21 Big Data Analytics

In this chapter, we introduce big data in all its glory and show how it expands the mission of the DW/BI system. We conclude with a comprehensive list of big data best practices.

Chapter 21 discusses the following concepts:

- Comparison of two architectural approaches for tackling big data analytics
- Management, architecture, modeling, and governance best practices for dealing with big data

Big Data Overview

What is *big data*? Its bigness is actually not the most interesting characteristic. Big data is structured, semistructured, unstructured, and raw data in many different formats, in some cases looking totally different than the clean scalar numbers and text you have stored in your data warehouses for the last 30 years. Much big data cannot be analyzed with anything that looks like SQL. But most important, big data is a paradigm shift in how you think about data assets, where you collect them, how you analyze them, and how you monetize the insights from the analysis.

The big data movement has gathered momentum as a large number of use cases have been recognized that fall into the category of big data analytics. These use cases include:

- Search ranking
- Ad tracking
- Location and proximity tracking
- Causal factor discovery
- Social CRM
- Document similarity testing

- Genomics analysis
- Cohort group discovery
- In-flight aircraft status
- Smart utility meters
- Building sensors
- Satellite image comparison
- CAT scan comparison
- Financial account fraud detection and intervention
- Computer system hacking detection and intervention
- Online game gesture tracking
- Big science data analysis
- Generic name-value pair analysis
- Loan risk analysis and insurance policy underwriting
- Customer churn analysis

Given the breadth of potential use cases, this chapter focuses on the architectural approaches for tackling big data, along with our recommended best practices, but not specific dimensional designs for each use case.

Conventional RDBMSs and SQL simply cannot store or analyze this wide range of use cases. To fully address big data, a candidate system would have to be capable of the following:

1. Scaling to easily support petabytes (thousands of terabytes) of data.
2. Being distributed across thousands of processors, potentially geographically dispersed and potentially heterogeneous.
3. Storing the data in the original captured formats while supporting query and analysis applications without converting or moving the data.
4. Subsecond response time for highly constrained standard SQL queries.
5. Embedding arbitrarily complex user-defined functions (UDFs) within processing requests.
6. Implementing UDFs in a wide variety of industry-standard procedural languages.
7. Assembling extensive libraries of reusable UDFs crossing most or all the use cases.
8. Executing UDFs as relation scans over petabyte-sized data sets in a few minutes.
9. Supporting a wide variety of data types growing to include images, waveforms, arbitrarily hierarchical data structures, and collections of name-value pairs.
10. Loading data to be ready for analysis, at very high rates, at least gigabytes per second.

11. Integrating data from multiple sources during the load process at very high rates (GB/sec).
12. Loading data into the database before declaring or discovering its structure.
13. Executing certain streaming analytic queries in real time on incoming load data.
14. Updating data in place at full load speeds.
15. Joining a billion-row dimension table to a trillion-row fact table without preclustering the dimension table with the fact table.
16. Scheduling and executing complex multi-hundred node workflows.
17. Being configured without being subject to a single point of failure.
18. Having failover and process continuation when processing nodes fail.
19. Supporting extreme, mixed workloads including thousands of geographically dispersed online users and programs executing a variety of requests ranging from ad hoc queries to strategic analysis, while loading data in batch and streaming fashion.

In response to these challenges, two architectures have emerged: extended RDBMSs and MapReduce/Hadoop.

Extended RDBMS Architecture

Existing RDBMS vendors are extending the classic relational data types to include some of the new data types required by big data, as shown by the arrows in the Figure 21-1.

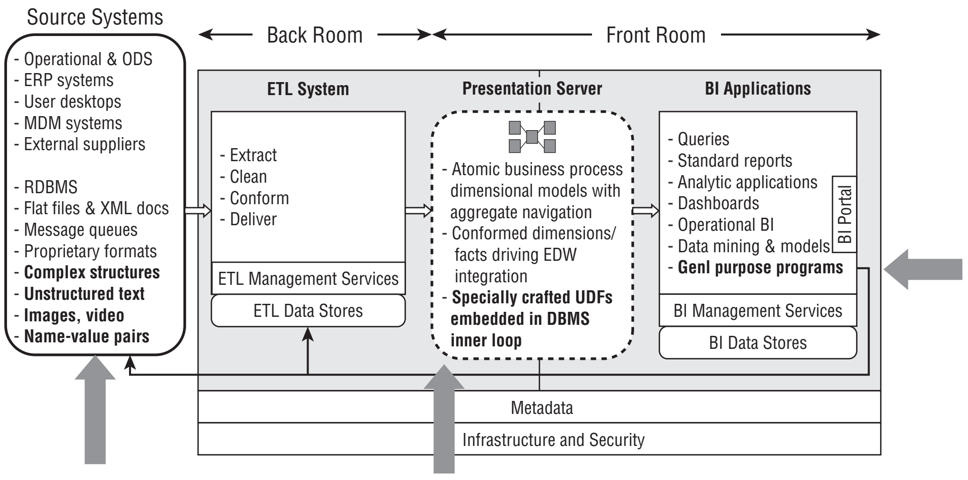

Figure 21-1: Relational DBMS architecture showing big data extensions.

Existing RDBMSs must open their doors to loading and processing a much broader range of data types including complex structures such as vectors, matrices, and custom *hyperstructured data*. At the other end of the spectrum, the RDBMSs need to load and process unstructured and semistructured text, as well as images, video, and collections of name-value pairs, sometimes called *data bags*.

But it is not sufficient for RDBMSs to merely host the new data types as *blobs* to be delivered at some later time to a BI application that can interpret the data, although this alternative has always been possible. To really own big data, RDBMSs must allow the new data types to be processed within the DBMS inner loop by means of specially crafted user-defined functions (UDFs) written by business user analysts.

Finally, a valuable use case is to process the data twice through the RDBMS, where in the first pass the RDBMS is used as a *fact extractor* on the original data, and then in the second pass, these results are automatically fed back to the RDBMS input as conventional relational rows, columns, and data types.

MapReduce/Hadoop Architecture

The alternative architecture, MapReduce/Hadoop, is an open source top-level Apache project with many components. MapReduce is a processing framework originally developed by Google in the early 2000s for performing web page searches across thousands of physically separated machines. The MapReduce approach is extremely general. Complete MapReduce systems can be implemented in a variety of languages; the most significant implementation is in Java. MapReduce is actually a UDF execution framework, where the "F" can be extraordinarily complex. The most significant implementation of MapReduce is Apache Hadoop, known simply as Hadoop. The Hadoop project has thousands of contributors and a whole industry of diverse applications. Hadoop runs natively on its own *Hadoop distributed file system* (*HDFS*) and can also read and write to Amazon S3 and others. Conventional database vendors are also implementing interfaces to allow Hadoop jobs to be run over massively distributed instances of their databases.

NOTE A full discussion of the MapReduce/Hadoop architecture is beyond the scope of this book. Interested readers are invited to study the in depth big data resources available on our website at `www.kimballgroup.com`.

Comparison of Big Data Architectures

The two big data architecture approaches have separate long-term advantages and are likely to coexist far into the future. At the time of this writing, the characteristics of the two architectures are summarized in the Figure 21-2.

Extended Relational DBMS	MapReduce/Hadoop
Proprietary, mostly	Open source
Expensive	Less expensive
Data must be structured	Data does not require structuring
Great for speedy indexed lookups	Great for massive full data scans
Deep support for relational semantics	Indirect support for relational semantics, e.g., Hive
Indirect support for complex data structures	Deep support for complex data structures
Indirect support for iteration, complex branching	Deep support for iteration, complex branching
Deep support for transaction processing	Little or no support for transaction processing

Figure 21-2: Comparison of relational DBMS and MapReduce/Hadoop architectures.

Recommended Best Practices for Big Data

Although the big data marketplace is anything but mature, the industry now has a decade of accumulated experience. In that time, a number of best practices specific to big data have emerged. This section attempts to capture these best practices, steering a middle ground between high-level motherhood admonitions versus down-in-the-weeds technical minutiae specific to a single tool.

Having said that, one should recognize that the industry has a well-tested set of best practices developed over the last 30 years for relationally-based data warehouses that surely are relevant to big data. We list them briefly. They are to:

- Drive the choice of data sources feeding the data warehouse from business needs.
- Focus incessantly on user interface simplicity and performance.
- Think dimensionally: Divide the world into dimensions and facts.
- Integrate separate data sources with conformed dimensions.
- Track time variance with slowly changing dimensions (SCDs).
- Anchor all dimensions with durable surrogate keys.

In the remainder of this section, we divide big data best practices into four categories: management, architecture, data modeling, and governance.

Management Best Practices for Big Data

The following best practices apply to the overall management of a big data environment.

Structure Big Data Environments Around Analytics

Consider structuring big data environments around analytics and not ad hoc querying or standard reporting. Every step in the data pathway from original source to analyst's screen must support complex analytic routines implemented as user-defined functions (UDFs) or via a metadata-driven development environment that can be programmed for each type of analysis. This includes loaders, cleansers, integrators, user interfaces, and finally BI tools, as further discussed in the architectural best practices section.

Delay Building Legacy Environments

It's not a good idea to attempt building a legacy big data environment at this time. The big data environment is changing too rapidly to consider building a long-lasting legacy foundation. Rather, plan for disruptive changes coming from every direction: new data types, competitive challenges, programming approaches, hardware, networking technology, and services offered by literally hundreds of new big data providers. For the foreseeable future, maintain a balance among several implementation approaches including Hadoop, traditional grid computing, pushdown optimization in an RDBMS, on-premise computing, cloud computing, and even the mainframe. None of these approaches will be the single winner in the long run. Platform as a service (PaaS) providers offer an attractive option that can help assemble a compatible set of tools.

Think of Hadoop as a flexible, general purpose environment for many forms of ETL processing, where the goal is to add sufficient structure and context to big data so that it can be loaded into an RDBMS. The same data in Hadoop can be accessed and transformed with Hive, Pig, HBase, and MapReduce code written in a variety of languages, even simultaneously.

This demands flexibility. Assume you will reprogram and rehost all your big data applications within two years. Choose approaches that can be reprogramed and rehosted. Consider using a metadata-driven codeless development environment to increase productivity and help insulate from underlying technology changes.

Build From Sandbox Results

Consider embracing sandbox silos and building a practice of productionizing sandbox results. Allow data scientists to construct their data experiments and prototypes using their preferred languages and programming environments. Then, after proof of concept, systematically reprogram these implementations with an IT turnover team. Here are a couple of examples to illustrate this recommendation:

The production environment for custom analytic programming might be MatLab within PostgreSQL or SAS within a Teradata RDBMS, but the data scientists might be building their proofs of concept in a wide variety of their own preferred languages and architectures. The key insight here: IT must be uncharacteristically tolerant of the range of technologies the data scientists use and be prepared in many cases to re-implement the data scientists' work in a standard set of technologies that can be supported over the long haul. The sandbox development environment might be custom R code directly accessing Hadoop, but controlled by a metadata-driven driven ETL tool. Then when the data scientist is ready to hand over the proof of concept, much of the logic could immediately be redeployed under the ETL tool to run in a grid computing environment that is scalable, highly available, and secure.

Try Simple Applications First

You can put your toe in the water with a simple big data application, such as backup and archiving. While starting with a big data program, and searching for valuable business use cases with limited risk and when assembling the requisite big data skills, consider using Hadoop as a low-cost, flexible backup and archiving technology. Hadoop can store and retrieve data in the full range of formats from totally unstructured to highly structured specialized formats. This approach may also enable you address the *sunsetting* challenge where original applications may not be available in the distant future (perhaps because of licensing restrictions); you can dump data from those applications into your documented format.

Architecture Best Practices for Big Data

The following best practices affect the overall structure and organization of your big data environment.

Plan a Data Highway

You should plan for a logical *data highway* with multiple caches of increasing latency. Physically implement only those caches appropriate for your environment. The data highway can have as many as five caches of increasing data latency, each with its distinct analytic advantages and trade-offs, as shown in Figure 21-3.

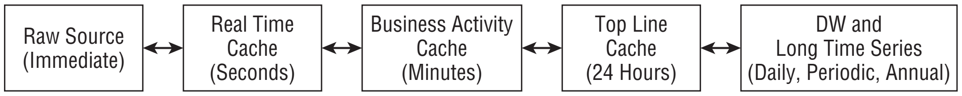
Raw Source (Immediate)
Real Time Cache (Seconds)
Business Activity Cache (Minutes)
Top Line Cache (24 Hours)
DW and Long Time Series (Daily, Periodic, Annual)

Figure 21-3: Big data caches of increasing latency and data quality.

Here are potential examples of the five data caches:

- **Raw source applications:** Credit card fraud detection, immediate complex event processing (CEP) including network stability and cyber attack detection.
- **Real time applications:** Web page ad selection, personalized price promotions, on-line games monitoring.
- **Business activity applications:** Low-latency KPI dashboards pushed to users, trouble ticket tracking, process completion tracking, "fused" CEP reporting, customer service portals and dashboards, and mobile sales apps.

- **Top line applications:** Tactical reporting, promotion tracking, midcourse corrections based on social media buzz. *Top line* refers to the common practice by senior managers of seeing a quick top line review of what has happened in the enterprise over the past 24 hours.
- **Data warehouse and long time series applications:** All forms of reporting, ad hoc querying, historical analysis, master data management, large scale temporal dynamics, and Markov chain analysis.

Each cache that exists in a given environment is physical and distinct from the other caches. Data moves from the raw source down this highway through ETL processes. There may be multiple paths from the raw source to intermediate caches. For instance, data could go to the real-time cache to drive a zero latency-style user interface, but at the same time be extracted directly into a daily top line cache that would look like a classic operational data store (ODS). Then the data from this ODS could feed the data warehouse. Data also flows in the reverse direction along the highway. We'll discuss implementing backflows later in this section.

Much of the data along this highway must remain in nonrelational formats ranging from unstructured text to complex multistructured data, such as images, arrays, graphs, links, matrices, and sets of name-value pairs.

Build a Fact Extractor from Big Data

It's a good idea to use big data analytics as a fact extractor to move data to the next cache. For example, the analysis of unstructured text tweets can produce a whole set of numerical, trendable sentiment measures including share of voice, audience engagement, conversation reach, active advocates, advocate influence, advocacy impact, resolution rate, resolution time, satisfaction score, topic trends, sentiment ratio, and idea impact.

Build Comprehensive Ecosystems

You can use big data integration to build comprehensive ecosystems that integrate conventional structured RDBMS data, documents, e-mails, and in-house, business-oriented social networking. One of the potent messages from big data is the ability to integrate disparate data sources of different modalities. You get streams of data from new data producing channels such as social networks, mobile devices, and automated alert processes. Imagine a big financial institution handling millions of accounts, tens of millions of associated paper documents, and thousands of professionals both within the organization and in the field as partners or customers. Now set up a secure social network of all the trusted parties to communicate as business is conducted. Much of this communication is significant and should be saved in a queryable way. You could capture all this information in Hadoop, dimensionalize it (as you see in the following modeling best practices), use it in the course of business, and then back it up and archive it.

Plan for Data Quality

You can plan for data quality to be better further along the data highway. This is the classic trade-off of latency versus quality. Analysts and business users must accept the reality that very low latency (that is, immediate) data is unavoidably dirty because there are limits to how much cleansing and diagnosing can be done in very short time intervals. Tests and corrections on individual field contents can be performed at the fastest data transfer rates. Tests and corrections on structural relationships among fields and across data sources are necessarily slower. Tests and corrections involving complex business rules range from being instantaneous (such as a set of dates being in a certain order) to taking arbitrarily long times (such as waiting to see if a threshold of unusual events has been exceeded). And finally, slower ETL processes, such as those feeding the daily top line cache, often are built on fundamentally more complete data, for example where incomplete transaction sets and repudiated transactions have been eliminated. In this case, the instantaneous data feeds simply do not have the correct information.

Add Value to Data as Soon as Possible

You should apply filtering, cleansing, pruning, conforming, matching, joining, and diagnosing at the earliest touch points possible. This is a corollary of the previous best practice. Each step on the data highway provides more time to add value to the data. Filtering, cleansing, and pruning the data reduces the amount transferred to the next cache and eliminates irrelevant or corrupted data. To be fair, there is a school of thought that applies cleansing logic only at analysis run time because cleansing might delete "interesting outliers." Conforming takes the active step of placing highly administered enterprise attributes into major entities such as customer, product, and date. The existence of these conformed attributes allows high value joins to be made across separate application domains. A shorter name for this step is "integration!" Diagnosing allows many interesting attributes to be added to data, including special confidence tags and textual identifiers representing behavior clusters identified by a data mining professional.

Implement Backflow to Earlier Caches

You should implement backflows, especially from the data warehouse, to earlier caches on the data highway. The highly administered dimensions in the data warehouse, such as customer, product, and date, should be connected back to data in earlier caches. Ideally, all that is needed are unique durable keys for these entities in all the caches. The corollary here is that Job One in each ETL step from one cache to the next is to replace idiosyncratic proprietary keys with the unique durable keys so that analysis in each cache can take advantage of the rich upstream content with a simple join on the unique durable key. Can this ETL step be performed even when transferring raw source data into the real time cache in less than a second? Maybe….

Dimension data is not the only data to be transferred back down the highway toward the source. Derived data from fact tables, such as historical summaries and complex data mining findings, can be packaged as simple indicators or grand totals and then transferred to earlier caches on the data highway.

Implement Streaming Data

You should implement streaming data analytics in selected data flows. An interesting angle on low latency data is the need to begin serious analysis on the data as it streams in, but possibly far before the data transfer process terminates. There is significant interest in streaming analysis systems, which allow SQL-like queries to process the data as it flows into the system. In some use cases, when the results of a streaming query surpass a threshold, the analysis can be halted without running the job to the bitter end. An academic effort, known as continuous query language (CQL), has made impressive progress in defining the requirements for streaming data processing including clever semantics for dynamically moving time windows on the streaming data. Look for CQL language extensions and streaming data query capabilities in the load programs for both RDBMSs and HDFS deployed data sets. An ideal implementation would allow streaming data analysis to take place while the data is loaded at gigabytes per second.

Avoid Boundary Crashes

You should implement far limits on scalability to avoid a *boundary crash*. In the early days of computer programming, when machines had pathetically small hard drives and real memories, boundary crashes were common and were the bane of applications development. When the application ran out of disk space or real memory, the developer resorted to elaborate measures, usually requiring significant programming that added nothing to the application's primary function. Boundary crashes for normal database applications have more or less been eliminated, but big data raises this issue again. Hadoop is an architecture that dramatically reduces programming scalability concerns because you can, for the most part, indefinitely add commodity hardware. Of course, even commodity hardware must be provisioned, plugged in, and have high bandwidth network connections. The lesson is to plan far ahead for scaling out to huge volumes and throughputs.

Move Prototypes to a Private Cloud

Consider performing big data prototyping on a public cloud and then moving to a private cloud. The advantage of a public cloud is it can be provisioned and scaled up instantly. In those cases in which the sensitivity of the data allows quick in-and-out prototyping, this can be effective. Just remember not to leave a huge data set online with the public cloud provider over the weekend when the programmers have gone home!

However, keep in mind that in some cases in which you are trying to exploit data locality with rack-aware MapReduce processes, you may not use a public cloud service because it may not provide the data storage control needed.

Strive for Performance Improvements

Search for and expect tenfold to hundredfold performance improvements over time, recognizing the paradigm shift for analysis at high speeds. The openness of the big data marketplace has encouraged hundreds of special purpose tightly coded solutions for specific kinds of analysis. This is a giant blessing and a curse. When free from being controlled by a big vendor's RDBMS optimizer and inner loop, smart developers can implement spot solutions that are truly 100 times as fast as standard techniques. For instance, some impressive progress has been made on the infamous "big join" problem in which a billion-row dimension is joined to a trillion-row fact table. The challenge is these individual spot solutions may not be part of a unified single architecture.

One very current big data theme is visualization of data sets. "Flying around" a petabyte of data requires spectacular performance! Visualization of big data is an exciting new area of development that enables both analysis and discovery of unexpected features and data profiling.

Another exciting application that imposes huge performance demands is "semantic zooming without pre-aggregations," in which the analyst descends from a highly aggregated level to progressively more detailed levels in unstructured or semistructured data, analogous to zooming in on a map.

The important lesson behind this best practice is that revolutionary advances in your power to consume and analyze big data can result from 10x to 100x performance gains, and you have to be prepared to add these developments to your suite of tools.

Monitor Compute Resources

You should separate big data analytic workloads from the conventional data warehouse to preserve service level agreements. If your big data is hosted in Hadoop, it probably doesn't compete for resources with your conventional RDBMS-based data warehouse. However, be cautious if your big data analytics run on the data warehouse machine because big data requirements change rapidly and inevitably in the direction of requiring more compute resources.

Exploit In-Database Analytics

Remember to exploit the unique capabilities of in-database analytics. The major RDBMS players all significantly invest in in-database analytics. After you pay the price of loading data into relational tables, SQL can be combined with analytic extensions in extremely powerful ways. In particular, PostgreSQL, an open source database, has extensible syntax for adding powerful user defined functions in the inner loop.

Data Modeling Best Practices for Big Data

The following best practices affect the logical and physical structures of the data.

Think Dimensionally

By thinking dimensionally, we mean dividing the world into dimensions and facts. Business users find the concept of dimensions to be natural and obvious. No matter what the format of the data, the basic associated entities such as customer, product, service, location, or time can always be found. In the following best practice you see how, with a little discipline, dimensions can be used to integrate data sources. But before getting to the integration finish line, you must identify the dimensions in each data source and attach them to every low-level atomic data observation. This process of dimensionalization is a good application for big data analytics. For example, a single Twitter tweet "Wow! That is awesome!" may not seem to contain anything worth dimensionalizing, but with some analysis you often can get customer (or citizen or patient), location, product (or service or contract or event), marketplace condition, provider, weather, cohort group (or demographic cluster), session, triggering prior event, final outcome, and the list goes on. Some form of automated dimensionalizing is required to stay ahead of the high-velocity streams of data. As we point out in a subsequent best practice, incoming data should be fully dimensionalized at the earliest extraction step in as close to real time as possible.

Integrate Separate Data Sources with Conformed Dimensions

Conformed dimensions are the glue that holds together separate data sources and enable them to be combined in a single analysis. Conformed dimensions are perhaps the most powerful best practice from the conventional DW/BI world that should be inherited by big data.

The basic idea behind conformed dimensions is the presence of one or more enterprise attributes (fields) in the versions of dimensions associated with separate data sources. For instance, every customer-facing process in an enterprise will have some variation of a customer dimension. These variations of the customer dimension may have different keys, different field definitions, and even different granularity. But even in the worst cases of incompatible data, one or more enterprise attributes can be defined that can be embedded in all the customer dimension variations. For instance, a customer demographic category is a plausible choice. Such a descriptor could be attached to nearly every customer dimension, even those at higher levels of aggregation. After this has been done, analyses on this customer demographic category can cross every participating data source with a simple sort-merge process after separate queries are run against the different data sources. Best of all, the step of introducing the enterprise attributes into the separate databases can be done in an incremental, agile, and nondisruptive way as described in Chapter 8: Customer Relationship Management and Chapter 19: ETL Subsystems and Techniques. All existing analysis applications will continue to run as the conformed dimension content is rolled out.

Anchor Dimensions with Durable Surrogate Keys

If there is one lesson we have learned in the data warehouse world, it is not to anchor major entities such as customer, product, and time with the natural keys defined by a specific application. These natural keys turn out to be a snare and a delusion in the real world. They are incompatible across applications and are poorly administered, and they are administered by someone else who may not have the interests of the data warehouse at heart. The first step in every data source is to augment the natural key coming from a source with an enterprisewide durable surrogate key. Durable means there is no business rule that can change the key. The durable key belongs to the DW/BI system, not to the data source. Surrogate means the keys themselves are simple integers either assigned in sequence or generated by a robust hashing algorithm that guarantees uniqueness. An isolated surrogate key has no applications content. It is just an identifier.

The big data world is filled with obvious dimensions that must possess durable surrogate keys. Earlier in this chapter when we proposed pushing data backward down the data highway, we relied on the presence of the durable surrogate keys to make this process work. We also stated that Job One on every data extraction from a raw source was to embed the durable surrogate keys in the appropriate dimensions.

Expect to Integrate Structured and Unstructured Data

Big data considerably broadens the integration challenge. Much big data will never end up in a relational database; rather it will stay in Hadoop or a grid. But after you are armed with conformed dimensions and durable surrogate keys, all forms of data can be combined in single analyses. For example, a medical study can select a group of patients with certain demographic and health status attributes and then combine their conventional DW/BI data with image data (photographs, X-rays, EKGs, and so on), free form text data (physician's notes), social media sentiments (opinions of treatment), and cohort group linkages (patients with similar situations), and doctors with similar patients.

Use Slowly Changing Dimensions

You should track time variance with slowly changing dimensions (SCDs). Tracking time variance of dimensions is an old and venerable best practice from the data warehouse world. Chapter 5: Procurement makes a powerful case for using SCD techniques for handling time variance. This is just as important in the big data world as it is in the conventional data warehouse world.

Declare Data Structure at Analysis Time

You must get used to not declaring data structures until analysis time. One of the charms of big data is putting off declaring data structures at the time of loading into Hadoop or a data grid. This brings many advantages. The data structures may not be understood at load time. The data may have such variable content that a single data structure either makes no sense or forces you to modify the data to fit into a structure. If you can load data into Hadoop, for instance, without declaring its structure, you can avoid a resource intensive step. And finally, different analysts may legitimately see the same data in different ways. Of course, there is a penalty in some cases because data without a declared structure may be difficult or impossible to index for rapid access, as in an RDBMS. However, most big data analysis algorithms process entire data sets without expecting precise filtering of subsets of the data.

This best practice conflicts with traditional RDBMS methodologies, which puts a lot of emphasis on modeling the data carefully before loading. But this does not lead to a deadly conflict. For data destined for an RDBMS, the transfer from a Hadoop or data grid environment and from a name-value pair structure into RDBMS named columns can be thought of as a valuable ETL step.

Load Data as Simple Name-Value Pairs

Consider building technology around name-value pair data sources. Big data sources are filled with surprises. In many cases, you open the fire hose and discover unexpected or undocumented data content, which you must nevertheless load at gigabytes per second. The escape from this problem is to load this data as simple name-value pairs. For example, if an applicant were to disclose her financial assets, as illustrated with Figures 8-7 and 8-8, she might declare something unexpected such as “rare postage stamp = $10,000.” In a name-value pair data set, this would be loaded gracefully, even though you had never seen “rare postage stamp” and didn’t know what to do with it at load time. Of course, this practice meshes nicely with the previous practice of deferring the declaration of data structures until past load time.

Many MapReduce programming frameworks require data to be presented as name-value pairs, which makes sense given the complete possible generality of big data.

Rapidly Prototype Using Data Virtualization

Consider using data virtualization to allow rapid prototyping and schema alterations. Data virtualization is a powerful technique for declaring different logical data structures on underlying physical data. Standard view definitions in SQL are a good example of data virtualization. In theory, data virtualization can present a data source in any format the analyst needs. But data virtualization trades off the cost of computing at run time with the cost of ETL to build physical tables before run time. Data virtualization is a powerful way to prototype data structures and make rapid alterations or provide distinct alternatives. The best data virtualization strategy is to expect to materialize the virtual schemas when they have been tested and vetted and the analysts want the performance improvements of actual physical tables.

Data Governance Best Practices for Big Data

The following best practices apply to managing big data as a valuable enterprise asset.

There is No Such Thing as Big Data Governance

Now that we have your attention, the point is that data governance must be a comprehensive approach for the entire data ecosystem, not a spot solution for big data in isolation. Data governance for big data should be an extension of the approach used to govern all the enterprise data. At a minimum, data governance embraces privacy, security, compliance, data quality, metadata management, master data management, and the business glossary that exposes definitions and context to the business community.

Dimensionalize the Data before Applying Governance

Here is an interesting challenge big data introduces: You must apply data governance principles even when you don't know what to expect from the content of the data. You may receive data arriving at gigabytes per minute, often as name-value pairs with unexpected content. The best chance at classifying data in ways that are important to your data governance responsibilities is to dimensionalize it as fully as possible at the earliest stage in the data pipeline. Parse it, match it, and apply identity resolution on-the-fly. We made this same point when arguing for the benefits of data integration, but here we advocate against even using the data before this dimensionalizing step.

Privacy is the Most Important Governance Perspective

If you analyze data sets that include identifying information about individuals or organizations, privacy is the most important governance perspective. Although every aspect of data governance looms as critically important, in these cases, privacy carries the most responsibility and business risk. Egregious episodes of compromising the privacy of individuals or groups can damage your reputation, diminish marketplace trust, expose you to civil lawsuits, and get you in trouble with the law. At the least, for most forms of analysis, personal details must be masked, and data aggregated enough to not allow identification of individuals. At the time of this writing, special attention must be paid when storing sensitive data in Hadoop because after data is written to Hadoop, Hadoop doesn't manage updates very well. Data should either be masked or encrypted on write (persistent data masking) or data should be masked on read (dynamic data masking).

Don't Choose Big Data over Governance

Don't put off data governance completely in the rush to use big data. Even for exploratory big data prototype projects, maintain a checklist of issues to consider when going forward. You don't want an ineffective bureaucracy, but maybe you can strive to deliver an agile bureaucracy!

Summary

Big data brings a host of changes and opportunities to IT, and it is easy to think that a whole new set of rules must be created. But with the benefit of big data experience, many best practices have emerged. Many of these practices are recognizable extensions from the DW/BI world, and admittedly quite a few are new and novel ways of thinking about data and the mission of IT. But the recognition that the mission has expanded is welcome and is in some ways overdue. The current explosion of data-collecting channels, new data types, and new analytic opportunities mean the list of best practices will continue to grow in interesting ways.