# Kimball Data Warehouse Toolkit

## Ch 19 – ETL Subsystems and Techniques

* The **ETL system consumes a disproportionate share of the time + effort required to build a DW/BI environment**
* **Development is challenging b/c so many outside constraints put pressure on its design: business requirements, source data realities, budget, processing windows, + skill sets of the available staff**
* Yet it can be hard to appreciate just why the ETL system is so complex + resource-intensive.
* Everyone understands the 3 letters: You get the data out of its original source location (E), do something to it (T), + then load it (L) into a final set of tables for the business users to query
* When asked about the best way to design + build an ETL system, many designers say, “Well, that depends.”
* It **depends on the source, on limitations of the data, on the scripting languages + ETL tools available, on the staff ’s skills, + on the BI tools**
* But the **“it depends” response = dangerous b/c it becomes an excuse to take an unstructured approach to developing an ETL system, which in the worse-case scenario results in an undifferentiated mess of tables, modules, processes, scripts, triggers, alerts, + job schedules**
* **This “creative” design approach should not be tolerated**
* With the wisdom of hindsight from thousands of successful DWs, a set of ETL best practices have emerged, so there is no reason to tolerate an unstructured approach.
* Careful consideration of these best practices has revealed **34 subsystems are required in almost every dimensional DW back-room**
* No wonder the ETL system takes such a large % of the DW/BI development resources
* *This chapter is drawn from The Data Warehouse Lifecycle Toolkit, Second Edition (Wiley, 2008)*
* **Concepts:**
* **Requirements + constraints** to be considered ***before* designing** the ETL system
* 3 subsystems focused on extracting data from source systems
* 5 subsystems to deal with value-added cleaning + conforming, including dimensional structures to monitor quality errors
* 13 subsystems to deliver data into now-familiar dimensional structures, such as a subsystem to implement techniques
* 13 subsystems to help manage the production ETL environment

### Round Up the Requirements

* **Establishing the architecture of an ETL system begins with one of the toughest challenges: rounding up the requirements** 🡪 gathering + understanding *all* the known **requirements, realities, + constraints affecting the ETL system**
* The list of requirements can be pretty overwhelming, but it’s **essential to lay them on the table *before* launching into the development of the ETL system**
* The **ETL system requirements are mostly constraints you must live with + adapt your system to**
* Within the framework of these requirements, there are opportunities to make your own decisions, exercise judgment, + leverage creativity, but **the requirements dictate the core elements that the ETL system must deliver**
* The following **10 sections describe the major requirements areas** that impact the design + development of the ETL system.
* Before launching the ETL design + development effort, provide a short response for each of the following ten requirements
* Point of this exercise = *any one of them can be a showstopper at some point in the project*

#### Business Needs

* From an ETL designer’s view, **business needs = the DW/BI system users’ information requirements**
* We use the term “*business needs”* somewhat narrowly here to mean **the information content that business users need to make informed business decisions**
* Because **business needs *directly* drive the choice of data sources + their subsequent transformation** in the ETL system, ETL team *must* understand + carefully examine them
* **NOTE: Maintain a list of the KPIs uncovered during the business requirements definition that the project intends to support, as well as the drill-down + drill-across targets required when a business user needs to investigate “why?” a KPI changed**

#### Compliance

* Changing legal + reporting requirements have forced many organizations to tighten their reporting + provide proof that reported numbers are: accurate, complete, + have not been tampered with
* Of course, DW/BI systems in regulated businesses, such as telecommunications, have complied w/ regulatory reporting requirements for years
* But certainly the whole tenor of financial reporting has become much more rigorous for everyone
* **NOTE**: In consultation with your legal department or chief compliance officer (if you have one) + the BI delivery team, **list all data and final reports subject to compliance restrictions**
* List those data inputs + data transformation steps for which you must maintain the **“chain of custody”** showing + proving that final reports were derived from the original data delivered from your data sources
* List the data that you must provide **proof of security** for the copies under your control, both offline + online
* List those data copies you must **archive**, + list the **expected usable lifetime** of those archives

#### Data Quality

* 3 powerful forces have converged to put data quality concerns near the top of the list for executives
* 1) The long-term cultural trend that says, “If only I could see the data, then I could manage my business better” continues to grow
* Today’s knowledge workers believe instinctively that **data is a crucial requirement for them to function in their jobs**
* 2) Most organizations understand their **data sources are profoundly distributed, typically around the world, + that effectively integrating a myriad of disparate data sources is required**
* 3) The **sharply-increased demands for compliance** mean careless handling of data will not be overlooked or excused.
* **NOTE:** List those data elements whose quality is known to be unacceptable, + list whether an agreement has been reached with the source systems to correct the data before extraction
* List those data elements discovered during data profiling, which will be continuously monitored and flagged as part of the ETL process

#### Security

* Security awareness has increased significantly in the last few years across IT but often remains an afterthought + an unwelcome burden to most DW/BI teams.
* The basic rhythms of the DW are at odds with the security mentality
* A DW seeks to publish data widely to decision makers, whereas the security interests assume data should be restricted to those with a need to know
* Additionally, security must be extended to physical backups
* If the media can easily be removed from the backup vault, then security has been compromised as effectively as if the online passwords were compromised.
* **During requirements roundup, the DW/BI team should seek clear guidance from senior management as to what aspects of the DW/BI system carry extra security sensitivity**
* If these issues have never been examined, it’s likely the question will be tossed back to the team
* That’s the moment when an experienced security manager should be invited to join the design team
* **Compliance requirements are likely to overlap security requirements**, so it may be wise to combine these 2 topics during the requirements roundup
* **NOTE:** Expand the compliance checklist to encompass known security + privacy requirements

#### Data Integration

* Data integration = a huge topic for IT because, ultimately, it aims to make all systems seamlessly work together
* The “360 degree view of the enterprise” is a familiar name for data integration
* In many cases, **serious data integration must take place among the organization’s primary transaction systems before data arrives at the DW’s back door**
* But **rarely is that data integration complete, unless the organization has a comprehensive and centralized master data management (MDM) system, + *even then* it’s likely other important operational systems exist outside the primary MDM system**
* **Data integration usually takes the form of conforming dimensions + conforming facts in the DW**
* **Conforming dimensions = establishing common dimensional attributes across separated databases, so drill-across reports can be generated using these attributes**
* **Conforming facts = making agreements on common business metrics such as KPIs across separated databases so these numbers can be compared mathematically by calculating differences + ratios**
* **NOTE: Use the bus matrix of business processes to generate a priority list for conforming dimensions (columns of the bus matrix)**
* **Annotate each row of the bus matrix with whether there is a clear executive demand for the business process to participate in the integration process, + whether the ETL team responsible for that business process has agreed**

#### Data Latency

* **Data latency** = **how quickly source system data must be delivered to the business users via the DW/BI system**
* Obviously, data latency requirements **have a huge effect on the ETL architecture**
* **Clever processing algorithms, parallelization, + potent hardware can speed up traditional batch-oriented data flows**
* But at some point**, if the data latency requirement is sufficiently urgent, the ETL system’s architecture must convert from batch to micro-batch or streaming-oriented**
* This switch isn’t a gradual or evolutionary change, but **a major paradigm shift in which almost every step of the data delivery pipeline must be re-implemented**
* **NOTE**: **List all legitimate + well-vetted business demands for data that must be provided on a daily basis, on a many times per day basis, w/in a few seconds, or instantaneously**
* **Annotate each demand with whether the business community understands the data quality trade-offs associated with their particular choice**
* Chapter 20 discusses data quality compromises caused by low latency requirements

#### Archiving and Lineage

* Archiving + lineage requirements were hinted at in the previous compliance + security sections
* Even without the legal requirements for saving data, **every DW needs *various* copies of old data, either for comparisons w/ new data to generate change capture records or reprocessing**
* Recommended to **stage the data (writing to disk) after each major activity of an ETL pipeline**: after it’s been extracted, cleaned + conformed, + then delivered
* So, when does staging turn into **archiving** = **data kept indefinitely on some form of permanent media?**
* **Simple answer** = a conservative answer 🡪 *All* **staged data should be archived unless a conscious decision is made that specific data sets will never be recovered in the future**
* **Almost always less problematic to read data from permanent media than to reprocess the data through the ETL system at a later time**
* And, of course, it **may be impossible to reprocess the data according to the old processing algorithms if enough time has passed or the original extraction cannot be re-created**
* And while we’re at it, **each staged/archived data set should have accompanying metadata describing the origins + processing steps that produced the data**
* Again, **the tracking of this lineage is explicitly required by certain compliance requirements but should be part of every archiving situation**
* **NOTE:** **List the data sources + intermediate data steps that will be archived, together w/ retention policies, and compliance, security, + privacy constraints**

#### BI Delivery Interfaces

* **Final step for the ETL system = the handoff to the BI applications**
* Take a strong + disciplined position on this handoff 🡪 the **ETL team, working closely w/ the modeling team, must take responsibility for the content + structure of the data that makes the BI applications simple + fast**
* This attitude is more than a vague motherhood statement 🡪 **it’s irresponsible to hand off data to the BI application in such a way as to increase the complexity of the application, slow down query or report creation, or make the data seem unnecessarily complex to the business users**
* The most elementary + serious error is to hand across a full-blown, normalized physical model + walk away from the job
* This is why we go to such lengths to **build *dimensional* structures that comprise the final handoff**
* The **ETL team + data modelers need to closely work w/ the BI application developers to determine the *exact* requirements for the data handoff**
* Each BI tool has certain sensitivities that should be avoided + certain features that can be exploited if the physical data is in the right format
* *The same considerations apply to data prepared for OLAP cubes*
* **NOTE**: **List all fact + dimension tables that will be *directly* exposed to your BI tools**
* This **should come directly from the dimensional model specification**
* Also, **list all OLAP cubes + special database structures required by BI tools**
* And **list all known indexes + aggregations you have agreed to build to support BI performance**

#### Available Skills

* Some ETL system design decisions must be made on the basis of available resources to build + manage the system
* *Don’t build a system that depends on critical C++ processing modules if those programming skills aren’t in-house or can’t be reasonably acquired*
* Likewise, you may be much more confident in building the ETL system around a major vendor’s ETL tool if you already have those skills in-house and know how to manage such a project
* **Big decision = whether to hand-code the ETL system or use a vendor’s ETL package**
* Technical issues + license costs aside, don’t go off in a direction that your employees + managers find unfamiliar without **seriously considering the decision’s long-term implications**
* **NOTE: Inventory your department’s OS, ETL tool, scripting language, programming language, SQL, DBMS, + OLAP skills so you understand how exposed you are to a shortage or loss of these skills**
* List those skills required to support your current systems and your likely *future* systems

#### Legacy Licenses

* Finally, in many cases, major design decisions will be made implicitly by senior management’s insistence that you use existing **legacy licenses**
* In many cases, this requirement is one you can live with because the **environmental advantages are clear to everyone**
* But **in a few cases, the use of a legacy license for ETL development is a mistake**
* This is a difficult position to be in, + if you feel strongly enough, you may need to bet your job
* If you must approach senior management + challenge the use of an existing legacy license, be well prepared in making the case, + be willing to accept the final decision or possibly seek employment elsewhere
* **NOTE:** List your *legacy* OS, ETL tool, scripting language, programming language, SQL, DBMS, and OLAP licenses + whether their exclusive use is mandated or merely recommended

### The 34 Subsystems of ETL

* With an understanding of the **existing requirements, realities, + constraints**, you’re ready to learn about the **34 critical subsystems that form the architecture for every ETL system**
* This chapter describes all 34 subsystems with **equal emphasis**
* Next chapter describes the practical steps of implementing those subsystems needed for each particular situation
* Although using the industry vernacular (ETL) to describe these steps, the **process really has 4 major components:**
* **Extracting**
* Gathering raw data from the source systems + usually writing it to disk in the ETL environment *before any significant restructuring of the data takes place*
* Subsystems 1-3 support the extracting process
* **Cleaning + conforming**
* Sending source data through a series of processing steps in the ETL system to improve the quality of the data received from the source, + merging data from 2+ sources to create + enforce conformed dimensions + conformed metrics
* Subsystems 4-8 describe the architecture required to support the cleaning + conforming processes
* **Delivering**
* Physically structuring + loading the data into the presentation server’s target dimensional models
* Subsystems 9-21 provide the capabilities for delivering the data to the presentation server
* **Managing**
* Managing the related systems + processes of the ETL environment in a coherent manner
* Subsystems 22-34 describe the components needed to support the ongoing management of the ETL system

### Extracting: Getting Data into the Data Warehouse

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

#### Subsystem 1: Data Profiling

* **Data profiling** = **the technical analysis of data to describe its content, consistency, + structure**
* In some sense, any time you perform a SELECT DISTINCT investigative query on a database field, you are doing data profiling
* There are a variety of tools specifically designed to do powerful profiling
* It **probably pays to invest in a tool rather than roll your own b/c the tools enable many data relationships to be easily explored with simple UI gestures**
* Can be much more productive in the data profiling stages of a project using a tool rather than hand coding all the data content questions.
* Data profiling plays 2 distinct roles: **strategic** and **tactical**
* As soon as a candidate data source is identified, a light profiling assessment should be made to determine its suitability for inclusion in the DW + provide an early go/no go decision
* Ideally, this **strategic assessment** **should occur immediately after identifying a candidate data source during the business requirements analysis**
* Early disqualification of a data source is a responsible step that can earn you respect from the rest of the team, even if it is bad news
* A late revelation that the data source doesn’t support the mission can knock the DW/BI initiative off its tracks (+ be a potentially fatal career outcome), especially if this revelation occurs months into a project
* After the basic strategic decision is made to include a data source in the project, a lengthy **tactical data profiling effort** should occur to squeeze out as many problems as possible
* Usually, this task begins during the data modeling process + extends into the ETL system design process
* Sometimes, the ETL team is expected to include a source w/ content that hasn’t been thoroughly evaluated
* Systems may support the needs of the production processes, yet present ETL challenges, because fields that aren’t central to production processing may be unreliable + incomplete for analysis purposes
* Issues that show up in this subsystem result in detailed specifications that are either
* 1) Sent back to the originator of the data source as requests for improvement or
* 2) Form requirements for the data quality processing described in subsystems 4-8
* The **profiling step provides the ETL team with guidance as to how much data-cleaning machinery to invoke + protects them from missing major project milestones due to the unexpected diversion of building systems to deal with dirty data**
* **Do data profiling upfront** + **use the results to set the business sponsors’ expectations regarding realistic development schedules, limitations in the source data, + the need to invest in better source data capture practices**

#### Subsystem 2: Change Data Capture System

* During the DW’s initial historic load, capturing source data content changes is not important because you load all data from a point in time forward
* However, **many DW tables are so large that they cannot be refreshed during every ETL cycle**
* **Must have a capability to transfer only the *relevant* changes to the source data since the last update**
* **Isolating the *latest* source data is called change data capture (CDC)**
* Idea behind CDC is simple enough: **Just transfer the data that has changed since the last load**
* But **building a good CDC system is not as easy as it sounds**
* **Key goals for the CDC subsystem are**:
* **Isolate the *changed* source data** to allow selective processing rather than a complete refresh
* **Capture *all* changes** (deletions, edits, + insertions) made to the source data, including changes made through nonstandard interfaces
* **Tag changed data with reason codes** to distinguish error corrections from true updates
* **Support compliance tracking** with additional metadata.
* **Perform the CDC step as early as possible**, preferably before a bulk data transfer to the DW
* Capturing data changes is far from a trivial task
* **Must carefully evaluate your strategy for *each* data source**
* Determining the appropriate strategy to identify changed data may take some detective work
* Data profiling tasks described earlier can help the ETL team make this determination
* There are **several ways to capture source data changes**, each effective in the appropriate situation, including:
* **Audit Columns**
* In *some* cases, the source system includes **audit columns** that **store the date + time a record was added or modified**
* These columns are **usually populated via database triggers** that are fired off automatically as records are inserted or updated
* ***Sometimes***, for performance reasons, the columns are **populated by the source application** instead of database triggers
* **When these fields are loaded by any means *other than* database triggers, pay special attention to their integrity, analyzing + testing each column to ensure it’s a reliable source to indicate change**
* **If you uncover any NULL values, must find an alternative approach for detecting change**
* **Most common situation that prevents the ETL system from using audit columns = when the fields are populated by the source application, BUT the DBA team allows back-end scripts to modify data**
* If this occurs in your environment, you face a **high risk of missing changed data during the incremental loads**
* Finally, **need to understand what happens when a record is *deleted* from the source because querying the audit column may not capture this event**
* **Timed Extracts**
* With a **timed extract**, you **typically select all rows where the create or modified date fields equal SYSDATE – 1 (meaning all of yesterday’s records)**
* However, **loading records based purely on time is a common mistake made by inexperienced ETL developers** 🡪 **This process is horribly unreliable**
* Time-based data selection **loads duplicate rows when restarted from mid-process failures**
* Means **manual intervention + data cleanup is required if a process fails for any reason**
* Meanwhile, if the **nightly load process fails to run + skips a day, there’s a risk that the missed data will never make it into the DW**
* **Full Diff Compare**
* A **full diff compare****keeps a full snapshot of yesterday’s data, + compares it, record by record, against today’s data to find what changed**
* Good news = this technique is thorough: You are ***guaranteed* to find every change**
* Obvious bad news = in many cases, this technique is **very resource-intensive**
* If a full diff compare is required, **try to do the comparison on the source machine, so you don’t have to transfer the entire table or database into the ETL environment**
* *Of course, the source support folks may have an opinion about this*
* Also, investigate using **cyclic redundancy checksum (CRC) algorithms** to quickly tell if a complex record has changed without examining each individual field
* **Database Log Scraping**
* **Log scraping** **effectively takes a snapshot of the database redo log at a scheduled point in time (usually midnight) + scours it for transactions affecting the tables of interest for the ETL load**
* **Sniffing involves a polling of the redo log, capturing transactions on-the-fly**
* **Scraping** the log for transactions is **probably the messiest of all techniques**
* It’s not uncommon for transaction logs to get full + prevent new transactions from processing
* When this happens in a production transaction environment, knee-jerk reaction from the DBA may be to empty the log so that business operations can resume
* **But when a log is emptied, all transactions within them are lost**
* If you’ve exhausted all other techniques + find log scraping is your last resort for finding new or changed records, persuade the DBA to create a special log to meet your specific needs
* **Message Queue Monitoring**
* In a message-based transaction system, the **queue is monitored for all transactions against the tables of interest**
* The **contents of the stream are similar to what you get with log sniffing**
* One benefit of this process = **relatively low overhead***, assuming the message queue is already in place*
* *However*, **there may be no replay feature on the message queue**
* If the connection to the message queue is lost, you lose data

#### Subsystem 3: Extract System

* Obviously, extracting data from source systems = a fundamental component of an ETL architecture
* If extremely lucky, all the source data will be in a single system that can be readily extracted using an ETL tool
* **More common situation = each source might be in a different system, environment, and/or DBMS**
* The **ETL system might be expected to extract data from a wide variety of systems involving many different types of data + inherent challenges**
* Organizations needing to extract data from mainframe environments often run into issues involving COBOL copybooks, EBCDIC to ASCII conversions, packed decimals, redefines, OCCURS fields, and multiple + variable record types
* Other organizations might need to extract from sources in RDBMSs, flat files, XML sources, web logs, or a complex ERP system
* Each scenario presents a variety of possible challenges
* **Some sources, especially older legacy systems, may require the use of different procedural languages than the ETL tool can support or the team is experienced with**
* In this situation, request the owner of the source system extract the data into a flat file format
* **NOTE:** Although XML-formatted data has many advantages because it is self-describing, you **may not** **want XML for large, frequent data transfers**
* The payload portion of a typical XML formatted file can < 10% of the total file
* Exception to this recommendation could be where the XML payload is a complex deeply hierarchical XML structure, such as an industry standard data exchange
* In these cases, the DW/BI team must decide whether to “shred” the XML into a large number of destination tables or persist the XML structure within the DW
* Recent advances in RDBMS vendors’ support for XML via XPath have made this latter option feasible
* **2 primary methods for getting data from a source system**: as a **file** or a **stream**
* If the source is an aging mainframe system, it is often easier to extract into files + then move those files to the ETL server
* **NOTE**: If the **source data is unstructured, semi-structured, or even hyper-structured “big data,”** then **rather than loading such data as an un-interpretable RDBMS “blob,” it is often more effective to create a MapReduce/Hadoop extract step that behaves as an ETL fact extractor from the source data, *directly* delivering loadable RDBMS data**
* If you use an ETL tool + the source data is in a database (*not necessarily an RDBMS*), you may set up the extract as a **stream** **where the data flows out of the source system, through the transformation engine, and into the staging database as a single process**
* By contrast, an **extract to file** **approach consists of 3-4 discrete steps: Extract to the file, move the file to the ETL server, transform the file contents, + load the transformed data into the staging database**
* **NOTE**: Although the **stream extract is more appealing, extracts to file have some advantages**
* They are **easy** **to restart at various points**
* **Save the extract file** + you can **rerun the load** without impacting the source system
* Can also **easily encrypt + compress the data before transferring across the network**
* Finally, it’s **easy to verify that all data has moved correctly by comparing file row counts before + after the transfer**
* Generally, **recommended to do a data transfer utility such as FTP to move the extracted file**
* **Data compression** **is important if large amounts of data need to be transferred over a significant distance or through a public network**
* In this case, the **communications link is often the bottleneck**
* If too much time is spent transmitting the data, compression can reduce the transmission time by 30-50% or more, depending on the nature of the original data file
* **Data encryption** **is important if data is transferred through a public network, or even internally in some situations**
* If this is the case, it is **best to send *everything* through an encrypted link** + not worry about what needs to be secure + what doesn’t
* **Remember to compress before encrypting because encrypted files do not compress very well**

### Cleaning and Conforming Data

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

#### Improving Data Quality Culture and Processes

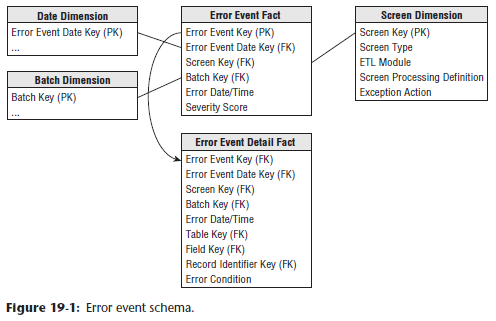
* It is tempting to blame the original data source for any + all errors that appear downstream.
* Perhaps you can fix data quality problem by **imposing constraints** on the data entry UI’s
* This approach provides a hint about how to think about fixing data quality, because a technical solution often avoids the real problem
* Suppose SSN fields for customers were often blank or filled with garbage on an input screen
* Someone comes up with brilliant idea to require input in the 999-99-9999 format, + to cleverly disallow nonsensical entries such as all 9s
* What happens? The data entry clerks are forced to supply valid SSNs to progress to the next screen, so when they don’t have a customer’s number, *they type in an artificial number that passes the roadblock*
* Michael Hammer, in his revolutionary book *Reengineering the Corporation* (Collins, revised 2003), struck the heart of the data quality problem with a brilliant observation
* Paraphrasing Hammer: “**Seemingly small data quality issues are, in reality, important indications of broken business processes**.”
* Not only does this insight correctly focus your attention on the source of data quality problems, but it also shows you the way to the solution
* **Technical attempts to address data quality will not prevail unless they are part of an overall quality culture that must come from the top of an organization**
* The famous Japanese car manufacturing quality attitude permeates every level of such organizations, + quality is embraced enthusiastically by all levels, from CEO to the assembly line worker
* To cast this in a data context, imagine a company such as a large drugstore chain, where a team of buyers contracts with thousands of suppliers to provide the inventory
* The buyers have assistants, whose job it is to enter the detailed descriptions of everything purchased by the buyers, + these descriptions contain dozens of attributes
* But the problem is the assistants have a deadly job + are judged on how many items they enter per hour + have almost no awareness of who uses their data
* Occasionally, the assistants are scolded for obvious errors
* But more insidiously, the data given to the assistants is itself incomplete + unreliable
* Ex: There are no formal standards for toxicity ratings, so there is significant variation over time + over product categories for this attribute
* *How does the drugstore improve data quality?*
* Here is a **9-step template for *any* organization addressing data quality**:
* 1) Declare a **high-level commitment** to a data quality culture
* 2) Drive **process reengineering** at the executive level
* 3) Spend **money** to **improve the data entry environment**
* 4) Spend **money** to **improve application integration**
* 5) Spend **money** to **change how processes** **work**
* 6) Promote **end-to-end team awareness**
* 7) Promote **interdepartmental** **cooperation**
* 8) Publicly **celebrate data quality excellence**
* 9) Continuously **measure + improve data quality**
* At the drugstore, money needs to be spent to improve the data entry system, so it provides the content + choices needed by the buyers’ assistants
* The company’s executives need to assure the buyers’ assistants that their work is important + affects many decision makers in a positive way
* Diligent efforts by the assistants should be publicly praised + rewarded
* And end-to-end team awareness + appreciation of the business value derived from quality data is the final goal

#### Subsystem 4: Data Cleansing System

* The **ETL data cleansing process is often expected to fix dirty data, yet at the same time the DW is expected to provide an accurate picture of the data as captured by the organization’s production systems**
* **Striking the proper balance between these conflicting goals is essential**
* One of our goals in describing the cleansing system is to offer a **comprehensive architecture for cleansing data, capturing data quality events, as well as measuring + ultimately controlling data quality in the DW**
* Some organizations may find this architecture challenging to implement, but **it is important for the ETL team to make a serious effort to incorporate as many of these capabilities as possible**
* If new to ETL and find this a daunting challenge, you might well wonder, **“What’s the minimum I should focus on?”**
* The answer is to **start by undertaking the best possible data profiling analysis**
* The results of this effort **can help understand the risks of moving forward with potentially dirty or unreliable data + help determine how sophisticated the data cleansing system needs to be**
* The **purpose of the cleansing subsystems is to marshal tech to support data quality**
* **Goals for the subsystem should include**:
* **Early diagnosis + triage** of data quality issues
* **Requirements for source systems + integration efforts** to supply better data
* Provide **specific descriptions of data errors** expected to be encountered in ETL
* **Framework for capturing all data quality errors + precisely measuring data quality metrics over time**
* Attachment of **quality confidence metrics** to final data
* **Quality Screens**
* The **heart of the ETL architecture is a set of quality screens that act as diagnostic filters in the data flow pipelines**
* Each quality screen is a test, + **if the test against the data is successful, *nothing happens* + the screen has no side effects**
* But **if the test fails, it must drop an error event row into the error event schema + choose to either halt the process, send the offending data into suspension, or merely tag the data**
* Although all quality screens are architecturally similar, it’s convenient to divide them into **3 types, in ascending order of scope**
* Jack Olson, in his seminal book *Data Quality*: *The Accuracy Dimension* (Morgan Kaufmann, 2002), **classified data quality screens into 3 categories**:
* **1) Column screens** **test the data within a single column**
* Usually simple, somewhat obvious tests, such as testing whether a column contains unexpected NULL values, if a value falls outside of a prescribed range, or if a value fails to adhere to a required format
* **2) Structure screens test the relationship of data *across columns***
* **2+ attributes may be tested to verify they implement a hierarchy**, such as a series of many-to-one relationships
* Structure screens **also test FK/PK relationships between columns in 2 tables**
* Also includes **testing whole *blocks* of columns to verify they implement valid postal addresses, for example**
* **3) Business rule screens implement more complex tests that do not fit the simpler column or structure screen categories**
* Ex: a customer profile may be tested for a complex time-dependent business rule, such as requiring a lifetime platinum frequent flyer to have been a member for at least 5 years + have flown more than 2 million miles
* Business rule screens **also include aggregate threshold data quality checks**, such as checking to see if a statistically improbable number of MRI examinations have been ordered for minor diagnoses like a sprained elbow
* In this case, the screen throws an error only after a threshold of such MRI exams is reached
* **Responding to Quality Events**
* Again: **each quality screen has to decide what happens when an error is thrown, w/ 3 choices:**
* **1) Halting the process**
* Obviously a pain because it **requires manual intervention** to diagnose the problem, restart or resume the job, or abort completely
* **2) Sending the offending record(s) to a suspense file for later processing**
* Often a poor solution because it is **not clear when or if these records will be fixed + re-introduced to the pipeline**.
* **Until the records are restored to the data flow, overall integrity of the database is questionable because records are missing**
* **Recommended to not using the suspense file for *minor* data transgressions**
* **3) Merely tagging the data with the error condition** **+ passing it through to the next step in the pipeline**
* This is **by far the best choice, *whenever possible***, + it **often works well**
* **Bad *fact* table data can be tagged with the audit dimension** (described in subsystem 6)
* **Bad *dimension* data can *also* be tagged using an audit dimension**, or, in the case of missing or garbage data, can be **tagged with unique error values in the attribute itself**

#### Subsystem 5: Error Event Schema

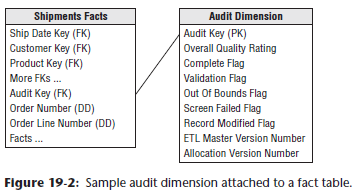
* **Error event schema** **= a *centralized* dimensional schema whose purpose is to record *every* error event thrown by a quality screen *anywhere* in the ETL pipeline**
* Although *we* focus on DW ETL processing, this approach can be used in generic data integration (DI) applications where data is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



* The ***main* table is the error event fact table** whose **grain is every error thrown/produced by a quality screen anywhere in the ETL system**
* Remember: **the grain of a fact table = the physical description of why a fact table row exists**
* Thus, **every quality screen error produces exactly 1 row in this table, + every row in the table corresponds to an observed error**
* The **dimensions of the error event fact table** include the **calendar date** of the error, the **batch job** in which the error occurred, and the **screen that produced** the error
* The **calendar date** is NOT a minute + second timestamp of the error, but rather **provides a way to constrain and summarize error events by the usual attributes of the calendar, such as weekday or last day of a fiscal period**
* The **error date/time fact** is a ***full* relational date/time stamp that specifies *precisely* when the error occurred**
* Useful for **calculating the time interval between error events** because you can take the difference between 2 date/time stamps to get the number of seconds separating events
* The **batch dimension** **can be generalized to be a “processing” step in cases in which data is *streamed*, rather than batched**
* The **screen dimension** **identifies precisely what the screen criterion is + where the code for the screen resides**
* Also **defines what to do when the screen throws an error** (Ex: halt the process, send the record to a suspense file, or tag the data)
* The **error event fact table** also has a **single column PK**, shown as the **error event key**
* This **surrogate key**, like dimension table PKs, is **a simple integer assigned sequentially as rows are added to the fact table**
* This key column is **necessary in those situations in which an enormous burst of error rows is added to the error event fact table all at once** (Hopefully this won’t happen)
* The **error event schema includes a 2nd error event detail fact table at a lower grain**
* Each row in this table **identifies *an individual field* in a specific record that participated in an error**
* Thus, a complex structure or business rule error that triggers a *single* error event row in the higher-level fact table may generate *many* rows in this error event detail fact table
* The 2 error event fact tables are tied together by the error event key, which is a FK in the lower grain table
* The **error event detail table identifies the table, record, field, + precise error condition**
* Thus, a complete description of complex multi-field, multi-record errors is preserved by these tables
* The **error event detail table could also contain a precise datetime stamp to provide a full description of aggregate threshold error events where many records generate an error condition over a period of time**
* You should now appreciate that **each quality screen has the responsibility for populating these tables at the time of an error.**

#### Subsystem 6: Audit Dimension Assembler

* **Audit dimension = a special dimension assembled in the back-room by the ETL system for each fact table** (discussed in Chapter 6: Order Management)
* The audit dimension below **contains the metadata context at the moment when a specific fact table row is created** (elevated metadata to real data)



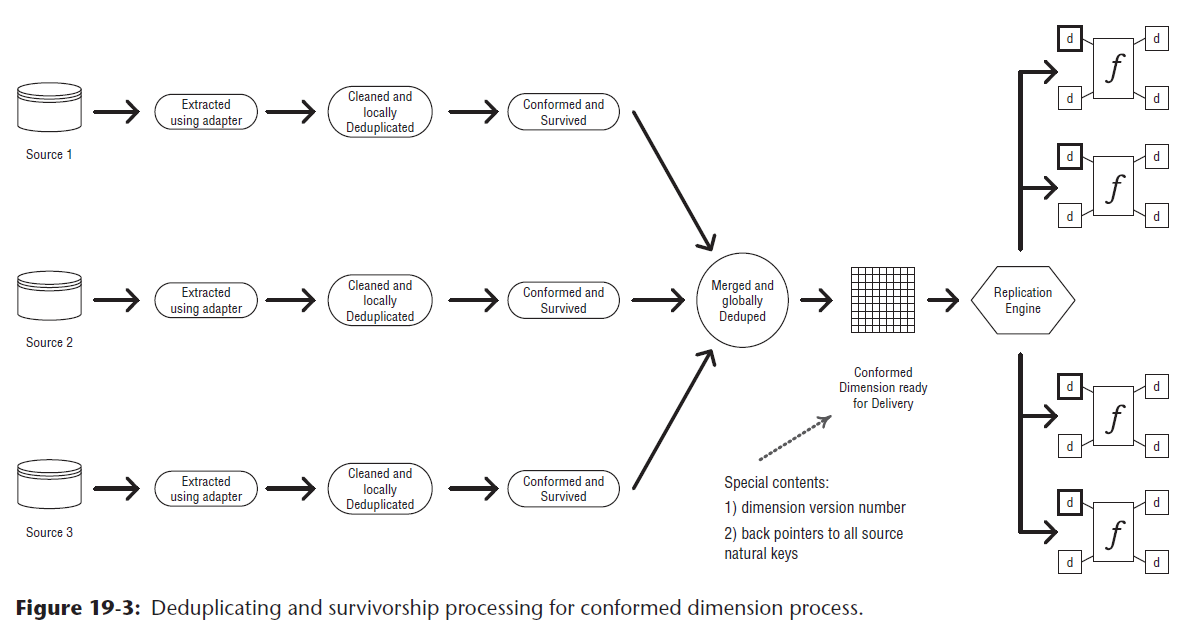
* To visualize how audit dimension rows are created, imagine the above shipments fact table is updated once per day from a batch file
* Suppose today you have a perfect run with no errors flagged
* In this case, you’d generate only 1 audit dimension row, + it would be attached to every fact row loaded today
* All the categories, scores, and version numbers would be the same.
* Now let’s relax the strong assumption of a perfect run
* If you had some fact rows whose discount dollars triggered an out-of-bounds error, then one more audit dimension row would be needed to flag this condition.

#### Subsystem 7: Deduplication System

* **Often dimensions are derived from *several* sources,** a common situation for organizations that have many customer-facing source systems that create + manage separate customer master tables
* Customer information may need to be merged from several lines of business *and* outside sources
* *Sometimes*, the data can be matched through identical values in some key column
* **However, even when a definitive match occurs, other columns in the data might contradict one another, requiring a decision on which data should survive**
* Unfortunately, **there is seldom a universal column that makes the merge operation easy**
* **Sometimes, the only clues available are the similarity of *several* columns**.
* The different sets of data being integrated + the existing dimension table data **may need to be evaluated on different fields to attempt a match**
* Sometimes, a match may be based on fuzzy criteria, such as names + addresses that may nearly match except for minor spelling differences.
* **Survivorship** is **the process of combining a set of matched records into a unified image that combines the highest quality columns from the matched records into a conformed row**
* Involves **establishing clear business rules that define the priority sequence for column values from all possible source systems to enable the creation of a single row with the best-survived attributes**
* **If the dimensional design is fed from multiple systems, you *must* maintain separate columns with back references (such as natural keys) to all participating source systems used to construct the row**
* There are a variety of data integration + data standardization tools to consider if you have difficult deduplicating, matching, + survivorship data issues
* These tools are quite mature and in widespread use

#### Subsystem 8: Conforming System

* **Conforming** consists of **all the steps required to align the content of some or all the columns in a dimension with columns in similar or identical dimensions in other parts of the DW**
* Ex: In a large organization you may have fact tables capturing invoices + customer service calls that both utilize the customer dimension
* It is **highly likely the source systems for invoices + customer service have separate customer databases, + it is likely there will be little guaranteed consistency between the 2 sources of customer information**
* The **data from these 2 customer sources needs to be conformed to make some *or* all the columns describing customer share the same domains**
* **NOTE**: **The process of creating conformed dimensions aligns with an agile approach**
* **For 2 dimensions to be conformed, they must share at least 1 common attribute w/ the same name + same contents**
* Can start w/ a single conformed attribute like Customer Category + systematically add this column in a non-disruptive way to customer dimensions in each customer-facing process
* As you augment each customer-facing process, you expand the list of processes that are integrated + can participate in drill-across queries
* Can also incrementally grow the list of conformed attributes, such as city, state, + country
* **All this can be staged to align with a more agile implementation approach**
* The **conforming subsystem is responsible for creating + maintaining the conformed dimensions + conformed facts described in Chapter 4: Inventory**
* To accomplish this, **incoming data from *multiple* systems needs to be combined + integrated, so it is structurally identical, deduplicated, filtered of invalid data, + standardized in terms of content rows in a conformed image**
* A large part of the conforming process = the deduplicating, matching, + survivorship processes previously described
* The conforming process flow combining deduplicating + survivorship processing is shown below:



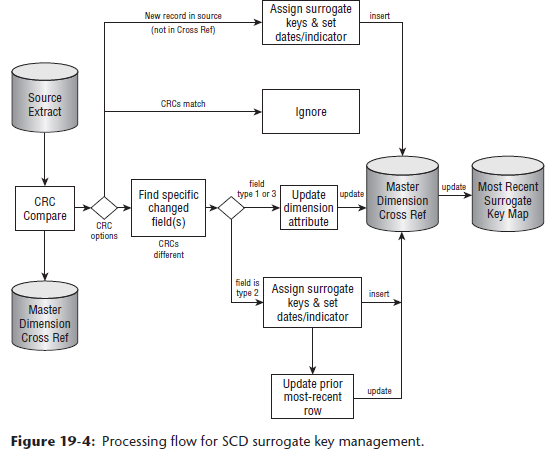
* The process of defining + delivering conformed dimensions + facts is described later in subsystems 17 (dimension manager) and 18 (fact provider)

### Delivering: Prepare for Presentation

* The ***primary* mission of the ETL system = the handoff of dimension + fact tables in the delivery step**
* For this reason, the **delivery subsystems are the most pivotal subsystems in the ETL architecture**
* Although there is considerable variation in source data structures and cleaning + conforming logic, the **delivery processing techniques for preparing the dimensional table structures are more defined + disciplined**
* Use of these techniques is critical to building a successful dimensional DW that is reliable, scalable, + maintainable
* **Many of these subsystems focus on *dimension* table processing, as dimension tables are the heart of the DW since they provide the context for the fact tables + hence for all the measurements**
* Although **dimension tables** are usually smaller than fact tables, they’re critical to the success of the DW/BI system as they **provide the entry points into the fact tables**
* **Delivering process begins w/ cleaned + conformed data** resulting from subsystems just described
* For many dimensions, the **basic load plan** is relatively simple: **perform basic transformations to the data to build dimension rows for loading into the target presentation table**
* Typically includes surrogate key assignment, code lookups to provide appropriate descriptions, splitting or combining columns to present the appropriate data values, or joining underlying 3NF table structures into denormalized flat dimensions
* Preparing fact tables is certainly important because **fact tables hold the key measurements of the business that the users want to see.**
* Fact tables can be large + time-consuming to load
* However, **preparing fact tables for presentation is typically more straightforward**

#### Subsystem 9: Slowly Changing Dimension Manager

* **1 of the more important elements of the ETL architecture = the capability to implement SCD logic**
* The ETL system *must* determine **how to handle an attribute value that has changed from the value already stored in the DW**
* If the revised description is determined to be a legitimate + reliable update to previous information, the appropriate SCD technique must be applied
* As described in Chapter 5: Procurement, when the DW receives notification that an existing row in a dimension has changed, there are **3 basic responses**: **type 1 overwrite, type 2 add a new row, + type 3 add a new column**
* The **SCD manager** should **systematically handle the time variance in the dimensions using these three techniques, as well as the other SCD techniques**
* Also, the SCD manager **should maintain appropriate housekeeping columns for type 2 changes**
* Below shows overall processing flow for handling surrogate key management for processing SCDs



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

##### Type 1: Overwrite

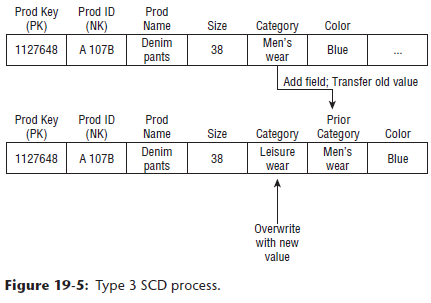
* **Type 1 SCD technique** = a **simple overwrite** **of one or more attributes in an existing dimension row**
* **Take the *revised* data from the CDC system + overwrite the dimension table contents w/ it**
* Appropriate when **correcting data** or when **there is no business need to keep the history of previous values**
* Ex: You may receive a corrected customer address, + in this case, overwriting is the right choice
* Note that **if the dimension table includes type 2 change tracking, you should overwrite the affected column in *all existing rows for that particular customer***
* **Type 1 updates must be propagated forward from the earliest permanently-stored staging tables to *all* affected staging tables, so if any of them are used to recreate the *final* load tables, the effect of the overwrite is preserved**
* Some ETL tools contain ***UPDATE else INSERT* functionality**
* This may be **convenient for the developer but can be a performance killer**
* **For maximum performance, existing row UPDATEs should be segregated from new row INSERTs**
* **If type 1 updates cause performance problems, consider disabling database logging or use of the DBMS bulk loader**
* **Type 1 updates invalidate any aggregates built upon the changed column, so the dimension manager (subsystem 17) must notify the affected fact providers (subsystem 18) to drop + rebuild the affected aggregates**

##### Type 2: Add New Row

* **Type 2 SCD = the standard technique for accurately tracking changes in dimensions + associating them correctly with fact rows**
* Supporting type 2 changes **requires a *strong* CDC system to detect changes *as soon as they occur***
* **Type 2 updates = copy the previous version of the dimension row + create a *new* dimension *row* with a *new surrogate key***
* *If there is NOT a previous version of the dimension row, create a new one from scratch*
* Then **update new row w/ columns that’ve changed + add any other columns that’re needed**
* This is the **main workhorse technique for handling dimension attribute changes that *need to be tracked over time***
* The **type 2 ETL process must *also* update the most recent surrogate key map table, assuming the ETL tool doesn’t automatically handle this**
* These little 2-column tables are of **immense importance when loading fact table data**
* Subsystem 14, the surrogate key pipeline, supports this process.
* Refer to the processing flow for handling surrogate key management figure from before to see the lookup + key assignment logic for handling a changed dimension row during the extract process
* In this example, **the CDC process (subsystem 2) uses a CRC compare to determine which rows have changed in the source data since the last update**
* *If lucky, you already know which dimension records have changed + can omit this CRC compare step*
* **After you identify rows that have changes in type 2 attributes, you can generate a new surrogate key from the key sequence + update the surrogate key map table**
* **When a new type 2 row is created, you need a pair of time stamps at the least, as well as an optional change description attribute**
* **The pair of time stamps defines a span of time from the beginning effective time to the ending effective time when the complete set of dimension attributes is valid**
* **More sophisticated treatment** of a type 2 SCD row involves **adding *five* ETL housekeeping columns**
* Referring to the processing flow figure, this *also* **requires the type 2 ETL process to find the prior effective row + make appropriate updates to these housekeeping columns**:
* Change Date (change date as foreign key to date dimension outrigger)
* Row Effective Date/Time (exact date/time stamp of change)
* Row End Date/Time (exact date/time stamp of next change, defaults to 12/31/9999 for most current dimension row)
* Reason for Change column (optional attribute)
* Current Flag (current/expired)
* **NOTE:** It is **possible that back-end scripts are run w/in the transaction database to modify data *without updating the respective metadata fields* such as the last\_modified\_date**
* Using *these* fields for the dimension time stamps can cause inconsistent results in the DW
* **Always use the system or as-of date to derive the type 2 effective time stamps**
* **Type 2 process does NOT change history as type 1 does, so type 2 changes don’t require rebuilding affected aggregate tables *as long as the change was made “today” + not backward in time***
* **NOTE**: Kimball Design Tip #80([www.kimballgroup.com](http://www.kimballgroup.com) under the Tools + Utilities tab for this book title) provides in-depth guidance on adding a row change reason code attribute to dimension tables

##### Type 3: Add New Attribute

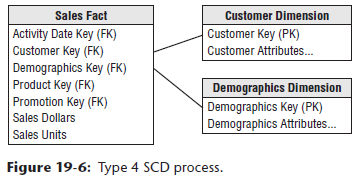
* The **type 3 technique** is designed to support attribute **“soft” changes** that **allow a user to refer either to the old value of the attribute or the new value**
* Ex: If a sales team is assigned to a newly named sales region, there may be a need to track the old region assignment, as well as the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



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

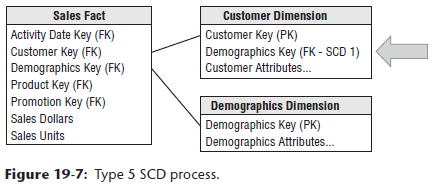
##### Type 4: Add Mini-Dimension

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



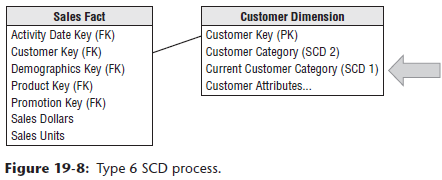
##### Type 5: Add Mini-Dimension + Type 1 Outrigger

* **Type 5 technique builds on the type 4 mini-dimension by also embedding a type 1 reference to the mini-dimension in the primary dimension**
* **Allows accessing the current values in the mini-dimension directly from the base dimension without linking through a fact table**
* The ETL team must **add the type 1 key reference in the base dimension** + must **overwrite this key reference in all copies of the base dimension whenever the current status of the mini-dimension changes over time**
* Below shows how a type 5 SCD is implemented:



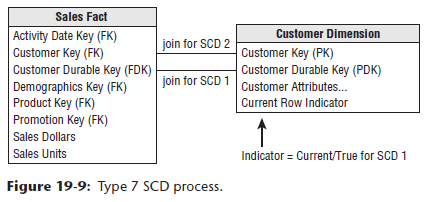
##### Type 6: Add Type 1 Attributes to Type 2 Dimension

* **Type 6 technique has an *embedded attribute* that is an *alternate value* of a normal type 2 attribute in the base dimension**
* Usually such an attribute is simply a type 3 alternative reality, but in this case **the attribute is systematically overwritten whenever the attribute is updated**
* Below shows how a type 6 SCD is implemented:



##### Type 7: Dual Type 1 and Type 2 Dimensions

* **Type 7 technique = a normal type 2 dimension paired w/ a specially constructed fact table that has *both* a *normal* FK to the dimension for type 2 historical processing, + also a foreign *durable* key (FDK in the figure below) that is used alternatively for type 1 current processing, connected to the durable key in the dimension table labeled PDK**



* The **dimension table *also* contains a current row indicator that indicates whether the particular row is the one to be used for current SCD 1 perspective.**
* The ETL team ***must* augment a normally constructed fact table with this constant value foreign durable key**

#### Subsystem 10: Surrogate Key Generator

* Chapter 3: Retail Sales 🡪 **strongly recommended to use surrogate keys for all dimension tables**.
* This **implies you need a robust mechanism for *producing* surrogate keys in the ETL system**
* The surrogate key generator should ***independently* generate surrogate keys for *every* dimension**
* Should be **independent of database instance *and* able to serve distributed clients**
* **Goal of the surrogate key generator = to generate a *meaningless* key, typically an integer, to serve as the PK for a dimension row.**
* Although it may be tempting to create surrogate keys via **database triggers**, this technique **may create performance bottlenecks**
* If the DBMS is used to assign surrogate keys, it is **preferable for the ETL process to directly call the database sequence generator**
* **For improved efficiency, consider having the ETL tool generate + maintain the surrogate keys**
* Avoid temptation of concatenating the operational key of the source system + a datetime stamp
* Although this seems simple, it is fraught with problems + ultimately will not scale

#### Subsystem 11: Hierarchy Manager

* It is **normal for a dimension to have multiple, simultaneous, embedded hierarchical structures**
* These multiple hierarchies **simply coexist in the same dimension as dimension attributes**
* All that’s **necessary** **=** **every attribute be single-valued in the presence of the dimension’s PK**
* Hierarchies are either **fixed** or **ragged**
* A **fixed-depth hierarchy** has a **consistent number of levels + is simply modeled + populated as separate dimension attributes for each of the levels**
* **Slightly-ragged hierarchies** (like postal addresses) are **most often modeled *as a fixed hierarchy***
* ***Profoundly*-ragged hierarchies** = **typically found w/ organization structures that are unbalanced + of indeterminate depth**
* The data model + ETL solution required to support these needs **require the use of a bridge table containing the organization map**
* **Snowflakes/normalized data structures** are **NOT recommended for the presentation level**
* *However*, the use of a normalized design **may be appropriate in the ETL *staging area* to assist in the maintenance of the ETL data flow for populating + maintaining the hierarchy attributes**
* The ETL system is **responsible for enforcing the business rules to assure the hierarchy is populated appropriately in the dimension table**

#### Subsystem 12: Special Dimensions Manager

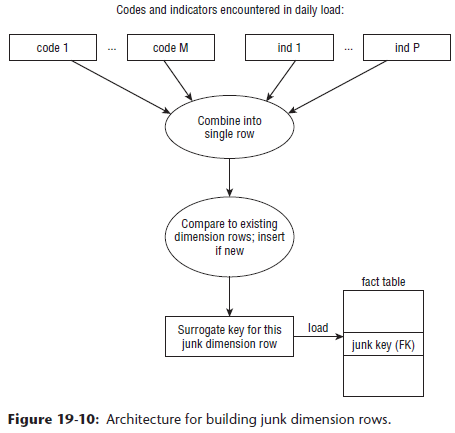
* The special dimensions manager is a **catch-all subsystem**: a placeholder in the ETL architecture for supporting an organization’s specific dimensional design characteristics.
* Some organizations’ ETL systems require all the capabilities discussed below, whereas others will be concerned with few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

##### Date/Time Dimensions

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

##### Junk Dimensions

* **Junk dimensions** are made up from **text + miscellaneous flags left over in the fact table after you remove all the critical attributes**
* **2 approaches for creating junk dimensions** in the ETL system
* **1) If the theoretical number of rows in the dimension is fixed + known, the junk dimension can be created *in advance***
* 2) In other cases, it **may be necessary to create newly observed junk dimension rows on-the-fly while processing fact row input**
* As illustrated in below, this process **requires assembling the junk dimension attributes and comparing them to the existing junk dimension rows to see if the row already exists**
* **If *not*, a new dimension row must be assembled, a surrogate key created, + the row loaded into the junk dimension on-the-fly during the fact table load process**



* **NOTE**: Kimball Design Tip #113([www.kimballgroup.com](http://www.kimballgroup.com) under the Tools + Utilities tab for this book title) provides more in-depth guidance on building and maintaining junk dimension tables

##### Mini-Dimensions

* As just discussed in subsystem 9, **mini-dimensions = a technique used to track dimension attribute changes in a large dimension when the type 2 technique is infeasible**, like in a customer dimension
* From an ETL perspective, **creation of the mini-dimension is similar to the junk dimension process previously described**
* Again, there are **2 alternatives**
* 1) Building *all* valid combinations in advance
* 2) Recognizing + creating new combinations on-the-fly
* Although **junk dimensions are usually built from the fact table input, mini-dimensions are built from dimension table inputs**
* The **ETL system is responsible for maintaining a multi-column surrogate key lookup table to identify the base dimension member + appropriate mini-dimension row to support the surrogate pipeline process described in Subsystem 14, Surrogate Key Pipeline**
* Keep in mind that **very large, complex customer dimensions often require *several* mini-dimensions**
* **NOTE**: Kimball Design Tip #127([www.kimballgroup.com](http://www.kimballgroup.com) under the Tools + Utilities tab for this book title) provides more in-depth guidance on building + maintaining mini-dimension tables

##### Shrunken Subset Dimensions

* **Shrunken dimensions = conformed dimensions that are a subset of rows and/or columns of one of the base dimensions**
* The ETL data flow should **build conformed shrunken dimensions *from the base dimension*, rather than independently, to assure conformance**
* The **PK for the shrunken dimension, however, *must* be independently generated**
* If you **attempt to use a key from an “example” base dimension row, you will get into trouble if this key is retired or superseded**
* **NOTE**: Kimball Design Tip #137 ([www.kimballgroup.com](http://www.kimballgroup.com) under the Tools + Utilities tab for this book title) provides more in-depth guidance on building shrunken dimension tables

##### Small Static Dimensions

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

##### User Maintained Dimensions

* **Often the DW requires that totally new “master” dimension tables be created**.
* These dimensions **have no formal system of record, + rather are custom descriptions, groupings, + hierarchies created by the business for reporting + analysis purposes**
* The ETL team often ends up with stewardship responsibility for these dimensions, but this is typically not successful because the ETL team is not aware of changes that occur to these custom groupings, so the dimensions fall into disrepair and become ineffective
* **Best-case scenario = have the appropriate business user department agree to own the maintenance of these attributes**
* The **DW/BI team needs to provide a UI for this maintenance**
* Typically, this takes the form of a simple application built using the company’s standard visual programming tool
* The **ETL system should add default attribute values for new rows, which the user owner needs to update**
* If these rows are loaded into the DW before they are changed, they still appear in reports with whatever default description is supplied
* **NOTE:** The ETL process should create a unique default dimension attribute description that shows someone hasn’t yet done their data stewardship job
* Favor a label that concatenates the phrase Not Yet Assigned with the surrogate key value: “Not Yet Assigned 157”
* That way, multiple unassigned values do not inadvertently get lumped together in reports and aggregate tables
* This also helps identify the row for later correction

#### Subsystem 13: Fact Table Builders

* **Fact tables hold the measurements of an organization + dimensional models are deliberately built around these numerical measurements**
* The fact table builder subsystem focuses on the ETL architectural requirements to effectively build the **3 primary types of fact tables: transaction, periodic snapshot, accumulating snapshot**
* An **important requirement for loading fact tables is maintaining referential integrity with the associated dimension tables**
* The surrogate key pipeline (subsystem 14) is designed to help support this need

##### Transaction Fact Table Loader

* The **transaction grain represents a measurement event defined at a particular instant.**
* A line item on an invoice is an example of a transaction event + a scanner event at a cash register is another
* In these cases, the **time stamp in the fact table is very simple: either a single daily grain FK OR a *pair* consisting of a daily grain FK together with a datetime stamp, depending on what the source system provides + the analyses require**
* The **facts in a *transaction* fact table MUST be true to the grain + should describe only what took place *in that instant***
* **Transaction grain fact tables = the largest + most detailed of the 3 types of fact tables**
* The **transaction fact table loader receives data from the CDC system + loads it w/ the proper dimensional FKs**
* The pure addition of the most current records = the easiest case: simply bulk loading new rows into the fact table
* **In most cases, the target fact table should be partitioned by *time* to ease the administration + speed the performance of the table**
* An **audit key, sequential ID, or date/time stamp column should be included to allow backup or restart of the load job**
* The **addition of late arriving data is more difficult, requiring additional processing capabilities described in subsystem 16**
* In the event it is necessary to **update existing rows**, this process **should be handled in 2 phases:**
* **1) Insert the corrected rows *without* overwriting or deleting the original rows**
* **2) Then delete the old rows in a second step**
* **Using a sequentially-assigned single surrogate key for the fact table makes it possible to perform the two steps of insertion followed by deletion**

##### Periodic Snapshot Fact Table Loader

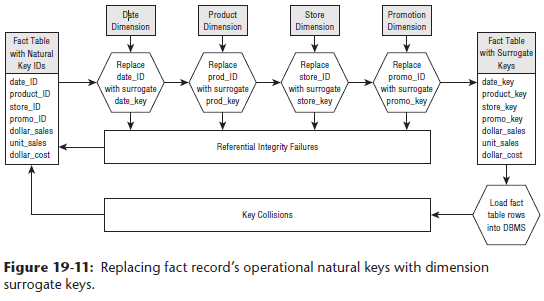
* The **periodic snapshot grain represents a regular repeating measurement/set of measurements**, like a bank account monthly statement
* This fact table **also has a single date column, representing the overall period**
* The **facts in a periodic snapshot fact table** **MUST be true to the grain + should describe only measures appropriate to the timespan defined by the period**
* Periodic snapshots are a **common fact table type** + are frequently used for account balances, monthly financial reporting, + inventory balances
* **Periodicity of a periodic snapshot is typically daily, weekly, or monthly**
* Periodic snapshots have **similar loading characteristics to those of transaction grain fact tables**
* The **same processing applies for inserts + updates**
* **Assuming data is promptly delivered to the ETL system, all records for each periodic load can cluster in *the most recent time partition***
* **Traditionally, periodic snapshots have been loaded en masse at the end of the appropriate period**
* Ex: A credit card company might load a monthly account snapshot table w/ the balances in effect at the end of the month
* **More frequently, organizations will populate a hot rolling periodic snapshot**
* **In addition to the rows loaded at the end of every month, there are special rows loaded with the most current balances in effect as of the previous day**
* As the month progresses, the current month rows are continually updated w/ the most current information + continue in this manner rolling through the month
* Note that the **hot rolling snapshot can sometimes be difficult to implement if the business rules for calculating the balances at the period end are complex**
* Often these complex calculations are dependent on other periodic processing outside the DW, + there is not enough information available to the ETL system to perform these complex calculations on a more frequent basis

##### Accumulating Snapshot Fact Table Loader

* The **accumulating snapshot grain represents the current evolving status of a process that has a finite beginning + end**
* Usually, these **processes are of short duration + therefore don’t lend themselves to the periodic snapshot**
* Order processing is the classic example of an accumulating snapshot 🡺 an order is placed, shipped, + paid for within one **reporting period**
* The transaction grain provides too much detail separated into individual fact table rows, + the periodic snapshot just is the wrong way to report this data
* **Design + administration of the accumulating snapshot is quite different from the first 2 fact table types**
* **All accumulating snapshot fact tables have a set of dates which describe the typical process workflow**
* Ex: An order might have an order date, actual ship date, delivery date, final payment date, + return date
* In this example, **these 5 dates appear as 5 *separate* date-valued foreign surrogate keys**.
* When the order row is first created, the 1st of these dates is well defined, but perhaps none of the others have yet happened
* This **same fact row is subsequently revisited as the order winds its way through the order pipeline**
* Each time something happens, the accumulating snapshot fact row is **destructively modified**
* The **date FKs are overwritten, + various facts are updated**
* **Often the 1st date remains inviolate because it describes when the row was created, but all the other dates may well be overwritten, sometimes more than once**
* Many RDBMSs utilize **variable row lengths**
* **Repeated updates to accumulating snapshot fact rows may cause the rows to grow due to these variable row lengths**, affecting the residency of disk blocks
* **May be worthwhile to occasionally drop + reload rows after the update activity to improve performance**
* An **accumulating snapshot fact table is an effective way to represent finite processes with well-defined beginnings and endings**
* *However*, the **accumulating snapshot, by definition, is the most recent view**
* **Often it makes sense to utilize all 3 fact table types to meet various needs**
* Periodic history can be captured w/ periodic extracts, + all the infinite details involved in the process can be captured in an associated transaction grain fact table
* The **presence of many situations that violate standard scenarios or involve repeated looping though the process would prohibit the use of an accumulating snapshot**

#### Subsystem 14: Surrogate Key Pipeline

* **Every ETL system must include a step for replacing the operational natural keys in the incoming fact table row w/ the appropriate dimension surrogate keys**
* **Referential integrity (RI) = for each FK in the fact table, an entry exists in the corresponding dimension table**
* If there’s a row in a sales fact table for product surrogate key 323442, you *need* to have a row in the product dimension table with the same key, or you won’t know what you’ve sold
* You have a sale for what appears to be a nonexistent product
* Even worse, without the product key in the dimension, a business user can easily construct a query that will omit this sale without even realizing it
* The **key lookup process should result in a match for EVERY incoming natural key or a default value**
* **In the event there is an unresolved referential integrity failure during the lookup process, you need to feed these failures back to the responsible ETL process for resolution, as shown below**



* Likewise, the **ETL process needs to resolve any key collisions that might be encountered during the key lookup process.**
* **After the fact table data has been processed + just before loading into the presentation layer, a surrogate key lookup needs to occur to substitute the operational natural keys in the fact table record with the proper current surrogate key**
* **To preserve referential integrity, always complete the updating of the dimension tables first**
* **In that way, the dimension tables are *always* the legitimate source of PK’s you must replace in the fact table (refer to the figure above)**
* **Most direct approach = to use the *actual dimension table* as the source for the most current value of the surrogate key corresponding to each natural key**
* **Each time you need the current surrogate key, look up all rows in the dimension w/ the natural key = the desired value, + then select the surrogate key that aligns w/ the historical context of the fact row using a current row indicator or begin + end effect dates**
* Current hardware environments offer nearly unlimited addressable memory, making this approach practical
* **During processing, each natural key in the incoming fact record is *replaced* w/ the correct current surrogate key**
* **Don’t keep natural keys in the fact row 🡪 a fact table needs to contain only the surrogate key**
* **Do not write the input data to disk until *all* fact rows have passed *all* the processing steps**
* If possible, all required dimension tables should be pinned in memory, so they can be randomly accessed as each incoming record presents its natural keys
* As illustrated at the bottom of the above figure, the **surrogate key pipeline needs to handle key collisions in the event you attempt to load a duplicate row**
* This is an **example of a data quality problem appropriate for a traditional structure data quality screen, as discussed in subsystem 4**
* In the event a key collision is recognized, the **surrogate key pipeline process needs to choose to halt the process, send the offending data into suspension, or apply appropriate business rules to determine if it is possible to correct the problem, load the row, + write an explanatory row into the error event schema**
* **NOT:** **A slightly different process is needed to perform surrogate key lookups if you need to reload history or if you have a lot of late arriving fact rows, because you don’t want to map the most current value to a historical event**
* In this case, you **need to create logic to find the surrogate key that applied at the time the fact record was generated**
* This means finding the surrogate key where the fact transaction date is between the key’s effective start date + end date
* **When all fact table natural keys have been replaced w/ surrogate keys, a fact row is ready to load**
* **The keys in the fact table row have been chosen to be proper FKs, + the fact table is guaranteed to have referential integrity with respect to the dimension tables**

#### Subsystem 15: Multivalued Dimension Bridge Table Builder

* Sometimes a fact table must support a dimension that takes on multiple values at the lowest granularity of the fact table, as described in Chapter 8: CRM
* **If the grain of the fact table cannot be changed to directly support this dimension, then the multivalued dimension must be linked to the fact table via a bridge table**
* Bridge tables are common in healthcare, in sales commission environments, + for supporting variable-depth hierarchies, as discussed in subsystem 11
* The **challenge for the ETL team is building + maintaining the bridge table**
* As multivalued relationships to the fact row are encountered, the ETL system has the **choice of either making each set of observations a unique group or reusing groups when an identical set of observations occurs**
* **Unfortunately, there is no simple answer for the right choice**
* **In the event the multivalued dimension has type 2 attributes, the bridge table must *also* be time varying**, such as a patient’s time variant set of diagnoses.
* One of the bridge table constructs presented in Chapter 10: Financial Services was the inclusion of a **weighting factor** to support properly weighted reporting from the bridge table
* In many cases, the weighting factor is a familiar allocation factor, but in other cases, the identification of the appropriate weighting factor can be problematic because **there may be no rational basis for assigning the weighting factor**
* **NOTE:** Kimball Design Tip #142 ([www.kimballgroup.com](http://www.kimballgroup.com) under the Tools + Utilities tab for this book title) provides more in-depth guidance on building and maintaining bridge tables

#### Subsystem 16: Late Arriving Data Handler

* **DW’s are usually built around the ideal assumption that measured activity (fact records) arrive in the DW at the same time as the context of the activity (dimension records)**
* When you have both the fact records *and* the *correct* contemporary dimension rows, you have the luxury of first maintaining the dimension keys + then *using* these up-to-date keys in the accompanying fact rows
* **However, for a variety of reasons, the ETL system may need to process late arriving fact or dimension data**
* In some environments, there **may need to be special modifications to the standard processing procedures to deal with late arriving facts**, namely fact records that come into the DW very much delayed
* This is **messy because you have to search back in history to decide which dimension keys were in effect when the activity occurred**
* In addition, you **may need to adjust any semi-additive balances in subsequent fact rows**
* In a heavily compliant environment, it is **also necessary to interface with the compliance subsystem because you are about to change history**
* Late arriving *dimensions* occur when the activity measurement (fact record) arrives at the DW *without its full context*
* In other words, the statuses of the dimensions attached to the activity measurement are ambiguous or unknown for some period of time
* If living in the conventional batch update cycle of 1 or more days’ latency, you can usually just wait for the dimensions to be reported
* Ex: The identification of the new customer may come in a separate feed delayed by several hours + you may just be able to wait until the dependency is resolved.
* **But in many situations, especially real-time environments, this delay is not acceptable**
* ***Cannot* suspend the rows + wait for the dimension updates to occur, as the business requirements demand that you make the fact row visible before knowing the dimensional context**
* The ETL system needs additional capabilities to support this requirement
* Using customer as the problem dimension, the **ETL system needs to support 2 situations**
* 1) To support late arriving type 2 dimension updates
* In this situation, you need to add the *revised* customer row to the dimension with a new surrogate key + then go in and destructively modify any subsequent fact rows’ FK to the customer table
* The effective dates for the affected dimension rows also need to be reset
* In addition, you need to scan forward in the dimension to see if there have been any subsequent type 2 rows for this customer + change this column in any affected rows.
* 2) When you receive a fact row with what appears to be a valid customer natural key, but you have not yet loaded this customer in the customer dimension
* It would be possible to load this row pointing to a default row in the dimension table
* This approach has the same unpleasant side effect discussed earlier of requiring destructive updates to the fact rows’ FKS when the dimension updates are finally processed
* Alternatively, if you believe the customer is a valid, but not yet processed customer, you should assign a new customer surrogate key with a set of dummy attribute values in a new customer dimension row
* You then return to this dummy dimension row at a later time + make type 1 overwrite changes to its attributes when you get complete information on the new customer
* At least this step avoids destructively changing any fact table keys.
* There is no way to avoid a brief provisional period in which the dimensions are “not quite right.”
* But **these maintenance steps can minimize the impact of the unavoidable updates to the keys and other columns**

#### Subsystem 17: Dimension Manager System

* The **dimension manager = a centralized authority who prepares + publishes conformed dimensions to the DW community**
* **A conformed dimension is by necessity a *centrally* managed resource**
* Each conformed dimension **must have a single, consistent source**
* It is the dimension manager’s responsibility to administer + publish the conformed dimension(s) for which he has responsibility
* There *may be multiple dimension managers in an organization, each responsible for a dimension*
* The **dimension manager’s responsibilities include** the following ETL processing:
* Implement the common descriptive labels agreed to by the data stewards + stakeholders during the dimension design.
* Add new rows to the conformed dimension for new source data, generating new surrogate keys
* Add new rows for type 2 changes to existing dimension entries, generating new surrogate keys
* Modify rows in place for type 1 changes + type 3 changes, without changing the surrogate keys
* Update the version number of the dimension if any type 1 or type 3 changes *are* made
* Replicate the revised dimension simultaneously to all fact table providers.
* It is easier to manage conformed dimensions in a single tablespace DBMS on a single machine because there is only one copy of the dimension table
* **However, managing conformed dimensions becomes more difficult in multiple tablespace, multiple DMBS, or multimachine distributed environments**
* In these situations, the dimension manager must carefully manage the simultaneous release of new versions of the dimension to every fact provider
* **Each conformed dimension should have a version number column in each row that is overwritten in every row whenever the dimension manager releases the dimension**
* **This version number should be utilized to support any drill-across queries to assure that the same release of the dimension is being utilized**

#### Subsystem 18: Fact Provider System

* The **fact provider is responsible for *receiving* conformed dimensions from the dimension managers**
* The **fact provider owns the administration of one or more fact tables + is responsible for their creation, maintenance, + use**
* **If fact tables are used in *any* drill-across applications, then by definition the fact provider must be using conformed dimensions provided by the dimension manager**
* The **fact provider’s responsibilities are more complex + include**:
* Receive or download replicated dimension from the dimension manager
* In an environment in which the dimension cannot simply be replicated but must be locally updated, the fact provider must process dimension records marked as new + current to update current key maps in the surrogate key pipeline + also process any dimension records marked as new but postdated
* Add all new rows to fact tables after replacing their natural keys w/ correct surrogate keys.
* Modify rows in all fact tables for error correction, accumulating snapshots, + late arriving dimension changes.
* Remove aggregates that have become invalidated.
* Recalculate affected aggregates
* If the new release of a dimension does not change the version number, aggregates have to be extended to handle only newly loaded fact data
* If the version number of the dimension *has* changed, the entire historical aggregate may have to be recalculated
* Quality ensure all base + aggregate fact tables
* Be satisfied the aggregate tables are correctly calculated.
* Bring updated fact and dimension tables online.
* Inform users that the database has been updated
* Tell them if major changes have been made, including dimension version changes, postdated records being added, + changes to historical aggregates

#### Subsystem 19: Aggregate Builder

* **Aggregates are the single most dramatic way to affect performance in a large DW environment**
* **Aggregation, like an index = specific data structure created to (significantly) improve performance**
* The **ETL system needs to effectively build + use aggregates *without causing significant distraction or consuming extraordinary resources + processing cycles***
* **Avoid architectures in which aggregate navigation is built into the proprietary query tool**
* From an ETL viewpoint, the **aggregation builder needs to populate + maintain aggregate fact table rows + shrunken dimension tables where needed by aggregate fact tables**
* The **fastest update strategy is incremental**, but **a major change to a dimension attribute may require dropping + rebuilding the aggregate**
* In some environments, it may be faster to dump data out of the DBMS + build aggregates w/ a SORT utility rather than building the aggregates *inside* the DBMS
* Additive numeric facts can be aggregated easily at extract time by calculating break rows in one of the sort packages
* **Aggregates must *always* be consistent with the atomic base data**
* The fact provider (subsystem 18) is responsible for taking aggregates off -line when they are not consistent with the base data.
* **User feedback on the queries that run slowly is critical input to designing aggregations**
* Although you can depend on informal feedback to some extent, **a log of frequently attempted slow-running queries should be captured**
* Also, try to ID the nonexistent slow-running queries that never made it into the log b/c they never ran to completion, or aren’t even attempted due to known performance challenges

#### Subsystem 20: OLAP Cube Builder

* **OLAP servers present dimensional data in an intuitive way, enabling a range of analytic users to slice + dice data**
* **OLAP is a sibling of dimensional star schemas in the RDB, with intelligence about relationships + calculations defined on the server that enable faster query performance + more interesting analytics from a broad range of query tools**
* **Don’t think of an OLAP server as a competitor to a relational DW, but rather an extension**
* **Let the RDB do what it does best: Provide storage + management**
* **The relational dimensional schema should be viewed as the foundation for OLAP cubes** *if you elect to include them in your architecture*
* The process of feeding data from the dimensional schema is an integral part of the ETL system, + the **relational schemas are the best and preferred source for OLAP cubes**
* **Many OLAP systems do *not* directly address referential integrity or data cleaning, so the preferred architecture = load OLAP cubes *after* completion of conventional ETL processes**
* Note that **some OLAP tools are more sensitive to hierarchies than relational schemas**
* It is **important to strongly enforce the integrity of hierarchies within dimensions *before* loading an OLAP cube**
* Type 2 SCDs fit an OLAP system well b/c a new surrogate key is just treated as a new member
* **Type 1 SCDs that restate history do NOT fit OLAP well**
* **Overwrites to an attribute value can cause all the cubes using that dimension to be reprocessed in the background, become corrupted, or be dropped**

#### Subsystem 21: Data Propagation Manager

* **Data propagation manager = responsible for the ETL processes required to present conformed, integrated enterprise data from the DW presentation server to other environments for special purposes**
* Many organizations need to extract data from the presentation layer to share w/ business partners, customers, and/or vendors for strategic purposes
* Similarly, some organizations are required to submit data to various government organizations for reimbursement purposes, such as healthcare organizations that participate in the Medicare program
* Many organizations have acquired package analytic applications
* Typically, these applications *cannot* be pointed directly against the existing DW tables, so data needs to be extracted from the presentation layer + loaded into proprietary data structures required by the analytic applications
* Finally, most data mining tools do not run directly against the presentation server
* They need data extracted from the DW + fed to the data mining tool in a specific format
* **All the situations previously described require extraction from the DW/BI presentation server, possibly some light transformation, + loading into a target format (in other words, ETL)**
* **Data propagation should be considered a part of the ETL system + ETL tools should be leveraged to provide this capability**
* **What’s *different* here is that the *requirements of the target are not negotiable* 🡪 you *must* provide the data as specified by the target**

### Managing the ETL Environment

* **A DW/BI environment** can have a great dimensional model, well-deployed BI applications, + strong management sponsorship, but it **cannot be a success until it can be relied upon as a dependable source for business decision making**
* **One of the goals for the DW/BI system = to build a reputation for providing timely, consistent, + reliable data to empower the business**
* To achieve this goal, the **ETL system must constantly work toward fulfilling 3 criteria:**
* **Reliability**
* **The ETL processes must consistently run + must run *to completion* to provide data on a timely basis that is trustworthy at any level of detail**
* **Availability**
* **The DW must meet its SLAs, + it should be + and available as promised**
* **Manageability**
* **A successful DW is never done, it constantly grows + changes along with the business, + the ETL processes need to gracefully evolve as well.**
* **ETL *management* subsystems = the key components of the architecture to help achieve the goals of reliability, availability, + manageability**
* Operating + maintaining a DW in a professional manner is not much different than any other systems operations: **Follow standard best practices, plan for disaster, + practice**

#### Subsystem 22: Job Scheduler

* Every EDW should have a robust **ETL scheduler**
* The ***entire* ETL process should be managed, to the extent possible, through a *single*, *metadata-driven* job control environment**
* Major ETL tool vendors package scheduling capabilities into their environments
* If you elect not to use the scheduler included with the ETL tool, or do not use an ETL tool, you need to utilize existing production scheduling or perhaps manually code the ETL jobs to execute
* **Scheduling is much more than just launching jobs on a schedule**
* **Scheduler needs to be aware of + control the relationships + dependencies between ETL jobs**
* Needs to **recognize when a file or table is ready to be processed**
* If the organization is processing in real time, you need a scheduler that supports your selected real-time architecture
* The job control process **must also capture metadata regarding the progress + statistics of the ETL process during its execution**
* Finally, the scheduler **should support a fully automated process, including notifying the problem escalation system in the event of any situation that requires resolution**
* The infrastructure to manage this can be as basic (and labor-intensive) as a set of SQL stored procedures, or as sophisticated as an integrated tool designed to manage + orchestrate multiplatform data extract + loading processes
* If you use an ETL tool, it should provide this capability
* In any case, you need to set up an environment for creating, managing, + monitoring the ETL job stream such that the **job control services needed include**:
* **Job definition**
* 1st step in creating an operations process = to **have some way to define a series of steps as a “job”** + to **specify some *relationship* among jobs**
* This is where the **execution flow** of the ETL process is written
* In many cases, if the load of a given table fails, it can impact your ability to load tables that depend on it
* Ex: If the customer table is not properly updated, loading sales facts for new customers that did not make it into the customer table is risky
* In some databases, it is impossible
* **Job scheduling**
* At a minimum, the environment needs to provide standard capabilities, such as **time-based** and **event-based scheduling**
* **ETL processes are often based on some upstream system event**, such as the successful completion of the general ledger close or the successful application of sales adjustments to yesterday’s sales figures
* This **includes the ability to monitor database flags, check for the existence of files, + compare creation dates.**
* **Metadata capture**
* No self-respecting systems person would tolerate a black box scheduling system
* The folks responsible for running the loads will demand a workflow monitoring system (subsystem 27) to understand what is going on
* The **job scheduler needs to capture information about what step the load is on, what time it started, + how long it took**
* In handcrafted ETL systems, this can be accomplished by having each step write to a log file
* The ETL tool **should capture this data *every time an ETL process executes***
* **Logging**
* i.e., **collecting information about the *entire* ETL process**, not just what is happening at the moment
* Log information **supports the recovery + restarting of a process in case of errors during the job execution**
* Logging to text files is the minimum acceptable level, but try to **prefer a system that logs to a database because the structure makes it easier to create graphs + reports**
* Also makes it possible to create time series studies to help analyze + optimize the load process
* **Notification**
* After the ETL process has been developed + deployed, it should execute in a hands-off manner + should run without human intervention, without fail
* If a problem *does* occur, the control system needs to interface to the problem escalation system (subsystem 30).
* **NOTE: *Somebody* needs to know if anything unforeseen happened during the load, especially if a response is critical to continuing the process**

#### Subsystem 23: Backup System

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

##### Backup

* Even if you have a fully redundant system w/ a universal power supply, fully RAID-ed disks, + parallel processors w/ failover, **some system crisis will eventually visit**
* Even with perfect hardware, someone can always drop the wrong table (or database)
* At the risk of stating the obvious, **it is better to prepare for this than to handle it on-the-fly**
* A **full-scale backup system needs to provide the following capabilities**:
* **High performance**
* The backup needs to fit into the allotted timeframe, which may include online backups that don’t impact performance significantly, including real-time partitions
* **Simple administration**
* The administration interface should provide tools that easily allow you to identify objects to back up (including tables, tablespaces, + redo logs), create schedules, + maintain backup verification + logs for subsequent restore
* **Automated, lights-out operations**
* The backup facility must provide storage management services, automated scheduling, media + device handling, reporting, + notification.
* The backup for the DW is usually a *physical* backup, an image of the database at a certain point in time, including indexes + physical layout information

##### Archive and Retrieval

* **Deciding what to move *out of* the DW is a cost-benefit issue**
* It costs money to keep the data around, + it takes up disk space and slows the load + query times
* On the other hand, business users just might need this data to do some critical historical analyses
* Likewise, an auditor may request archived data as part of a compliance procedure
* **Solution = *Not* to throw the data away but to put it some place that costs less but is still accessible**
* **Archiving is the data security blanket for the DW**
* **Cost of online disk storage is dropping so rapidly that it makes sense to plan many of archiving tasks to simply write to disk**
* Especially if disk storage is handled by a separate IT resource, the requirement to “migrate + refresh” is replaced by “refresh.”
* You need to make sure that you can interpret the data at various points in the future
* **How long it takes data to get stale depends on industry, business, + particular data in question**
* In some cases, it is fairly obvious when older data has little value
* Ex: In an industry with rapid evolution of new products + competitors, history doesn’t necessarily help you understand today or predict tomorrow
* After a determination has been made to archive certain data, the issue becomes: “what are the long-term implications of archiving data?”
* Obviously, you need to leverage existing mechanisms to physically move the data from its current media to another media + ensure it can be recovered, along with an **audit trail** that accounts for the accesses and alterations to the data
* *But what does it mean to “keep” old data?*
* Given increasing audit + compliance concerns, you may face archival requirements to preserve this data for 5, 10, or perhaps even 50 years
* *What media should you utilize? Will you be able to read that media in future years?*
* Ultimately, you may find yourself implementing a library system capable of archiving + regularly refreshing the data, + then migrating it to more current structures + media.
* Finally, if archiving data from a system that’s no longer going to be used, may “**sunset**” the data by extracting it from the system + writing it in a vanilla format *independent of the original application*
* You might need to do this if the license to use the application will terminate

#### Subsystem 24: Recovery and Restart System

* After the ETL system is in production, **failures can occur for countless reasons beyond the control of the ETL process**: Network failure, database failure, disk failure, memory failure, data quality failure, unannounced system upgrade, etc.
* To protect yourself from these failures, you **need a solid backup system** (subsystem 23) ***and* a companion recovery + restart system**
* **Must plan for unrecoverable errors during the load because they *will* happen**
* The **system should anticipate this + provide crash recovery, stop, + restart capability**
* First, look for appropriate tools + design processes to minimize the impact of a crash
* Ex: A load process should commit relatively small sets of records at a time + keep track of what has been committed
* The size of the set should be adjustable because the transaction size has performance implications on different DBMSs.
* The recovery + restart system is used, of course, for either resuming a job that has halted or for backing out the whole job + restarting it
* *This system is significantly dependent on the capabilities of the backup system*
* When a failure occurs, the initial knee-jerk reaction is to attempt to salvage whatever has processed + restart the process from that point
* This requires an ETL tool with solid + reliable checkpoint functionality, so it can perfectly determine what has processed + what has not to restart the job at exactly the right point
* **In many cases, it may be best to back out any rows that have been loaded as part of the process + restart from the beginning.**
* often recommend **designing fact tables with a single column primary surrogate key, which is a simple integer that is assigned in sequence as rows are created to be added to the fact table**
* With the fact table surrogate key, you **can easily resume a load that is halted or back out all the rows in the load by constraining on a range of surrogate keys**
* **NOTE**: **Fact table surrogate keys have a number of uses in the ETL back-room**
* 1) As just described, **can be used as the basis for backing out or resuming an interrupted load**
* 2) **Provides immediate + unambiguous identification of a single fact row without needing to constrain multiple dimensions to fetch a unique row**
* 3) **Updates to fact table rows can be replaced by inserts plus deletes because the fact table surrogate key is now the actual key for the fact table**
* Thus, a row containing updated columns can be inserted into the fact table without overwriting the row it is to replace
* When all such insertions are complete, underlying old rows can be deleted in a single step
* 4) **The fact table surrogate key is an ideal parent key to be used in a parent/child design**
* A fact table surrogate key appears as a FK in the child, along w/ the parent’s dimension FKs
* **The longer an ETL process runs, the more you must be aware of vulnerabilities due to failure**
* Designing a modular ETL system made up of efficient processes that are resilient against crashes + unexpected terminations can reduce the risk of a failure resulting in a massive recovery effort
* Careful consideration of when to physically stage data by writing it to disk, along w/ carefully crafted points of recovery + load date/time stamps or sequential fact table surrogate keys enable you to specify appropriate restart logic.

#### Subsystem 25: Version Control System

* The **version control system** is **a “snapshotting” capability for archiving + recovering all the logic + metadata of the ETL pipeline**
* It controls check-out + check-in processing for *all* ETL modules + jobs
* It **should support source comparisons to reveal differences between versions**
* This system **provides a librarian function for saving + restoring the complete ETL context of a single version**
* In certain highly-compliant environments, it will be equally important to archive the complete ETL system context alongside the relevant archived + backup data
* Note that **master version numbers need to be assigned for the overall ETL system**, just like software release version numbers
* **NOTE**: You ***must*** **have a master version number for each part of the ETL system as well as one for the system as a whole**
* And **you *must* be able to restore yesterday’s complete ETL metadata context if it turns out there is a big mistake in the current release**

#### Subsystem 26: Version Migration System

* After the ETL team gets past the difficult process of designing + developing the ETL process + completes the creation of the jobs required to load the DW, the **jobs must be bundled + migrated to the next environment (from Dev to Test and on to Prod) according to the lifecycle adopted by the organization**
* The **version migration system** needs to interface to the version control system to control the process + back out a migration if needed
* It **should provide a single interface for setting connection information for the entire version**
* **Most organizations isolate Dev, Test, and Prod environments**
* **Need to be able to migrate a complete version of the ETL pipeline from Dev, into Test, + finally into Prod**
* **Ideally, the Test system is identically configured to its corresponding Prod system**
* **Everything done to the Prod system should have been designed in Dev + the deployment script tested on the Test environment**
* **Every back-room operation should go through rigorous scripting + testing, whether deploying a new schema, adding a column, changing indexes, changing the aggregate design, modifying a database parameter, backing up, or restoring**
* **Centrally managed front-room operations such as deploying new BI tools, deploying new corporate reports, + changing security plans should be equally rigorously tested + scripted if the BI tools allow it**

#### Subsystem 27: Workflow Monitor

* **Successful DWs are consistently + reliably available**, as agreed to with the business community
* To achieve this goal, **the ETL system must be constantly monitored to ensure the ETL processes are operating efficiently and the DW is being loaded on a consistently timely basis**
* The **job scheduler (subsystem 22) should capture performance data every time an ETL process is initiated**
* This data is part of the process metadata captured in the ETL system
* The **workflow monitor** **leverages the metadata captured by the job scheduler to provide a dashboard + reporting system taking many aspects of the ETL system into consideration**
* **Monitor job status for *all* job runs initiated by the job scheduler 🡪 pending, running, completed, + suspended jobs, and also capture the historical data to support trending performance over time**
* **KPI’s include number of records processed, summaries of errors, + actions taken**
* Most ETL tools capture the metrics for measuring ETL performance
* Be sure to **trigger alerts whenever an ETL job takes significantly more or less time to complete than indicated by the historical record**
* In combination with the job scheduler, the **workflow monitor should also track performance + capture measurements of the performance of infrastructure components** including CPU usage, memory allocation + contention, disk utilization + contention, buffer pool usage, database performance, and server utilization + contention
* Much of this information is process metadata about the ETL system + should be considered as part of the overall metadata strategy (subsystem 34)
* The **workflow monitor has a more significant strategic role than you might suspect** 🡪 It is **the starting point for the analysis of performance problems across the ETL pipeline**
* **ETL performance bottlenecks can occur in many places, + a good workflow monitor shows where the bottlenecks are occurring**
* Chapter 20, discusses many ways to improve performance in the ETL pipeline, but this list is more or less ordered starting with the most important bottlenecks:
* Poorly indexed queries against a source system or intermediate table
* SQL syntax causing wrong optimizer choice
* Insufficient RAM causing thrashing
* Sorting in the RDBMS
* Slow transformation steps
* Excessive I/O
* Unnecessary writes followed by reads
* Dropping and rebuilding aggregates from scratch rather than incrementally
* Filtering (i.e., CDC) applied too late in the pipeline
* Untapped opportunities for parallelizing and pipelining
* Unnecessary transaction logging especially if doing updates
* Network traffic and file transfer overhead

#### Subsystem 28: Sorting System

* **Certain common ETL processes call for data to be sorted in a particular order**, such as aggregating and joining flat file sources
* Because **sorting is such a fundamental ETL processing capability, it is called out as a separate subsystem to ensure it receives proper attention as a component of the ETL architecture**
* There are a variety of technologies available to provide sorting capabilities
* An ETL tool can undoubtedly provide a sort function, the DBMS can provide sorting via the SQL SORT clause, + there are a number of sort utilities available
* Sorting simple delimited text files with a dedicated sort package is awesomely fast
* These packages typically allow a single read operation to produce up to 8 different sorted outputs
* Sorting can produce aggregates where each break row of a given sort is a row for the aggregate table, and sorting + counting is often a good way to diagnose data quality issues
* **Key = to choose the most efficient sort resource to support the requirements w/in your infrastructure**
* The **easy answer for most organizations is to simply utilize the ETL tool’s sort function**
* However, in some situations it may be more efficient to use a dedicated sort package
* Although ETL + DBMS vendors claim to have made up much of the performance differences

#### Subsystem 29: Lineage and Dependency Analyzer

* 2 increasingly important elements being requested of the ETL system are the ability to track both the lineage and dependencies of data in the DW/BI system:
* **Lineage** **= Beginning w/ a specific data element in an intermediate table or BI report, identify the source of that data element, other upstream intermediate tables containing that data element + its sources, + all transformations that data element + its sources have undergone**
* **Dependency = Beginning with a specific data element in a source table or an intermediate table, identify all downstream intermediate tables + final BI reports containing that data element or its derivations + all transformations applied to that data element + its derivations**
* **Lineage analysis is often an important component in a highly compliant environment where you must explain the complete processing flow that changed any data result**
* This means the ETL system must display the ultimate physical sources + all subsequent transformations of any selected data element, chosen either from the middle of the ETL pipeline or on a final delivered report
* **Dependency analysis is important when assessing changes to a source system + the downstream impacts on the DW and ETL system**
* This implies the ability to display all affected downstream data elements + final report fi elds affected by a potential change in any selected data element, chosen either in the middle of the ETL pipeline or an original source (dependency)

#### Subsystem 30: Problem Escalation System

* **Typically, ETL team *develops* the ETL processes + quality assurance team *tests* them thoroughly before they are turned over to the group responsible for day-to-day systems operations**
* To make this work, the **ETL architecture needs to include a proactively-designed problem escalation system similar to what is in place for other production systems.**
* After the ETL processes have been developed + tested, the **1st level of operational support for the ETL system should be a group dedicated to monitoring production applications**
* The ETL development team becomes involved only if the operational support team cannot resolve a production problem.
* Ideally, you have developed ETL processes, wrapped them into an automated scheduler, + have robust workflow monitoring capabilities peering into the ETL processes as they execute
* **The execution of the ETL system should be a hands-off operation**
* Should run like clockwork without human intervention and without fail
* **If a problem *does* occur, the ETL process should *automatically* notify the problem escalation system of any situation that needs attention or resolution.**
* This automatic feed may take the form of simple error logs, operator notification messages, supervisor notification messages, + system developer messages
* The ETL system may notify an individual or a group depending on the severity of the situation or the processes involved
* ETL tools can support a variety of messaging capabilities including e-mail alerts, operator messages, and notifications to mobile devices
* Each notification event should be written to a database used to understand the types of problems that arise, their status, + resolution
* This data forms part of the process metadata captured by the ETL system (subsystem 34)
* **Need to ensure that organizational procedures are in place for proper escalation, so that every problem is resolved appropriately.**
* In general, the support structure for an ETL system should follow a fairly standard support structure
* 1st level support = typically a help desk that’s the first point of contact when a user notices an error
* The help desk is responsible for resolution whenever feasible
* If the help desk cannot resolve the issue, the 2nd level support is notified, typically a SysAdmin or DBA on the production control technical staff capable of supporting general infrastructure failures
* The ETL manager is the 3rd level support + should be knowledgeable to support most issues that arise in the ETL production process
* Finally, when all else fails, the ETL developer should be called in to analyze the situation and assist with resolution

#### Subsystem 31: Parallelizing/Pipelining System

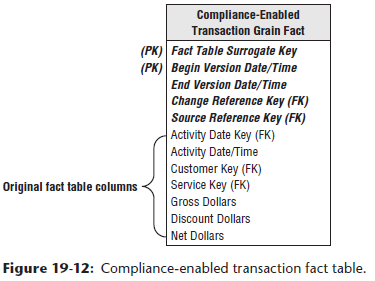
* The **goal of the ETL system, in addition to providing high quality data, is to load the DW *within the allocated processing window***
* In large organizations w/ huge data volumes + a large portfolio of dimensions + facts, **loading the data w/in these constraints can be a challenge**
* The **paralleling/pipelining system provides capabilities to enable the ETL system to deliver within these time constraints.**
* The **goal of this system = to take advantage of multiple processors or grid computing resources commonly available**
* **Highly desirable, + in many cases *necessary*, that parallelizing + pipelining be automatically invoked for *every* ETL process unless specific conditions preclude it from processing in such a manner, such as waiting on a condition in the middle of the process.**
* **Parallelizing = a powerful performance technique at every stage of the ETL pipeline**
* Ex: The extraction process can be parallelized by logically partitioning on ranges of an attribute.
* **Verify that the source DBMS handles parallelism correctly + doesn’t spawn conflicting processes**
* **If possible, choose an ETL tool that handles parallelizing of intermediate transformation processes automatically**
* In some tools it is necessary to hand-create parallel processes, which is fine until you add additional processors, + the ETL system then can’t take advantage of the greater parallelization opportunities unless you modify ETL modules by hand to increase the number of parallel flows

#### Subsystem 32: Security System

* Security is an important consideration for the ETL system
* **A serious security breach is much more likely to come from *within* the organization** than from someone hacking in from the outside
* Although we don’t like to think it, folks on the ETL team present as much a potential threat as any group inside the organization
* We recommend **administering *role-based* security on *all* data + metadata in the ETL system**
* To support compliance requirements, may need to prove that a version of an ETL module hasn’t been changed or show who made changes to a module
* **Enforce comprehensive authorized access to all ETL data + metadata by individual + role**
* Also, **maintain a historical record of all accesses to ETL data + metadata by individual + role**
* Another issue to be careful of = the **bulk data movement process**
* **If you move data across the network, even if it is within the company firewall, it pays to be careful**
* **Make sure to use data encryption or a file transfer utility that uses a secure transfer protocol**
* Another back-room security issue to consider is admin access to the Prod DW server and software
* There’s situations where no one on the team had security privileges +, in other cases, everyone had access to everything.
* **Obviously, many members of the team should have privileged access to the Dev environment, but the Prod DW should be strictly controlled**
* On the other hand, someone from the DW/BI team needs to be able to reset the DW machine if something goes wrong
* Finally, **the backup media should be guarded w/ as much security surrounding them as the online systems**

#### Subsystem 33: Compliance Manager

* In highly-compliant environments, supporting compliance requirements is a significant new requirement for the ETL team
* **Compliance in a DW involves “maintaining the chain of custody” of the data**
* In the same way a police department must carefully maintain the chain of custody of evidence to argue that the evidence has not been changed or tampered with, the **DW must also carefully guard the compliance-sensitive data entrusted to it from the moment it arrives**
* Furthermore, the **DW must always show the *exact* condition and content of such data at any point in time that it may have been under the control of the DW**
* The DW **must also track who had authorized access to the data**
* Finally, when the suspicious auditor looks over your shoulder, you need to link back to an archived + time-stamped version of the data as it was originally received, which you have stored remotely with a trusted third party
* If the DW is prepared to meet all these compliance requirements, the stress of being audited by a hostile government agency or lawyer armed w/ a subpoena should be greatly reduced
* **Compliance requirements may mean you cannot actually change any data, for any reason**
* **If data must be altered, then a new version of the altered records must be inserted into the database**
* **Each row in each table therefore must have begin + end time stamps that accurately represents the span of time when the record was the “current truth.”**
* The big impact of these compliance requirements on the DW can be expressed in simple dimensional modeling terms: **Type 1 + type 3 changes are dead** (i.e., all changes become inserts, no more deletes or overwrites)
* Below shows how a fact table can be augmented so that overwrite changes are converted into a fact table equivalent of a type 2 change



* The original fact table consisted of the lower 7 columns starting with activity date + ending with net dollars
* The original fact table allowed overwrites
* Ex: Perhaps there is a business rule that updates the discount + net dollar amounts after the row is originally created
* In the original version of the table, history is lost when the overwrite change takes place, and the chain of custody is broken.
* To convert the fact table to be compliance-enabled, 5 columns are added, shown in bold
* A fact table surrogate key is created for each original *unmodified* fact table row
* This surrogate key, like a dimension table surrogate key, is just a unique integer that is assigned as each original fact table row is created
* The begin version datetime stamp is the exact time of creation of the fact table row
* Initially, the end version datetime is set to a fictitious date/time in the future
* The change reference is set to “original,”
* The source reference is set to the operational source
* **When an overwrite change is needed, a *new* row is added to the fact table with the same fact table surrogate key, + the appropriate regular columns changed**, such as discount dollars and net dollars
* The begin version datetime column is set to the exact datetime when the change in the database takes place, + the end version datetime is set to a fictitious date/time in the future
* The end version datetime of the original fact row is now set to the exact datetime when the change in the database takes place
* The change reference now provides an explanation for the change, + the source reference provides the source of the revised columns
* Referring to the design above, **a specific moment in time can be selected + the fact table constrained to show exactly what the rows contained at that moment**
* Alterations to a given row can be examined by constraining to a specific fact table surrogate key + sorting by begin version date/time.
* The compliance machinery is a significant addition to a normal fact table (refer to above)
* If the compliance-enabled table is actually used for only demonstrating compliance, then a normal version of the fact table w/ just the original columns can remain as the main operational table, w/ the compliance-enabled table existing only in the background
* **The compliance-enabled table doesn’t need to be indexed for performance because it will not be used in a conventional BI environment.**
* Now, don’t assume all data is now subject to draconian compliance restrictions 🡪 It is essential you receive firm guidelines from the chief compliance officer before taking any drastic steps
* The **foundation of a compliance system is the interaction of several of subsystems already described married to a few key technologies and capabilities**:
* **Lineage analysis**
* Show where a final piece of data came from to prove the original source data + the transformations including stored procedures and manual changes
* This requires full documentation of all the transforms + the technical ability to rerun the transforms against the original data
* **Dependency analysis**
* Show where an original source data element was ever used
* **Version control**
* It may be necessary to rerun the source data through the ETL system in effect at the time, requiring the exact version of the ETL system for any given data source
* **Backup and restore**
* Of course, the requested data may have been archived years ago + need to be restored for audit purposes
* Hopefully, you archived the proper version of the ETL system alongside the data, so both the data + the system can be restored
* It may be necessary to prove the archived data hasn’t been altered
* During the archival process, the data can be hash-coded + the hash and data separated
* Have the hash codes archived separately by a trusted 3rd party
* Then, when demanded, restore the original data, hash code it again, + then compare to the hash codes retrieved from the trusted 3rd party to prove the authenticity of the data
* **Security**
* Show who has accessed or modified the data and transforms
* Be prepared to show roles + privileges for users
* Guarantee the security log can’t be altered by using a write once media.
* **Audit dimension**
* The audit dimension ties runtime metadata context directly w/ the data to capture quality events at the time of the load

#### Subsystem 34: Metadata Repository Manager

* The ETL system is responsible for the use + creation of much of the metadata involved in the DW/BI environment
* **Part of the overall metadata strategy should be to *specifically* capture ETL metadata, including the process metadata, technical metadata, + business metadata**
* Develop a balanced strategy between doing nothing + doing too much
* Make sure there’s time in the ETL development tasks to capture + manage metadata
* And finally, make sure someone on the DW/BI team is assigned the role of metadata manager + owns the responsibility for creating + implementing the metadata strategy