# Kimball Data Warehouse Toolkit

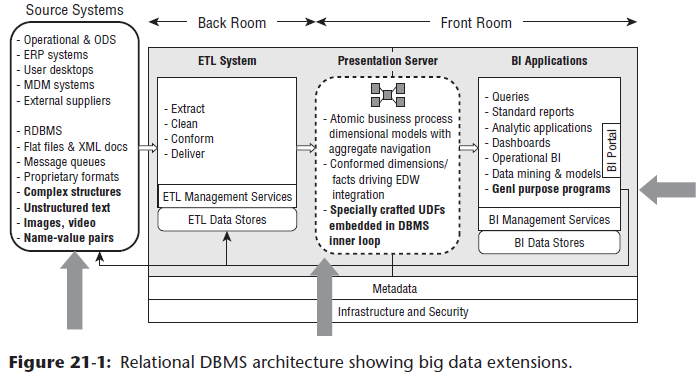
## Ch 21 – Big Data Analytics

### Big Data Overview

* What is “big” data? 🡺 Its bigness is actually not the most interesting characteristic
* Big data = structured, semi-structured, unstructured, + raw data in many different formats, in some cases looking totally different than the clean scalar numbers + text that’s been stored in DWs 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, + 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
* Including: Search ranking, ad tracking, location + 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 + intervention, computer system hacking detection + intervention, online game gesture tracking, big science data analysis, generic name-value pair analysis, loan risk analysis + insurance policy underwriting, customer churn analysis, etc.
* 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 PB (thousands of TB) of data.
* 2. Being distributed across thousands of processors, potentially geographically dispersed + potentially heterogeneous.
* 3. Storing the data in the original captured formats while supporting query + analysis applications w/out 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 GB per second
* 11. Integrating data from multiple sources during a 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 pre-clustering the dimension table with the fact table.
* 16. Scheduling + 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 + streaming fashion.
* In response to such challenges, 2 architectures have emerged: **extended RDBMSs** + **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 below



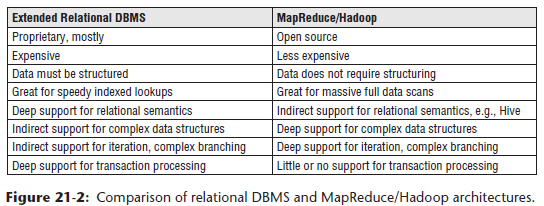
* Existing RDBMSs must open their doors to loading + processing a much **broader range of data types** including complex structures such as vectors, matrices, + custom **hyper-structured data**
* At the other end of the spectrum, RDBMSs **need to load and process unstructured + semi-structured text**, as well as images, video, + collections of name-value pairs, sometimes called **data bags**
* But it is ***not* sufficient for RDBMSs to merely host the new data types as blobsto 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 1st pass the RDBMS is used as a *fact extractor* on the original data, + then in the 2nd pass, these results are automatically fed back to the RDBMS input as conventional relational rows, columns, + data types**

#### MapReduce/Hadoop Architecture

* The alternative architecture, MapReduce/Hadoop = an open source top-level Apache project w/ 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**.
* **Most significant implementation of MapReduce = Apache Hadoop, known simply as Hadoop**
* The Hadoop project has thousands of contributors + a whole industry of diverse applications.
* Hadoop runs natively on its own **Hadoop distributed file system (HDFS)** + can also read + write to Amazon S3 + others
* Conventional database vendors are also implementing interfaces to allow Hadoop jobs to be run over massively distributed instances of their databases

#### Comparison of Big Data Architectures

* The two big data architecture approaches have separate long-term advantages + are likely to coexist far into the future
* At the time of this writing, the characteristics of the two architectures are summarized below



### Recommended Best Practices for Big Data

* Although the big data marketplace is anything but mature, the industry now has a decade of accumulated experience + a number of best practices specific to big data have emerged, w/ 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 DW that surely are relevant to big data:
* Drive the choice of data sources feeding the DW from business needs.
* Focus incessantly on **user interface simplicity + performance**.
* **Think dimensionally**: Divide the world into dimensions + facts.
* Integrate separate data sources with **conformed dimensions**.
* Track time variance with **SCDs**
* Anchor all dimensions with **durable surrogate keys.**
* Can divide big data best practices into 4 categories: management, architecture, data modeling, + 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

* **Structure big data environments around analytics** + *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 UDFs or via a metadata-driven development environment that can be programmed for each type of analysis
* This includes loaders, cleansers, integrators, user interfaces, + 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.
* **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, + 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-prem computing, cloud computing, + 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 + context to big data so it can be loaded into an RDBMS
* The same data in Hadoop can be accessed + transformed with Hive, Pig, HBase, + MapReduce code written in a variety of languages, even simultaneously
* This **demands flexibility 🡪 Assume you will reprogram + rehost all your big data applications within 2 years**
* **Choose approaches that can be reprogramed + rehosted**
* Consider using a metadata-driven codeless development environment to increase productivity + help insulate from underlying technology changes

##### Build From Sandbox Results

* Consider embracing **sandbox silos + building a practice of productionizing sandbox results**
* Allow data scientists to construct their data experiments + prototypes using their preferred languages + programming environments
* Then, after POC, systematically reprogram these implementations with an IT turnover team
* Here are a couple of examples to illustrate this recommendation:
* Ex: The production environment for custom analytic programming might be MatLab w/in PostgreSQL or SAS within a Teradata RDBMS, but the data scientists might be building their POCs in a wide variety of their own preferred languages + architectures
* The **key insight here: IT must be uncharacteristically tolerant of the range of tech data scientists use + 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 POC, 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

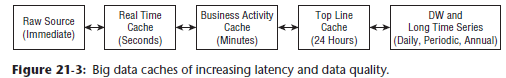
* You can put your toe in the water with a simple big data application, such as backup + archiving
* While starting with a big data program, + searching for valuable business use cases w/ limited risk + when assembling the requisite big data skills, consider using Hadoop as a low-cost, flexible backup + archiving technology.
* Hadoop can store + 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 + organization of your big data environment

##### Plan a Data Highway

* You should plan for a **logical data highwaywith multiple caches of increasing latency**
* **Physically implement only those caches appropriate for your environment**
* The data highway can have as many as 5 caches of increasing data latency, each w/ its distinct analytic advantages + trade-offs, as shown below



* Here are potential examples of the 5 data caches:
* **Raw source applications:** Credit card fraud detection, immediate complex event processing (CEP) including network stability, 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 + dashboards, + 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
* **DW and long time series applications:** All forms of reporting, ad hoc querying, historical analysis, master data management, large scale temporal dynamics, + 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 UI, 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 DW
* **Data also flows in the reverse direction along the highway**
* **Much of the data along this highway must remain in non-relational formats** ranging from unstructured text to complex multi-structured 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**
* Ex: The analysis of unstructured text tweets can produce a whole set of numerical, trend-able 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, + idea impact

##### Build Comprehensive Ecosystems

* 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, + automated alert processes
* Imagine a big financial institution handling millions of accounts, tens of millions of associated paper documents, + thousands of professionals both w/in the organization + 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 + should be saved in a query-able way**
* Could capture all this information in Hadoop, dimensional-ize it (as sees in modeling best practices), use it in the course of business, + then back it up + archive it

##### Plan for Data Quality

* **Can plan for data quality to be better further along the data highway** 🡪 This is the **classic trade-off of latency versus quality**
* **Analysts + business users must accept the reality that very low latency (i.e., immediate) data is unavoidably dirty because there are limits to how much cleansing + diagnosing can be done in very short time intervals**
* Tests + corrections on individual field contents can be performed at the fastest data transfer rates
* Tests + corrections on structural relationships among fields + across data sources are necessarily slower
* Tests + 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)
* 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 + 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

* **Apply filtering, cleansing, pruning, conforming, matching, joining, + diagnosing at the earliest touch points possible** (a corollary of the previous best practice)
* **Each step on the data highway provides more time to add value to the data**
* **Filtering, cleansing, + pruning the data reduces the amount transferred to the next cache + 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, + 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 + textual identifiers representing behavior clusters identified by a data mining professional

##### Implement Backflow to Earlier Caches

* **Implement backflows, *especially from the DW*, to earlier caches on the data highway**
* The **highly-administered dimensions in the DW, such as customer, product, + 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 + complex data mining findings, can be packaged as simple indicators or grand totals + then transferred to earlier caches on the data highway**

##### Implement Streaming Data

* **Implement streaming data analytics in *selected* data flows**
* An interesting angle on low latency data = 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 + 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 GB/sec**

##### Avoid Boundary Crashes

* **Implement far limits on scalability to avoid a boundary crash**
* In the early days of CPU programming, when machines had very small hard drives + real memories, boundary crashes were common + 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, + 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 + then moving to a private cloud**
* The **advantage of a public cloud = it can be provisioned + 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 w/ 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 + expect 10X to 100X performance improvements over time, recognizing the paradigm shift for analysis at high speeds
* The **openness of the big data marketplace has encouraged 100s 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 100X as fast as standard techniques
* Ex: 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 = 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 PB of data requires spectacular performance
* Visualization of big data is an exciting new area of development that enables both analysis + discovery of unexpected features + data profiling
* Another exciting application that imposes huge performance demands = “semantic zooming without pre-aggregations,” in which the analyst descends from a highly-aggregated level to progressively more detailed levels in unstructured or semi-structured 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, + 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 DW to preserve SLAs**
* If your big data is hosted in Hadoop, it probably doesn’t compete for resources with your conventional RDBMS-based DW
* However, be cautious if your big data analytics run on the DW machine because big data requirements change rapidly + 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 integration, you must identify the dimensions in each data source and attach them to every low-level atomic data observation**.
* This process of **dimensional-ization is a good application for big data analytics**
* Ex: A single Twitter tweet “Wow! That is awesome!” may not seem to contain anything worth dimensional-izing, 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, etc.
* **Some form of *automated* dimensional-izing is required to stay ahead of the high-velocity streams of data**
* As pointed out in a subsequent best practice, **incoming data should be fully dimensional-ized at the earliest extraction step in as close to real time as possible**

##### Integrate Separate Data Sources with Conformed Dimensions

* **Conformed dimensions = the glue that holds together separate data sources + enable them to be combined in a single analysis**
* **Conformed dimensions = 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 = the presence of 1+ enterprise attributes (fields) in the versions of dimensions associated with separate data sources**
* Ex: 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, field definitions, even granularity
* **But even in the worst cases of incompatible data, 1+ enterprise attributes can be defined that can be embedded in all the customer dimension variations**
* Ex: 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, + non-disruptive way** as described in Chapters 8 and 19
* **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 **1 lesson we have learned in the DW 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 + a delusion in the real world 🡪 incompatible across applications, are poorly administered, + are administered by someone else who may not have the interests of the DW at heart**
* The **1st step in every data source is to augment the natural key coming from a source with an enterprise-wide durable surrogate key that belongs to the DW/BI system, NOT to the data source**
* **“Durable” means there is no business rule that can change the key**
* **“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 w/ 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
* 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 an RBD, rather it will stay in Hadoop or a grid
* But **after you are armed w/ conformed dimensions + durable surrogate keys, all forms of data can be combined in single analyses**
* Ex: A medical study can select a group of patients w/ certain demographic + health status attributes + then combine their conventional DW/BI data w/ image data (photographs, X-rays, EKGs, etc.), free form text data (physician’s notes), social media sentiments (opinions of treatment), cohort group linkages (patients with similar situations), + doctors w/ similar patients

##### Use Slowly Changing Dimensions

* **Track time variance with SCDs, an old and venerable best practice from the DW world (Ch. 5)**
* This is just as important in the big data world as it is in the conventional DW 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 = putting off declaring data structures at the time of loading into Hadoop or a data grid**, whichbrings 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 fi t into a structure
* If you can load data into Hadoop, for instance, without declaring its structure, you can avoid a resource intensive step
* 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 w/ 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 + 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 tech around name-value pair data sources**
* Big data sources are filled w/ surprises 🡪 In many cases, you open the fire hose + discover unexpected or undocumented data content, which you must nevertheless load at GB/sec
* **The escape from this problem is to load this data as simple name-value pairs**
* Ex: If an applicant were to disclose her financial assets, 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 = 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 w/ the cost of ETL to build physical tables before run time.**
* **Data virtualization = a powerful way to prototype data structures + make rapid alterations or provide distinct alternatives**
* **Best data virtualization strategy = to expect to materialize the virtual schemas when they have been tested + 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

* 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, MDM, + the business glossary that exposes definitions + context to the business community**

##### Dimensional-ize 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**
* May receive data arriving at GB/min., often as name-value pairs w/ unexpected content
* The **best chance at classifying data in ways that’re important to your data governance responsibilities is to dimensional-ize it as fully as possible at the earliest stage in the data pipeline**
* **Parse it, match it, + apply identity resolution on-the-fly**
* Made this same point when arguing for the benefits of data integration, but here we advocate against even *using* the data before this dimensional-izing 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 + business risk
* Egregious episodes of compromising the privacy of individuals or groups can damage your reputation, diminish marketplace trust, expose you to civil lawsuits, + get you in trouble w/ the law
* At the least, **for most forms of analysis, personal details must be masked, + data aggregated enough to not allow identification of individuals**
* **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