scheduling packages to reduce the learning curve for beginning developers and to expedite development time for seasoned ETL experts. But most enterprise system operations environments need the ability to execute ETL jobs from a command-line interface. The reality is that the team that supports the daily operations also supports many other applications and cannot be expected to learn a different interface to support each. Therefore, your ETL application must allow processes to be executed from a command-line interface for the data warehouse ETL to be supported by your system-operations staff. Note that the major ETL tool suites all allow command-line execution as an option for these reasons.

**Notification and Paging**

Once the ETL has been developed and deployed, its execution must be a hands-off operation. It should run like clockwork, without any human intervention and without fail. If a problem with the process does occur, the support group must be notified electronically. Your ETL solution must have the ability to notify different groups or people depending on the job or the type of failure. As we write this book, wireless PDAs and smart phones are exploding. These devices seem likely to be standard equipment for operational personnel. The displays on these devices can display complex text and graphical information, and the operator can issue commands to the ETL system remotely. See the warning that follows!

E-mail notification and paging must be completely automated. There is simply not enough time to wait for the key support personnel to be notified manually. Automated notification can be achieved in one of three ways:

* **Integrated ETL tool**. Some of the major ETL products offer paging and notification features natively in their scheduling application. Features are usually not very robust, but they are getting better. At a minimum, you need to differentiate between successful loads and failures and page-appropriate personnel accordingly. Also, messages should automatically send vital information about the failure (for example, job name, time of failure, rows loaded, rows failed, and last error message dynamically).
* **Third-party messaging application**. A number of companies offer urgent messaging products dedicated to supporting 24/7 system operations to minimize downtime. Additionally, operations management/monitoring tools often include notification features that can be utilized if your operations-support team utilizes such a tool.
* **Custom scripts**. You have the option of manually scripting the e-mail notification portion of the execution strategy at the operating-system level. The scripts must interact with the ETL jobs and be triggered as necessary.

NOTE

When designing your custom e-mail notification system, use scripts with embedded e-mail addresses with extreme caution. Scripts can be read on the file system as simply as a text file. Scripts are vulnerable to having e-mail addresses hijacked by spammers who can saturate the e-mail recipients with junk mail. Use encryption techniques or a solution from a secure product whenever possible.

**Nested Batching**

A batch is a group of jobs or programs that run together as a single operation. Usually, ETL jobs are grouped together—or batched—to load a single data mart. And the data warehouse, composed of a collection of data marts, is loaded with a batch of data mart load batches. The technique of loading batches of batches is known as *nested* batching. Nested batching can involve several layers of ETL jobs. For example, a single dimension can require several ETL jobs to load it due to severe complexity within the data or business rules. Those dimension jobs are grouped together to run in a single batch. That batch is included in another batch to load the rest of the dimensions for the data mart. The data mart batch is then incorporated into the data warehouse batch, making the batch three layers deep. No logical limit to the depth of nested batching exists.

ETL jobs are typically executed in nested batches. You will rarely run a single, standalone ETL job in a production environment. A data mart usually requires at least one job for every dimension and the fact table. As you can see, multiple levels of nested batching are common while loading the data warehouse. Therefore, your solution must be able to manage nested batches. Batch management includes the following:

* **Graphical interface**. ETL batches typically become quite complex due to the nature of the nesting required to load the data warehouse. Select a batch-management tool that has the capability to navigate through your nested batches as easily as navigating through a directory structure in Windows Explorer. Without a graphical representation of the nested batches, management can become unwieldy. Developers should be able to create, delete, edit, and schedule batches through a GUI, as well as move jobs and nested batches among outer batches by dragging and dropping them. Batch management is best achieved graphically, although a logical naming standard must accompany the graphics. Visualization of the dependencies between batches is crucial to maintaining a clear understanding of which jobs belong in each batch and also to identifying dependencies between batches.
* **Dependency management**. A dependency occurs when the execution of one job is contingent upon the successful completion of another. Rules of dependencies between jobs are defined in the execution strategy and must be enforced at runtime by the ETL scheduling system. Your batch-management tool must have the ability to stop a batch dead in its tracks upon a failed job if business rules so require. For example, if a dimension job fails, you must not proceed to load the fact table. Not all scenarios require such a strict batch-halt policy. For example, if an outrigger fails, it is usually still okay to load its associated dimension. The batch-management tool should be robust enough to set dependencies on a batch-by-batch basis as business rules dictate.
* **Parameter sharing**. Values of parameters might need to be passed from one job to another or set once at the outermost batch and used globally throughout the nested batches. The batch manager must include parameter-management functionality. More information regarding parameter management is discussed in a section dedicated to that topic later in this chapter.
* **Graceful restart**. What happens if a job fails in the middle of its execution? How do you know exactly what has been loaded and what has not? Upon restart, the batch-management tool must be able to systematically identify which rows have been processed and loaded and process only the rest of the input data. Special attention must be paid to the load process at times of midprocess failure. In general, the ETL system should have a number of staging points (steps in the process where data has been written to the disk) if for no other reason than to support a restart scenario. Also, special care should be taken if one of the ETL steps involves manual intervention and correction of data. These manual steps must at least be preserved in a log so that they can be reapplied if the ETL processing step must be rerun.
* **Sequential/Concurrent execution**. In some cases, it is necessary to load tables sequentially. For instance, when tables have dependencies between them, you must load the parent before you can load child tables. Outriggers associated with specific dimensions are a good example of this parent-child sequencing, as well as normal dimensions and facts. You cannot load a fact until all dimensions are loaded. Also, sometimes you need to load tables in sequence rather than concurrently to distribute server resources. If you attempt to load all dimensions in a data mart at once, you might bring the ETL server to its knees by overloading its resources. Conversely, in cases of long-running processes, you can separate a job into several smaller jobs run concurrently to improve load performance. Assuming appropriate resources are available, run as many independent processes concurrently as possible to maximize processing and minimize the load window. More information on concurrent and parallel processing is detailed later in this chapter.
* **Pre/Post-execution activity**. Simply launching scripts before or after an ETL process is not execution management. The batch manager must be able to realize that a preprocess script has executed successfully before it launches the core ETL job. Moreover, it must trigger only post-process scripts if the core ETL job completes without failure. Lastly, scripts must be able to be executed at the batch level as well as the job level. This is especially important for batches run concurrently, because a different job might complete last each time the batch is executed. Nevertheless, you might need a post-process script to fire off only after *all* jobs are complete.
* **Metadata capture**. All metadata within the control of the batch manager must be captured, stored, and published. In a best-case scenario, metadata should be stored in an open repository that can be shared with other applications. Each ETL job has a scheduled execution time and frequency, its parameters, and recovery procedures, which are all forms of metadata that must be presented and easily obtained by those who need to support the load processes as well as business users. At a minimum, metadata must have reporting abilities so users and developers have insight into the operational aspects of the data warehouse ETL. Refer to [Chapter 9](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch09.html) for an in-depth view of ETL metadata.

NOTE

ETL tools are becoming better at failure recovery, but graceful restart is an extremely difficult requirement that we have not yet seen perfected. In many cases, it is still safest to truncate or delete the information that has been partially loaded as a result of midprocess failures and begin the failed ETL process from the beginning. If you entrust your tool set to automatically pick up where it left off, it is recommended that extra time be spent auditing the data of the completed process to ensure data quality and integrity.

**Parameter Management**

The ETL system moves through different environments throughout its development lifecycle. Since the lifecycle includes testing the code within the ETL system, you cannot alter the code between environments. Therefore, hard-coded parameters are not acceptable while coding variables in the ETL system. Parameters are a way to substitute values in code that would otherwise be constant. A robust ETL scheduling system must have the ability to manage and pass parameters to ETL jobs as they are executed. Parameters add flexibility to ETL jobs so they can gracefully change environments or extraction criteria without reprogramming the application. For example, the natural course of an ETL job is to be developed in a development environment, tested in a test environment, and ultimately migrated to a production environment where it supports the production data warehouse.

Each of the environments in the ETL lifecycle has dedicated source, staging, and target databases; file systems; and directory structures. By making each of these environment changes parameter driven, the ETL system can pass the jobs through the environments without changing code to point to relevant files or databases. You must parameterize environment variables and allow the scheduler to pass the applicable values to those variables at run time.

A good list of items to parameterize includes:

* Server name
* Database or instance name
* Schema description file name
* Database connection information (without the password in plain text!)
* The root directory or folder in which to find useful control files
* Metadata database-connection information

Your scheduler must be able to manage two kinds of parameters:

* **Global parameters**. A global parameter is a single parameter that supports many ETL jobs. Naturally, ETL jobs can have many global parameters. For example, the target database name should be set globally; otherwise, you are forced repeatedly to maintain the parameter for each job that loads the data warehouse.
* **Local parameters**. Local parameters live only within an ETL job. Local parameters can be set to change variables within a single job without affecting other jobs in its batch. An example of a local parameter is the setting of the earliest date that should be retrieved from the source table.

Native ETL tool schedulers are the best bet to obtain a robust parameter-management system because the scheduler is usually natively integrated with the ETL engine. Native integration of the ETL engine and the scheduler makes communication between the two components remarkably efficient. Architectures that involve parameter management by a third-party vendor are not as efficient but might provide more flexibility. ETL solutions that do not support parameters that can be set at runtime fail the Manageability criteria of RAM.

In an enterprise environment, it's important to produce metadata for the parameters in your ETL jobs. If you don't have a robust parameter-management system, parameters can be maintained in flat files on your files system. By utilizing flat files, the operations teams can simply update a parameter file without at all invading the ETL system.

**ETL Scheduler Solution Options**

In the previous section, we describe the functionality that one should expect of an enterprise ETL scheduling system. A few options that achieve the same functionally exist. In this section, we offer five options to select from when you are building your ETL scheduler solution:

1. Integrated ETL tool
2. Third-party scheduler
3. Operating system
4. Real-time execution
5. Custom application

The key is to select a solution robust enough to meet all the criteria you think you'll need based on your knowledge of the jobs that have been created to load your data warehouse thus far, yet fits within the budget of the data warehouse initiative. The final criterion is to consider in-house expertise. The next sections evaluate each of the five options.

**Integrated ETL Tool**

Virtually all of the dedicated ETL tools incorporate a scheduling system to execute ETL jobs created within their toolset. Some tools offer minimal functionality, while others are robust scheduling applications. If you are not forced into using a tool that your operations-support team has already established as the *standard* scheduling tool and your ETL tool contains a robust scheduler, it is most beneficial to use your integrated ETL scheduler. Benefits of an integrated solution include:

* **Product support by your ETL vendor**. Utilize a single Service Level Agreement (SLA) for both applications. Those with IT experience are familiar with the passing of the buck that occurs with multivendor solutions. Funny how problems are never the fault of the vendor on the phone. (*It must be a compatibility issue caused by the other product*.)

Using a single vendor or product suite can improve vendor support and expedite the resolution of technical issues.

* **Integration of scheduler and ETL engine**. Integrated suites are designed to pass parameters between components and natively enforce dependencies between jobs. Dependency between jobs, meaning the execution of one job depends on the successful completion of another job or set of jobs, is a crucial to properly loading the data warehouse and recovering from ETL failures.
* **Knowledge of toolset within ETL group**. Since the ETL toolset is the specialty of the ETL team, they can set up the execution strategy without learning another application. Moreover, once an ETL job has been thoroughly tested, it is rare that it fails in production. When jobs do fail, the ETL team usually needs to get involved at some level of capacity. By keeping ETL scheduling within the domain of the ETL toolset, the team can easily jump into the support role and help recover any failed ETL processes.

**Third-Party Scheduler**

Many production-support departments standardize on a single scheduling system that all applications must adapt to. In some enterprise environments, the data warehouse is treated like any other application and must abide by the rules set by the production-support team. In these cases, the ETL is triggered by a scheduling system that supports all applications throughout the enterprise. Operating an enterprise-scheduling system is a specialty beyond the scope of the ETL team's knowledge. The ETL team needs to work closely with the production-support team in cases where failure recovery is not straightforward.

NOTE

If your production-support team insists that they execute ETL jobs via their standardized enterprise scheduling application, make sure it has the required functionality to properly support your ETL execution strategy, including dependencies between jobs, parameter management, and notification and alerts.

**Operating System**

It's not uncommon for the ETL process to be executed by native operating system scheduling systems such as Unix Crontab or the Windows Scheduler. Even if you have a state-of-the-art ETL product, many production-support groups require scripts to execute the ETL jobs in production because it is a common denominator of all applications throughout the enterprise. Virtually any application can be executed via a line command or script at the operating-system level. In the Windows world, very elaborate batch or .BAT or VBScript or JScript files can be constructed to manage the execution of ETL processes. On Unix, Crontab is used to launch jobs. Operating-system schedulers can execute the ETL job directly or by way of a script.

As most programmers know, the power of scripting is not trivial. One can build very robust application-type logic with scripting languages. Most of the RAM criteria can be met with scripting. Moreover, Perl, VBScript, or JavaScript can be run on Unix or Windows to handle complex business logic while executing jobs that load the data warehouse. In fact, scripting languages can most likely provide the functionality of the logic within the jobs, too. However, we still recommend a robust ETL tool for building and maintaining ETL jobs. The shortfall of using scripting instead of a dedicated ETL scheduling tool is its lack of metadata. Any useful information regarding the ETL schedule lies within the scripts. One needs to be a programmer to decipher the information within the script. Two techniques can be utilized to maintain metadata within the execution scripts.

* **Spreadsheets**. The ETL manager or programmer must maintain a spreadsheet that contains important metadata, including parameters, jobs within the batch, timing of the execution, and so on.
* **Tables**. A dynamic scripting solution is metadata driven. All pertinent metadata is stored in tables (either database or flat) and is passed to scripts at runtime. Metadata-driven scripts are an achievable goal that should be built and utilized when integrated ETL schedulers are not an option.

**Real-Time Execution**

If part of your data warehouse is real-time enabled, you need to select one of the mechanisms detailed in [Chapter 11](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch11.html) to support your real-time requirements. It is rare that an entire enterprise data warehouse is loaded in real time. Often, some segments of the data warehouse are loaded real-time, while others are batched and processed periodically. Special attention must be paid to the integration of the two types of ETL techniques to ensure a seamless, cohesive solution.

**Custom Application**

Building a custom scheduling solution is always an option. However, we have not come across a reasonable justification for a custom scheduling application—but that doesn't stop some from building them anyway. If you choose to execute all of your jobs via scripts, it might be worthwhile to build an application to manage them, but building a custom GUI for this purpose would be overkill. Usually, scripting programs, along with metadata tables, are a viable solution for custom ETL scheduling.

Load Dependencies

Defining dependencies between jobs is perhaps the most important aspect of batching ETL jobs. If a subdimension load job fails, perhaps you can continue to load a dimension, but if a dimension load fails, should you continue to load the fact table? It's usually not recommended. A dependency set between jobs is metadata that the load process must be aware of. Operational metadata of this sort is needed for the operation of the ETL to function properly. A fact table ETL process will load erroneously—missing key data—if it is executed before all of its dimensions are successfully loaded. Moreover, if the fact table is not designed to perform updates, all of the erroneous data must be manually *backed out* or deleted before the process can restart. Manual intervention is the costliest approach to rectifying failed ETL loads. Much of that cost can be avoided by declaring enforceable dependency rules between ETL jobs.

Dependency holds true between bridge tables and dimensions—and hierarchy mapping tables and dimensions. Use the preceding list as a reference for job-dependency definitions. In a nutshell:

* Do not load dimensions without successfully completing their subdimensions.
* Do not load bridge tables without successfully completing their dimensions.
* Do not load fact tables without loading all parents, including bridge tables and dimensions.

However, keep this clever data warehouse aphorism in mind: For each rule of thumb, there are four more fingers to consider. For example, if a dimension is designed to update the foreign key that associates itself to a subdimension, it is not necessary to stop loading the data mart because a subdimension load has failed, as long as the scheduler issues a warning whenever a job does not complete as expected.

Metadata

Imagine if your local railroad ran its service without publishing its train schedule. How would anyone know when to catch the train? Running an execution strategy without publishing its metadata is equally detrimental to its users. Earlier in this chapter, we told you that your scheduler must capture metadata for the contents and schedule of batches and nested batches and that this metadata must be available to business users as well as to the data warehouse team. Batch metadata serves as the train schedule for the data warehouse. It should predict when users should expect data to arrive and become available for use.

The scheduling system should also let users know when data will be arriving late. This notification is different from the failure notification discussed earlier in this chapter. Data-availability metadata is a crucial aspect of communication and a key mechanism for setting user expectations. Meta-data used to notify users of data arrival falls under the category of *process metadata*. Process metadata captures the operational statistics on the ETL process. It typically includes measures such as the count of rows loaded successfully, rows rejected, elapsed time, rows processed per second, and the row's estimated time of completion. It is important process metadata because it helps to set user expectations—just like giving announcements at the train station.

Metadata collected during the cleaning and conforming steps serves several operational roles. It serves to advise the ETL team whether the data is fit to be delivered to the end user community. The data in the audit dimension is meant to be combined with normal data in specially instrumented data-quality reports, both for instilling confidence in the reported results and supporting compliance reporting. Finally, the cleaning and conforming metadata is a direct indicator of action items for improving the data quality of the original sources.

All metadata within control of the batch manager must be captured, stored, and published. In a best-case scenario, metadata should be stored in an open repository that can be shared with other applications. At a minimum, metadata must have reporting capabilities so users and developers have insight into the operational aspects of the data warehouse ETL.

Migrating to Production

The migration process can vary depending on many variables, including politics, technical infrastructure, and the ETL toolset. In general, the ETL team is usually part of the development side of things and should avoid the distractions associated with providing first-level production support for the data warehouse, unless your organization is large enough to warrant a dedicated production-support ETL team.

For the purpose of this chapter, assume that the ETL team exists only in development and hands its work over to a production-support team when the jobs are ready for production. Again, these processes can vary depending on your organizational structure and the tools employed in your environment. This section should be used as a guide to the finishing touches of the ETL lifecycle.

Operational Support for the Data Warehouse

It's interesting how many books and articles talk about how the data warehouse team needs to maintain the data warehouse. In reality, at least in our experience, the data warehouse team—including the ETL team—are analysts as well as developers. They gather all of the business requirements, analyze the findings, and build the data warehouse. Once it is built, they usually hand it off to another team that monitors and maintains the production environment.

The data warehouse architect and data modelers are responsible for the dimensional data model, and the ETL manager is responsible for populating the dimensionally designed data warehouse.

The ETL development team builds the processes to load the data warehouse, and the quality-assurance (QA) team thoroughly tests them according to the written test plans. The data warehouse needs to be transitioned to the group within your organization that can support its day-to-day operations. If you are a small company or the data warehouse is still in its infancy, the development team may in fact support the operation of the ETL in production. But as the data warehouse grows—more data marts are added to it—the development team needs to be alleviated from the distractions of supporting the operational aspects of the production environment.

NOTE

Once the ETL process is developed and tested, the first level of operational support for the data warehouse and ETL should be provided by a group dedicated to monitoring production operations—not the data warehouse development team. The data warehouse team should be called only if the operational support team has exhausted all troubleshooting procedures without resolution.

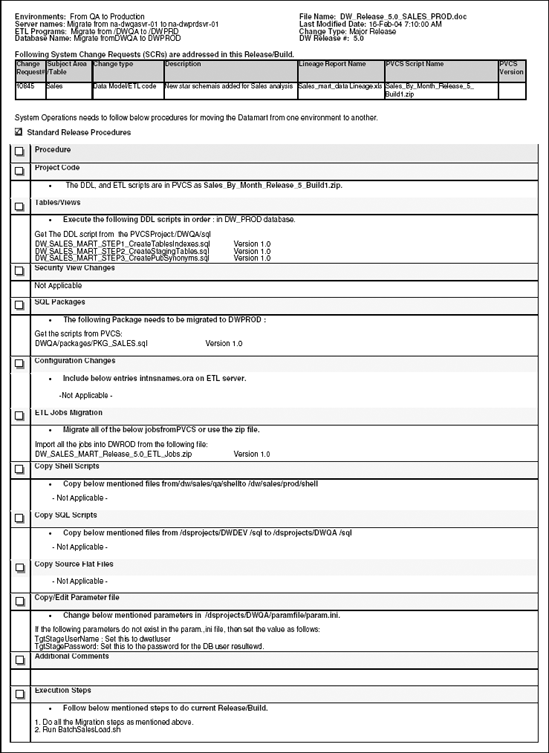
Bundling Version Releases

Once the ETL team gets past the terrific challenges of developing the ETL process and manages to complete the creation of the jobs required to load the data warehouse or data mart, the jobs must be bundled and migrated to the next environment according to the lifecycle that has been implemented by your data warehouse management team.

NOTE

Have a discussion with your ETL tool vendor about exactly this step of your ETL deployment. Does the tool support incremental scripting of all edits, so that you can migrate your test system into development in a single command? Or do all the files have to be opened, examined, closed, and transferred one at a time?

With each data warehouse release, the development team should produce a release procedures document similar to the one displayed in [Figure 8.1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html#data_mart_release_document).



**Figure 8.1. Data mart release document**

The data mart release document introduces the release and provides technical details required to migrate and support the release. The document includes the following:

* **Environment**. This section contains the source and the target environment. Environment migrations are usually Development to Test or Test to Production. If you have more environments such as dedicated User Acceptance or QA, those will also be included in this section, depending on where your project is in its lifecycle.
* **Server name**. The physical names of the servers in the environments participating in the migration. This can list the ETL and DW servers.
* **ETL Programs**. Lists the directory where the programs reside. If you are using an ETL tool, use the component to identify the correct programs or jobs for the release.
* **Database Name**. The database the migration is coming from and going to. This is usually Development to QA or QA to Production.
* **Documentation File Name**. The name of the file that contains information about the migration, including step-by-step recovery procedures
* **Last Modified Date**. The last time the Release Document has been modified
* **Change Type**. The description of the type of release. Types include major, minor, or patch. See [Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html) for a complete explanation of release types.
* **Release Number**. This is the version that the data warehouse becomes as a result of the release.
* **Change Request Numbers**. This corresponds to the requests addressed and included in the deployment as a result of your scope-management procedures.
* **Procedures**. The procedure is a step-by-step guide to migrate the jobs. The standard information usually provided on the release document includes:
  + **Project Code**. The area in the version-management tool to find the code to build the release in the data warehouse
  + **Tables/Views**. The Data Definition Language (DDL) that created the new structures and indexes for the new data mart
  + **Security**. Any new security policies or requirements to support the release
  + **SQL Packages**. Database store procedures used by the ETL
  + **Configuration Changes**. Global settings or entries required for the release, such as TNSNames (Oracle's aliases for remote database names) or Object Database Connectivity (ODBC) connections.
  + **ETL Jobs**. Where to get the ETL jobs required for the release. Usually specifies an area in your version-control manager or ETL tool.
  + **Shell Scripts**. Preprocess and post-process OS shell scripts that the ETL depends on
  + **SQL Scripts**. Preprocess and post-process SQL scripts that the ETL depends on
  + **Flat Files**. A list of flat files, including their path for new source or staging files
  + **Edits to Parameter File**. For environments with managed parameters, this is where you list the new source system databases or any new changed parameters for the ETL process to utilize.
  + **Additional Comments**. Any further instructions or comments about the release to help the system operations team with the migration
  + **Execution Steps**. Explicit instructions for the execution of the job. For new data marts or independent ETL processes, you can specify the schedule and frequency of the run.

Once the data mart release document is complete, the ETL team walks through the document with the implementation team. During the migration, the ETL team should be on standby for any emergency issues that might affect production. Immediate response by the ETL team might not be a key requirement for the first release of the data warehouse. But as the data warehouse becomes recognized as a mission-critical application, any downtime can be detrimental to the organization and must be avoided. Once the migration is complete, the ETL team can return to their regular development tasks for the next phase of the data warehouse. Ongoing support for the production environment should be provided by a dedicated production-support team. The next section walks you through the different support levels required for a data warehouse.

Supporting the ETL System in Production

The beginning of this chapter explains how to get the ETL processes running and shared recommendations on several techniques for scheduling, batching, and migrating ETL jobs. This section of the chapter concentrates on supporting the ETL once it is in Production. Generally speaking, support for any software implementation essentially has three levels:

1. **First-level support**. First-level support is usually a Help Desk type of situation. If a failure occurs or a user notices an error in data, first-level support is notified. Armed with the appropriate procedures provided by the ETL team, the first-level support team makes every attempt to resolve the situation before escalating it to the next level.
2. **Second-level support**. If the Help Desk cannot resolve the support issue, the system administrator or DBA is usually notified. The second level of support is usually technically proficient and can support general infrastructure type failures.
3. **Third-level support**. If the production operations technical staff cannot resolve an issue, the ETL manager is the next to be called. The ETL manager should have the knowledge to resolve most issues that arise in production. Sometimes the ETL manager converses with developers or external vendors for advice on certain situations.
4. **Fourth-level support**. When all else fails, go directly to the source. Fourth-level support demands the expertise of the actual developer of the ETL job to analyze the code to find a bug or resolve an issue. If the issue involves a potential bug in a vendor application, the vendor is called in to support its product.

In smaller environments, it is acceptable—and common—to combine support levels three and four. However, that combination puts more of a burden and coordination factor on the second-level support team. It is not advised to call an ETL developer every time a job fails. First-level support should be more than merely a phone service and must make every effort to resolve production issues before they are escalated to the next service level.

Achieving Optimal ETL Performance

Okay, you've thoroughly read this book and implemented a luminous ETL solution. But wait; you are not done yet! As your data warehouse expands, you must ensure that your ETL solution can grow along with it. A scalable ETL solution means that the processes you've designed have the ability to process much larger volumes of data without requiring redesign. Your designs must execute efficiently to achieve the performance required to process loads far greater than the size of their current volume. Scalability and performance are attributes that cannot be overlooked when designing the data warehouse ETL.

ETL developers, DBAs, and the data warehouse management team will benefit from this chapter because it outlines strategies that monitor and improve existing ETL implementations. The chapter assumes you've already done your analysis, designed your logical data lineage, and implemented the physical ETL process. It dives right into the details and techniques that should be applied to new and existing ETL jobs to obtain optimal technical performance.

Toward the end of this chapter, you'll find tips on how to tackle security issues in the data-staging area. We specifically address vulnerability during File Transfer Protocol (FTP) and offer techniques for encrypting and decrypting data in-stream to provide a secure yet efficient environment.

Upon completion of this chapter, you'll be able to offer expert ETL tuning techniques to your data warehouse team. The techniques offered in this chapter are specific to the data-staging environment and are not intended to be used to optimize the presentation layer of the data warehouse.

NOTE

If you need information on optimizing the target data warehouse, we recommend that you read *Essential Oracle8i Data Warehousing: Designing, Building, and Maintaining Oracle Data Warehouses* by Gary Dodge and Tim Gorman (Wiley 2000).

Estimating Load Time

Estimating the initial load of the data warehouse to bring all history from the transaction system into the data warehouse can be overwhelming, especially when the time frame can run into weeks or even months. Throughout this book, we present the extract, transform, and load processes as a complete package, according to the goal of the ETL process in its entirety. However, when you estimate a large initial load, it is necessary to divide the sections of the ETL system into its three discrete processes.

* Extracting data from source systems
* Transforming data into the dimensional model
* Loading the data warehouse

Estimating Extraction Process Time

Surprisingly, extracting data from the source system can consume the greater part of the ETL process. The historic load for the data warehouse extracts an enormous amount of data in a single query and online transaction processing (OLTP) systems are just not designed to return those voluminous data sets. However, the breath-of-life historic database load is quite different from the daily incremental loads.

But in any case, transaction systems are not designed to pull data in the way required to populate fact tables. ETL extraction processes often require overhead-intensive methods such as views, cursors, stored procedures, and correlated subqueries. It is crucial to estimate how long an extract will take before it is kicked off. Estimating the extract time is a difficult metric to calculate. The following obstacles prevent straightforward estimation procedures:

* **Hardware in the test environment is usually substantially smaller than in production**. Estimates based on executions of the ETL processes on the test environment can be significantly skewed because of the hardware difference between the test and production servers.
* **Since the ETL jobs that extract historic data can take days to run, it is impractical to perform a test run of the complete data set**. We have seen projects where an extract job ran and ran until it eventually failed; then it would be restarted and run again until it failed again. Days to weeks went by without an ounce of productivity.

To overcome the difficulties of dealing with massive volumes of data, you need to break the extract process into two smaller components:

* **Query response time**. The time it takes from the moment the query is executed to the moment the data begins to be returned
* **Dataset retrieval time**. The measurement of time between the first and last record returned

Since the initial extract load is likely to be massive, it's recommended that you take a portion of the data for your estimation. In most cases, the fact table you are loading is partitioned. We recommend that you retrieve enough data to populate an entire fact table partition for your sample. Use a partition worth of data for your sample because database partitions should divide data—more or less—into equal portions and provide a clean benchmark. Once the time to extract one partition is obtained, multiply that number by the number of partitions allocated for the fact table to estimate the total extraction time. The caveat with this method is that it combines the query response time with the data retrieval time, which can skew estimates.

**Calculating Query Response Time**

The best approach to isolating the two processes is to utilize a querymonitoring tool. Most of the ETL tools have a monitoring tool built into their application. Keep in mind that if you are using an ETL tool, reference tables are loaded into memory before the main extraction begins. Therefore, you need to separate the cache process from the raw extract process as well.

**Calculating Data Retrieval Time**

Once the extract query starts to return data, begin timing exactly how long it takes to load a portion of the data. Select a portion that makes sense for your situation. If you've got 200 million rows to load, perhaps one million would be a good test. Stop the job when exactly one million rows have been loaded and check the elapsed time. Then multiply the elapsed time by 200 (200 million rows total/1 million test rows) to derive the data retrieval portion of the extraction estimate for the entire historic load.

NOTE

Once the extraction job has been thoroughly tested, insist that the sample job for the estimate is performed in the production environment to prevent skewed results that would occur by running the process in a smaller technical infrastructure.

Estimating Transformation Process Time

One would expect that manipulating data would be a time-intensive effort. Surprisingly, most of the actual transformations of data are done in memory at an amazing rate of speed. Relative to its sister processes, extraction and load, the time it takes to physically transform data can be inconsequential.

If you are using stored procedures that utilize cursors, consider redesigning your system. Depending on your circumstances, it might be best to utilize an ETL tool and minimize the use of database-stored procedures for your transformation processes. Most of the ETL process should be consumed by I/O (physically reading and writing to disk). If the transformation time is not significantly less than that of the extract and load processes, you might have a major bottleneck in your transformation logic.

The easiest way to estimate transformation time is to gather the extract estimate and the load estimate and then run the entire process. Once you have those statistics, subtract the duration of the extract and load processes from the complete process time. The difference is the time spent on the transformation process.

Estimating Loading Process Time

When you calculate the load time, you need to ensure that delays aren't being caused by the transformation of the data. Even though you may be pipelining the data from the transformation to the load process, for the purpose of the estimate, you need to touch down the data to a flat file after it's been transformed.

Many variables affect load time. The two most important factors to consider are indexes and logging. Make sure the environment during the test exactly matches the physical conditions that exist in production. Like data retrieval, the data load is processed proportionately. That means you can bulk load a sample data set—say 1 million of 200 million—and then multiply that duration by 200 to derive the complete load time estimate.

Vulnerabilities of Long-Running ETL processes

The purpose of an ETL process is to select data from a source transaction system, transform it, and load it into the data warehouse. The goal of the ETL team is to design efficient processes that are resilient against crashes and unexpected terminations while accomplishing those tasks.

Horizontal versus Vertical ETL System Flows

ETL systems are inherently organized either horizontally or vertically. In a horizontal organization, a given extract-clean-conform-deliver job runs from the start to completion with little or no dependency on other major data flows. Thus, a customer-orders ETL job could run to completion, but the inventory tracking ETL job could fail to complete. This may leave the decision makers in the organization with an inconsistent and unacceptable situation.

In a vertical ETL system organization, several ETL jobs are linked together so that comparable steps in each job run to completion and wait for the other jobs to get to the same point. Using our example, both customer orders and inventory tracking would need to complete the extract step before either would advance to the cleaning, conforming, and especially the delivery steps.

Determining whether your ETL system should be horizontally or vertically organized depends on two big variables in your environment:

1. Detailed data dependencies that require several ETL jobs to progress through the steps in parallel. (That is, if the inventory tracking job fails, maybe the customer-orders job could have undefined product codes.)
2. The sensitivity the end user community might have to partially updated data (for example, orders being updated but shipments lagging by a day)

Analyzing the Types of Failure

Unfortunately, to execute and complete successfully, the ETL process depends on many components. Once an ETL process is in production, failures are typically due to reasons beyond the control of the process itself. Leading causes for production ETL failures include:

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

To familiarize you with the variables in your environment that may pose a threat to your ETL processes, this section discusses each of the ETL vulnerabilities and offers tips on minimizing your risks.

Network Failure

The network is the physical infrastructure that connects all components of the data warehouse. Each server, whether it is for a database or an application, connects via the internal corporate network. With the miles of cabling, routing, and nodes, the risk of network faults always exists. Network failures will never be completely unavoidable, but the ETL can take measures to minimize vulnerability to network failures.

A precaution to reduce your vulnerability is to put the ETL engine on the same server as the target data warehouse database. Obviously, this choice raises the issue of resource contention between ETL jobs and end user queries, but in the realm of minimizing network failures this practice eliminates 50 percent of network traffic because the data can pass from the ETL engine to the target data warehouse on the internal bus of the server. In many cases this co-residency makes sense if conventional data warehouse querying happens during the day while ETL processes take over most of the system resources at night.

Database Failure

Remember, the initial load does not only happen at the beginning of the data warehouse implementation. If you have an enterprise data warehouse implemented using the Data Warehouse Bus Architecture, each phase of the data warehouse requires an initial load. Each data mart needs to have historic data loaded into it before it is loaded incrementally.

A physical database failure is known as unscheduled downtime. With today's available technology, where virtually everything is redundant, un-scheduled downtime can and should be avoided. Make sure you have a comprehensive Service Level Agreement (SLA) that specifies your unscheduled downtime rate requirements.

Moreover, a database does not have to be physically down to be perceived as down. One of the goals of the ETL team is to conduct the required processes to load the data warehouse yet remain transparent to users. If an ETL process has a table locked or has the temp space pegged, the experience by the user is a failure. Perceived database failures are as detrimental to the reputation of the data warehouse as physical failures.

Disk Failure

The storage of data in the data warehouse is perhaps the most vulnerable component of the data warehouse to all potential points of failure. Typically, three disk groups are involved in the ETL process:

* **Source system disk**. Typically, the ETL process merely reads data from the source system disk and risk of failure is minimal. However, use extra caution while extracting the initial history from the system. An extract with many years of history can be a quite large data set. If you run complex extract queries with multiple joins and order by or group by clauses, you may exceed the disk space allocated for these types of operations. To reduce your vulnerability, work with the source system DBA team to monitor temp-space usage while you perform test runs of the history load process. Make sure ample space is allocated before you run the whole process.
* **Staging area**. The staging area is the workbench of the ETL process. Generally, it contains a number of staged files representing different steps in the flow of data from the source to the final dimensional targets. Data would normally be staged immediately after each of the major steps of extracting, cleaning, conforming, and preparing for delivery. The process reads and writes to this area for several reasons, including for data persistence and safekeeping, as well as a holding cell for data in the midst of transition. The data-staging area can be the size of the source system and the data warehouse combined. However, this is rarely the case. But keep in mind that the possibility is there and that the data-staging database is often off the radar for the data warehouse and DBA teams. As a precaution, periodically check the available space in the data-staging environment to ensure you are not running too low.
* **Data warehouse storage**. The data warehouse can grow much faster than initially anticipated. Quite often, space for indexes and temp space is underestimated and allocated space is exceeded. When the ETL process tries to write to unallocated disk, the process crashes. It is very difficult to recover from errors that involve disk space. To prevent running out of space, you need to *lie* to your DBA team and exaggerate the volumetric estimate of the initial load and three-month size estimate of the data warehouse. We used to double our estimates, but now, after a few close calls, we triple our volumetric to be safe. We recommend tripling the estimate of your initial load to avoid potential catastrophe. Trust us; the space will not go to waste.

NOTE

It is not enough to simply measure various storage capacities and utilizations. The ETL team should monitor these numbers regularly and do something when the alarm thresholds are reached. We have seen alarms set at 90 percent of capacity, but then the warning gets ignored for six weeks. Boom!

Memory Failure

Memory can fail in any of the three environments:

* Source system
* Staging area
* Data warehouse

These environments are equally vulnerable to overloading their allocated memory and failing the ETL processes. A memory overload will not necessarily *crash* your process, but it will slow it down tremendously when it starts to utilize *virtual memory*—when the operating system writes data intended to be in random access memory (RAM) to disk. Make sure you consult with your ETL application vendor to obtain recommended cache settings for your particular history load.

If you have a hardware breakdown, you'll need to correct the problem and restart your process from the beginning (unless your ETL tool can recover gracefully).

Temp Space

Temp space is the area of the database used whenever you sort or join data. Temp space is often *blown out* when data warehouse type queries are run on a database environment set up for transactional processing. Fear of blowing out temp space is one of the primary reasons that historic data should be extracted from the source system into a staging area in its simplest form and then transformed further in the dedicated staging environment. If you fill up the temp space in any of your database environments, your process will stop dead.

When temp space failures arise, the DBA team needs to allocate more space, and the process must be restarted. Depending on where and when the temp space failure occurs, data cleanup is almost always required. Since the data warehouse is designed for queries, temp space should be plentiful. The issue typically occurs in the source system environment, where the ETL team unfortunately does not have any control.

Data Space

In the data warehouse, data should be stored separately from indexes to alleviate contention and lessen the burden of managing space. A practical solution to estimate the size of your historic load is to load a small sample of data. Loading a single partition of a partitioned table is a good benchmark. Then you can multiply the amount of space consumed by loaded data by the number of partitions in your table. [Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html) illustrates a volumetric worksheet that provides more information on estimating data space.

NOTE

Make sure the data-staging databases and data warehouse databases have ample space available to load your historic data before the massive load is kicked off. Running out of disk space is an ungraceful failure that requires manual data cleanup and a restart of the load process.

Index Space

Estimating space for indexes is a tricky science because indexes do not grow proportionately like table data. We won't go into too much detail on index space; the calculations are fairly complex, and the data warehouse architect, data modeler, and DBA should have the index space created appropriately in the data warehouse before you start. As a rule of thumb, make sure there is at least as much space allocated for the indexes as there is for the base table.

NOTE

When you load voluminous historic data, drop the target table's indexes before the load begins; and rebuild them after the load completes. By dropping and rebuilding the indexes, not only does performance improve, but you are insulated from load failure if the index space runs out of room. Once the table is loaded, you can always allocate more space to the indexes and rebuild them without affecting the data in the table.

Flat File Space

Just as the space allocated for data in a database can be exceeded, space on your file system must be monitored to avoid exceeding the space allocated to your flat files. Fortunately, the space requirement in the staging area is allocated according to the ETL team requirements, so if you've followed the recommendations for estimating file system requirements in [Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html), you should be in a safe position for processing your historic loads. Some robust ETL tools include a *checkpoint* feature that guarantees that any record to reach a certain point in the process is written to disk for safekeeping. But remember, those checkpoint files are written to the file system and might well be the culprit that is filling up your file space. Checkpoint, cache, hash, flat, temporary, or any intermediate data staging file can fill your disk space. If you exceed allocated space and your process crashes for any of these files, it is recommended that you begin the process from the beginning once additional space has been granted rather than attempt to salvage already processed data.

Data-Quality Failure

Data-quality failure in production can either be an easily detected catastrophic administrative failure, such as missing fields or referential integrity violations, or it can be a threshold of data-quality warnings reached gradually in the course of a long run. A data-quality failure in production should be an unusual event and generally requires expert intervention. Perhaps the job can run to completion with known unusual data or perhaps the job needs to be backed out and the source data fixed.

Unannounced System Upgrade

Perhaps the only good news about unannounced system upgrades is that they are usually dramatic and obvious. The ETL job hangs. There is often no simple fix at the scrimmage line. If the reason for the suspended job cannot be fixed quickly, provision must be made for rolling back the system upgrade. This situation is no different from any software change; for critical systems, you must perform the most robust regression tests on a test system before installing the changes on your production ETL systems.

Recovery Issues

Whenever a process fails, it is a knee-jerk reaction for the ETL team to try to salvage whatever has been processed up to the point of failure. If you are lucky and are using an ETL tool that has checkpoint functionality, you may be able to restart your process, and it will magically pick-up where it left off when it failed. Notwithstanding vendor claims, we'd be hesitant to rely on checkpoint technology. If a process fails midstream, it's good practice to clean up the data and begin the process from the beginning. Divide your processes into subsets of data to make recovery issues cleaner and more efficient; instead of reprocessing the entire history load, you merely have to reprocess a single subset of the data.

Minimizing the Risk of Load Failures

Here are some rules of thumb for processing historic data.

* **Break-up processes**. Use dates or ranges or the natural key to break up the process into smaller manageable chunks. In the event of a failure, only that portion of data needs to be reloaded.
* **Utilize points of recovery**. Write data to a flat file for safekeeping after every major intermediate process (for example, upon extract, significant transformation, or once surrogate keys are assigned).
* **Load in parallel**. Not only the data load but also all of the components of the ETL should run in parallel to reduce the time it takes to process data.
* **Maintain metadata**. Operational metadata (for example, the last date loaded or the number of rows loaded) is crucial to detecting the status of each of the components of the ETL during a failure.

Purging Historic Data

When any database application is designed, a matrix is created to track the processes that insert, update, delete, and select the data. The matrix is commonly referred to as a CRUD (Create, Read, Update, and Delete) Matrix. The CRUD Matrix ensures that every entity has a process to perform each of the four ways to manipulate data. While developing application software, it is most common that the *D* in the matrix is overlooked. That means that the data gets entered and can be changed and read but that there is no formal process for deletion. When no formal process is developed to purge history, one of two things usually happens: Back-end scripts are run against the system to delete the history, or the records stay in the system indefinitely. As you might imagine, neither of these solutions is suitable for a data warehouse.

A purge process must be laid out as each subject area is planned. If volume is relatively small and ten or more years of future data will not affect performance, the ETL need not be developed right away. However, the purge-policy metadata must still be collected and published with the initial implementation.

Archiving data warehouse data should be done by the DBA, not by the ETL team. However, the permanent deletion of data from the data warehouse must be executed by the ETL team. Business rules surrounding deleted data must be enforced by a thoroughly tested and quality-assured ETL process.

Monitoring the ETL System

The business depends on the data warehouse to be refreshed at an agreed interval (or continuously). Failure to fulfill that responsibility puts the reliability and dependability of the data warehouse in question. Therefore, the data warehouse cannot succeed without efficient and persistent data feeds. The ETL team must monitor and evaluate the ETL jobs to ensure they are operating efficiently and the data warehouse is being loaded in an effective manner.

ETL monitoring takes many aspects of the process into consideration. Resources outside the scope of the ETL system such as hardware and infrastructure administration and usage, as well as the source and target environments, play crucial parts in the overall efficiency of the ETL system. Here, we introduce several ETL performance indicators that tell you how well (or not so well) your processes are executing. The indicators are part of operational metadata and should be stored in a repository so they can be analyzed over time by the ETL team.

Measuring ETL Specific Performance Indicators

Those of you with exposure to system or database administration are aware that there are measurements specific to environments captured to ensure that they are performing properly. As you might expect, the ETL system has its own set of performance indicators. ETL indicators are specific to the physical movement and management of the actual data. They are a step below the typical performance indicators. By *below*, we mean they do not measure at the operating-system level or hardware-resource level but within the ETL process itself.

The measurement most indicative of ETL efficiency is the actual time it takes to process data. Remember: The goal of the ETL system, besides creating quality information, is to load the data warehouse within the allotted load window. But if a job takes 20 minutes to complete, is that good? There's really no way of knowing unless you know how many records were processed during that time. For example, 20 minutes is fantastic if you are processing 50 million rows but less than adequate if the job processes only 100 rows. Following are ETL-specific measurements that prove to be useful while investigating load performance.

* **Duration in seconds**. This straightforward calculation is the basis of all other calculations. The duration is the difference between the start time and the end time of an ETL process in seconds. For example, if a process is kicked off at 4:00 a.m. and completes at 4:15 a.m., its duration is 900 seconds.
* **Rows processed per second**. This is equivalent to the rows loaded per second calculation, except in cases where the source data is larger than the target, as in the case of aggregate loads. Then it is the same as the rows read per second. A sample calculation of rows per second is 1,000,000 rows processed in 15 minutes (1000000 / (15 \* 60)) = 1111.11 rows/sec.
* **Rows read per second**. The row count of the result of the SQL that retrieves the data from the source system divided by the duration in seconds. The data is then fed through the downstream transformation processes in the ETL pipeline where the row count can increase or decrease depending on the process.
* **Rows written per second**. The count of rows committed to the target table after they have been transformed, divided by duration in seconds. In cases with multiple target tables, it is the sum of all rows inserted into all tables divided by duration in seconds. The rows written can be greater or less than the rows read.
* **Throughput**. Throughput is the rows processed per second multiplied by the number of bytes in each row. Throughput, as with all performance measurements, is an approximation that should be used as a baseline for improving processes.

Most of the major ETL tools provide the necessary metrics to measure the performance of the ETL. You should instrument your ETL system to trigger an alert for any ETL job that takes dramatically more or less time to run than the historical experience would predict.

NOTE

Bottlenecking occurs when throughput of a process diminishes due to a component of a process not being able to handle the output volume sent by a prior component in the process stream. For example, a bulk loader can feed 1000 rows per second to disk, but the disk may write only 800 rows per second. Therefore, throughput bottlenecking occurs at the disk component of the process. As a result, the entire process can be only as fast as its slowest component.

Measuring Infrastructure Performance Indicators

The next component to examine is the infrastructure of the data-staging area. Many process metrics are available through different monitoring software packages or can be written to logs in hand-coded ETL solutions. Only a few vendor solutions intentionally offer direct indications that ETL performance is being affected. The crucial measurements for ETL performance can be obtained only by monitoring the processes as they run. Measurements that offer a direct indication that there may be a bottleneck in the process include the following:

* CPU usage
* Memory allocation
* Server contention

Naturally, network traffic and other known infrastructure performance indicators can affect ETL performance. Unfortunately, their measurements are so volatile that they are not stable or consistent enough to use reliably. Moreover, the origin of network traffic is extremely hard to identify or duplicate in a test environment. If you suspect that you are experiencing network issues, contact your network administrator for assistance.

CPU Usage

The ETL process runs on the central processing unit (CPU) of its server. CPUs are processors or chips that actually facilitate the computations required to operate the computer, make software run, and accomplish the ETL goals. Most ETL servers contain more than one processor to handle the enormous amount of computations required for the extraction, transformation, and load of data warehouse data. You are not likely to find CPU-usage reporting within your ETL tool, as it is outside the scope of the ETL system. However, if you are running your process on Unix, you can use the SAR –u command to list the usage of each processor in your system.

On Windows-based operating systems, you can use the Performance Monitor, a graphical user interface that allows you to add new counters to the already available performance logs. In Windows XP, the Performance Monitor can be found in the Administrative Tools applet in the Control Panel. To add a new counter, open the Performance Monitor, right-click the System Monitor Details pane, and then click Add Counters. From there you can select *Processor* as the performance object and select the relevant counters. The Performance Monitor will create a log file that captures statistics of your CPU usage for analysis.

If you find that your CPUs are often reaching their capacity during the ETL process, you need to add possessors. These CPU monitoring utilities also help you to reveal if the ETL process is being distributed evenly across all available processes.

Memory Allocation

Random access memory (RAM) can be allocated to an ETL process in many different places. First of all, the memory has to be physically available on your server. If you are purchasing an ETL tool, your vendor should be able to provide hardware specifications and recommend how much RAM you are likely to need to process your load volume with their toolset. Once the RAM is installed in your server, the memory must be allocated to your processes. In most ETL tools, memory usage can be specified at the job or batch level as well.

You must have appropriate memory allocated to each ETL process for ultimate efficiency. If your process is constantly reading and writing to disk instead of processing data in memory, your solution will be much slower. Appropriate memory allocation in your process affects transformation performance more than any other setting, so take extra care to ensure the setting is correct.

Your ETL tools should be able to tell you how much memory is allocated to each process, how much the process actually uses, and also how much spills over to virtual memory (cached to disk). Make sure this operational metadata is provided by the ETL tool during your proof-of-concept. If much of the data that should be in RAM is getting written to disk, you need to allocate more memory to the process or to get more physical memory installed on your server.

Some ETL tools make memory management completely transparent to the development team, while others might require certain configurations to be set manually. Memory settings that can be found in some of the major ETL tools include:

* **Shared memory**. When ETL engines read and write data, they use a dedicated area in memory called *shared memory*. Shared memory is where data queues up before it enters or exits the physical transformation portion of the ETL process. If not enough shared memory is allocated to your job, excessive disk caching occurs. Conversely, if too much shared memory is allocated, an unnecessarily large amount of RAM is reserved and taken away from other processes. Your ETL vendor should provide guidelines on how to calculate optimal shared memory settings depending on the size of the data being processed. Some ETL engines may attempt to manage shared memory dynamically. Look for a tool that allows you to override the system-assigned setting for special situations that the engine may not be able to assess.
* **Buffer block size**. The buffer block setting is a key element to consider when allocating performance-related settings. The proper setting of buffer block size depends on the row size of the data in the transformation stream. If your tool requires or allows custom adjustments to the buffer block size, your ETL vendor can recommend calculations for optimal settings.

NOTE

When a program requires more RAM than is physically available, the operating system (or application) writes data that doesn't fit into memory onto disk. As the overflowed data is needed, the program must read from and write to disk instead of utilizing RAM. *Virtual memory* is commonly referred to as page swapping because memory is stored in pages and as more pages are required, they are swapped between disk and RAM. Page swapping is a performance killer and should be avoided during ETL processing. If continual page swapping is detected, more RAM must be added to the ETL server and process.

If you opt—at least for the time being—to hand-code the ETL, you can manually monitor memory usage with the vmstat command. Among other key measurements, the vmstat command reports virtual and real memory usage as well as paging activity and disk operations.

Server Contention

Another potential performance killer is server contention. Contention occurs when more than one process attempts to use the same resource. You can encounter contention for memory, disk access, or data access. The most common offender of contention is when two ETL processes try to access the same data. The ETL processes can cause deadlocks. A deadlock happens when process A attempts to lock out process B while process B attempts to lock out process A, and the system simply hangs. Usually, the DBMS does a good job at managing data-access contention, but it *will* happen at some point in an ETL developer's career. The ETL system is most vulnerable to server contention when ETL processes run concurrently but not in parallel. When this happens, the processes constantly compete for resources, and collision is imminent. Your best defense is to avoid concurrent processes unless each process has dedicated process streams and appointed data partitions.

Memory Contention

When many applications or processes are running on the same server, they each need physical RAM to operate. Unfortunately, RAM is a limited resource, and the individual processes must contend for it. If you are running ETL processes concurrently, you may run into memory-contention problems. The memory-allocating settings at the job level become crucial when they are processed concurrently. Each ETL product has its own recommendations for alleviating memory contention. As a rule of thumb, minimize the memory allocated to small jobs, leaving room for larger jobs such as Type 2 slowly changing dimensions, bridge tables, and facts.

The ETL tools should be able to manage memory usage within its tool and avoid memory contention. In any case, the tool should provide the operational metadata to expose memory contention. If it doesn't, the Unix SAR command can assist in detecting memory contention. The SAR command is especially useful to detect memory usage of processes running beside the ETL tool and competing for the same RAM. If your process warrants it (and the budget allows it), make sure that the ETL engine is the only process running on your server during the data warehouse load window.

The efficiency of an ETL processes is questioned when it does not complete within the load window. Effective monitoring can usually reveal that most load delays are not the result of ETL inefficiency but of external processes running at the same time as the ETL and competing for server resources.

Disk Contention

Most disks have a limit on the number of accesses and the amount of data they can read or write at any given time. When that limit is reached, the ETL processes have to wait in line to access the disk. If you place many tables loaded concurrently in the same data files on the same disk, hot spots can occur. A *hot spot* is an area on disk repeatedly accessed for reading or writing. If you are using Oracle, you can use the following SQL to detect hot spots in your source, staging, or target databases:

select d.name datafile\_name, f.phyrds reads\_count, f.phywrts

writes\_count

from v$datafile d, v$filestat f

where f.file# = d.file#

order by greatest(f.phyrds, f.phywrts) desc

Since this query sorts the result set by the number of reads and writes in descending order, the data files hit the most rise to the top. If you have a few data files disproportionately larger than the rest, you need to reconfigure the physical attributes of your staging tables so they are distributed more evenly.

Disk contention occurs outside the database as well. ETL engines utilize temporary fields and implicitly create files to hold transient data. Furthermore, developers explicitly create staging tables, configuration, and parameter files on the file system. Constant reading and writing to these files can cause unwanted disk contention. For information about your disk activity from the operating system point of view, use the IOSTAT Unix command. The IOSTAT command lists each disk and pertinent information:

* Name of the disk
* Reads per second
* Writes per second
* Kilobytes read per second
* Kilobytes written per second
* Average number of transactions waiting for service (queue length)
* Average number of transactions actively being serviced (removed from the queue but not yet completed)
* Average service time, in milliseconds
* Percent of time there are transactions waiting for service (queue nonempty)
* Percent of time the disk is busy (transactions in progress)

Information about how to resolve disk contention is provided later in this chapter.

Database Contention

Database contention can be most problematic if the ETL processes attempt to update records in the same table at the same time. Essentially, managing database contention is the job of the DBMS, but at times processes that contend for the same resource can block each other out, causing them to wait indefinitely. Refer to your specific DBMS reference manual or contact your local DBA for the best procedure for detecting database contention.

Processor Contention

Sometimes, an attempt at parallel processing at the software level can cause problems if your hardware is not configured to run parallelized. When you have multiple processes—more than the number of processes available—attempting to run at the same time, you can overload the CPUs and cause critical performance issues. You can use the SAR command on Unix or PerfMon on Windows to capture statistics on the CPU usage.

Measuring Data Warehouse Usage to Help Manage ETL Processes

If you refer to the supply-chain analogy we provide earlier in this chapter, you'll notice that we've identified four key components that lend a hand in transforming raw data to the customer in a useful format for consumption. So far, we have described how to monitor the activity within the scope of the ETL system as well as in the hardware and infrastructure of the ETL environment. Now we outline important measures within the presentation layer of the data warehouse.

The measurements in this section indirectly affect the ETL system but are important to capture and analyze because the do have an impact on the load processes. Hopefully, the measurements in the list that follows soon are already being captured by the data warehouse team to help manage their user experience and prune the data warehouse of data stored but not used.

The ETL team should take advantage of data-warehouse usage reports and look for opportunities to rearrange the load schedule, modify the load frequency, or eliminate the maintenance of jobs that load dormant tables. For example, if a table is accessed only on the first of the month, it should not be updated daily. Another efficiency gain can be achieved by analyzing index usage. A large portion of ETL processing includes rebuilding indexes in the data warehouse after each load. If usage analysis determines that certain indexes are never utilized, their reconstruction should be dropped from the ETL process. Usage metrics that support ETL job management include:

* **Table usage**. The contents of a table-usage report can vary, but a useful report contains a list of each table, a timestamp to represent the first and last time the table is accessed, the count of queries that reference the table, and the count of distinct users that query the table. Tables accessed first should be made available first. Tables used only once a month can be dropped from the daily load process and switched to a monthly frequency. Tables that appear to have continuous usage, except for when the table is being refreshed, are candidates for *high availability* techniques. Tables highly available have a duplicate structure loaded in the background. Once the load is complete, the names of the two identical structures are switched. This technique leaves the table online while the refresh takes place.
* **Index usage**. Indexes are key performance enhancers for data warehouse tables but are a burden to the ETL because in many cases they must be dropped and rebuilt with each data load. When a data warehouse architect builds the dimensional structures for the presentation layer, he or she has a tendency to index as many columns as possible to prevent bad performance experiences by a user. The fact is that many indexed columns are never constrained on and the indexes are never utilized. An index-usage report reveals dormant indexes whose demise can be negotiated with the data warehouse team.
* **Aggregate usage**. Aggregates are typically built in the same vein as indexes—when in doubt, build it. But just as the case with indexes, some aggregates are built but never utilized. Or over time, they become less interesting and fall dormant. An aggregate-usage report can identify aggregates that are no longer used and should be dropped.
* **Dormant data**. The dormant-data report is always interesting because the data warehouse is created as a result of user interviews that find out what data elements are needed to perform the analysis required to do their job. Yet it's inevitable that tables refreshed by the ETL every day lay unused. Even if a table is used, certain columns may never be selected. We always find fact table column usage to be interesting because it's so common to find that the most complicated derived measures are not used because their definitions are not properly conveyed to the user community. A dormant data report can help the ETL team identify and question the effectiveness of measures and dimension attributes that are never selected.

You have several ways to gather statistics on the usage of the data warehouse. Some database management systems offer usage information natively. However, be sure to test performance with the usage-reporting functionality turned off versus having it turned on; it may affect query response and ETL load time. A noninvasive way to track usage statistics is to employ a middleware such as Teleran Technologies (www.teleran.com). These data warehouse monitoring tools capture SQL and data outside of the database at the network-packet level. We're sure there are other tools that provide database usage statistics. Try typing **data warehouse usage tracking** in www.google.com to find a list of vendors in this space. Also, a list of monitoring vendors is available at the companion Web site to this book.

Tuning ETL Processes

To best understand how to optimize ETL processes, you must be familiar with how databases work. Some functionality available in most database management systems should not be used in the data warehouse environment. And some features are hardly ever used in transaction systems that are not only applicable but also preferred in the data warehouse environment.

Many design decisions are based on the volume of data being moved by the process being developed. For example, if you have a very small dimension with minimal volatility, it is okay to have your incremental process update the existing data in the dimension with SQL UPDATE statements. But if the dimension is a monster with 20 million rows of highly volatile data, it is probably more efficient to truncate and bulk load the table.

Use the volumetric report created by the data warehouse architect or project manager that documents how much data will be loaded into the data warehouse initially and the planned growth to occur six months after implementation for capacity planning and to identify scalability expectations of the ETL system. Then follow up by documenting the actual growth over time.

The next few sections highlight the functionality of databases that are unnecessary overhead and should be avoided in a controlled ETL environment and provide faster but otherwise equivalent solutions for your implementation.

Explaining Database Overhead

Before relational databases were created, data was stored in sequential or flat files. Those files were known for having notoriously bad data quality. The bad data quality stemmed from repeating groups and elements, lack of primary keys, and no enforced relationships between tables. Everything was repeated throughout the database. Additionally, no validation of data existed at the database level. In those days, a database was nothing more than a collection of disconnected sequential files. In short, the situation was a mess.

In 1970, E. F. Codd invented the relational algebra that served as the basis for the design of relational database systems. Relational databases enforce referential integrity, data uniqueness, primary keys, check constraints, foreign keys, and so on. The result is much cleaner, more reliable data, but much slower operations. Each of the features of the relational database adds significant overhead to transactions. *Overhead* is additional processing by a program to perform behind-the-scenes error checking and controls.

In this section, we discuss different database features that you are likely to encounter as a member of an ETL team, and we offer suggestions to overcome database overhead. This is not a substitute for DBA training. In fact, the content of this section will not help you become a database administrator. The purpose of this chapter is to help the ETL team and those who are already DBAs to understand special considerations for optimizing ETL processes. Much of the work involved in optimizing the ETL is outside of the database. Portions of this chapter lend some insight into how databases handle large sets of data and offer tips and techniques for speeding up (or avoiding) those processes.

Inserts, Updates, Deletes

Data manipulation language (DML) has four main verbs: select, insert, update, and delete. Each of the four DML utilities has the ability to manipulate data in a database differently. Remember that DBMSs are primarily designed to survive transaction failures. Therefore, as a precaution, virtually every DBMS maintains a rollback log. A rollback log records DML transactions and provides a mechanism to *undo*changes that occur as a result of a DML submission. In the case of a midtransaction failure, the DBMS automatically rolls back the half-finished transaction, leaving the data in the exact state it was in before the transaction began.

The Effects of Logging

Each of the four types of DML affects the rollback log in a different way. Select statements don't get written to the log, because they don't alter the existing data. In most databases, insert statements are written to the log just in case data is inadvertently entered or the transaction fails midstream, the DBMS can simply rollback the entry instead of having to delete or clean it up. Updates and deletes both require writing to the rollback log. Deletes require some overhead because they store the *old* records before the deletes occur. Updates require the most overhead of all DML statements and are extremely slow to process.

The Effects of Indexes

The data warehouse is utilized because it is substantially faster and more reliable than the transaction system. The speed advantage that the data warehouse offers is due to a number of key features:

* Dimensional data model that allows purpose-built indexing of dimensions and facts
* Aggressive index strategy
* Physically stored aggregate records
* Query parallelism

Design techniques and benefits of dimensional data models are sprinkled throughout this book and are available in a wide range of others. In this section, we'd like to talk for a minute about indexes.

Indexes are the backbone of query-response time in the data warehouse. Every query that hits the database utilizes at least one index. Unfortunately, inasmuch as indexes help users query the data warehouse, the ETL team is burdened with managing the existing indexes during the ETL process. Index management accounts for a substantial portion of most ETL processes.

Before we dive into the techniques for managing indexes during the data warehouse load, it's important that we review the different types of indexes available in most databases. Primarily, you find two types of indexes, and it's important to understand the distinction between the two types:

* **B-tree indexes**. B-tree, or balanced tree, indexes store the key values and pointers in an inverted tree structure. B-tree indexes are optimal for columns with very high cardinality. By *high cardinality*, we mean the count of distinct values. Inverted tree structures utilize an extremely effective *divide and conquer* technique of sifting through data to find a specified value (or range of values). B-tree indexes are great for optimizing known queries but are fairly inflexible at supporting ad-hoc environments such as data warehouses. B-tree indexes are deemed inflexible because you cannot combine indexed columns on the fly to dynamically create compound indexes to resolve new, unexpected queries. All indexes must be made in advance. The DBA must attempt to guess which columns might be constrained. Moreover, the order in which the columns are positioned determines whether they are utilized or not. The result is the DBA team must make many, many compound B-tree indexes, many containing the same columns in different orders.
* **Bitmap indexes**. Bitmap indexes function completely different from B-tree indexes. Bitmap indexes are better suited for lower cardinality columns. Many single column bitmap indexes can dynamically join together to create necessary compound indexes to support ad-hoc queries. Because of their flexibility, it is common practice to create single-column bitmap indexes on each surrogate key in fact tables in the data warehouse.

Now we want to turn back to how indexes affect DML transactions. Every entry of a B-tree index contains one, and only one, rownum that points back to the corresponding record in the base table. Conversely, bitmap indexes contain a range of rownums for each value in the index. If a value is updated, every record that corresponds to the range of rownums is locked. Ultimately, each time a record is updated, a tremendous burden is put on the database to manage all of the row locking going on. Unfortunately, fact tables usually consist only of bitmap indexes, and doing massive updates creates massive headaches because of the excessive overhead and extremely poor performance.

In conclusion, it's recommended to drop all bitmap indexes before you begin to manipulate data in fact tables. It's further recommended to partition fact tables so you have to drop only the local index in the current partition (assuming the table is partitioned on the date key). B-tree indexes are not excluded from being dropped before the ETL process executes. Statistics show that in most cases, dropping data warehouse indexes, loading the tables, and rebuilding the indexes is substantially faster than loading tables with scores of indexes enabled.

Addressing Constraints and Foreign Keys

Foreign keys in relational DBMSs enforce integrity in data between tables. For example, you cannot enter an order status unless that status is an existing valid value in the order status table. But in the data warehouse, the transaction has already occurred and has been validated by the source system.

In a nutshell, all foreign keys and constraints should be disabled in the data warehouse, especially during the ETL process. Convincing the DBA team that it is okay to drop foreign keys can be a political challenge. You must walk through the ETL process with the DBA team and explain that fact records simply cannot exist without getting the surrogate keys from their associated dimensions. Also, point out that dimension natural keys are looked up to ensure they are not inserted more than once. Once the DBA team is convinced that the ETL is truly a controlled and managed environment, they realize that database constraints and foreign keys are superfluous elements that slow down the ETL process without offering much benefit.

The fastest way to load data into the data warehouse is to enable the database to bulk load it by following these four database-preparation steps:

* Eliminate as many DML statements as possible.
* Disable rollback logging.
* Drop all existing indexes.
* Eliminate database foreign keys and constraints.

Once the four steps are complete, you are ready to utilize the database bulk-load utility. A guide for utilizing bulk loaders can be found in [Chapter 7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html).

ETL System Security

Database security in the data-staging area is much simpler to enforce than in the data warehouse presentation database. Generally speaking, no one is allowed to read or write to the data-staging area besides the ETL engine or program. Most databases utilize roles and users to enforce security at the database level. A *role* is a mechanism that allows a security administrator to group together many users that require the same database-access privileges. Each user has his or her own userID and authentication mechanism. When the user is authenticated to the system, he or she is handed the privileges associated with his or her role.

Without roles, a security administrator would have to explicitly grant every person appropriate privileges individually. A user is an individual that uses the database.

Typically, it is sufficient to create a single data warehouse administrative role with the following privileges:

* Select, Insert, Update, and Delete all objects
* TRUNCATE TABLE
* Utilize bulk loader
* Drop and create indexes

If there is highly sensitive data, such as compensation rates or sales leads, the data should be encrypted by the source system before it is extracted by the ETL team. Normally, column-level security is handled by the use of database views that sit on top of a table and conceal the sensitive columns. However, the ETL team must have the ability to select data from the source system and also be able to perform any DML required in the data warehouse. Therefore, views are not an appropriate mechanism for hiding sensitive data. Furthermore, the ETL team should not be responsible for encrypting sensitive data. Securing or encrypting sensitive data from the ETL team is the responsibility of the source system security administrator.

Securing the Development Environment

In the development environment, everyone on the ETL team is granted the privileges of the DWETL role (all DML and TRUNCATE on all objects and so forth). This is where all staging tables are created. Even though the data-staging area is owned by the ETL team, sometimes table creation is controlled by the data warehouse architect or DBA. In some cases the ETL architect has the authority to create tables in the data-staging area without further approval.

Furthermore, any ETL team member can add, delete, or modify files in the dedicated directories on the file system. No one outside ETL team should have access to data-staging environments.

Securing the Production Environment

The ETL team typically has read-only access to production. Sometimes, in very secure environments, such as banking, they have no access at all. The DWETL role exists in production, but the only the user ID and password used by the ETL engine is created in production. If for some extreme reason, an ETL team member must have write access to production, such as to fix a bug that exists only in production, it should be granted temporarily and revoked as soon as the fix is complete.

FTP Vulnerabilities

You must lock out everyone from the FTP *inbox*. Only the FTP process can write or delete within the specified directory. Moreover, only the ETL engine is allowed to read the directory. If there is an emergency situation, temporary access should be granted to a predetermined administrator whose privilege is revoked as soon as the issue is resolved.

Encryption/Decryption

The biggest performance inhibitor is writing to and reading from disk (I/O). However, you must encrypt data in movement for security purposes. Usually, the following steps are followed to secure data in movement:

1. Encrypt data and store encrypted data on disk.
2. Read and transfer encrypted data across networks.
3. Store encrypted data on data-staging server.
4. Decrypt data and store on disk.
5. Transform and load decrypted data into data warehouse.

It may be possible to reduce this to three steps by encrypting the source data as it is read into memory, then transferring the encrypted data, and finally decrypting the data as it enters the transformation and load routines.

As you can see, the second solution does not touch the disk from the time the data is read until it is loaded into the data warehouse. This process is known as *in-stream decryption*. Some ETL tools support in-stream decryption functionality. If you need to encrypt or decrypt in-stream to improve performance, make sure you select a tool that supports the functionality; otherwise, you can write your own applet in Java. In the case of the Java applet, make sure that your ETL toll can at least embed an external process, such as the in-stream decryption applet, without incurring more I/O.

Short-Term Archiving and Recovery

There are many reasons to keep the various data-staging results from the ETL system. In this book, we have identified needs for short-term restart capabilities, comparisons of day-to-day extracts to detect differences when you don't have a proper change data capture system, as well as legal and financial auditing requirements. All of these archiving and recovery scenarios should be familiar challenges to the IT staff. Back the data up to your current media and make sure it can be recovered. Make sure you have a believable audit trail that accounts for all accesses and alterations to the data. Make sure the data is physically secure, and protect the archived data as strongly as you protect the on line data.

But what if we are supposed to keep the data for many years?

Long-Term Archiving and Recovery

One of the oaths we take as data warehouse managers is that we will *preserve history*. In many ways, we have become the archivists of corporate information. We don't usually promise to keep all history on-line, but we often claim that we will store it somewhere for safekeeping. Of course, storing it for safekeeping means that we will be able to get history back out again when someone is interested in looking at it.

Most of us data warehouse managers have been so busy bringing up data warehouses, avoiding stovepipe data marts, adapting to new database technologies, and adapting to the explosive demands of the Web that we have relegated our archiving duties to backing up data on tapes and then forgetting about the tapes. Or maybe we are still appending data onto our original fact tables and we haven't really faced what to do with old data yet.

But across the computer industry there is a growing awareness that preservation of digital data is not being done yet, and that it is a serious problem and a hard problem.

*Does a Warehouse Even Need To Keep Old Data?*

Most data warehouse managers are driven by urgent needs of departments like marketing, who have very tactical concerns. Few marketing departments care about data that is more than three years old, because our products and our markets are changing so quickly. It is tempting to think only of these marketing clients and to discard data that no longer meets their needs.

But with a little reflection we realize we are sitting on a lot of other data in our warehouses that absolutely must be preserved. This data includes:

* Detailed sales records, for legal, financial, and tax purposes
* Trended survey data where long-term tracking has strategic value
* All records required for government regulatory or compliance tracking
* Medical records that in some cases must be preserved for 100 years!
* Clinical trials and experimental results that may support patent claims
* Documentation of toxic waste disposal, fuel deliveries, and safety inspections
* All other data that may have historical value to someone, sometime

Faced with this list, we have to admit that a plan is needed for retrieving these kinds of data five, ten, or maybe even 50 years in the future. It begins to dawn on us that maybe this will be a challenge. How long do mag tapes last, anyway? Are CD-ROMs or DVDs the answer? Will we be able to read the formats in the future? We have some eight-inch floppies from just a few years ago that are absolutely unrecoverable and worthless. All of a sudden, this is sounding like a difficult project.

Media, Formats, Software, and Hardware

As we begin to think about really long-term preservation of digital data, our world begins to fall apart. Let's start with the storage media. There is considerable disagreement about the practical longevity of physical media like mag tapes and CD-ROM disks, with serious estimates ranging from as little as five years to many decades. But, of course, our media may not be of archival quality, and they may not be stored or handled in an optimum way. We must counterbalance the optimism of vendors and certain experts with the pragmatic admission that most of the tapes and physical media we have today that are more than ten years old are of doubtful integrity.

Any debates about the physical viability of the media, however, pale when compared to the debates about formats, software, and hardware. All data objects are encoded on physical media in the *format of the day*. Everything from the density of the bits on the media, to the arrangement of directories, and finally to the higher-level application-specific encoding of the data is a stack of cards waiting to fall. Taking our eight-inch floppies as examples, what would it take to read the embedded data? Well, it would take a hardware configuration sporting a working eight-inch floppy drive, the software drivers for an eight-inch drive, and the application that originally wrote the data to the file.

Obsolete Formats and Archaic Formats

In the lexicon of digital preservationists, an obsolete format is no longer actively supported, but there is still working hardware and software extant that can read and display the content of the data in its original form. An archaic format has passed on to the nether realm. Our eight-inch floppies are, as far as we are concerned, an archaic format. We will never recover their data. The Phoenician writing system known as Linear A is also an archaic format that has apparently been lost forever. Our floppies may be only slightly easier to decipher than Linear A.

Hard Copy, Standards, and Museums

A number of proposals have been made to work around the format difficulties of recovering old data. One simple proposal is to reduce everything to hard copy. In other words, print all your data onto paper. Surely, this will side step all the issues of data formats, software, and hardware. While for tiny amounts of data this has a certain appeal, and is better than losing the data, this approach has a number of fatal flaws. In today's world, copying to paper doesn't scale. A gigabyte printed out as ASCII characters would take 250,000 printed pages at 4000 characters per page. A terabyte would require 250,000,000 pages! Remember that we can't cheat and put the paper on a CD-ROM or a mag tape, because that would just reintroduce the digital format problem. And finally, we would be seriously compromising the data structures, the user interfaces, and the behavior of the systems originally meant to present and interpret the data. In many cases, a paper backup would destroy the usability of the data.

A second proposal is to establish standards for the representation and storage of data that would guarantee that everything can be represented in permanently readable formats. In the data warehouse world, the only data that remotely approaches such a standard is relational data stored in an ANSI-standard format. But almost all implementations of relational databases use significant extensions of the data types, SQL syntax, and surrounding metadata to provide needed functionality. By the time we have dumped a database with all its applications and metadata onto a mag tape, even if it has come from Oracle or DB2, we can't be very confident that we will be able to read and use such data in thirty years or fifty years. Other data outside of the narrow ANSI-standard RDBMS definition is hopelessly fragmented. There is no visible market segment, for instance, that is coalescing all possible OLAP data storage mechanisms into a single physical standard that guarantees lossless transfer to and from the standard format.

A final somewhat nostalgic proposal is to support museums, where ancient versions of hardware, operating systems, and applications software would be lovingly preserved so that old data could be read. This proposal at least gets to the heart of the issue in recognizing that the old software must really be present in order to interpret the old data. But the museum idea doesn't scale and doesn't hold up to close scrutiny. How are we going to keep a Digital Data Whack 9000 working for 50 years? What happens when the last one dies? And if the person walking in with the old data has moved the data to a modern medium like a DVD ROM, how would a working Digital Data Whack 9000 interface to the DVD? Is someone going to write modern drivers for ancient pathetic machines? Maybe it has an eight-bit bus.

Refreshing, Migrating, Emulating, and Encapsulating

A number of experts have suggested that an IT organization should periodically refresh the storage of data by moving the data physically from old media onto new media. A more aggressive version of refreshing is migrating, where the data is not only physically transferred but is reformatted in order to be read by contemporary applications. Refreshing and migrating do indeed solve some of the short-term preservation crises because if you successfully refresh and migrate, you are free from the problems of old media and old formats. But taking a longer view, these approaches have at least two very serious problems. First, migrating is a labor-intensive, custom task that has little leverage from job to job and may involve the loss of original functionality. Second, and more serious, migrating cannot handle major paradigm shifts. We all expect to migrate from version 8 of an RDBMS to version 9, but what happens when the world is taken over by heteroschedastic database systems (HDS's)? The fact that nobody, including us, knows what an HDS is, illustrates our point. After all, we didn't migrate very many databases when the paradigm shifted from network to relational databases, did we?

Well, we have managed to paint a pretty bleak picture. Given all this, what hope do the experts have for long-term digital preservation? If you are interested in this topic and a serious architecture for preserving your digital data warehouse archives for the next 50 years, you should read Jeff Rothenberg's treatise *Avoiding Technological Quicksand, Finding a Viable Technical Foundation for Digital Preservation*. This is a report to the Council on Library and Information Resources (CLIR). The 41-page report can be retrieved as an Adobe PDF file by linking to www.clir.org/pubs/reports/rothenberg. Very well written and very highly recommended.

As a hint of where Jeff goes with this topic, he recommends the development of emulation systems that, although they run on modern hardware and software, nevertheless faithfully emulate old hardware. He chooses the hardware level for emulation because hardware emulation is a proven technique for recreating old systems, even ones as gnarly as electronic games. He also describes the need to encapsulate the old data sets along with the metadata needed for interpreting the data set, as well as the overall specifications for the emulation itself. By keeping all these together in one encapsulated package, the data will travel along into the future with everything that is needed to play it back out again in 50 years. All you need to do is interpret the emulation specifications on your contemporary hardware.

The library world is deeply committed to solving the digital-preservation problem. Look up *embrittled documents* on the Google search engine. Their techniques need to be studied and adapted to our warehouse needs.

Summary

In this chapter, we have provided an overview of the operations framework of typical ETL systems. The first half of the chapter was devoted to major scheduling approaches. Most of the second half dealt with managing performance issues as your system grows and gets more complicated. Finally, we proposed a simple framework for ETL system security.

Chapter 9. Metadata

Metadata is an interesting topic because every tool space in the data warehouse arena including business intelligence (BI) tools, ETL tools, databases, and dedicated repositories claims to have a metadata solution, and many books are available to advise you on the best metadata strategies. Yet, after years of implementing and reviewing data warehouses, we've yet to encounter a true end-to-end metadata solution. Instead, most data warehouses have manually maintained pieces of metadata that separately exist across their components. Instead of adding to the metadata hoopla, this chapter simply covers the portions of metadata that the ETL team needs to be aware of—either as a consumer or a producer. We propose a set of metadata structures that you need to support the ETL team.

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → *Architecture* → *Implementation* → Release to Ops

Data Flow: *Extract* → *Clean* → *Conform* → *Deliver*

Because the ETL system is the center of your data warehouse universe, it often assumes the responsibility of managing and storing much of the metadata for the data warehouse. One might think that there is no better place than the ETL system for storing and managing metadata because the environment must already know the specifics of all data to function properly. And the ETL process is the creator of the most important metadata in the data warehouse—the data lineage. The *data lineage*traces data from its exact location in the source system and documents precisely what transformation is done to it before it is finally loaded. The data lineage includes the data definition of the source system database and also of the final resting place in the data warehouse. If you are using an ETL tool, attributes other than the data lineage can also live in the ETL environment. But can the ETL environment really capture and manage *all* metadata in the data warehouse? No way.

If you'd like to explore data warehouse metadata in more detail, a noteworthy text is *Metadata Solutions: Using Metamodels, Repositories, XML, and Enterprise Portals to Generate Information on Demand* by Adrienne Tannenbaum (Addison Wesley 2002). We want to start by trying to define exactly what is meant by *metadata*.

Defining Metadata

A leading cause for the difficulty behind metadata implementations is that the exact definition of metadata is ambiguous, and defining exactly what metadata is a very difficult task. In 1998 we tackled the definition of metadata in the *Data Warehouse Lifecycle Toolkit* book. In looking over those words, we still find them surprisingly relevant. Here's what we wrote:

Metadata—What Is It?

Metadata is an amazing topic in the data warehouse world. Considering that we don't know exactly what it is, or where it is, we spend more time talking about it, more time worrying about it, and more time feeling guilty that we aren't doing anything about it. Several years ago, we decided that metadata is any data about data. This wasn't very helpful, because it didn't paint a clear picture in our minds as to what exactly this darn stuff was. This fuzzy view gradually clarified and recently we have been talking more confidently about the *back room metadata* and *front room metadata*. Back room metadata is process related and guides the extraction, cleaning, and loading processes. Front room metadata is more descriptive and helps make our query tools and report writers function smoothly. Of course, process and descriptive metadata overlap, but it is useful to think about them separately.

Back room metadata presumably helps the DBA bring the data into the warehouse and is probably also of interest to business users when they ask where the data comes from. Front room metadata is mostly for the benefit of end user, and its definition has been expanded to not only be the oil that makes our tools function smoothly but a kind of dictionary of business content represented by all data elements.

Even these definitions, as helpful as they are, fail to give the data warehouse manager much of a feeling for what he or she is supposed to do. But one can apply a traditional IT perspective to metadata. At the very least, we should:

* Make a nice annotated list of all of it
* Decide just how important each part is
* Take responsibility for it or assign that responsibility to someone else
* Decide what constitutes a consistent and working set of it
* Decide whether to make it or buy it
* Store it somewhere for backup and recovery
* Make it available to people who need it
* Quality-assure it and make it complete and up to date
* Control it from one place
* Document all of these responsibilities well enough to hand this job off (soon)

The only trouble is that we haven't really said what metadata is yet. We do notice that the last item in the preceding list really isn't metadata; it's data about metadata. With a sinking feeling, we realize we probably need *meta meta data data*.

Source System Metadata

To understand this better, let's try to make a complete list of all possible types of metadata. We surely won't succeed at this first try, but we will learn a lot. First, let's go to the source systems, which could be mainframes, separate nonmainframe servers, users' desktops, third-party data providers, or even on-line sources. We will assume that all we do here is read source data and extract it to a data-staging area that could be on the mainframe or could be a downstream machine.

Source specifications:

* Repositories
* Source schemas
* Copy books
* Proprietary or third-party source schemas
* Print spool file sources
* Old format for archived mainframe data
* Relational source system tables and DDL
* Spreadsheet sources
* Lotus Notes databases
* Presentation graphics (for example, PowerPoint)
* URL source specifications:
* Source Descriptive Information:
  + Ownership descriptions of each source
  + Business descriptions of each source
  + Update frequencies of original sources
  + Legal limitations on the use of each source
  + Access methods, access rights privileges, and passwords for source access
* Process Information:
  + Mainframe or source system job schedules
  + The COBOL/JCL or C or Basic or other code to implement extraction
  + The automated extract-tool settings if we use such a tool
  + Results of specific extract jobs, including exact times content and completeness

Data-Staging Metadata

Now let's list all the metadata needed to get data into a data-staging area and prepare it for loading into one or more data marts. We may do this on the mainframe with hand-coded COBOL or by using an automated extract tool. Or we may bring the flat file extracts more or less untouched into a separate data-staging area on a different machine. In any case, we have to be concerned about metadata describing

Data Acquisition Information:

* Data transmission scheduling and results of specific transmissions
* File usage in the data-staging area, including duration volatility and ownership

Dimension Table Management:

* Definitions of conformed dimensions and conformed facts
* Job specifications for joining sources, stripping out fields, and looking up attributes
* Slowly changing dimension policies for each incoming descriptive attribute (for example, overwrite create new record or create new field)
* Current surrogate key assignments for each production key, including a fast lookup table to perform this mapping in memory
* Yesterday's copy of a production dimension to use as the basis for Diff Compare

Transformation and Aggregation:

* Data-cleaning specifications
* Data enhancement and mapping transformations (for example, expand abbreviations and provide detail)
* Transformations required for data mining (for example, interpret nulls and scale numerics)
* Target schema designs, source to target data flows, and target data ownership
* DBMS load scripts
* Aggregate definitions
* Aggregate usage statistics, base table usage statistics, and potential aggregates
* Aggregate modification logs

Audit, Job Logs, and Documentation:

* Data lineage and audit records (where *EXACTLY* did this record come from and when)
* Data transform run time logs, success summaries, and time stamps
* Data transform software version numbers
* Business descriptions of extract processing
* Security settings for extract files extract software and extract metadata
* Security settings for data transmission (that is, passwords certificates)
* Data-staging area archive logs and recovery procedures
* Data-staging archive security settings.

DBMS Metadata

Once we have finally transferred data to the data warehouse or data mart DBMS, another set of metadata comes into play:

* DBMS system table contents
* Partition settings
* Indexes
* Disk-striping specifications
* Processing hints
* DBMS-level security privileges and grants
* View definitions
* Stored procedures and SQL administrative scripts
* DBMS backup, status-backup procedures and backup security.

Front Room Metadata

In the front room, we have metadata extending to the horizon, including

* Business names and descriptions for columns tables groupings and so on
* Precanned query and report definitions
* Join specification tool settings
* Pretty print tool specifications (for relabeling fields in readable ways)
* End user documentation and training aids, both vendor supplied and IT supplied
* Network security user privilege profiles
* Network security authentication certificates
* Network security usage statistics, including log on attempts access attempts and user ID by location reports
* Individual user profiles with link to human resources to track promotions transfers resignations that affect access rights
* Link to contractor and partner tracking where access rights are affected
* Usage and access maps for data elements, tables, and views reports
* Resource charge-back statistics
* Favorite Web sites (as a paradigm for all data warehouse access)

Now we can see why we didn't know exactly what metadata was all about. It is everything, except for the data itself. All of a sudden, data seems like the simplest part. In a sense, metadata is the DNA of the data warehouse. It defines all elements and how they work together.

So, how do you capture and manage these forms of metadata? You don't. At least the ETL team doesn't. Over the past few decades, consortiums, alliances, committees, organizations, and coalitions have been formed to solve the metadata quandary. To this day, no universal solution exists. We have found that as an ETL team member you need certain metadata to do your job, and that it is convenient to focus on selected items from the preceding list and organize them into three major categories:

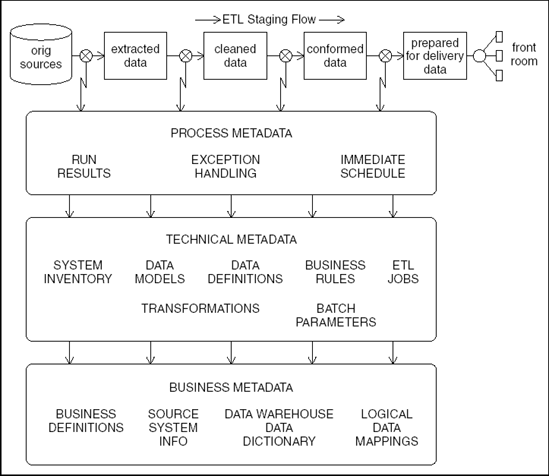
1. **Business metadata**. Describing the meaning of data in a business sense
2. **Technical metadata**. Representing the technical aspects of data, including attributes such as data types, lengths, lineage, results from data profiling, and so on
3. **Process execution metadata**. Presenting statistics on the results of running the ETL process itself, including measures such as rows loaded successfully, rows rejected, amount of time to load, and so on

In addition to the three categories of metadata, you need to consider another aspect of metadata: standards. Standards are another attempt by IT to make work throughout your organization consistent and maintainable. In [Chapter 10](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html), we define the role of the metadata librarian and propose a set of responsibilities for this person. Your organization most likely has many standards in place, and you can adopt specific data warehouse and ETL standards from the recommendations found throughout this book. If you seek to further investigate metadata standards, we've provided sources for your perusal in the next section.

As you read this chapter, please refer to [Figure 9.1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch09.html#metadata_sources_in_the_back_room_of_the), showing the three main categories of ETL system metadata, together with each of the separate metadata tables discussed in the text.

Here is a comprehensive list of the places so far in this book where we have urged you to collect and use metadata:

* Origins and processing steps for each staged data set ([Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html))
* Metadata repository as an advantage of the vendor-supplied ETL tool ([Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html))
* Need for metadata architecture: source tables, cleaning, and processes ([Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html))
* Presenting useful metadata to end users ([Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html))
* Extract transformations applied ([Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html))
* Compliance metadata ([Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html))
* XML data descriptions of metadata ([Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html))



**Figure 9.1. Metadata sources in the back room of the data warehouse**

* Lack of metadata in flat files ([Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html))
* Impact analysis metadata ([Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html))
* Planning for building metadata describing lineage, business definitions, technical definitions, and processes ([Chapter 2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch02.html))
* Logical data map ([Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html))
* Capturing calculations for derived data during extract ([Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html))
* Source database descriptions ([Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html))
* ETL tools reading ERP system metadata ([Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html))
* Results of data profiling ([Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html))
* Error event tracking fact table ([Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html))
* Audit dimension ([Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html))
* Column survivorship (conforming results) table ([Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html))
* Surrogate key highest value ([Chapter 5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch05.html))
* Aggregate navigator data ([Chapter 6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html))
* Process data to kick off OLAP cube building ([Chapter 6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch06.html))
* Bulk loader control file ([Chapter 7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch07.html))
* Metadata supporting recovery processes ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))
* Job schedule metadata ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))
* Database connection information ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))
* ETL system parameters ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))
* Job dependencies ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))
* Job operational statistics such as performance and resource use ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))
* MetaData repository reporting ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))
* Table purge policies ([Chapter 8](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch08.html))

We hope you aren't too dismayed by these long lists. The rest of this chapter makes a serious attempt to propose specific metadata structures for tracking this kind of information!

Business Metadata

The assignment of who is responsible for business metadata in the data warehouse is often argued. Some say it is the responsibility of the data warehouse business analyst and should be created during the requirementsgathering process. Others believe the source system business analyst should create business terms because most data warehouse attributes originate in the source systems. Yet others think it is part of the data modeler's tasks to create and maintain business metadata because it is a natural part of the logical data model.

It's not up to you to settle those arguments, but there is some business metadata that the ETL team influences and must be maintained to accurately reflect reality. The ETL team should not be responsible for generating business metadata but must communicate changes that need to be applied to the appropriate personnel. From the ETL perspective, business metadata is *proxy* metadata. Proxy metadata is obtained from one system and made available to another without any explicit manipulation. Some business-intelligence tools are designed to look into ETL repositories to get business definitions to present to its users—making the tool a one-stop shop for data and data's metadata.

To keep things interesting, we must make you aware that the data warehouse can be in a situation where different business definitions are defined for the same attribute. Remember, by design, that data in the data warehouse is sourced from multiple systems, and each system can potentially have a different definition for the same attribute. For example, the marketing department defines a customer as anyone who has a registered account with the company, while the sales department might consider only persons who have actually made purchases to be customers. As a recommended practice, the data warehouse manager should bring all of the owners of an overlapping element into a room and have them agree on a single definition from an *enterprise* standpoint. We describe this process throughout the book as *conforming* business definitions, labels, and measures of the enterprise. That meaning might be from one of the source systems or an entirely new one. The enterprise definition is stored in the data warehouse and in ETL tool.

NOTE

The ETL team is part of the back room of the data warehouse and should not get involved in creating business metadata. However, you should understand the purpose of the data you are working with and review business definitions as you need them.

Business Definitions

Often, the data warehouse team gets caught up in designing the architectural database for query performance and optimizing the ETL process for load performance. Once the database is designed and loaded, the team concentrates on the business intelligence of the data warehouse, making elegant user interfaces and slick graphical reports. But to the business, the most important ingredient in the data warehouse recipe is the definition of the elements available to them. If the user cannot make sense of data, as sophisticated as it may be, it has absolutely no value to the organization.

Business definitions are a crucial requirement to the completion of the data warehouse. Not only are end users dependent on business definitions; the ETL team needs business definitions to give context to the data they are loading. You will notice how important the business definitions are to the ETL team if you attempt to present them with the data model and data lineage before the business definitions are complete. If you rush to complete the data lineage so the ETL development can begin, you'll be forced to continually explain the purpose of the main data elements in the data model to the ETL team before they have enough information to work on them. Therefore, shift the development priorities away from expediting the data lineage and to concentrating on the association of the business definitions with the data warehouse data elements before the data lineage is handed off to the ETL team. A simple business definition matrix includes three main components:

* **Physical table and column name**. The business interpretation of data elements in the data warehouse is based on the actual table and column names in the database. The physical names need not be presented to end users if the BI tool presents only the business names and completely hides the physical instance of the data structures. However, the ETL team deals only with physical names and needs this information to associate the business definitions to the appropriate data elements.
* **Business column name**. The database stores data elements in technical terms consisting of prefixes, suffixes, and underscores. The business community needs a translation of the technical names to names that make sense and provide context. For example, the business name for EMP STTS CD might be Employee Status Code or just Employee Status. We discourage the use of cryptic names. Remember, the business name is how the column is represented in the BI tool. Furthermore, the business name often becomes column and row headings in user reports.
* **Business definition**. A business definition is one or two sentences that describe the business meaning of an attribute. Each attribute in the data warehouse must have a business definition associated with it. If the business cannot define the attribute, it usually indicates that the attribute has no analytic value and probably does not need to be stored in the data warehouse. If the business demands that the attribute remain in the data warehouse, a business definition must be defined for it.

The business definition matrix can be as simple as a three-column spreadsheet. However, you should strive to make this particular metadata as centralized and sharable as your technical environment allows. Virtually all major ETL tools support capturing and storing business metadata. The ETL tool should work with your data-modeling tool and database to obtain business definitions and with your BI tool to present business names and definitions to your end users.

Source System Information

The ETL team needs to know every intimate detail of each table it accesses for the data warehouse. Imagine that you need to fill an open position on your team. Before you hire someone, you have your HR department find candidates, prescreen them, introduce them to you, and arrange for formal interviews so you can ensure they are the right fit for the position. During the interviews, you identify any weaknesses that might need some work before candidates are deemed satisfactory for the position.

When you populate a data warehouse, the data modeler creates the position; the data warehouse architect finds and prescreens the source. Then data must be analyzed thoroughly to identify its weaknesses, and a plan of action must be initiated to make data satisfactory enough to be included in the data warehouse. Some data will be perfect; other data elements will need some transformation or be rejected because of inadequate quality.

When you analyze source systems, you need certain pieces of metadata. At a minimum, you need the following metadata attributes:

* **Database or file system**. The name commonly used when referring to a source system or file. This is not the technical server or database instance. Names such as *Sales database* or *Inventory Management System* are common values for this piece of metadata.
* **Table specifications**. The ETL team needs to know the purpose of the table, its volume, its primary key and alternate key, and a list of its columns.
* **Exception-handling rules**. You must be informed of any data-quality issues and advised on how they should be handled by the ETL process.
* **Business definitions**. The infamous yet rarely available business definitions. Do your best to have these provided to you. These one-to-two-sentence definitions are invaluable when you are trying to make sense of data.
* **Business rules**. Every table should come with a set of business rules. Business rules are required to understand the data and to test for anomalies. Every business rule must be tested, and each exception to a rule must be documented and resolved by the source system or the ETL process. Forms of business rules include an account of when a table receives new rows, updates, or deletions. If you are lucky, business rules are enforced in the source database management system (DBMS) by way of referential integrity, check constraints, and triggers.

Investigating source systems takes a substantial amount of time during the data-analysis phase of the data warehouse project. A lack of source system metadata incurs excessive research and troubleshooting by the data warehouse team. To curb cost overruns, all source system metadata must be available to the ETL team before the development of any ETL process.

Data Warehouse Data Dictionary

When we refer to the data dictionary, we are not referring to the DBMS catalog. The data warehouse data dictionary is a list of all of the data elements in the data warehouse and their business descriptions. Similar to the source system business definitions, the data warehouse data dictionary contains the physical table and column names and the business names and definitions. In the case of data warehouse business metadata, spreadsheets are insufficient. Many data warehouse environments depend on the ETL repository to store the data dictionary because BI tools are designed to look there to obtain metadata for presentation.

NOTE

Many BI tools are designed to work cohesively with ETL metadata repositories. When you are selecting your toolset, make sure that your ETL tool has an open repository that can be read by any query tool or at least comes with an adapter or broker for this purpose.

Logical Data Maps

The logical data map is the lifeline of the ETL team. Read [Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html) for detailed information about logical data maps from an ETL work-specification perspective. From a metadata perspective, the logical data map consists of the source-to-target mapping that explains, logically, exactly what happens to the data from the moment it is extracted from its origin to when it is loaded into the data warehouse.

The logical data map is a crucial piece of metadata. The ETL team first uses the document as a functional specification to create the physical ETL jobs, then again to validate the work with the end users. The document provides guidance when questions arise during user-acceptance testing (UAT). It is also used when data goes through quality-assurance (QA) testing, when the ETL team provides a walkthrough of each mapping with the QA team. Finally, the ETL team reviews the document with the DBA team to provide information about the data transformations so they can support the processes in the event of a failure.

NOTE

CROSS-REFERENCE Refer to [Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html) for the exact metadata elements created in the logical data map and recommendations on how to create, maintain, and utilize the information.

Technical Metadata

Technical metadata serves many purposes and is the most interesting type of metadata to the ETL team. Technical metadata can encompass everything from column names and data types to storage and the configuration of your RAID arrays. As a member of the ETL team, you needn't be too concerned with hardware configurations and the like. Most interesting are the physical attributes of the data elements in all databases involved in the data warehouse. To get essential metadata, seek out the physical data model of each of your source systems and the target data warehouse. If your sources include flat files, you need the file layouts for each file you will be working with.

System Inventory

The ETL team must have access to and thoroughly understand the technical metadata of each system in the data warehouse environment to accurately build the physical ETL jobs. The technical definition of data is probably what technicians think of first when they are asked about metadata. After all, that really is data about data—it is the container and the structure of the data. The ETL team must be aware of data definitions for at least three environments:

* Source databases
* Staging-area tables (for example, extracted, cleaned, conformed, prepared for delivery)
* Data warehouse presentation area

It is possible that an entity relationship diagram is provided for each environment. At a minimum, a listing that includes the following elements for each system is required.

* **Tables**. An exhaustive list of tables or files that are—or might be—required during the extract and load process. Often, only source system tables in the logical data mapping are provided. Yet, there are *associative* tables that are usually not specified but are required. Whenever the source system has many-to-many relationships, a well-designed system has associative tables to enable the relationship.
* **Columns**. For each table, you will need a list of the required columns for your data mapping. Hopefully, the source system DBA can provide a list of only the columns that you need. It works best if the column listing excludes unnecessary columns.
* **Data types**. Each column is defined with a data type. Data types vary among different database systems. Luckily, most dedicated ETL tools implicitly convert equivalent data types. For example, an INTEGER from SQL Server automatically becomes a NUMBER when it is loaded into Oracle. Be aware that a DBA can define custom data types known as User Defined Datatypes. User Defined Datatypes are based on the database's core data types but can extend their definition to include things such as column length, whether the data type can accept NULL values, or whether the data type contains special characters, for example, telephone numbers.
* **Relationships**. Relational databases are designed to support referential integrity (RI). RI is enforced by relationships designed between tables to ensure unique and consistent data entry. Relationships are outlined in data models by linking foreign keys (FK) in a table to primary keys (PK) in another.

Data Models

Data models in the form of physical schema diagrams (either normalized or dimensional) are really just a graphical display of metadata but not particularly metadata itself. However, they can be an invaluable asset to the ETL team arsenal because they enable you to quickly identify and confirm relationships at a glance. Even though table joins are depicted in logical data mapping, you cannot count on it being perfectly complete. We recommend that you hang the physical schema diagrams from all of the source systems (and the data warehouse) on the wall of your office. Having physical schema diagrams hung around you alleviates sifting through rolls of diagrams piled in a corner. Furthermore, the hung diagrams make an instant reference point that expedites the resolution of questions and issues—not to mention the positive impression it makes on managers!

Data Definitions

Data definitions must be consistent between each of their potential data stores. Each time your data *touches down* to a database or file, it is vulnerable to data truncation or corruption. If data definitions are not alike between environments, the ETL team must explicitly convert the data to avoid catastrophe. In addition to the attributes listed in the previous section, the following data-definition metadata elements must also be supplied to the ETL team.

* **Table name**. The physical name of the table or file that contains the data
* **Column name**. The physical name of the column within a table or file
* **Data type**. Data elements with a table are categorized into types. Common data types are Numeric, Character, Date, and Binary. You will find that custom data types are allowed in most DBMSs. Custom data types are based on common types and usually enforce formatting rules. Data types are mutually exclusive and cannot coexist within a single column.
* **Domain**. The set of values allowed to be entered into a column is known as its domain. The domain can be enforced by a foreign key, check constraints, or the application on top of the database. If the application enforces the domain, a list of allowed values must be provided by the programming team to the ETL team.
* **Referential integrity**. If your data comes from a database, you will most likely find foreign keys that point to a primary key of another table to ensure the data is unique. If referential integrity (RI) is enforced at the application level, the programming team must provide the RI rules. Typically, RI in the data warehouse is deemed unnecessary overhead because all data is entered in a controlled fashion—the ETL process—and does not have RI enforced at the database level.
* **Constraints**. Constraints are another form of a physical implementation of a business rule. Database constraints can eliminate NULL values, enforce foreign key look-ups, ensure compliance with allowed values, and so on.
* **Defaults**. A default in the context of ETL metadata is the assignment of a string, number, date, or bit, in the case when the actual value is not available. In your source system, column defaults are usually assigned at the database level. In the data warehouse, the defaults should be assigned in the ETL process. It is recommended that defaults in the data warehouse are used consistently.
* **Stored procedures**. Stored procedures, which store prewritten SQL programs in the database, offer great insight to how your source data is used. Each data warehouse project inevitably involves analyzing stored procedures that exist in the source systems.
* **Triggers**. A trigger is an SQL procedure automatically executed by the DBMS when a record is added, deleted, or updated. Like stored procedures, triggers offer information about how data is used. Triggers often enhance foreign key constraints by adding additional checks to records added to a table. Triggers also load audit tables when records are altered or deleted from a table. Audit tables loaded by triggers are a vital source of deleted data for the data warehouse.

Business Rules

Business rules can be categorized as business or technical metadata. We like to refer to business rules as *technical* because they are the essence of the ETL process—which is very technical. Each and every business rule must be coded in the ETL process. Business rules can include anything from allowed values to default values to calculations for derived fields. In source systems, business rules are enforced by stored procedures, constraints, or database triggers. But most often, business rules exist only in the application code. In older, especially mainframe, environments, the actual source code no longer exists and only the compiled portion of the application remains. In those cases, business rules are extremely difficult to obtain. It usually requires interviewing data analysts and programmers who support the application. The metadata for business rules varies between functional or technical documentation and source code in the native programming language of the application or pseudocode.

Business rules must be incorporated into logical data mapping. Sometimes business rules are omitted from logical data mapping and go unnoticed until the first attempt at the ETL process is complete and the exclusions are detected by users during UAT. As new business rules are learned, the metadata in the logical data mapping must be updated to reflect the new rules.

ETL-Generated Metadata

So far in this chapter, we have focused on metadata created outside of the ETL environment and provided to the ETL team from other sources. The remainder of this chapter addresses metadata generated by the ETL team and used either within the team to manage the ETL processes or by end users or other data warehouse members to better understand the data within the data warehouse.

As ETL physical processes are built, specific metadata must be generated to capture the inner workings of each process. ETL metadata falls into four main categories:

* **ETL job metadata**. The ETL job is a container that stores all of the transformations that actually manipulate the data. The job metadata is very valuable because it contains the data lineage of the elements in the data warehouse. Every ETL task—from extraction to load and all transformations in between—is captured in the ETL job metadata.
* **Transformation metadata**. Each job consists of many transformations. Any form of data manipulation within a job is performed by a dedicated transformation.
* **Batch metadata**. Batching is a technique used to run a collection of jobs together. Batches should have the ability to be configured to run sequentially or in parallel. Also, batches should be able to contain subbatches. Subbatches are common in data warehousing. You may have a batch that loads dimensions and a batch that loads facts. Those batches can be batched together to load a specific data mart. Batches are scheduled to run periodically or according to any triggering mechanism.
* **Process metadata**. Each time a batch is executed, process metadata is generated. Process metadata is crucial for depicting whether the data warehouse was loaded successfully or not.

If you are not new to ETL, you've probably noticed that the metadata categories are not listed in container order but in the order that they are generally built or generated in the ETL process. For example, even though batches contain jobs, you must build your jobs before you can batch them. Each type of ETL metadata contains its own specific attributes that need to be created, maintained, and published. The subsequent sections examine the specifics of the technical metadata in the ETL environment.

For consistency throughout this book, we refer to the physical implementation of the data mapping as a *job* and their containers as *batches*. The terms may be consistent with some ETL tools but not others. Our generic designation of the terms does not infer any particular technology. If your technology calls your physical source-to-target mappings something other than *job*, substitute *job* with your own term for the purposes of capturing its metadata.

NOTE

If you implement your ETL solution without a dedicated tool, you are not excused from generating the metadata outlined in this chapter. Tools are meant to ease the burden of these tasks. Without them, you will have to produce and maintain metadata manually. If you must create ETL metadata by hand, use spreadsheets kept in a version-control system such as PVCS or SourceSafe.

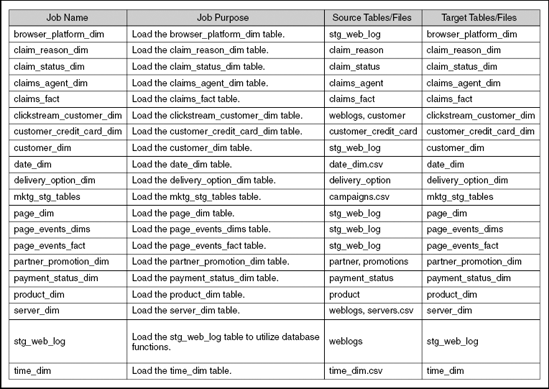
ETL Job Metadata

ETL metadata can be very technical, and end users tend to shy away from it. Source-to-target mapping usually contains programming code and can be cryptic to business users or nontechnical end users. However, source-to-target mapping is crucial for understanding the true data lineage of the data in the data warehouse. Metadata is sought after by the data warehouse team as well when credibility of the data is in question and its integrity needs to be proven.

NOTE

The optimal way to present source-to-target mappings is to utilize a dedicated ETL tool. These tools offer reports to present the information to users. Some tools are better at it than others. Make sure to ask your potential ETL vendor to show you how their tool handles the presentation of source-to-target mappings.

[Figure 9.2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch09.html#etl_job_metadata-id1) illustrates the elements of ETL job metadata that need to be created, stored, and shared. The following elements must be tracked to help manage your jobs and share their identity and functionality with your managers and users.



**Figure 9.2. ETL job metadata**

* **Job name**. The name of the physical ETL job
* **Job purpose**. Brief description of the primary focus of the process
* **Source tables/files**. The name and location of all source data
* **Target tables/files**. The name and location of all resulting data after it has been transformed
* **Reject file name**. The name and location of the file or table that stores any records intended to be loaded but are not in the ultimate target
* **Pre-processes**. Any other jobs or scripts on which the job is dependent to run before it can be executed
* **Post-processes**. Any other jobs or scripts that the job needs to run to complete its process

Jobs

A job is a collection of transformations that perform the physical extraction, transformation, and load routines. The metadata for a job is the physical source-to-target mapping. Jobs should be named after the target table or file that they are loading. If your ETL process has many segments, each job must contain a prefix that indicates its purpose. ETL jobs generally fall into one of three categories:

* **Extraction**. EXT\_<table name>. Indicates that the job's primary purpose is to extract data from the source system.
* **Intermediate stage (for example, cleaning and conforming)**. STG\_<table name>. The STG prefix signifies that the job does not touch the source or the target. It is an intermediate process that lives only in the staging area. If your process touches down in the staging area more than once, append a counter to the prefix (for example, STG1, STG2, STG3, and so on).
* **Target**. TRG\_<table name>. Indicates that the job loads the target data warehouse. Alternatively, we've seen these jobs named FAC\_<table name> and DIM\_<table name> to indicate if the target table is a fact or dimension, respectively. We don't see much value in that, but it is an acceptable convention if it provides value for your situation.

Transformation Metadata

Transformation metadata is information about the *construction* of the ETL process. The ETL developer spends most of his or her time constructing or reusing data transformations. Transformations are composed of custom functions, procedures, and routines that can include cursors, loops and memory variables, making them extremely difficult to document and offer as metadata. Any manipulation performed on your data during the ETL process is considered a transformation. If you are writing your ETL in SQL, you need to identify each of the distinct sections in your procedures and label them to be consistent with the common attributes of transformation metadata.

Dedicated ETL tools predefine transformations common to the data warehouse environment and provide them as part of their package. Prebuilt transformations expedite ETL development and implicitly capture transformation metadata. Common data transformations that exist in most ETL jobs include:

* **Source data extractions**. This could be as simple as an SQL SELECT statement or involve FTP or reading XML DTDs or mainframe Copy Books.
* **Surrogate key generators**. These can simply call a database sequence or involve complex routines that manage memory or involve third-party software. The last number inserted into the data warehouse is also metadata that needs to be maintained and presented as required.
* **Lookups**. Primarily used to get surrogate keys from the dimensions during a fact table load or referential integrity in the staging area. If you are using raw SQL, this involves all of your inner and outer joins and IN statements.
* **Filters**. This rule determines which rows are extracted and loaded. Metadata is the business rule or constraint used to apply the filter. Filters can be applied anywhere in the ETL process. It is a recommended practice to filter your data as early in the process as possible.
* **Routers**. Conditionally routes rows like a CASE statement
* **Union**. Merges two pipelines with compatible row sets
* **Aggregates**. When the fact table is not the same grain as the atomic-level transaction, you need to apply aggregates to source data. Metadata associated to the aggregate includes any calculations within the aggregate function, the function itself—count, sum, average, rank, and so on—and the columns on which the aggregate functions are grouped. The grouped columns declare the granularity of the aggregate.
* **Heterogeneous joins**. When your source's data comes from different systems, they are usually joined outside of any single database (unless you use a database *link*). The way you join heterogeneous systems outside of the DBMS environment needs to be defined and presented as metadata.
* **Update strategies**. The update strategy contains business rules that determine if a record should be added, updated, or deleted. It also contains the slowly changing dimension policy.
* **Target loader**. The target loader tells the ETL process which database, table, and column needs to be loaded. Additionally, this documents if any bulk-load utility is used to load data into the data warehouse.

Each transformation gets data, manipulates it to a certain degree, and passes it to the next transformation in the job stream. Metadata attributes that describe the transformation include:

* **Transformation name**. A single job can have a multitude of transformations, and identification of each is crucial for management and maintenance. Each transformation must have a unique name that is meaningful and complies with standard naming conventions. Transformation-naming conventions are outlined in this chapter.
* **Transformation purpose**. The purpose of the transformation must be easily identified. Many ETL tools color-code their predefined transformations. If you are hand-coding your ETL processes, make sure metadata is captured both within your code and your transformation matrix.
* **Input columns**. The data elements fed into the transformation
* **Physical calculations**. The actual code used to manipulate data
* **Logical calculations**. The textual equivalent of the physical calculations is required to make sense of the sometimes cryptic code required to physically manipulate data.
* **Output columns**. The result of the data transformation sent to the next transformation

Transformation Nomenclature

Transformations are the components of an ETL job. Each type of transformation requires a slightly different naming format. For maintainability, adhere to the following naming conventions while building your ETL transformations:

* **Source data extractions**. SRC\_<table name>
* **Surrogate key generators**. SEQ\_<name of surrogate key column being populated>
* **Lookups**. LKP\_<name of table being looked up or referenced>
* **Filters**. FIL\_<purpose of filter> (for example, FIL\_SUPPRESS\_BOTS to suppress hits on a Web site generated by bots)
* **Aggregates**. AGG\_<purpose of aggregate> (for example, AGG\_HITS\_BY\_MONTH to aggregate Web site hit counts to the monthly level)
* **Heterogeneous joins**. HJN\_<name of first table>\_<name of second table>
* **Update strategies**. UPD\_<type of strategy (INS, UPD, DEL, UPS)>\_<name of target table>
* **Target loader**. TRG\_<name of target table>

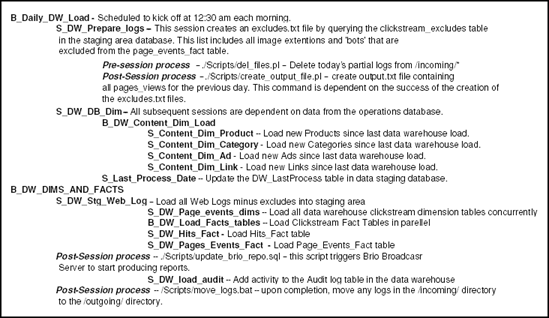
Batch Metadata

Once all of the jobs are designed and built, they need to be scheduled for execution. The load schedule is a crucial piece of metadata for the team responsible for incrementally loading the data warehouse. [Figure 9.3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch09.html#clickstream_data_mart_load_schedule) illustrates what a typical load schedule might look like for a clickstream data mart.

In [Figure 9.3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch09.html#clickstream_data_mart_load_schedule), notice that the load schedule contains information for the following metadata attributes:

Load Schedule

* **Dependent batches**. Batches are nested to contain several levels of batched jobs to execute many jobs in parallel or to maintain integrity between jobs. For example, a batch that loads dimensions of a data mart must complete successfully before fact table jobs can be executed. Remember, the data warehouse database does not enforce referential integrity. Dependent batches are one of the ways to enforce integrity in the data warehouse.



**Figure 9.3. Clickstream data mart load schedule**

* **Frequency**. Portions of the data warehouse are loaded monthly, weekly, daily, or are continuously fed data. This piece of metadata defines how often the batch is executed.
* **Schedule**. If a job is run daily, this metadata attribute captures the exact time the batch is executed. If it is run monthly, the exact day of the month is represented. Batches must have the ability to be scheduled on any give time of day, day of week, month, or year.
* **Recovery steps**. Actions required in the event of a midprocess failure. Recovery steps can be a lengthy process and are usually offered in a separate document. The steps to recover from a failed process must be walked through with the team that supports the operation of the execution of batched ETL jobs to ensure they understand the procedure.

Data Quality Error Event Metadata

[Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html) is an in-depth tutorial on capturing metadata describing data quality. The three main tables are depicted in detail in that chapter, but for uniformity, we list the data elements we captured in the cleaning and conforming steps.

First, the screen table includes:

* The **ETL Injection Stage** describes the stage in the overall ETL process in which the data-quality screen should be applied.
* The **Processing Order Number** is a primitive scheduling/dependency device informing the overall ETL master process the order in which to run the screens. Data-quality screens with the same Processing Order Number in the same ETL Injection Stage can be run in parallel.
* The **Severity Score** is used to define the error severity score to be applied to each exception identified by the screen.
* The **Exception Action** attribute tells the overall ETL process whether it should pass the record, reject the record, or stop the overall ETL process upon discovery of an error of this type.
* The **Screen Category Name** is used to group data-quality screens related by theme—such as *Completeness, Validation*, or *Out-of-Bounds*.
* The **SQL Statement** captures the actual snippet of SQL or procedural SQL used to execute the data quality check. If applicable, this SQL should return the set of unique identifiers for the rows that violate the data-quality screen—so that this can be used to insert new records into the Data Quality Error Event fact.

Second, the main error event fact table includes:

* The **Staged Record Identifier**, which uniquely identifies the error record
* The **Error Severity Score**, which assigns a severity from 1 to 100 to the error condition

The error event fact table has foreign keys to calendar date, time of day, ETL batch, table, and source system dimensions. These dimensions provide the context for the measures in the error event fact table.

The audit dimension includes the following fields described in [Chapter 4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch04.html):

* Overall Data Quality Score
* Completeness Score
* Validation Score
* Out of Bounds Score
* Number of Screens Failed
* Maximum Error Event Severity
* Cleaning and Conforming Timestamps, including the begin times and end times of specific job runs
* Overall ETL Timestamps, including the begin times and end times of complete end-to-end ETL jobs
* The Overall ETL Release Version Numbers to identify the consistent suite of ETL software tools in use at a point in time
* Other Audit Version Numbers such as allocation version, currency conversion logic version, and conforming logic version, depending on the business-rules environment

Process Execution Metadata

Virtually all process metadata in the data warehouse is generated by the ETL. Each time a job or batch is executed, statistics or indicators of success need to be captured. Load statistics are a vital piece of metadata that captures information about the *execution* of the ETL process and contains information about the actual load-process results.

Run Results

Metadata elements that help you understand the activity in your jobs or batches and rate the success of their execution include:

* **Subject name**. This can be the data mart or a description of a *batch* of programs being run for a specific area.
* **Job name**. The name of the program executed
* **Processed rows**. The total number and percentage of rows read and processed from the source system
* **Success rows**. The total number and percentage of rows loaded to the data warehouse
* **Failed rows**. The total number and percentage of rows rejected by the data warehouse
* **Last error code**. The code of the last database or ETL exception raised during the load process
* **Last error**. The textual description of the last database or ETL exception raised during the load process
* **Read throughput**. Throughput is used to measure the performance of the ETL process. It is normally represented in rows per second. This is used to capture if the source system is causing a bottleneck.
* **Write throughput**. Throughput is used to measure the performance of the ETL process. It is normally represented in rows per second. This is used to capture if the target data warehouse database is causing a bottleneck.
* **Start time**. The date, time, and second that the job is initiated
* **End time**. The date, time, and second that the job ends, regardless of its success
* **Elapsed time**. The difference between the Start time and End time. This is an important element for analyzing performance. In most cases, rows per second are not enough, because it can vary depending on the number of rows being loaded.
* **Source file name**. The name of the table or file where the data in the process originates. This can include more than one table or file.
* **Target file name**. The name of the table or file where the data in the process is targeted. This can include more than one table or file.

Process-execution metadata should be retained in a data store so trend analysis can be performed. Metadata can reveal bottlenecks in the process, and trending can expose portions of the data warehouse that lack the required scalability. Measures of data quality should also be trended.

Exception Handling

This data records exceptional conditions that arose in the running of the ETL system and what action was taken:

* **Subject name**. This can be the data mart or a description of a *batch* of programs being run for a specificarea
* **Job name**. The name of the program executed
* **Exception Condition**. One of a standard set of exception conditions
* **Severity**.
* **Action Taken**.
* **Operator**.
* **Outcome**.

Batches Scheduled

Batches are a collection of ETL jobs scheduled for execution. The name of the batch should reveal the subject area being loaded, the frequency that the jobs are run, and whether the jobs within the batch are executed sequentially or in parallel.

Metadata Standards and Practices

One aspect of metadata that is worth investigating is standards. Many organizations attempt to standardize metadata at various levels. If you are interested in standards on things such as naming conventions or domain standards, you may find the standards maintained by the Library of Congress to be helpful (www.loc.gov/standards/standard.html). Furthermore, links to additional standards organizations are offered. On their Web site, you'll find links to:

* **Metadata Encoding and Transmission Standard (METS)**. A standard for encoding descriptive, administrative, and structural metadata regarding objects within a digital library
* **American National Standards Institute (ANSI)**. The organization that coordinates the U.S. voluntary standardization and conformity-assessment systems
* **International Organization for Standardization (ISO)**. The body that establishes, develops, and promotes standards for international exchange

Metadata relative to the ETL process includes not only standards in values and conventions but also in methodology on storing and sharing metadata. Organizations geared toward the organization and storage of metadata include:

* **Dublin Core**. The Dublin Core Metadata Initiative (DCMI) is an open forum created to develop metadata standards. DCMI hosts periodic working groups and conferences that promote metadata standards and practices worldwide. More information about Dublin Core can be found on their Web site at www.dublincore.org.
* **Meta Data Coalition**. The now defunct Meta Data Coalition (MDC) was created in 1995 as a consortium of approximately 50 vendors and end users whose efforts attempted to provide technical direction to exchange metadata between products in the data warehouse environment. During its existence, MDC helped to establish and encourage consistent means of sharing metadata. In 2000, MDC folded into the Object Management Group (OMG), who has a much larger agenda in the metadata space and combined MDC's objectives with their own broader plan.
* **Common Warehouse Metamodel**. The Common Warehouse Metamodel (CWM) was created as a result of the merger of MDC and OMG. You can find detailed information about CWM in books dedicated to the topic and the OMG Web site www.omg.org/cwm.

Establishing Rudimentary Standards

To maintain manageable jobs for all of your enterprise data warehouse ETL processes, your data warehouse team must establish standards and practices for the ETL team to follow. Whether you follow the recommendations of the organizations in this section or follow the practices outlined throughout this book, at the most rudimentary level, your organization should adhere to standards for the following:

* **Naming conventions**. Corporations usually have namingconvention standards in place for their existing software and database development teams. The data warehouse may be required to follow those standards. In practice, we tend to deviate from those standards when we can, especially while naming columns. But often, you must conform to corporate policies. Therefore, any corporate policies must be documented and provided to the data warehouse team.
* **Architecture**. Best-practice guidelines need to be captured as metadata for your ETL environment. In many cases, high-level architecture decisions are made before the ETL team is established or on the recommendation of your ETL vendor. Decisions such as whether you should run your ETL engine on your data warehouse server or on a dedicated box, whether you should have a persistent staging area (PSA), or whether your target data warehouse should be normalized or dimensional should be based on best practices that need to be established, documented, and followed.
* **Infrastructure**. Should your solution be on Windows or UNIX, mainframe or AS/400? Corporate standards influence the decision-making process for the products and hardware used for the ETL. Some ETL engines run only on UNIX or the mainframe, while others run on Windows. You must have established corporate infrastructure metadata before any purchasing decision is made for your ETL environment. Once your environment is established, the metadata of its infrastructure must be documented and reviewed with your internal infrastructure support team.

Naming Conventions

Naming conventions for the objects in the data warehouse environment should be established before the ETL team begins any coding. Conventions for tables, columns, constraints, indexes, checks, and so on should be offered by the existing DBA team within your organization or the data warehouse manager. The ETL team must adopt any conventions that exist in the rest of the data warehouse environment.

NOTE

If you decide that your existing corporate naming conventions are not applicable to your ETL environment, you must document your alternative naming conventions and seek approval from your corporate standards committee. Once approved, the corporate standards committee should incorporate the new ETL-specific conventions with their existing naming standards.

None of the organizations we recognize earlier in this section recommend naming standards specific to the ETL process. If your ETL processes consist only of raw SQL, follow your in-house programming practices and naming standards, with the exception of conventions explicitly defined in this chapter. In this section, we share conventions that prove to be effective in the ETL environment.

NOTE

The naming conventions recommended in this section include the common transformations found in most ETL tools. Use these conventions regardless of your tool, even if you are coding your ETL by hand. Tools can come and go—do not set up new standards each time you change tools. Use the vendor-recommended naming conventions for transformations that are not mentioned in this book. It is acceptable to slightly modify these naming conventions for your purposes—the important thing is be consistent within your environment.

Impact Analysis

One of the key advantages of maintaining ETL metadata is to enable impact analysis. Impact analysis allows you to list all of the attributes in the data warehouse environment that would be affected by a proposed change. Ultimately, you should be able to analyze the impact of a change that would occur in any component of the data warehouse and list all of the attributes in all of the other components. An impact analysis solution must be able to answer the following questions:

* Which ETL jobs depend on this staging table?
* Is this table in the source system used by the data warehouse?
* Would the deletion of this source system column affect the ETL process?
* Which source systems populate this dimension?
* Which ETL jobs and data warehouse tables will need to be modified if we change this data type from VARCHAR(2000) to CLOB?

Tools designed specifically for ETL should be able to answer all of these questions. Without an ETL tool, you need to maintain spreadsheets to capture every table and column from the source systems and their mapping into the data warehouse. Each time an ETL job is altered, the spreadsheet needs to be manually modified to stay current.

Summary

In this chapter, we have brought order to the traditional chaos of metadata surrounding a data warehouse, first by focusing only on the metadata needed to manage the ETL system, and next by dividing the ETL metadata into three categories.

1. **Business metadata**. Describing the meaning of the data in a business sense and consisting of separate tables tracking business definitions, source system information, the data warehouse dictionary, and logical data mapping
2. **Technical metadata**. Representing the technical aspects of data, including attributes such as data types, lengths, and lineage, and consisting of separate tables tracking system inventory, data models, data definitions, business rules, ETL jobs definitions, specific data transformations, and batch job definitions
3. **Process execution metadata**. Presenting statistics on the results of running the ETL process itself, including measures such as rows loaded successfully, rows rejected, and amount of time to load. We proposed particularly important process metadata in the cleaning and conforming steps, including the screen dimension table, the error event fact table, and the audit dimension table. All of this metadata consists of separate tables tracking run results, exception handling, and the immediate operational schedule.

Chapter 10. Responsibilities

In this chapter, we discuss managing the development and administration of a successful ETL system. We could have put this chapter at the beginning of the book, before the myriad responsibilities of the ETL system were discussed thoroughly, but we think by putting it at the end of the book, you will better be able to visualize how to manage a team effectively.

The first part of this chapter looks at planning and leadership issues, and the second part descends into more detail of managing the ETL system. Many of these perspectives were developed in *Data Warehouse Lifecycle Toolkit*.

NOTE

PROCESS CHECK Planning & Design:

Requirements/Realities → Architecture → Implementation → Release to Ops

Data Flow: *Extract* → *Clean* → *Conform* → *Deliver*

Planning and Leadership

In some ways, the data warehouse and ETL process are just like any other software development project. When a data warehouse team is established, it usually requires three specialists. The following list contains common roles required to initiate a data warehouse project. The list includes the primary role and the secondary role (in parentheses) that the same individual can perform on small teams.

* **Data Modeler (Project Manager)**. The data modeler must be specially trained in dimensional data modeling and educated in the principles of dimensional modeling.
* **ETL Architect/Programmer (DW Architect)**. The ETL programmer is usually a SQL and database expert as well as an architect. This person establishes the technical infrastructure of the ETL system and data warehouse environment and designs the physical ETL processes.
* **Application Specialist (Business Analyst)**. This person gathers and documents the business, analytical, and reporting requirements. This specialist writes the front-end interface and initial reports for the data warehouse. This position is often called the business intelligence (BI) specialist.

When a data warehouse is kicked off, the often compact team of highly specialized individuals builds the foundation for what evolves into the most visible, widely used database application in your enterprise. Like any other substantial structure, without a thoroughly planned and methodical construction of the foundation, anything built subsequently is certain to topple.

Having Dedicated Leadership

The data warehouse is a complex entity that requires specialized knowledge that most enterprise IT managers don't quite understand. Initially, the data warehouse must have a dedicated project manager who has experience implementing a data warehouse using the principles of dimensional modeling. As your data warehouse evolves, each component and subcomponent must have a dedicated project manager. A mature data warehouse must a have distinct ETL, data-modeling, and business-intelligence managers as well as a dedicated project manager who oversees all of the *departments* of the data warehouse team to ensure a cohesive solution is implemented across all areas.

It's been argued that a single person can manage the entire data warehouse, but we strongly recommend that specialists for each area be appointed. Each area requires specialized skills that become diluted if someone tries to encompass them all. Remember, a single mind, no matter how strong, is not as strong as a group.

NOTE

A group is always stronger than an individual, but that does not mean that design decisions are made by voting! Design decisions are best made autocratically, so that consistency is maintained.

Planning Large, Building Small

When you are building a data warehouse from scratch, it is often difficult to imagine that it is going to evolve from the single data mart you are working on into a major enterprise, mission-critical application that has more exposure than any other application within your company. The big picture is commonly lost because data warehouses are usually built in an iterative approach. They start and complete a single business process or *data mart*, such as human resources or campaign management, before development of the next data mart begins.

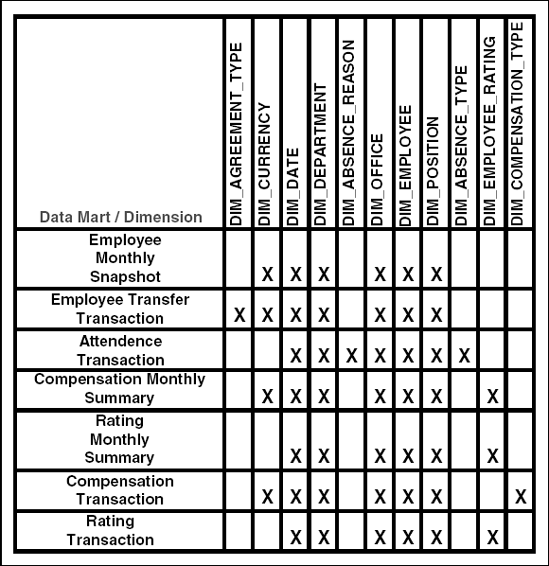
The data warehouse architect must utilize a methodology known as *data warehouse bus architecture*, which outlines the framework of the data warehouse so all of the resulting data marts work together in a cohesive fashion using conformed dimensions and conformed facts as we have described extensively in this book.

Part of the data warehouse bus architecture process includes devising a *data warehouse bus matrix*, a list of all the dimensions that need to be created and their associations to the various data marts in the data warehouse. The bus matrix helps the architect visualize which dimensions are shared or conformed across the various data marts in the data warehouse. Once the bus matrix is created, the physical data marts can be built one at a time. [Figure 10.1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html#data_warehouse_bus_matrix) illustrates a sample data warehouse bus matrix.

Just as certain dimensions are reused throughout the data warehouse, certain ETL routines are reused over and over when you are building the ETL processes. For example, your first ETL process most likely includes generating a surrogate key for a dimension. The code that generates the surrogate key can be reused to generate all surrogate keys in the data warehouse—just by using different parameters. If you come from a software development background, you may have heard the saying, *Write once, use many*. That adage means to reuse as much code as possible. Not only does reusable code cut down development time of subsequent processes; it ensures consistency across them.

You need an effective code-reusability strategy. Establish an environment that encourages developers to share ideas and to trust each other's work. The following tips can help you build a team environment.

* **Agree as a group on strategies**. Have regular meetings that discuss technical and functional strategies and solve problems as a group.
* **Share ideas as well as code**. Reinforce that ETL development is not a competitive sport. Work together and share issues with others on your team. We've spent hours in isolation agonizing over situations. Then, by simply explaining the scenario to someone else, the solution instantly came to mind. It's amazing how many ideas are born during simple conversation.



**Figure 10.1. Data warehouse bus matrix**

* **Use a repository**. Many ETL tools offer a repository for reusable code. Make sure your tool includes a repository that allows code to be reused and shared among various load routines and developers. If you have not yet invested in a dedicated ETL tool, at least use a source-code repository such as SourceSafe or PVCS. In mature installations, you need to develop multiple individual repositories which then must be managed as a single virtual repository.

Once your team is trained to work together and all of your core routines are in your repository, development efficiency is sure to increase. Also, working together helps developers understand the big picture of the project. Avoid isolating developers by subject area. Over time, each developer becomes an expert in the specific areas he or she develops. It is advantageous for the team to be exposed to other areas for which they are not directly responsible. Broadening the scope of ETL developers promotes cross-functional planning and builds morale within the team.

NOTE

Your ETL team should be encouraged to share and reuse as much of their work as possible. Make sure appropriate metadata is associated to all of the sharable code in your repository. Metadata is crucial for identifying the purpose of the code and providing instructions for its use. But be realistic in your expectations for literally reusing code across separate operating systems and DBMS platforms.

Hiring Qualified Developers

Skilled ETL developers are invaluable assets to your organization. However, skill alone does not qualify someone as an expert. We've interviewed many potential ETL developers over the years who knew various tools inside and out but could not grasp dimensional concepts such as hierarchy mapping tables. Developers must have the ability to comprehend new techniques quickly and implement them with minimal hand-holding. When we interview, we tend to spend less time talking about using tool features and more time on problem solving—technical and functional. We find that candidates with intelligence and character make much better ETL team members than those with only technical skill.

During candidate interviews, ask a specific question you know the interviewee does not know the answer to. Watch to see how he or she works it out. Remember, it's not whether the candidate gets the answer right, but the process he or she uses to solve it. The reality is that ETL and data warehousing can be quite complex and are quite specialized. Still, it's not splitting atoms. (If it were, a scientist would have to provide a specification!) So, when you are building your team, make sure that your developers are motivated to grow technically and professionally. They must be able to grow with you and your project and be able to accept and adapt new techniques and strategies.

Building Teams with Database Expertise

Part of the responsibility of the ETL manager is to inventory all of the source systems within your enterprise and align the appropriate skill sets in your development team according to the existing databases. If you use a dedicated ETL tool, staffing your team with specific database expertise might not be as critical. But even with the best toolsets, you never seem to get away from rolling up your sleeves and writing raw SQL at some point in the ETL development process.

Listing specific SQL coding tips and techniques is beyond the scope of this book—there are several SQL books on the market—but be advised that SQL is the foundation of any DBMS query. Tools alone cannot adequately fulfill all of your ETL requirements. When you interview potential candidates for your ETL team, be sure they are proficient in the specific *flavors* of SQL required both for your transaction DBMSs as well as the system you have chosen for your main ETL processing.

NOTE

Each source system DBMS that you encounter requires knowledge and implementation of specialized SQL syntax. Make sure your team has the specialized skills to navigate the various databases and that your ETL tool can seamlessly integrate native SQL code in the ETL process without leaving the application.

Don't Try to Save the World

The ETL system is just a portion of the data warehouse project, and some things that happen within it are beyond your control. Accept that the data warehouse is not, nor will be, perfect. You are going to encounter dirty data, and in many cases, you will not get the political backing to clean it. Our philosophy is that the best way to get data cleansed is to expose it. You've been asked to build the data warehouse because the existing data has been so difficult to analyze. Chances are that much of the data you are publishing with your ETL process has not been exposed before, especially to the extent that it is via the data warehouse. If your petition for ultimate data quality is ignored, be patient. As soon as blemished data is published, managers have to start explaining the anomalies, and you will witness a change in heart about the quality of the data and receive the support you need.

Enforcing Standardization

In large ETL projects, it is imperative that you establish standards early on. Without standards, developers write inconsistent ETL jobs and cause the maintenance of existing code to be horrendous. The ETL team must standardize their development techniques to provide a consistent and maintainable code environment. The following list contains areas of the ETL process that need standardization most:

* **Naming conventions**. Establish and enforce a standardized convention for naming objects and code elements in your ETL process. Begin by adopting existing naming standards in your organization. Add to your existing conventions with those recommended by your ETL tool vendor.
* **Best practices**. Document and follow best practices for building your ETL routines. Make standards for things such as the ordinal position of transformations in your routines or the best ways to recover from a failed process. This book is full of recommended strategies for your ETL processes. Standards to consider are:
* **Generating surrogate keys**. If you decide to use the database, ETL tool, or any other mechanism to generate surrogate keys, be consistent throughout your ETL jobs.
* **Looking up keys**. You may use mapping tables, look to the physical dimensions, or use other staging techniques to associate natural keys to their surrogates. Pick one and stick with it. If you mix techniques, maintaining these routines is a nightmare.
* **Applying default values**. Several approaches and values are acceptable means of defaulting missing values. Remember that missing values or NULL values have to be handled carefully in the data warehouse because those values can cause *blank* column or row headings on reports and because some databases do not include NULLs in their indexes. It's best to check incoming records for NULL values and to substitute them with actual values such as the single character ? during the ETL process.

Monitoring, Auditing, and Publishing Statistics

ETL statistics are invaluable to anyone who uses the data warehouse. If your data warehouse has a dedicated Web site—and it should—make sure it includes the daily statistics of your ETL processes. Users often want to know exactly when a table has been loaded or if any rows were rejected. Most ETL tools generate load statistics automatically. Make sure your tool has the ability to automatically publish the required statistical information upon completion of the daily data load.

NOTE

CROSS-REFERENCE A list of the statistical elements that should be published as part of your metadata strategy can be found in [Chapter 9](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch09.html).

Maintaining Documentation

Documentation of what your ETL processes do is an invaluable asset to the data warehouse team and has become mandatory in many phases of financial and regulatory-reporting data warehouses. Even with the most thorough logical data-map specification, only the developer knows *exactly* what is done to the data between the data warehouse and its source. It's the responsibility of the ETL team to maintain documentation for the lineage of each data element in the data warehouse with rigorous change control. Some documentation comes in the form of metadata, but not all forms of documentation are considered formal metadata. Metadata is a complicated entity that already has several books to explain its capacity. Regardless of how you categorize it, several pieces of documentation must exist and be maintained and published. Often, descriptions of processes cannot be captured in the allotted fields in the various tools designed to capture this information. Inevitably, you need to provide documentation that explains your ETL processes in Word documents, Excel spreadsheets, PowerPoint presentations, and so on. Use a version control system such as SourceSafe or PVCS to maintain the integrity of your documentation.

Providing and Utilizing Metadata

Metadata is crucial for sharing and reusing ETL processes. Virtually all ETL tools have the ability to capture and utilize metadata. Don't do your team an injustice by creating processes without metadata. Each time you create a new ETL process, keep this in mind: If it's not captured by metadata, it doesn't exist. This is true in established data warehouse environments. If you don't expose your work via metadata, someone else on your team may recreate from scratch something that you've already created and tested.

What's more, the ETL tool repository is often the home of the metadata repository. Some ETL tools have the ability to transmit existing metadata from other tools such as data-modeling or reporting tools and associate elements for impact analysis. Some business-intelligence tools can utilize the ETL repository to integrate metadata with the data warehouse user interface. If the metadata in the ETL environment is published, it must be maintained, or the toolset can publish out-of-date information to its users.

Keeping It Simple

If you think there has to be an easier way to so something, there usually is. When you are building your ETL processes, take a step back from time to time and look at your work from a design perspective. Is the design straightforward, or does it have complexities that could be avoided? The more complex your processes are, the more difficult they will be to maintain. Moreover, complex ETL designs are almost impossible to *evolve*. As business needs or source systems change, your ETL jobs must be adaptable enough to change with them. We were called on a project once where an ETL job was so complex it was untouchable. No one knew exactly what it did, so no one was able to modify it. It was so convoluted that we were hired to reverseengineer, document, and redesign it into a more streamlined, manageable process.

Optimizing Throughput

No real limitation exists as to how elaborate your ETL processes can be to transform your source data into usable information for the data warehouse.

However, a restriction on how long your jobs can take to process does exist. One of the challenges to being an ETL developer is to have the ability to extract, clean, conform, and load data within the allotted load window. The load window is the time set aside each night to run the ETL processes. Usually, the data warehouse is not available to its users during the load window, so there is always pressure to keep the load window as small as possible.

Managing the Project

The ETL process is a critical piece of the data warehouse project. Until now, it has been thought of as the least glamorous aspect of the project and typically did not receive the attention it deserved. In the early days of data warehousing, the primary focus was on front-end tools; then as the size of data warehouses began to grow, the dimensional data model became the next focal point. As the data warehouse reaches its next level of maturity, ETL is finally getting appropriate time in the spotlight.

Most designers agree that at least 70 percent of the entire data warehouse project is dedicated to the ETL process. Managing the team that builds these tenacious processes responsible for transforming potentially billions of rows of unorganized data from disparate systems into a cohesive userfriendly information repository is an achievement that is highly regarded by technologists and executives alike. Managing the ETL team takes dedication and know-how.

The ETL manager position has been established to alleviate the overwhelming responsibility of the ETL process from the data warehouse project manager. Also, this position provides business sponsors with confidence that the ETL team can maintain a controlled, efficient environment to load the data warehouse with clean, consistent data. The tasks contained in this chapter should be read carefully by all members of the data warehouse team to make certain that they understand that the ETL process is not a trivial byproduct of the data warehouse project but rather the glue that holds the entire project together. To the ETL manager, this chapter offers the knowledge required to bring your ETL project to victory.

Responsibility of the ETL Team

At the most rudimentary level, the ETL team is responsible for extracting data from the source system, performing data transformations, and loading transformed data into the target data warehouse. More specifically, to achieve optimal ETL results, the following tasks are the responsibilities of the ETL team:

* Defining the scope of the ETL
* Performing source system data analysis
* Defining a data-quality strategy
* Working with business users to gather and document business rules
* Developing and implementing physical ETL code
* Creating and executing unit and QA test plans
* Implementing production
* Performing system maintenance

To effectively manage your team in the execution of the preceding tasks, we've outlined an actual project plan that incorporates these tasks and gives details of your functional responsibility for properly managing each.

Defining the Project

Although the ETL process is only one of the many components in the data warehouse lifecycle, it is the center of the data warehouse universe. Moreover, the ETL process is by far the most difficult component to manage. As users begin to see the resulting data in the beginning phases of the project, you will be faced with an onslaught of change requests. Without a properly executed project plan and change-management strategy, managing the ETL process will seem impossible, a never-ending task that could delay the project to the point of failure. As you go about defining your project, keep the following guidelines in mind:

* **For a seamless process, the management of the ETL must be closely coupled with the other components within the data warehouse lifecycle**. From the standpoint of those who work with it regularly, the data warehouse is never really finished. As new requirements are initiated, the modeling team, ETL team, and reporting team must work together to effectively accomplish these new goals and complete the tasks that lie ahead. Steps outlined in this chapter should be reused as each subject area is added to the ever-evolving data warehouse. Nailing down the methods outlined in this chapter is crucial to properly managing these iterative processes.
* **Be realistic when estimating completion dates and defining scope**. Do not let data modelers or business sponsors who do not have the knowledge to make an informed decision dictate the time frame of the ETL effort. Use the project plan in this chapter as a guide and make sure your business users and sponsors are aware of exactly what is involved in *loading* your data warehouse.
* **Make sure the ETL team is an active participant in the data warehouse project kick-off meeting**. Such a meeting can be a venue where you introduce the ETL team to key business users and discuss the ETL-specific goals, roles and responsibilities, and timeframes. Create an environment that fosters collaboration. This meeting helps participants understand project needs, and it gives you the opportunity to manage expectations.

Planning the Project

*What is a plan? A plan is a method of action, procedure, or arrangement. It is a program to be done. It is a design to carry into effect, an idea, a thought, a project or a development. Therefore, a plan is a concrete means to help you fulfill your desires. —Earl Prevette*

It is your ultimate goal as an ETL manager to successfully manage the ETL process and integrate the process into all other phases of the lifecycle. You might assume that managing the ETL process is identical to managing any other implementation, but it is quite different. In this section, we explain the methods that have helped us achieve successful implementations. Also, we expose many obstacles you may be faced with and provide suggestions to mitigate those risks to achieve your goals.

As a prerequisite to beginning the iterative portions of the project plan, you need to complete a few housekeeping responsibilities. These tasks include determining your ETL tool set and staffing your project team.

NOTE

The order in which these two tasks are executed is important. You want to select your ETL tool set prior to staffing your team. Doing so will enable you to recruit individuals who specialize in your selected tool set.

Determining the Tool Set

The ETL manager must determine whether it makes sense to build the ETL processes by hand or to purchase an ETL tool set, as discussed in [Chapter 1](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch01.html). There are many arguments for either case. However, with the success of enterprise data warehousing and the expectations of executive sponsors, we feel there is no time for hand-coding, especially when you consider the iterative nature of data warehousing. Even the smallest projects benefit from the transformation reusability of dedicated ETL tools. The features available in these tools, right out of the box, would take months to design manually, not to mention coding the actual data-transformation processes. The reduced development time obtained via these tools makes them viable solutions for any data warehouse project.

Furthermore, ETL tools are specifically designed for the task at hand. The most popular case we hear for building over buying is that programmers already know SQL. Why waste time learning a tool that essentially has the same result: moving data? Two analogies immediately come to mind when we hear this. First, if the only tool you know is a hammer, everything around you is treated like a nail. Setting screws becomes very difficult, laborious, and sloppy. The second is the secretary that didn't have time to learn word processing because she was too busy typing. As silly as this may sound, it is synonymous to not training your SQL programmers in dedicated state-of-the-art ETL tools to perform their assignments.

To aid in the decision-making process, we recommend documenting your tool-selection criteria. Establish proof-of-concept decisive factors such as throughput performance, ease of modification, and vendor support and then perform a proof-of-concept for the tools (including a comparison with hand-coding) that you feel may meet your criteria. Upon evaluation of your proof-of-concept results, you will be able to make a firm decision on whether to build or buy an ETL tool set. If you are purchasing, you will have a firm idea of exactly which tool fits your needs.

Staffing Your Project

A crucial factor in managing the ETL process is establishing a superior team. Your team members must possess the necessary skills to perform the duties expected of them. A properly trained team is vital to your success. Ensuring that all team members fit into the company culture and work well together is equally important.

Before data warehousing reached its current point of maturity, all duties of the project were typically performed by just a few data warehouse experts. These all-encompassing experts interviewed business users, documented requirements, designed the data model, loaded the database, and so on. As the data warehouse project evolves, we are discovering that each of these specific tasks requires a unique set of specialized skills and that no individual can achieve expertise in all of them.

ETL Team Roles and Responsibilities

Staffing the roles of the ETL team is an undertaking that must be reckoned with. If you have appropriate knowledge internally, you may be able to recruit or train your internal staff. Otherwise, you need to work with recruiters to find the appropriate expertise required to construct your team.

The following bulleted list explains the roles and responsibilities we've found to be fundamental to building an optimal ETL team.

NOTE

Staffing one person per specific role would be ideal. However, as circumstances dictate, it is realistic to have people play multiple roles by overlapping some of their responsibilities, depending on the size of your project. Remember when you are staffing the project team, your main goal is to ensure that all duties will be performed. You do not necessarily have to fill each role with a dedicated person.

* **ETL Manager**. This individual is responsible for the day-to-day management of ETL team and the on-going data warehouse maintenance as it relates to the ETL process. The ETL manager is accountable for managing the development of the data-extract, transform, and load processes within the data warehouse and oversees its testing and quality assurance. The ETL manager also develops standards and procedures for the ETL environment, including naming conventions and best-development and design practices.
* **ETL Architect**. Primary responsibilities for this individual include designing the architecture and infrastructure of the ETL environment and designing the logical data mappings for the ETL development team. This architect must have a strong understanding of the business requirements and the source operational systems. The ETL architect is responsible for resolving complex technical issues for the team and migrating ETL routines to production.
* **ETL Developer**. This individual is accountable for building the physical ETL processes. The ETL developer works closely with the architect to resolve any ambiguity in specifications before actual coding begins. The developer is responsible for creating functional ETL routines and testing their reliability to ensure that they comply with business requirements. There are usually several ETL developers assigned to a data warehouse project.
* **Systems Analyst**. The systems analyst is accountable for business requirements definition activities and documenting those requirements throughout the data warehouse lifecycle. The systems analyst works closely with all members of the data warehouse team and the business users.
* **Data-Quality Specialist**. Data-warehouse quality includes the quality of the content and the information structure within the data warehouse. The data-quality specialist typically reports to the ETL manager but may also report directly to the data warehouse project manager. The data-quality specialist primarily works with the systems analyst and the ETL architect to ensure that business rules and data definitions are propagated throughout the ETL processes.
* **Database Administrator (DBA)**. The DBA is primarily responsible for translating the logical database design into a physical structure and maintaining the physical database. Moreover, the DBA works very closely with the ETL team to ensure that new processes do not corrupt existing data. In some environments, the DBA actually owns the ETL process once it is migrated to production.
* **Dimension Manager**. The dimension manager is responsible for defining, building, and publishing one or more conformed dimensions to the extended data warehouse community that agrees to use the conformed dimensions. This is a truly centralized responsibility. Conformed dimensions must be version-stamped and replicated simultaneously to all fact table provider clients. There can be more than one dimension manager in an organization, since the data content of each dimension is largely independent. In any case, a given dimension is the responsibility of a single dimension manager.
* **Fact Table Provider**. The fact table provider owns a specific fact table. In a conformed dimension environment, the fact table provider receives periodic updates of dimensions from dimension managers, converts the natural keys in the fact tables to the dimension's surrogate keys, and exposes the fact table appropriately to the user community.

ETL Project Team Staffing Options

The old aphorism *you are only as good as your subordinates* holds special importance in a mission-critical environment like the ETL process of the data warehouse. An intelligent approach to preventing project failure is to build a superlative team to develop it. This section discusses various options available to you while building your ETL team.

Working with Recruiters

More often than not, you will need to look outside your organization while building your ETL team. Typically, organizations work with dedicated recruiters to seek the best candidates. But, just as a data warehouse needs to be fed complete, reliable information to be valuable to its users, you need to provide precise requirements to your recruiters for them to be effective. Be as detailed as possible when you supply job qualifications to ensure that you receive candidates that possess the skills and work habits you are looking for. Let the recruiter know the details of your environment, especially emphasizing your programming languages, vendor packages, and database systems. Also describe the dynamics of your team and exactly what type of person you are looking for. Provide the most detail possible to ensure that the candidates they send will meet your expectations.

Recruiting companies that specialize in data warehouse staffing will give you the benefit of working with recruiters who are knowledgeable in the data warehouse industry and the tools sets that support it. They are responsible for pre-screening candidates and weeding out under-qualified individuals before forwarding any resumes to you, limiting the number of lacking resumes and individuals you need to evaluate. You will be busy enough with many other tasks; the time saved using qualified recruiters is well worth their fees.

Hiring Internally versus Externally

There are advantages to building your team from either internal or external sources. The benefits of hiring internally include the following:

* The primary benefit of performing internal searches and hiring from within your organization is that internal individuals already have a strong understanding of your organizational structure and IT systems. They know who is responsible for what; who to go to for answers; and how to get things done politically. If you're lucky, they may already possess the skills needed to fill a specific role within the team. If an individual does not have the desired skill level but do have the potential and desire to be trained appropriately, this person may very well be a candidate worth considering.
* Providing internal employees who possess motivation with the opportunity to learn new things keeps them challenged and satisfies their needs.
* There is an economic benefit to hiring internally: It will most likely be more cost effective to hire from within than to go through a recruiter and incur placement fees, interview expense reimbursements, relocation costs, and so on.

If you hire externally, you hire an individual who possesses the skills you are looking for as well as experience using those skills in several different business cultures. This experience offers more than its face value. Experience saves you time and money while adding value to your team.

Selecting Team Members

Once you are armed with a handful of resumes, we recommend that you schedule a telephone interview to speak with potential candidates before bringing them in. Asking key questions over the phone instantly reveals their communication skills and level of understanding of the subject matter.

Candidates that pass phone screenings should be brought in for face-to-face interviews. Your candidates should be questioned not only by the ETL manager but also by technical developers, as well as by functional analysts. Having candidates meet both functional and technical individuals gives you the ability to gauge how broad their proficiency is. We've had many unpleasant experiences where team members were technically proficient but did not (and could not) grasp the functional picture. Their inability required extra work on the part of other team members to ensure their work actually met the business needs.

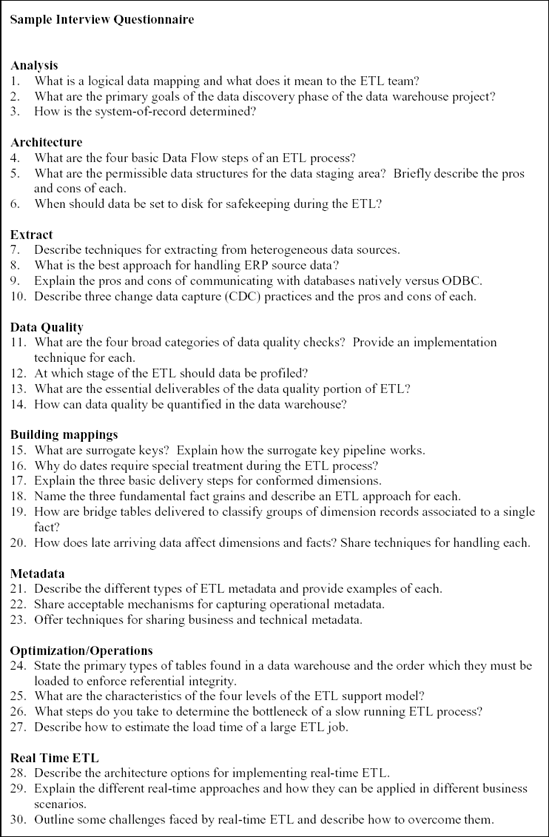
You need to be convinced, without a doubt, that all potential members of your team have sufficient knowledge in ETL process design, ample skill in required tool sets, and appropriate aptitude of business processes to comprehend functional requirements. The ability to work collaboratively with the rest of your team is crucial. Be sure to inquire about team dynamics on previous projects during your interviews.

During candidate screening and interviewing, it is essential that you and your recruiters are not only knowledgeable about the role they are seeking to fill but also know the appropriate questions to ask. [Figure 10.2](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html#sample_interview_questionnaire) includes an interview questionnaire that provides you with questions you need during the interview process. Using this questionnaire helps ensure your candidate's knowledge is sufficient for the specific role. Answers to the questionnaire are sprinkled throughout the book.

Building and Retaining a Winning ETL Team

Once you have staffed your team, your main responsibility as a manager begins. Retaining a first-rate team is among your biggest challenges. Superstar ETL personnel are in very high demand, and recruiters are not shy about poaching them from right under your nose. We find the best way to keep the majority of ETL developers and architects on our projects is to keep them challenged technically. In our experience, a bored technologist will be a departing one. It is your responsibility to assign projects that keep your team members interested and excited.

The tasks that the ETL developer accomplishes are not trivial. They step up to the plate acknowledging it is their responsibility to transform unorganized, disparate data into cohesive valuable information, an intense, sometimes grueling undertaking. Do not take them for granted. Be attentive of their needs; know what makes them tick and starts their fire. We've worked with some developers that just love to clean data. They find making consistent reliable data from garbage to be rewarding. Others cannot be bothered. They feel that if data is so important, it would be clean in the source; those developers would much rather be challenged with solving nearly impossible SQL puzzles like converting data from tremendously complex data models into simple dimensional ones. Other developers just love to race the clock. If an ETL process should take one week to develop, they work furiously to have it complete in just a few days. Part of your responsibility as manager is to know what kind of developers you have and keep them challenged.



**Figure 10.2. Sample interview questionnaire**

If team members are eager and able to accept more responsibility, give it to them. It's your duty to navigate each individual's desires and try to fulfill them. Also, you need to provide team members with the training they need to ensure they are the best at what they do. If you let your staff stagnate, they will leave the project and move on to a more challenging environment.

An effective approach to keeping in tune with your subordinates needs is to hold weekly status meetings. The ETL environment is a volatile one, and letting more than a week elapse without receiving your team's feedback on progress could be detrimental to the project. These dual-purpose meetings make sure team members are meeting their goals and that you are meeting yours. Give them responsibility, empowering them to make decisions where doing so makes sense. Moreover, you should foster an environment where team members can voice concerns, convey development needs, and so on. They must be able to rely on their ETL manager to take action on rectifying their problems. A well-managed staff is a satisfied one.

Outsourcing the ETL Development

Outsourcing IT responsibilities is a hot topic as we write. Yet the overall numbers are smaller than the talk would suggest. In 2003, of the $119 billion spent in the United States on IT budgets, less than five percent was reportedly vulnerable to outsourcing. In recent reports, some of the hoopla surrounding outsourcing savings is being offset by realizing that over time, managing outsourcing projects involves extra communication, travel to foreign countries, and resetting of expectations and deliverables that did not appear in the original financial savings projections. We are not saying that outsourcing is a bad idea, but we are cautioning that outsourcing is a tricky topic as far as the data warehouse is concerned.

The data warehouse must always respond to the data sources *de jure*, as well as to the changing priorities of management and the end user community. As we have said many times, the data warehouse is not a *project* (with specifications and a final delivery) but is rather a *process* that is on-going. Data warehouse development tasks are iterative and changing. In fact, that is one of the reasons we like the dimensional approach; it is the most resilient architecture for adapting to new surprises and changes in scope.

For these reasons, we are generally negative about outsourcing many of the data warehouse development tasks to remote parties and do not engage in regular contact with the source data suppliers as well as the end users. Remember that the data warehouse is a decision support system judged solely on whether it effectively *supports decisions*.

ETL system development does provide selected opportunities to outsource development tasks, if you have a specific transformation that can be well specified.

Project Plan Guidelines

Now that you have decided on your tool set and staffed your ETL project team, you are ready to dive into the heart of the ETL project plan. In this section, we provide a detailed project plan that any ETL team can utilize.

NOTE

Given the cyclical nature of data warehouse phases, the project plan can, and should, be reused with each phase of your data warehouse project. Consistent use of these guidelines enforces standards and ensures that no steps are forgotten.

Details of each step are explained in the remaining portion of this chapter. High-level steps for managing the ETL process are shown as a project plan in [Figure 10.3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html#etl_project_plan).

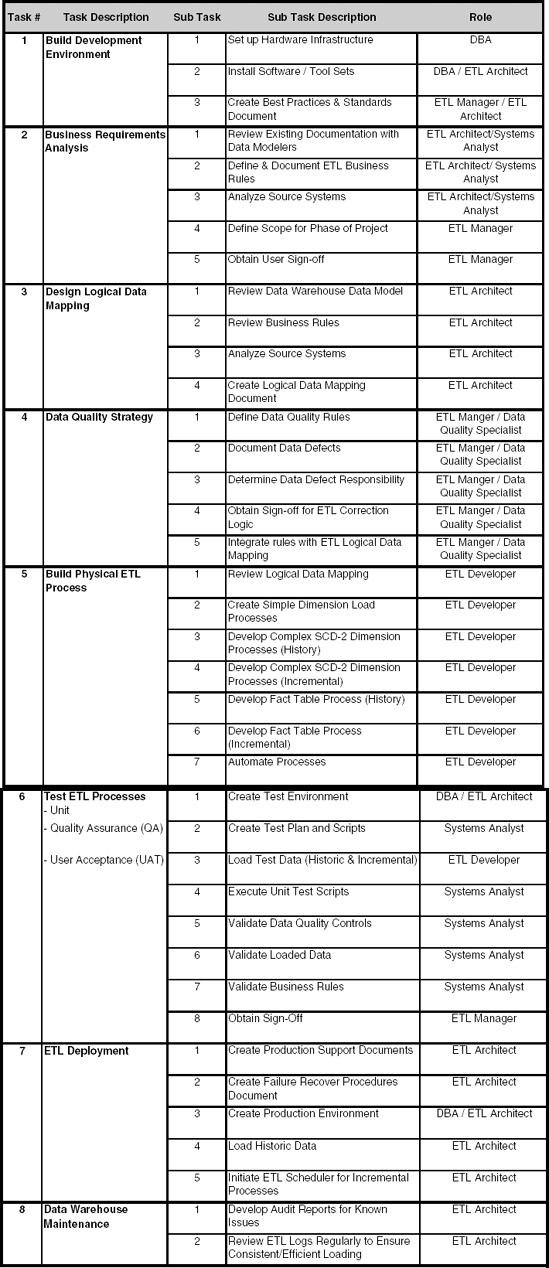
Building the Development Environment

To perform thorough data analysis and begin ETL development of any source system, it is good practice to have the DBA team build a development environment. Use of a separate development environment guarantees that data analysis and ETL development will not affect the production transaction system. Once the environment has been set up, the ETL architect and the DBA team work together to install the appropriate software and tool sets required to perform the analysis and development activities for your team.

Be sure to document the course of actions required and create the development environment during your first phase of the project. Documenting standards from lessons learned minimizes future errors and risk to the added systems during subsequent iterations.

Business Requirements Analysis

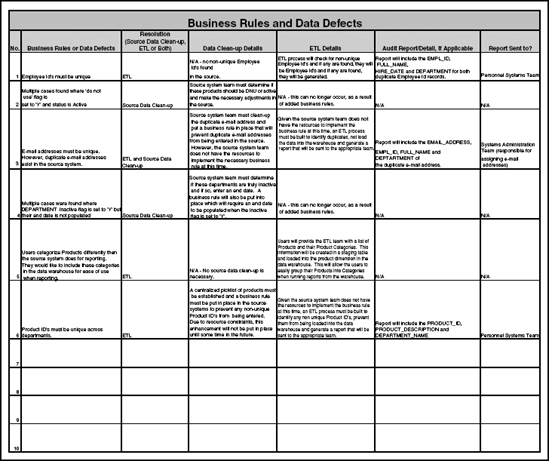
Although many of the business rules have been documented through analysis during data-modeling sessions, the ETL architect's is responsibility to take those rules to their completion. Typically, the ETL architect and the systems analyst review all existing documentation and meet with data modelers to discuss questions that arise.



**Figure 10.3. ETL Project Plan**

It is critical that the ETL architect and systems analyst have a solid understanding of the source systems and the data inside them. Be sure not to underestimate the time needed to complete this analysis, and keep in mind that the logical data mapping cannot be created until the source systems have been thoroughly analyzed. It is not uncommon for the ETL architect and systems analyst to meet with the data modelers, the source system DBAs, or system analysts for multiple sessions, depending on scope, to review details of the source systems. These sessions will facilitate the findings of business rules that will be used to build the logical data mappings and finally code the ETL process.

[Figure 10.4](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html#business_rules_and_data_defect_tracking) is a sample template for gathering and documenting business rules and data defects. The spreadsheet is broken out as such to allow the tracking of either data clean up or ETL details or both. We have combined the two into one template because the ETL architect typically has to tackle the business rules and data clean-up transformations simultaneously for a cohesive, integrated solution. It is important that these rules and transformations are thoroughly documented in detail, not only for purposes of coding but also because this document is the foundation of the creation of unit, system, QA, and user acceptance testing test cases. This metadata is also used for end user training and procedures documents.



**Figure 10.4. Business rules and data defect tracking spreadsheet.**

In theory, emphasis is always placed on documentation. Unfortunately, in reality it is common for project teams to start off with good intentions by creating the documentation, but they rarely go back and update the documents as things change. Do not fall into this trap. Keeping documentation up to date is crucial to the success of your project. Up-to-date documentation is mandatory to perform detailed impact analysis of future enhancements and subsequent phases of your project. Moreover, current documentation ensures that you have a handle on the data lineage of your warehouse. Maintaining your documentation may require time and effort, but consider the alternative: going back and trying to figure out what has changed within ETL processes, business rules, or data after the fact. It doesn't take a lot of imagination to visualize the wasted time, increased costs, and pure frustration that can be avoided by planning ahead and updating your documentation as modifications to your ETL processes are made.

Defining the Scope of the ETL project

Defining scope includes determining and documenting what will be included for each phase of the ETL process as it relates to subject areas, business rules, transformation, and data-cleansing strategies. It can also, and usually does, indicate what is not included in the phase. Documenting the scope of each phase and requiring business users to review and sign-off the scope documents aids in your management and prevents scope-creep.

Be realistic when defining each phase. Although your users expect you to fulfill your commitment, they will most likely make many changes and additions throughout the lifecycle of the phase. Changes to scope must be negotiated and prioritized, leaving low-priority changes for future phases. Keep potential scope-creep items on your radar when finalizing the scope documentation.

After business rules have been documented and scope has been defined, have a user-walkthrough of the documentation and obtain sign-off by the business users. Techniques for managing scope are discussed in the "Managing Scope" section of this chapter.

Designing the Logical Data Map

To help facilitate the design of the logical data map, the ETL architect must review the data warehouse data model and all business-rules documentation. Additional meetings may be needed to get answers for any remaining open questions and/or issues. If the existing business rules gathered during the business-requirements-analysis phase do not provide enough detail, the ETL architect needs to analyze the source systems manually. Once all questions and issues are resolved, the ETL architect creates the logical data-mapping document.

NOTE

CROSS-REFERENCE [Chapter 3](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch03.html) contains the exact details involved in creating the logical data map.

Defining a Data Quality Strategy

Other than being query friendly, the chief acceptance factor of any data warehouse is that the data is consistent and reliable. Therefore, conducting a data-quality assessment and defining a strategy is a vital part of the ETL process.

The ETL manager and the data-quality specialist are jointly responsible for defining data-quality rules. They are tasked with analyzing the quality of the source system data and documenting all identified data defects. This exercise not only ensures that the cleanest possible data is entering your data warehouse; it also benefits the source system from a data-quality perspective. This analysis exposes flaws in the source system applications and gives source system administrators the opportunity to make corrections to their application to prevent future data defects.

Options for cleaning data usually fall into two categories:

* Cleanse data at the source.
* Transform data in the ETL.

Cleansing data at the source is the most desirable and beneficial option. Unfortunately, it may be neither feasible, due to resource constraints, nor timely, due to the transaction application development lifecycle complexity and schedule. Additionally, data cleanup usually involves political navigation to settle on appropriate data defect correction activities.

After all details and deadlines are committed and agreed upon, document the details of each data-cleanup issue and associated cleanup resolution. Creating a project plan to track the issues to be cleansed in the source systems as well those to be cleansed by the ETL process will help you manage user expectations regarding data defects.

We recommend setting up weekly meetings with the administrators responsible for source system data cleanup. Use these meetings to review and update the project plan to accurately reflect progress and discuss any new findings. These meetings will determine the feasibility of cleanup and will provide a forum to agree on a strategy for cleaning up new findings. Be sure to stay on top of progress made to source-data cleansing, as your data warehouse is depending on source data being clean. It is a good idea to make the extra effort to query the source database to ensure the cleanup effort was successful.

NOTE

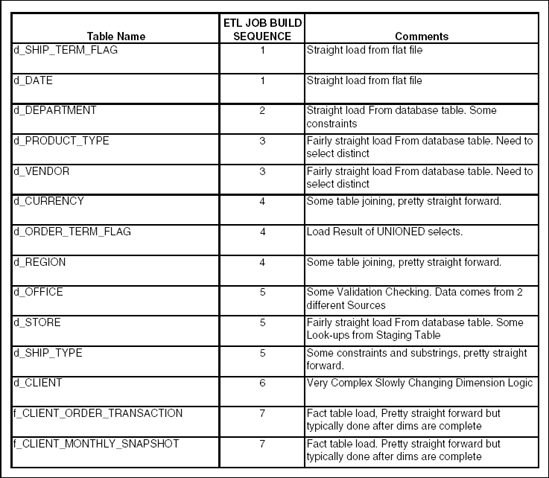
Although data is being cleansed in the source system, the source system owners may not have the ability to add business rules to prevent data from becoming dirty again. Therefore, it is a good idea for the systems analyst and ETL architect to meet to determine whether ETL code is necessary to prevent dirty data from entering the data warehouse. ETL code can either kick out or transform dirty data, depending on the business rules.

If ETL code is used as a preventative measure, whether through exclusion or transformation, it is a good idea to define audit reports for the ETL processes to capture and report dirty data. Such reports aid in the continual cleanup of the source data and provide a mechanism to tie the corrected data in the data warehouse back to its source. This metadata also serves as an audit trail that provides the ability to trace a data discrepancy to its place of origin, identifying the data owner responsible for its cleanup. Be sure to obtain user sign-off on the business rules and data cleanup logic being handled by the ETL.

Building the Physical ETL Process

Once the data analysis is complete and the business rules and logical data mappings are final, the ETL architect walks through the logical data mapping with the assigned ETL developer. This walkthrough ensures that the ETL developer understands the complete requirements before he or she begins coding. The ETL developer is responsible for forward engineering the logical data mapping into physical ETL routines. Whether SQL scripts are written or a dedicated ETL tool is used, the routines must be developed and tested and the resulting data must be validated by the developer before they are turned over to the ETL architect for migration.

When several routines are given to the developer at once, which is usually the case, an *ETL build sequence document* is usually prepared by the ETL architect for the developer to use as a guide. Shown in [Figure 10.5](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html#etl_build_sequence_document), the document contains a listing of the expected tables to be loaded, the ordinal position to build the processes, and comments as to what challenges should be expected in the routine. This document is especially important during the first phase of the project or for developers new to your team.



**Figure 10.5. ETL build sequence document**

Testing the ETL Processes

Most systems' lifecycle methodologies include three phases of testing. During your ETL, it is recommended that you follow the three-phase approach when going live with new source systems, subject areas, or any major release. Following are the three types of testing that should be conducted with each phase of your ETL project.

* **Unit Testing**. This testing occurs during and after development before going to QA testing. This testing is performed by the ETL developer and the systems analyst in the development environment.
* **Quality Assurance Testing (QA)**. This is the testing that typically occurs by a separate group within your organization in a separate environment mirroring production. The environment is created and controlled by the DBA and QA team members. This environment will be used to ensure all ETL processes are performing as expected, meeting all business rules and timeframe (load window) requirements. Given that it simulates the production environment, the QA group can validate that the ETL processes will work in production.
* **User Acceptance Testing (UAT)**. This phase typically occurs by your user group in a separate controlled environment created from the QA environment. This database is controlled by the DBA team members. In smaller organizations, after QA testing is complete, it is acceptable to open the environment to users for user-acceptance testing, reducing the cost of infrastructure maintenance and hardware. UAT is the testing phase that benefits the team by letting users have a hands-on look at the data to ensure processes are running as expected. At the end of UAT, obtain sign-off from your users. Once sign-off is received, you are ready to move to production.

NOTE

We've been on projects where the user-acceptance testing phase is bypassed for small build releases and bug fixes, going directly from quality-assurance testing to production. In these cases, users inevitably detect issues after code has been pushed into production. Excluding the user-acceptance testing phase is a short cut that prevents you from discovering issues that only a user might find before it is too late: in production.

When testing new ETL processes, be sure to have users test for known data issues and source system anomalies. Not only will this validate your efforts; exposure to the clean data will excite your users and make them eager to use the new data warehouse. Clean data tends to have some positive effects. Users will enthusiastically spread the word of the success of the ETL and data warehouse project, causing other subject areas to flock to the data warehouse project manager begging to be next in line for their data to be transformed and loaded into the data warehouse.

Developing Test Cases

While ETL development is taking place, using the business rules and data defects document, the systems analyst and ETL architect are jointly responsible for developing detailed test plans for unit testing, QA testing, and UAT.

Test plans should include cases that test all business-rule scenarios. Validating test results against expected results ensures that the ETL code is correct and the transformations are working as designed. Your test cases should deliberately try to load poor data into the data warehouse. The ETL process should either prevent data from entering or transform data and load it. In either case, an audit report should be generated. Even when poor data is not intentionally loaded, be sure to include queries that test data quality in the data warehouse to ensure that data-cleansing transformations are working as expected.

Most likely, issues will be identified during the validation of the test cases. Some of these issues may be bugs discovered in your code, and some may be fresh ideas triggered by the users' exposure to their data in a format that is new to them. It's not uncommon to receive user requests for new requirements during this phase that may need to be added as enhancements. Be careful: Data is not the only thing being tested here. Managing the initial ETL processes is a task in itself; add on bug fixes, additional requests, and ever-changing business rules and the process can become completely unmanageable. In-depth change management techniques are detailed in the "Managing Scope" section of this chapter.

A sample test case template is illustrated in [Figure 10.6](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html#test_case_template). It is intended to capture the requirements you are testing; the detailed steps to perform the test; the expected results; and the status of the test: pass or fail. Sample test cases are given to display the level of detail you should capture. This template should be used for all three phases of the testing process.

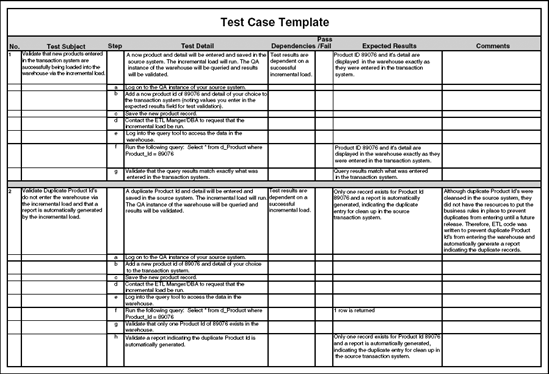
ETL Deployment

Next comes the moment you have all been waiting for: ETL deployment. To make the migration to production as seamless as possible, be sure to create production support documents. These documents should include following information:

* The final lineage report
* Procedures for running (and restarting) the incremental load process
* Details about the automated load schedule

It is important to create and deliver documented failure recovery procedures. Should a load process fail, users could have access to bad data or data that is not up to date. A plan must be in place to avoid this before the production environment is unleashed to users. Document and test your failure recovery procedures, so that when failures occur, you can quickly recover data and make it available for your users in a timely manner.

Work with the DBA team to create a stable production environment. Load your historical data and kick off the ETL incremental load processes with your production scheduler. Be sure to run tests on data in production (historical and incremental) to ensure data was successfully loaded.



**Figure 10.6. Test case template**

Maintaining the Data Warehouse (ETL)

Depending on how your organization is structured, the data warehouse project manager and DBA team are typically responsible for the ongoing maintenance of the data warehouse. However, you are the owner of the ETL process, and unless other arrangements have been made, its ongoing maintenance is your responsibility.

After you go *live* in production, it is important to continuously monitor your data warehouse for known content issues. Part of this maintenance includes the development of audit reports that will capture known issues. These audit reports should stem from the business rules and data defects document. The reports can automatically be sent to the appropriate contact personnel for action via e-mail.

Patches and upgrades are inevitable in any production IT environment. Such patches and upgrades are especially relevant in the data warehouse environment, where so many distinct tool sets are integrated for a single solution. Be diligent in applying patches and upgrades as necessary. It is recommended that you schedule regular system maintenance and perform these upgrades during this time. All patches and upgrades must go through the full development lifecycle, including unit testing in the development environment, QA testing, and user-acceptance testing. Passing the patches and upgrades through testing ensures that maintenance was performed correctly and that all processes are running as expected.

Keep your users abreast of new releases or enhancements as they are being rolled out to production. That communication helps users prepare for changes as they enter the data warehouse. Your users could be waiting for a specific release or enhancement. Giving them a heads up on the time frame of scheduled releases will boost their experience with the data warehouse.

Managing Scope

It won't be far into the project when you realize why defining scope and obtaining sign-off is so important. It's easy to lose control when you are trying to tackle the overwhelming bombardment of change requests.

Unmanaged ad-hoc changes to the ETL specifications can be detrimental to the success of the project. It is common to receive additional requirements during the development and testing phases. Issues will certainly be found and new ideas will most likely surface, all of which need to be implemented *immediately*. In our experience, when the data warehouse is unveiled, new wish lists and requirements excitedly begin to trickle in, picking up momentum exponentially as more subject areas are deployed. Before you know it, you will be bombarded with more work than you and your team can handle. Did someone say scope-creep? Creating a mechanism for tracking and managing these changes is crucial to your success. The next section provides the documents you need to track changes and recommends procedures that help you execute them.

Change Tracking

Implementing a process to track enhancement requests, bug fixes, or changes to the initially agreed scope is essential to your success as the ETL manager. Following is a list of elements that have proven to be significant while capturing and tracking change requests. You will want the ability to track and manage the following information even if it means creating a simple spreadsheet to do it. Capturing the following elements aids in the management of changes and helps minimize scope-creep.

NOTE

A system could easily be built in Microsoft Access or any small personal database application for this purpose. For larger groups, you can implement a small Visual Basic application or can leverage packaged systems that your organization has already invested in. And, of course, a good ETL tool may provide this capability.

* **Subject Area**. This is the name of the data mart (portion of the data warehouse) the request is being submitted for.
* **Request Date**. The date the request originates
* **Change Description**. This should capture a high-level description of the request.
* **Priority**. High, Medium, or Low. This is a negotiated rating of the importance of the request.
* **Change Type**. Indicates whether the request is for a new requirement or a change to an existing process
* **Status**. Status values can include anything that identifies the state of the request. Values we've used include New request, Developer investigating, Developer developing, More information needed, Cancelled, Passed unit testing, Passed QA, Passed UAT, Ready for production, and so on. The values in this field change throughout the life of the request.
* **Submitter**. The name of the person submitting the change request
* **Owner**. Indicates the person responsible for the request at a specific time. It usually begins with the data warehouse or ETL manager and then gets assigned to the appropriate developer, tester, or so on throughout the life of the request.
* **Version Found in**. This is the active version number at the time the bug is detected or the request is submitted.
* **Version Fixed in**. This is the number of the version that the request is packaged with when released to production.
* **State**. Open or Closed. Closed should be selected only when the Status is set to Released to production or Cancelled.
* **Date Closed**. This field should be populated at the time the state field is set to Closed.
* **Functional Description**. The information provided here should describe the user's experience prompting the request.
* **Technical Description**. The information provided here is usually filled in by power users or the ETL architect. It is used by the developer for coding the change or new requirement.

Having the ability to generate reports using the elements from this list is advantageous. If you have the resources to develop this as an IT system, consult with your team to get their input on the process, making sure the process meets the needs of everyone on your team.

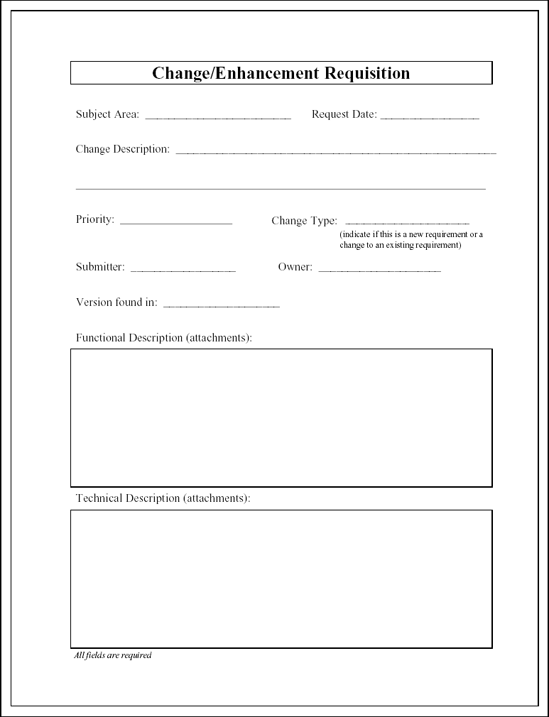
However, minimizing changes to the production data warehouse environment is a good practice, if you can achieve it. Keep in mind, the more changes made, the greater the risk of affecting other processes. We recommend holding regularly scheduled meetings with appropriate team members to discuss the priority of each request. It is important that you make your users realize how critical it is to minimize changes to production. Make sure they understand the cost and effort it takes to fulfill each request. Have them ask themselves the following questions: What is their return on investment? Are the benefits of the changes worth making? Can they justify the change requests?

Once it is agreed that a change should be made, you must discuss the impact of the change. If the change affects another ETL process or another area, a detailed impact analysis must occur. Proposed changes can result in multiple new changes to existing ETL processes. Be sure to add these changes to your new tracking system.

A sample change/enhancement requisition form is shown in [Figure 10.7](https://www.safaribooksonline.com/library/view/the-data-warehouse/9780764567575/ch10.html#change_solidus_enhancement_requisition_f). This form includes all of the necessary information you need to enter a new request and perform development to complete the request. This form, in conjunction with your change-request tracking system, supports effective management of the change-request process.

Scheduling Version Releases

Typically, a number of changes, enhancement requests, patches, and upgrades are bundled together as a single build or release. Each release must go through the full development lifecycle. Before going to production, be sure to unit, QA, and UAT test your changes. After the changes have passed the testing cycles, either the ETL architect migrates the routines or the DBA team pushes the code to production.



**Figure 10.7. Change/enhancement requisition form**

Tracking versions of your data warehouse is beneficial for troubleshooting problems discovered in production. Use the tracking mechanisms outlined earlier in this chapter to maintain control over your version releases. Normally, following standard-versioning techniques works well in the data warehouse/ETL environment. It is especially important for the ETL manager to adhere to this standard because much of the data warehouse code releases to production are created and deployed by the ETL team.

The version number consists of a series of three decimal delimited numbers (##.##.##). The first set of numbers signifies major releases; the second, minor releases; and the third, patches. For example, Version 1.2.1 means the data warehouse is in its first version and there have been two minor releases and one patch applied to it.

In the data warehouse environment, a *major version release* typically constitutes a new subject area or data mart that includes new facts, dimensions, and ETL processes. a *minor release* is defined as primarily ETL modifications, possibly including some minor structural database changes. *Patches* are usually a result of a *hot fix*, where a mission-critical error has been detected in the production environment and needs to be corrected immediately. If patches are bundled with minor changes or minor changes with a major, only the leftmost number in the series should be incremented and the right-hand numbers are reset. For example, if version 1.2.1 is in production and you have two patches, a minor change, and a major release scheduled for migration, bundling these changes would be considered a single major release. In this case, you would now be at release 2.0.0.

It is good practice to bundle and schedule major releases with enough time between to address hot fixes. With scheduled major releases, perhaps monthly, it is easier to bundle minor fixes into the controlled release environment to minimize code migrations.

Our recommended data warehouse versioning strategy is especially powerful when your project is using the data warehouse bus architecture. In such a case, each data mart in the bus matrix will be a major version release as it enters the physical data warehouse. If your data warehouse is at version 1.0.210, you are most likely not using this matrix and probably not sleeping at night, either.

Summary

In this chapter, we have finally stepped back a little from the myriad tasks of the ETL team to try to paint a picture of who the players are and what are they supposed to think about. We must keep in mind that this chapter and really the whole book are deliberately limited to the back-room concerns of the enterprise data warehouse.

We began by describing the planning and leadership challenges faced by the ETL team; then we descended into the specific tasks that these people face. In many cases, much more detail is provided in the main text of the book.