PERSISTENT

Best Practices in Data Management for Analytics Projects

Mataprasad Agrawal, Sr. Architect
Shirish Joshi, Sr. Architect
Fernando Velez, Chief Data Technologist

CTO Office, Persistent Systems Ltd.

SECOND QUARTER 2016

Contents

1. Introduction and Scope of this document

Under the joint sponsorship of Sid Chatterjee, Head Corporate CTO organization and Sameer Dixit, Head Analytics practice in the Services Unit Persistent Systems Ltd (PSL), Mataprasad Agrawal and Shirish Joshi, Senior analytics architects and Fernando Velez, Chief Data Technologist in the Corporate CTO organization, are striving to identify and document best data management practices for projects within the Analytics practice. These projects provide analytical reporting and dashboarding based on structured and/or ad hoc queries, as well as predictive analysis, across various technologies within analytics space.

This effort addresses a gap, as there is currently no definite way or process or best practices that are documented in this space. The intended audience for this document is IT professionals that need to know about the details of building and managing a data platform for analytics applications: this includes data architects, designers, developers, database administrators / dev-ops teams and managers. It will be of most use to a professional who has already had some exposure to data management, data warehousing and business intelligence. We do provide, as a minimum, a glossary in chapter 9 to explain the intended meanings concepts and acronyms, notably those in italics.

By data management we mean data acquisition, data integration and data governance activities to build a dedicated environment consolidating data from a growing number of sources for Business Intelligence, i.e., for analytics projects. The last part of the previous sentence is important, as there are other data management endeavors with a different final goal, such as creating master data for an organization. We are leaving data management for *Master Data Management (MDM)* out of the initial scope, but will point out some of the commonalities with analytics data management along the way.

The presence of a dedicated, target environment assumes that the data from multiple heterogeneous sources are integrated in advance and are stored in this environment. There is an alternative architecture, called data federation, which is to leave the data in place and query it live from the target environment running only analytics tools. However, experience shows that this works only in limited environments; in its general setting, poor query performance may be a showstopper and, it any case, it does not provide important capabilities such as maintaining data history and improving data quality.

This dedicated environment may be characterized along several dimensions:

    a.   Type: It may be either a *data mart*, a *data warehouse* or a *data lake*.

    b.   Deployment: On premise or in the cloud,

    c.   Industry: It may serve any vertical industry.

This best practices document covers all cases from the dimensions above. Besides, it is mostly tool agnostic.

2. Document Plan

As it can be appreciated, the scope defined above is quite daunting; besides, there already exist publications addressing most of these topics in detail. Therefore, rather than embarking on a long journey to rewrite a yet another book on data integration and data governance, we decided to pursue the following strategy for this document:

1. Pick a widely-known reference publication,

2. Summarize the most relevant points of the publication on selected topics, and

3. Enhance it on areas not covered by the publication and including our own experience.

www.persistent.com

2.1 Reference publication: Kimball's Data Warehousing Lifecycle

The reference publication we selected is "The Data Warehouse Lifecycle Toolkit", 2nd edition, by Ralph Kimball, Margy Ross, Warren Thornthwaite, Joy Mundy and Bob Becker [1]. This second edition is from 2008 and significantly updates and reorganizes the first edition, published 9 years before. It is a very practical field guide for designers, managers and owners of a data warehousing / business intelligence (DW/BI) system. It is well-known to PSL architects from the Analytics practice.

The book describes a coherent framework, the "Kimball Lifecycle", covering all the way from the original scoping and planning of an overall enterprise DW/BI system, through the detailed steps of developing and deploying, to the final steps of planning the next phases. The Lifecycle diagram below depicts the sequence of high level tasks required for effective DW/BI requirements gathering, design, development, and deployment.
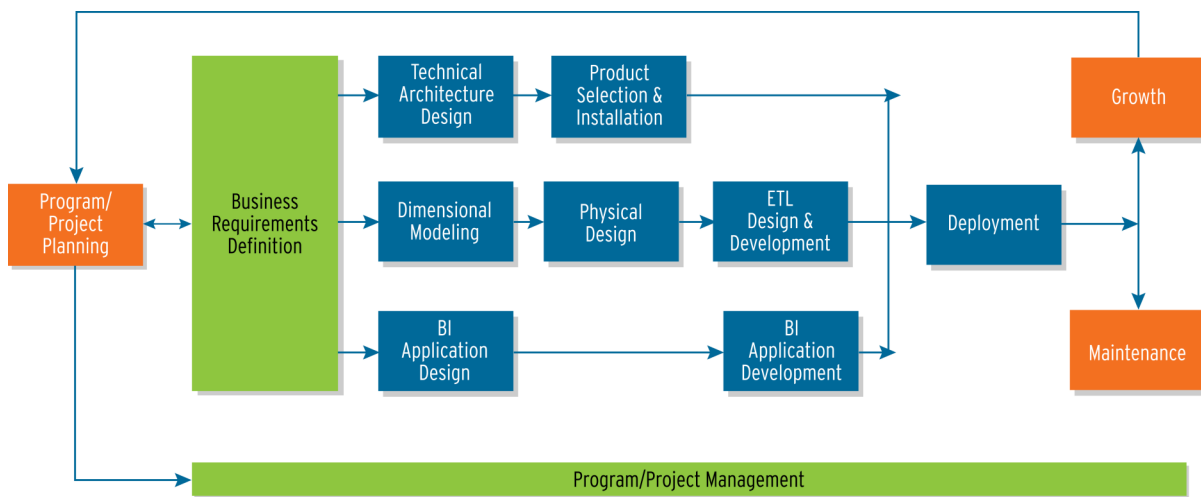
Figure 1: The Kimball Lifecycle diagram

After a planning and requirements gathering phase, the design and development activities are carried on three parallel tracks: a technology track, a data track and a BI track. The data track shows the journey of data from the sources to the target environment: It is extracted, transformed and loaded from the sources into the target environment to make it comply with a data model exposed to the BI applications. For reasons explained in chapter 5 below, the preferred model for analysis follows a dimensional model design, and it is also managed at the physical level for optimal performance and other non-functional requirements. These tracks converge at deployment, followed by maintenance activities and planning for growth.

It is important to note that the notion of data marts is also covered on this publication: it is now referred to as "business process dimensional model" (see Glossary). The reason for this change is because the authors consider that the term "data mart", in their own words, "has been marginalized by others to mean summarized departmental, independent non-architected datasets".

www.persistent.com

2.2 Enhancements to Reference Publication

Now, we turn onto the third point of our strategy, namely, how to enhance this reference publication. We believe we can add value to this reference publication on two important dimensions:

- Areas of interest to PSL developed recently since the book was published, and

- Our own experience as practitioners.

On the first point, during the last 8 years since this reference publication appeared, several tendencies have materialized and developed enormously, which we want to cover explicitly:

a. **Cloud deployments.** We mentioned this in the previous section (deployment type dimension). Until three years ago, data warehousing was essentially an on-premises initiative. Since then, cloud computing has had a tremendous impact on data warehousing and analytics, which we want to cover in this document.

b. **Big data.** We mentioned **data lakes** as a new type of target environment associated with big data engines / solutions. We will cover explicitly this topic in the big data section given its recent development (it is not covered in the reference publication).

c. **Self-service tools and agility.** This new set of tools and processes, democratizing access to data for users of varying technical skills, will allow more agility in the delivery of analytics projects, and involve business users more effectively to support data governance.

These are the three broad topics this paper is organized around on each area. We will describe the relevance of these topics for our scope in section 3 below in greater detail.

Finally, as mentioned in the second point of our strategy, we want to concentrate on a handful of important transversal topics (at the expense of others). After analyzing the current, internal knowledge of the technical scope at PSL, we have chosen the following four topics of data management:

1. Technical architecture

2. Dimensional data modeling

3. Data quality

4. Nonfunctional aspects to successfully run in production, such as performance, scalability and security.

The technical architecture addresses the technology track. The next two ones are the most important aspects of the data track in Kimball's lifecycle. And the non-functional aspects are necessary for a successful deployment, maintenance and growth, but they need to be considered from the start of any analytics project.

Concretely, we want to develop our best practices document as follows: each of these four transversal areas will be expanded into chapters 4 through section 7, with the following content:

1. A brief presentation of the topic, followed by a summary of its treatment through the requirements gathering, design, development, and deployment phases in the reference publication, including pointers to the chapters and pages where the topic is developed in detail. This will serve as a high-level view of the reference document from the vantage point of the topic in question.

2. Our value-add on each topic, both through

a. Our comments on the topic in any of the phases of the lifecycle of an analytics project, which are either not covered in the reference publication or which we want to specifically highlight, given our own experience as practitioners on the topic.

b. Our comments on the impact of the 3 recent tendencies not covered on the Kimball book, namely cloud deployments, big data and self-service tools and agility on the topic.

Driven by the pragmatic need of releasing a first version of this document sooner, we will initially emphasize the construction of the target environment (data warehouse/data mart), at the expense of its use through BI applications[1]. This means we will initially summarize each topic and provide our added value under the angle of building the target environment, ignoring the BI track.

3. Relevance of the new tendencies for our scope

Data management for analytics is undergoing radical change. The system landscape, the architectural patterns to perform data integration and improve data quality, as well as the underlying software technology to develop data integration and improve data quality are all changing. This cannot only be attributed to the pervasiveness of data-driven analysis: We believe in fact that this phenomenon is to be attributed to the convergence of three of the hottest topics in today's information technology industry:

(i)   The usage of cloud computing for analytics projects,

(ii) The explosion of available data for analysis, and

(iii) The recent rebirth of Artificial Intelligence, in particular machine learning.

We will dedicate a section below to each of the two first topics. As for machine learning, we also dedicate a specific subsection on this subject on the big data quality section (see 5.3.3.3 below), even though the subject is not strictly specific to big data nor to data quality domains, but because this technology is being increasingly applied in the intersection of the two domains. We will also illustrate that it also is contributing to make data integration more accessible to less technical users, so it has a definite impact on our third new tendency, namely self-service tools and agility.

We hope that after reading chapter 3 it will become immediately clear that the system landscape and the available underlying technology is dramatically changing in projects that must prepare and manage data for analytics. As for the architectural patterns for integrating data, section 4.4.1 below discusses Data Integration processing styles, ETL and ELT, which is very relevant to the presentation about cloud computing (section 3.1) and big data for analytics (section 3.2).

3.1 Cloud Computing for Analytics

Since cloud computing emerged, its effects on IT infrastructure, network services and applications have been enormous. Data warehousing and analytics is no exception and is even a key use case, as the cloud provides generous processor and storage resources to assure processing speed and volume scalability.

At PSL we are witnessing this change in most of the new analytics projects being launched with new customers. Most customers that engage with PSL do so because of we are seen as innovators which can help with cutting the time to market in their solutions. In that respect, our current experience in cloud deployments is the key, and best practices needs to be shared as widely as possible.

[1]By these we mean query and reporting tools, data mining tools, dashboards/scorecards, and analytic applications embedding query and reporting technology, as well as domain expertise on a particular business process.

www.persistent.com

We describe the impact of the cloud for analytics projects in two levels: first, globally, in this chapter; and second, in our transversal topics, in the dimensional modeling, data quality and non-functional aspects chapters. At a global level, we will look at the following market categories:

1. Cloud data warehouses

2. Cloud integration tools

3. Cloud / on premise application integration

3.1.1 Cloud data warehouses

Until three years ago, data warehousing was essentially an on-premises initiative, with data migration and security issues playing a large role in keeping warehoused storage of corporate information within the walls of organizations. The first vendor to challenge this status quo was Amazon, who launched in 2013 its Redshift data warehouse service. The main value proposition was low cost (about a tenth of the cost of traditional data warehouse solutions when it first came out), which has often been one of the most painful aspects of data warehousing. This is becoming possible, not only thanks to the pay-per-use model, but also because database vendors are increasingly switching from expensive proprietary hardware to low-cost commodity servers and storage technology[2].

The data warehouse as a service options now available in the market include IBM's dashDB and Microsoft's Azure SQL Data Warehouse. In addition, Oracle, Teradata and SAP offer cloud-based versions of their data warehouse platforms. New startups such as Snowflake, a columnar cloud DB built on top of AWS S3 offering SQL access to semi-structured data, were launched in 2015. Google has also introduced their own offer: BigQuery, the public implementation of Dremel, a (really) massively parallel, scalable query service for datasets of nested data, which is now positioned as an analytics data warehouse.

The most common use cases for cloud databases are (i) the traditional enterprise data warehouse, (ii) the big data/data lake use case which we describe in detail in section 3.2.2, and (iii) software as a service companies embedding analytic functionality within their cloud applications.

Most vendors offer both private and public cloud versions of their software. On public clouds, cloud warehouse vendors also have PaaS offers and IaaS offers. The former are fully managed data warehouses, while the latter allow customers to take advantage of cloud based hardware and storage, with clients retaining administrative rights on the DBMS warehouse running on this hardware[3], allowing customers to implement, for instance, hybrid cloud warehouses –more on this below.

The advertised benefits of public cloud data warehouses are:

- Faster time to market, especially because of the specialized personnel skills available at the provider to get a cloud data warehouse up and running, and to operate it.

- All the lower TCO benefits of generic cloud computing (e.g., pay per usage, multiple tenants).

- Periodic warehouses (load-process-discard) with associated cheaper payment by the hour, for both report generation and for testing environments for every target platform.

---

[2] For instance, Redshift is powered by a massively parallel processing (MPP) database system: ParAccel (which is now part of Actian), running on commodity servers, so performance is not being sacrificed just for a low-cost point.
[3] A good example is Microsoft: Azure SQL Database is a fully managed PaaS offer, and SQL Server in Azure VM is an IaaS offer.

- Some benefits of IaaS public clouds and hybrid clouds include:

    - Less expensive storage of backups

    - Customers can tune their particular requirements, in particular across SLAs such as high availability, disaster recovery, or providing on-demand, externally-provisioned scalability when computing demands spike.

The following benefits are common to private and public data warehouses:

- Elastic sandboxes and data marts for self-service workloads, for business analysts and data scientists.

- Faster setup of Proof-of-Concept (PoC) environments for sales and presales teams.

- Cheaper developer environments for new applications and for porting legacy applications.

The reasons to prefer private data warehouses include:

- Security: as private clouds are dedicated to a single organization, the hardware, storage and network can be designed to meet high security levels.

- Compliance, for instance, HIPAA in healthcare in the US and PCI in the global payment card industry, is much easier to achieve, for the same reason.

- Performance and scalability SLAs are easier to guarantee, as the hardware is no longer shared among several tenants.

    - In the case of external private cloud providers, if there is enough scale in terms of storage and hardware equipment, true elastic scaling can generally be provided (e.g., through MPP databases), which is not always the case in data centers internal to the company.

    - On the other hand, you have more control in internal private clouds for the hardware you want the data center to run. In particular, the customer is free to choose a particular DW appliance or scale-up server that is known to suit their needs, which external providers may not give you, as their default is to run on commodity hardware[4].

We have recently witnessed hybrid cloud environments gaining a lot of traction in the market. This may translate into demand of hybrid cloud data warehouses. Indeed, customers may have good reasons to prefer private clouds for specific areas of their warehouse (e.g., sensitive data areas). However, if other areas don't need, e.g., security or compliance as much, and could benefit from higher capacity at lower cost from a public cloud provider, hybrid architecture for this data warehouse is a possibility: data can be sliced such that only sensitive data stays private. Another driver may be SLAs such as high availability, disaster recovery and, well, performance[5].

3.1.2 Cloud data integration

In cloud warehousing deployments, the fundamental data management processes – data integration, data quality , data governance, master data management – still need to be applied to information that is warehoused in the cloud.

---

[4]This is not always true: some external providers may let you specify and customize the hardware.
[5]It was initially hard to negotiate SLAs with public cloud providers but, overall, they are getting better at this: for instance, some vendors now offer high availability. Make sure you understand this area if you are working with a private cloud provider as well.

www.persistent.com

Data integration and data quality vendors (Informatica, Talend, IBM, Pentaho), emerging players (SnapLogic, Iron ETL) and cloud providers (AWS, with Data pipeline and Kinesis, Microsoft with Azure Data Factory and Google with Cloud Data Flow and Cloud Data Proc) now offer cloud-based ETL tools. The public cloud services additionally support dynamic tenant provisioning, elastic scaling, and a web administration and monitoring GUI, as well as pay-per-use billing. Again, cost and ease of use are big factors driving adoption. License-based (on premise or private cloud) data integration tools can also connect on premise and cloud applications; they are costlier and they require additional hardware, but pose fewer risks to customers in use cases where security and /or compliance to regulations are needed. In both type of deployments, these tools generally provide services that enables the development, execution, and governance of bidirectional integration flows between on premise and cloud-based systems or cloud-to-cloud processes and services.

However, the big challenge for data in the cloud revolves around moving and integrating data in the cloud. Per our experience, moving on-premises data to the cloud is problematic on the following areas:

- **Large number of data sources.**

    Connectivity to a large number of sources is needed. Even if the connectivity part is the most mature in established vendor products (as on premises versions included web-service-based sources connectivity for a while now), the explosion of data sources is such that no list of prepackaged connectors will work in most cases. Because of this, providers offer development kits that enable customers to create custom adapters for non-supported systems.

- **Large data volumes.**

    Moving data on the internet is sometimes a stumbling block, for several reasons.

    - Performance. The public internet is slow. A possible approach is to use private network connection solutions, which allows organizations to directly connect to cloud providers rather than through connections established on the public Internet. See section 7.2.2 for more details.

    - Shared resources in a public cloud. Heavy-duty data loads or ETL operations will disturb tenants. This should be done during the nights or off-peak period.

Leading vendors like Amazon have taken a big leap in this space and are providing Batch Cloud Data Transfer services like Amazon Snowball Appliance, Bulk Import to S3 storage.

- **Legacy transforms.**

    Reusing existing transformation (the T of ETL) code to be applied on data being moved to the cloud is an issue, when the ETL tool that was used on premises is not the same as the tool retained for the cloud. In our experience, vendor's tools frequently change when changing system landscape, and there is no interoperability standard; and even within the same vendor, the on premises tool and the cloud tool are not always fully interoperable. Indeed, cloud tools do not always offer the same transformations as on premise tools; besides, some of the bells and whistles of traditional on-premises tools in terms of richness of transformations may have not made it to the cloud tool.

- **Firewalls.**

    Security issues arise when cloud providers integrate with on premise systems requiring IT administrators to open an external communications port. This creates a huge security hole, which is why some cloud integration providers have devised ways to tunnel through corporate firewalls without requiring administrators to open external ports.

www.persistent.com

For companies that are just starting to use cloud computing, most of the usage will be for new applications, with stand-alone being the low-hanging fruit. Those that have already critical cloud apps in use (e.g., salesforce.com), have already solved the cultural and some technical problems in transitioning to cloud computing, so data management in cloud databases will make more sense for them.

Much of the cloud integration challenge centers on connecting cloud and on-premises applications (not just moving data from an on premise database to a cloud data warehouse), which is developed in our next section.

3.1.3 Cloud Application Integration

In the early days, the first cloud application integration tools focused largely on integrating Salesforce.com, the earliest successful cloud application, with packaged ERs providers (SAP, Oracle) and other on premise applications, usually by moving files in batch from the cloud to on premise systems.

Today's organizations have adopted a variety of cloud and on premise applications to manage their business. IT departments are now expecting that hybrid system landscapes will become the standard way to manage the organization's IT assets in the future[6]. In fact, cloud applications will probably be favored over on premise applications when there is a choice, but on premise applications will not go away anytime soon[7]. On these hybrid landscapes, process and data integration across the cloud / on premise boundaries is a must to prevent *data silos*[8] and allow seamless end-to-end business processes.

To illustrate this trend with a specific example, let's use the CRM (Customer Relationship Management) paradigm. When they were first introduced, the basic thinking was that the company owns the relationship with the customer. An efficient CRM was one that would allow to easily create new fields and do better customer segmentation. Data quality management was focused on rules and control for create-read-update-delete (CRUD) operations. CRM is now changing towards a dynamic, highly personalized and multi-channel collection of interactions where customers enter in different kinds of engagements with a company. In this new setting, customers tend to be more self-enabled and interact with the company through different applications: first maybe a marketing app, then an onboarding app, then one or more services apps; some are deployed in the cloud, some on premise. It is no longer a CRM package that owns the relationship. It now becomes necessary to seamlessly manage these interactions. In this setting, best-in-breed customer MDM systems are not those that just govern customer master data through CRUD operations but those that are focused on automated completion of customer master data along the paths a customer takes in his journey with the company.

The following requirements on data integration and governance on these hybrid landscapes were drafted in large enterprise projects we have participated in the past, and they include:

- Real-Time Business Process Integration

- Agility, in particular in integrating new sources/targets

- Responsiveness to new functionality and technical changes

- Batch and real time data movements on both individual records and collections of records

---

[6] According to Gartner, 70% of enterprises will pursue a hybrid cloud hosting model by 2015 [6].
[7] A recent TechTarget survey [17] shows that more than three-quarters of organizations (77%) have either all or most applications running on-premises.
[8] See the Glossary section 9 for a definition of this term.

- Strong Data Governance

  - Trusted data through high data quality, data enrichment, integrity and interoperability

  - Full transparency on data movements through tracking and auditing

  - Data security and compliance

- End to end monitoring and support

Our experience is that the best architecture concept to handle this list of requirements is a Data Integration platform, preferably built with technology from cloud data integration vendors[9], that avoids point-to-point interfaces between applications. So rather than connecting applications A and B directly, they talk through this middleware platform, which could run on premise or on a private cloud. In theory, this platform could also run in a public cloud; however, if data exchanged through these platforms must comply to regulations, or if customers chose to store data in transit (some good reasons for doing that are mentioned in section 6.3.2.1 below), public cloud integration providers may be seen as a risky bet for these customers.

3.2 Big Data

The Big Data era is the direct consequence of our ability to generate and collect digital data at an unprecedented scale and, at the same time, our desire to analyze and extract value from this data in making data-driven decisions, thereby altering all aspects of society. Abundant data from the internet, from sensors and smartphones, and from corporate databases can be analyzed to detect unknown patterns and insights, for smarter, data-driven decision-making in every field. As the value of data increases dramatically when it can be linked and integrated with other data to create a unified representation, integrating big data becomes critical to realizing the promise of Big Data.

A word of caution is in order, though, as the term "Big Data Integration" may mean at least two different things. It was initially coined by the 2013 publication [10], and meant, for each domain such medical records, astronomy, or genomics, literally mining the web and attempting to integrate its thousands of sources per domain. Here we are using this term in our data warehousing setting, which is about (i) building a large, central data repository for multi-structured data from various data sources, including data of the traditional sort such as those generated by applications within an organization, but also of the "big data sort", i.e., internal web logs or document files, and external data generated by web sites such as social networks, or by sensor machines; and (ii) providing search, querying and analytics capabilities on top of this repository, for application use cases such as customer churn analysis, fraud detection (for instance, in financials), cohort group discovery (for instance, in healthcare), and the like.

This being said, some of the volume, velocity and variety challenges described in [10] still remain in our warehousing setting, albeit to a lesser degree:

- **Variety Challenge.** Data sources, even within the same domain, remain extremely heterogeneous both at the schema level regarding how they structure their data and at the instance level regarding how they describe the same real world entity, exhibiting considerable variety even for substantially similar entities. This impacts the entire integration effort. Variety at the local schema level means that it is difficult to come up with a target schema that aligns all the sources, which is the very first step of an integration project. As for detecting and resolving duplicates, techniques applicable for structured data have to evolve to cope with unstructured or semi-structured data such as tweets or social posts, as shown in section 6.3.3.1.

www.persistent.com

- **Volume Challenge.** It is expensive to warehouse all the data sources an organization wants to integrate, especially not knowing in advance the queries to be asked of the data. And for the areas where queries are known, assuming they can be aligned in a consistent schema, query execution may exhibit great latency on database sizes of hundreds of terabytes or petabytes if the internal architecture is not distributed and/or massively parallel. Existing techniques for data quality challenges such as duplicate records are also being challenged because of performance reasons on these volumes.

- **Velocity challenge.** Many sources systems want to integrate provide rapidly changing data, for instance sensor data or stock prices. Depending on the rate of change, it may be unfeasible to capture rapid data changes in a timely fashion, in particular if analysts and BI applications need low query latencies on the incoming data. On the other hand, data quality monitoring in such environments need new approaches than are just starting to get explored.

Conventional RDBMSs and SQL simply cannot store or analyze this wide range of use cases and cope with the volume, velocity and variety challenges posed by big data. In response to these challenges, two types of systems have emerged: NoSQL and Hadoop[10].

Hadoop is one of the most interesting and promising new technologies related to big data. Historically, the first use case for Hadoop was the storage and processing of semi-structured web log files stored in plain text. Since then, many more types of datasets relevant for analysis are being stored into Hadoop clusters. Although big data has passed the top of the Gartner's Hype Cycle, and based on the amount of attention that big data technology is receiving, one might think that adoption and deployment is already pervasive. The truth is that this technology is still in a relatively early stage of development. For instance, only recently did Apache Spark or Apache Flink, the new distributed engine of the Hadoop infrastructure, adopt declarative technology that makes complex querying both easy and fast. On the other hand, preparing big data for effective analysis beyond descriptive BI and into predictive BI (insights about what will happen) and prescriptive BI (what actions should be taken given these insights) requires highly developed skills of a special group of people – data scientists. These skills go beyond traditional data management and include advanced statistics and machine learning. Most companies are still trying to find the talent and the products to effectively manage big data in order to get tangible business benefits.

What about NoSQL systems? These are systems that also store and manipulate big data, but are not meant for complex querying or OLAP style aggregation but rather for operational systems, or for search-intensive applications. NoSQL databases are unstructured in nature, trading off stringent consistency requirements and modeled for querying patterns for speed and agility. As Hadoop, they store their data across multiple processing nodes, and often across multiple servers, but interactions are through very simple queries or APIs and updates involving one or a few records accessed by their key[11].

We have witnessed two main uses of Hadoop in integration projects: (i) as the area where raw data reflecting both entities and events arrive from operational processes, and where they are transformed for integration and sent down the chain to the target repository, i.e., where analytics workloads are run, and (ii) as the target repository itself. As we will see below, the Hadoop repository may play these two roles simultaneously, but then it will need other components to make this viable. We analyze these two use cases below.

---

[10] In all fairness, relational database vendors also tried offering an alternative, which is to extend their type systems through user defined data types (UDFs) that process unstructured and semi-structured data within the DBMS inner loop. However, these systems have not had much traction, mainly due to the volume challenge, which needs to be tamed through massive distribution and parallel computation.
[11] HBase, a NoSQL system from the Apache Foundation, is actually layered on top of Hadoop HDFS.

3.2.1 Hadoop for ETL workloads

When the transformations required by the analytics project require the execution of complex SQL queries and procedures, and/or when, for performance reasons, execution is required in the target repository which is not Hadoop but, e.g., an MPP relational database, then clearly Hadoop is an unlikely candidate: to start with, ANSI standard SQL and other relational technologies are not fully supported on Hadoop today. This situation may change in the future, given Hadoop's recent improvements on this respect (e.g., the introduction of declarative technology to Spark 2.0, see other query engines below), and if 3rd party tools make it easy to develop native Hadoop ETL workloads and provide high level data management abstractions. For instance, functionality such as Change Data Capture (CDC), merging UPDATEs and capturing history are much easier to achieve in relational databases.

At the other end of the spectrum, some ETL jobs may just need basic relational capabilities or even no relational capabilities at all, e.g., computation expressed in MapReduce (MR). ETL jobs that make simple aggregations and derived computations at a massive scale are well-suited to the Hadoop framework, and they can be developed for a fraction of the cost of a high-end ETL tool. Indeed, those were Hadoop's origins: MR was designed to optimize simple computations such as click sums and page views on massive data.

Today, numerous extensions to the Hadoop framework exist both for moving data and for querying and managing data.

- For moving data, the new tools below include visual interfaces to create, manage, monitor and troubleshoot data pipelines, a welcome change from the initial set of tools around Hadoop which only offered programmatic interfaces and were hard to use and debug.

  - Apache NiFi [32] is a platform for automating the movement of data between disparate systems. It supports directed graphs for data routing, transformation, and system mediation logic via a web interface. The workflow can be monitored, and provides data provenance features to track data from source to end.

  - Apache Storm [33] adds fast and reliable real-time data processing capabilities to Hadoop. It provides Complex Event Processing capabilities, processing and joining data from multiple streams, and rolling window aggregation functionality.

  - Kafka [34], also from the Apache foundation, is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system. Kafka is often used in place of traditional message brokers like JMS and AMQP because of its higher throughput, reliability and replication.

  - Airflow [35], from Airbnb, is an open source platform to programmatically author, schedule and monitor data pipelines as directed acyclic graphs (DAGs) of tasks. The rich GUI makes it easy to visualize and troubleshoot production pipelines, monitor progress.

  - For querying and managing data: Spark, Impala, Hive/Tez, and Presto. All of these engines have dramatically improved in one year. Both Impala and Presto continue to lead in BI-type queries and Spark leads performance-wise in large analytics queries. Hive/Tez with the new LLAP (Live Long and Process) feature has made impressive gains across the board and is close to the other engines now.

Finally, usage of high-end ETL tools or specialized ELT add-ins for Hadoop remain an option, especially if there are already present in the organization because, as mentioned above, these systems are now able to push down processing of high level flows into Hadoop, providing again development productivity and administrative control.

www.persistent.com

3.2.2 Hadoop for Analytics, or Data Lakes

Data lakes [7] is an emerging approach to extracting and placing big data for analytics, along with corporate data, in a Hadoop cluster which several components are layered in order to effectively enable data scientists and business analysts to extract analytical value out of the data.

In the original concept expressed in [7], one of the primary motivations of a data lake is not to lose any data that may be relevant for analysis, now or at some point in the future; the idea is to provide as large a pool of data as possible. Data lakes thus store all the data deemed relevant for analysis, and ingests in raw form, without conforming to any overarching data model. So, for instance, when a customer buys a product from the organization's e-commerce web front-end, not only we want to get the data of the customer's transaction from the underlying database, but we also want to include all the logged events that trace the customer's experience during the purchase on the web site. We might also be interested in gathering data from social sites to understand the experience that this customer might have with the product in the future[12].

Reference [5] is PSL's official "Data Lake story", explaining our point of view on this concept. What follows summarizes some sections in [5]. To be effective to analytics processing, the Hadoop environment uses a variety of processing tools to semantically **discover** and to **govern** the data, which include traceability through lineage metadata and tools to improve its overall quality. This allows finding the data needed to solve a business process on huge amounts of data, providing a level of trust to this data and making it effectively consumable. Finally, it allows tools and exposes APIs for consumers to extract business value through several types of consumer workloads, including traditional BI queries, exploration and machine learning / statistical workloads. The figure below is PSL's data lake reference architecture from [5].

Data Lake - Reference Architecture
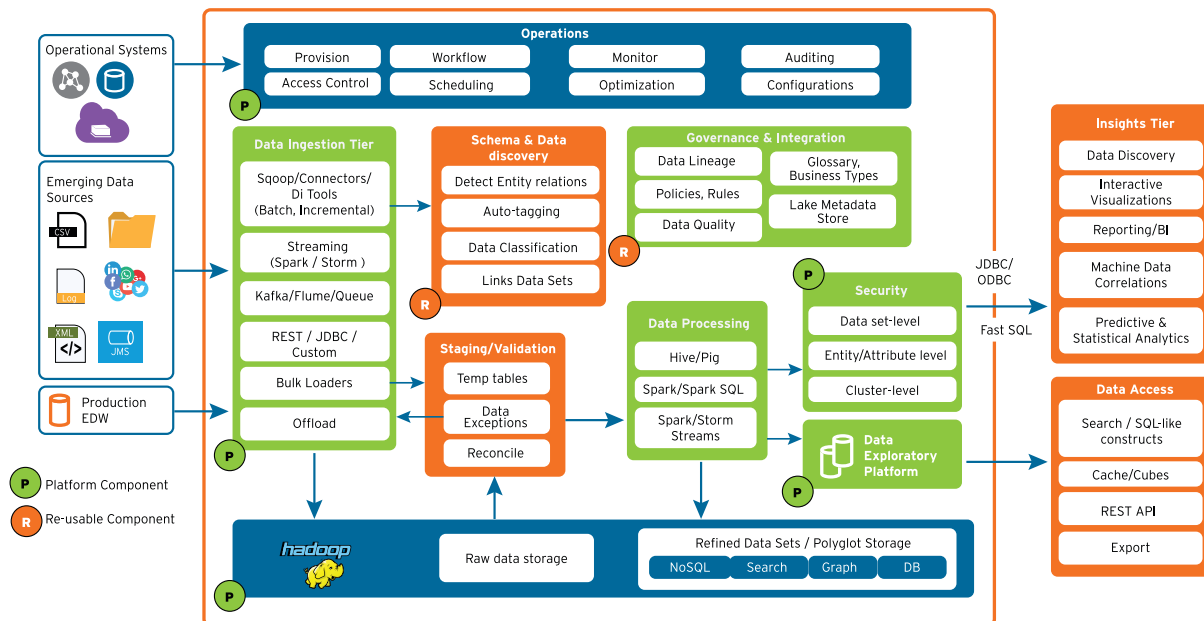
[12] Provided we are able to identify him/her as the same customer that bought the product, which may or may not be possible.

Data Lakes and Data warehouses

Data lakes are therefore far more flexible alternatives to data warehouses to gather all the data of an organization that is relevant for analysis, especially as they become more outward looking and increase their exposure to cloud and mobile applications. Indeed, data warehouses require upfront modeling, and only good quality data conforming to the model should be loaded. However, as we will see below, this does not mean that data is not modeled in the lake for further analysis, or that its quality can be disregarded: it just means that these two activities can be *deferred.*

As we already know, data warehouses are optimized for a different purpose: answers to the questions for which they have been designed can immediately be trusted and, because of more mature (but more expensive) engines, it can support fast response times for large numbers of concurrent users.

Data warehouses and data lakes should then be used for the purpose they were designed and, as such, they can co-exist in an organization's landscape: data from the warehouse can be fed into the lake; and conversely, data from the lake can be a source for the warehouse, or a data mart (a portion of a data warehouse specializing on a business process).

Roles in a Data Lake

A Data Lake owner (called custodians in [5]) is a role fulfilled by IT-savvy people, who can belong to the IT department of an organization or not, and who try to satisfy the needs of personas of two different roles:

- The data producers, whose role is that of the business head who owns the data specific to the business function. They are all about control: they worry about regulatory compliance issues, visibility about who uses their data, and privacy and access control, among others.

- The data consumers, who may be either data scientists or business analysts. They are the ones who discover, explore, visualize and ultimately get business value in the form of insights to executives. They are about flexibility: they want to quickly find relevant data for their use cases, consuming good quality data, supporting several analytics workloads, etc.

Data custodians can use several tools on the data lake to assure data producers that data is in safe hands and is being governed (controlled for access, quality and lineage, even if these controls are at consumption time) and monitored, while serving data consumers with all flexibility, so that they can discover, explore and visualize data sets to derive valuable insights.

3.2.3 Hadoop as a service

At the intersection of the cloud and big data there is a nascent but fast-growing market: Hadoop as-a-service, or HaaS. The biggest drivers are reducing the need for technical expertise and low upfront costs.

Amazon Elastic MapReduce (EMR), the HaaS service by AWS is the largest player. It provides a Hadoop based platform for data analysis with S3 as the storage system and EC2 as the compute system. Microsoft Azure HDInsight, Cloudera CDH3, IBM BigInsights for Apache Hadoop for Bluemix, EMC Pivotal HD are the primary competing HaaS services provided by global IT giants. Most of these competitors initially provided a Run It Yourself deployment option and are now proposing managed options. Altiscale, recently bought by SAP, is a pure play provider, meaning it provides complete running and management of the Hadoop jobs. Another pure play provider is Qubole.

www.persistent.com

3.3 Self-service tools and agility

Data integration continues to primarily be an IT-centric activity based on the data, database and technology know-how needed. Typically, data integration platforms are purchased, managed and used by IT groups responsible for BI, data warehousing, master data management and other data management programs.

On the other hand, business unit and department heads want faster, better, cheaper information processing. Analysts need data to solve an increasing number of business questions, and frequently the data they have at hand (LOB excel files, or corporate reports and dashboards) are not useful. They have to rely on IT data architects to either

(i)   Ingest and integrate new, disparate datasets into the warehouse,

(ii) Curate new datasets from existing warehouse data; in other terms, transform existing tables or views virtually (through views) or in a materialized way (through cleansing/integration transforms) into new layouts as requested by the analyst, and/or

(iii) Build reports from existing datasets in the warehouse for the analysts.

The IT architects are already short on capacity to manage an increasingly complex information technology landscape for the enterprise, and can't keep up with these (generally unplanned) analyst requests. This generates a bottleneck.

Today, we live in the era of self-service, and this applies to information as well. Business units and departments are eager to process information and applications themselves, without or with limited IT assistance or intervention, to get through this high latency problem to get at the data they need.

The first category of self-service tools that appeared in the market was self-service, interactive data visualization and discovery tools. They provide BI capabilities suitable for workgroup or personal use, where the key criteria are ease of use, flexibility and control over how to model and build content without having to go to IT. This area became the fastest growing area of BI in recent years, and it is not uncommon to see them in use in a significant portion of PSL's customer engagements.

However, these categories of tools only address point (iii) in the laundry list of IT architect tasks above, while leaving the remaining two tasks: finding, preparing the data for analysis and curating a dataset that an analyst can use, which basically is still 80% of the work. Of course, this does not go too far. Business users and analysts are now demanding access to self-service capabilities beyond data discovery and interactive visualization of IT-curated datasets, to include access to sophisticated data integration tools to prepare data for analysis.

What is meant by "data preparation" in this setting is the ability, through a data-driven experience, for the user

- To profile and visually be able to highlight the structure, distribution, anomalies and repetitive patterns in data, thanks to data mining and machine learning algorithms,

- To standardize, cleanse and de-duplicate the data, again thanks in part to the same technology,

- To enrich it with data that is functionally dependent of the data at hand (for instance, to determine missing postal codes with the available address data),

- To combine it with other datasets,

www.persistent.com

- To define calculations, filters, projections and/or aggregations, in order to implement a logical model suitable for an analytic application, and

- To be able to do this on a cloud or an on premise deployment.

In order for a business user to do this effectively, these environments must support a variety of other services:

- Discovery of datasets, at least of data behind the firewall in on premise deployments, and in the underlying repository in cloud deployments –these systems will also handle (if they don't do that today) discovery of datasets on the cloud,

- Data sampling,

- Tagging and cataloging datasets as well as searching cataloged datasets through tags,

- Collaboration workspaces where users can share transformations, datasets and queries; and

- Last but not least, a means for IT administrators and data architects
  - To monitor and govern the overall environment and
  - To operationalize business-driven data preparation workflows in the corporate data environment. Typically, data preparation is run on small sets or on a sample; operationalization takes care of making this work on larger volumes. It also deals with applying the preparation incrementally on new data from the sources, modeling the end result, integrating it in the data warehouse and securing it at the corporate level.

A new brand of self-service data preparation tools is now gaining significant traction and mindshare in the market. Gartner recognized this trend about 18 months ago, as a new market segment [8] which disrupts the analytics market by empowering the business with an agile means to helping users find, assess, combine, and prepare their data for consumption. Tools from this market segment include Triacta, Paxata, INFA Rev, Talend Data Preparation, Datameer SAP Agile Data Preparation and Tamr.

4. Technical Architecture

4.1 Technical Architecture Overview

Architecture, in the data warehousing world, answers the question "how will we fulfill the requirements". It describes the plan for what you want the DW/BI system to look like when it can be launched in production, and the way to develop the plan. Kimball [1] describes BI/DW architecture in chapters 4, 5, and 6, and calls out 3 distinct pieces.

1. **Data Architecture**. At the core of the data architecture there is the firm belief that "dimensional modeling is the most viable technique for delivering data for business intelligence because it addresses the twin non-negotiable goals of business understandability and query performance" ([1], page 233). The data architecture is built with the following blocks:

   - Enterprise DW Bus Matrix, a conceptual tool in where the rows correspond to business processes and the columns correspond to natural groupings of standardized descriptive reference data, or *conformed dimensions* in Kimball parlance. The matrix delivers the overall big picture of the data architecture for the DW/BI system, while ensuring consistency and integrating the various enterprise dimensional models.

www.persistent.com

- **Business Process Dimensional Model.** Each of the organization's business processes can be represented by a dimensional model (see section 5.1 for a details). As they are conformed, dimensions deliver the same content, interpretation and presentation regardless of the business process involved, so they can be shared across the enterprise DW environment, joining to multiple fact tables representing various business processes.

- **Physical Data Model**. This is the database model implementation to support the logical model enabling scalability, high performance and easy maintenance.

Data Architecture is covered in chapter 5 of this document, for the dimensional modeling part, and chapter 6 for the data quality improvement (data cleansing) part.

2. **Application (DW/BI system) Architecture**. This is the processes and tools we apply to the data, a combination of custom code, home grown utilities, and off-the-shelf tools answering the question *"how"*—how do we get at this data at its source, have it conform to the data architecture and meet the business requirements, and move it to a place that's accessible. The balance of the architecture has shifted towards off-the shelf tools, as they have become more capable of handling a broader range of complex warehouse requirements. Thus, when planning for the application architecture it is crucial to select the right products to support it.

In this section 4.2 we provide a quick description of the natural separation between the internal workings of the data warehouse –the back room – and its public BI face –the front room.

3. **Infrastructure and Security**. Infrastructure provides the underlying foundation for all the architecture elements described in the previous two points. This includes the server hardware, the disk subsystems, the networks, the database systems and the desktops. Security is a complex area, intertwined with networking infrastructure, that the higher-level components take for granted; it must be well understood, as it goes far beyond recognizing legitimate data warehouse users and giving them specific rights to look at some, but not all, of the data.

In section 4.4.1 below we will comment on our experience with MPP databases and SMP databases; also, chapter 7 focuses on all security aspects of a DW/BI system.

Creating the technical architecture is the first step after the requirements are finalized, it may be done in conjunction with the process of finalization of the requirements.

The value of technical architecture lies way beyond just its documentation. It is the only systematic way to ensure that requirements are being tracked and met, and ensuring that communication between all stakeholders is being carried out at the sufficient level of detail. The building blocks and dependencies are the basis for the project plan. The architecture also documents the decisions and tradeoffs made to address facets of flexibility, productivity and maintenance.

4.2 DW/BI system architecture model

The Kimball DW/BI system architecture introduced in chapter 4 of [1] separates the data and processes comprising the DW/BI system into the backroom extract, transformation and load (ETL) environment and the front room presentation area, as illustrated in the following diagram

www.persistent.com

Back Room → ← **Front Room** →

ETL System and Services	Presentation Servers	BI Application and Services
• Extract • Clean • Conform • Deliver (dimensionalize) • Load	• Atomic level business process dimensional models • Conformed dimensions via Enterprise Bus Matrix • Aggregates for performance • Real time layer	• Direct access querying • Standard reports • Analytic applications • Dashboards & scorecards • Operational BI • Data mining and models • BI portal
ETL Management Services		BI Management Services
ETL Data Stores		BI Data Stores

Metadata

The Kimball DW/BI system architecture focuses on the following components:

- **Backroom area** or ETL (Extract -Transform – Load) area ([1], pages 119-133). This area performs four major operations:

    - **Extracting** the data from the sources,

    - Performing **cleansing and conforming** transformations,

    - **Delivering** dimensional models to the presentation server and

    - **Managing** the ETL process and back room environment (job scheduling and monitoring, recovery/restart capabilities, versioning, and pipelining/parallelizing).

    An ETL tool may be built homegrown or may be used off-the-shelf, with the key parameters being the productivity support, usability and metadata driven approach. The data stores in this layer deal with process history, staging tables, master tables, reference tables, audit tables and error tables.

- **Presentation server** ([1], pages 133-141), also called the target DW server, is where the data complying to a logical dimensional model and queryable by the front room applications is stored. It is organized by business process, is atomically grained, is tied together by the enterprise data warehouse bus architecture. In general, because of the large datasets involved in most organizations, a second layer is necessary for summary level queries to return in real time (or at least, a reasonable amount of time): the aggregation layer, where data is pre-aggregated during the load process as opposed to being aggregated on the fly by analytic queries. Implementation choices for the aggregation layer include the same (or another) relational database as the one storing the atomically grained dimensional models, or an OLAP engine.

    The management features are those that govern the life of the data warehouse, which include performance monitoring, securing and auditing access to the server, backing up and recovering content, physical database maintenance and automated system operation.

www.persistent.com

- **Front room BI applications** ([1], pages 141-155). The front room is the public face of the DW/BI system. There's a broad range of of-the-shelf BI applications including ad hoc query and reporting BI tools, dashboarding and scorecarding tools, and more powerful analytic or mining/modeling applications.

  The BI management services include a querying service (by far the most important), which is based on a metadata service allowing to define a BI semantic layer simplifying and enhancing the presentation layer structures for the business user's benefit. Additional services include access control and security, usage monitoring, delivery of production-style predefined reports regularly to broad audiences, portal services and web and desktop access. The data stores in this layer is comprised of stored reports, analytic application caches, local user databases and/or spreadsheets, documents and presentations, authentication and authorization data.

- **Metadata:** Metadata is all the information that defines and describes the structures, operations, and contents of these three DW/BI system components ([1], pages 170-173). It comes in three different types:

  - **Technical metadata** defines the objects and processes which comprise the DW/BI system. For instance, source descriptions and job transformations from the backroom layer, database system tables and table partition settings from the presentation layer, and BI semantic layer definition and query/report definitions from the front room layer.

  - **Business metadata** describes the DW contents in user terms, including what data is available, where did it come from, what does it mean, `and how does it relate to other data. For instance, business rule definitions from the backroom layer, calculated measure definitions in the presentation layer, and conformed attribute value lists from the front room layer.

  - **Process metadata** describes the warehouse's operational results. For instance, quality results from the backroom layer, aggregate usage statistics from the presentation server, and report and query usage and execution from the front room layer.

4.3 Best Practices

This section lists some best practices from the reference book [1], they appear in the sequel in **boldface**.

4.3.1 Technical Architecture at Planning Stage

A DW/BI technical architect should **form and lead an Architecture Task Force** that includes someone with back room experience, and front room experience, of ideally 2-4 persons.

During the gathering of business requirements, it is a good idea to **include in the interview plan questions addressing architectural topics such as user counts, response time expectations, and analytic functionality.** To detail out a few architecture issues, it may be required to conduct follow-up interviews.

The DW/BI architect must **understand the existing/proposed technical environment** to identify elements that may be leveraged, and any limitations or boundaries that the environment may impose.

www.persistent.com

The business requirements that have an architecture implication should lead the architect to **create a draft Architecture Implications Document** such as the one in figure 5-5 in [1], page 186. Look for business requirements that imply the need for some capability or service or that set boundaries and baselines. The first column should list the major requirements and opportunities, the second column should list the functional implications: what functions and tools are needed to deliver, what are the service level implications for performance, availability, or data latency, what are the scale implications in terms of simultaneous users or data volumes. The third column should list the functional areas, and the fourth column should list the business value or priority.

By arranging the functionalities needed to support business requirements by priority, we arrive at the architecture phases that correspond to the iterations of the lifecycle. For each phase, identifying the clear functional requirements and the deliverables will help determine timing expectations. Thus, from the architecture implications document, we then **arrive at the architecture implementation phases.**

For the sub-systems that are unique to your organization and not delivered by tool vendors, you need to **define and specify each sub-system in detail** to build it, get someone to build it, or evaluate products to satisfy the requirements.

All the previous steps now lead the team to c**reate the application architecture plan document**, that spans the form and function, components and sub-systems, business rules and definitions, priorities, resources and implementation timeframes.

4.3.2 Technical Architecture at Design Stage

Selecting the Hardware Platform, DBMS, ETL tool, BI Tool and metadata and modeling tool has not been usually done by PSL on behalf of the customer. However, this is starting to change, especially for the tools at the end of the list, so the recommendations about the selection process are very useful:

- Understand the Product Purchasing Process

- Develop the Product Evaluation Matrix with functional requirements, priorities and scales

- Perform a market research

- Select a short list

- Evaluate the candidates

    - In-depth research

    - Hands-on evaluation

    - Product comparison prototypes

    - Evaluation shortcuts (existing standards and resources, team experience, political factors)

- Recommend a product

- Trial

Section 7.2.2 below provides additional advice for selecting tools and database technology from a performance point of view.

A best practice mentioned is to **appoint a person to the role of a metadata manager** responsible for creating and implementing the metadata strategy. The metadata strategy will include

- Surveying the landscape for the locations, formats, and uses of metadata across the DW/BI System

- Working with the data steward to educate the DW/BI team about the importance of metadata and the metadata strategy

- Identifying and/or define metadata that needs to be captured and managed

- Deciding on the location and the version identification for each metadata element

- Creating systems to capture any business or process metadata that does not currently have a home

- Creating programs or tools, or purchasing a tool to share and synchronize metadata as needed

- Designing and implementing the delivery approach for getting business metadata to the user community, and manage the metadata and monitor usage and compliance.

We will review this metadata management issue further in section 4.4.1 below.

4.3.3 Technical Architecture at Development Stage

At the development stage, activities center on the dimensional model development (logical and physical) described in sections 5.3.3 and 5.3.4, the ETL development to populate the physical model described in 5.3.5, the data cleansing development, section 6.2.4 and the performance and security aspects described in sections 7.2.4 and 7.2.8.

Integration of the various metadata repositories from the off-the-shelf selected tools is addressed in section 4.4.1.6 below.

We are ignoring for the moment all aspects of BI development in this first version of the document.

4.3.4 Technical Architecture at Deployment Stage

Kimball says (chapter 13, page 544) that when a developer gives you a demo working on a developer environment with the complete scope for the first time, you should consider that the project is only 25% done. Once it passes all unit tests, you have completed 50%. The third 25% is when you have validated and debugged a simulated production environment, including data quality assurance, performance and operations testing. The last 25% is delivering the system into production, which includes usability testing, documentation and training.

This is to contrast with typical testing effort estimation as a percentage of development time, which is anywhere between 25 to 40% in traditional application programming.

www.persistent.com

The way the paragraph is written can mislead the reader into believing that 75% of the time is spent after development, and that a DW/BI project must be executed in the waterfall style depicted. Modern agile methodologies include test driven development which stresses writing unit tests early and running them often through automated processes; while this happens, QA developers can focus on developing test datasets and end-to-end system tests early enough as well. At each sprint a tested, if possible end-to-end reduced scope should be delivered.

But we believe nevertheless that Kimball has a point. Data management for analytics is all about complex data and systems integration, cleansing bad data, loading and managing large data volumes, all of which are hard to get right; and besides, the true measure of success in a DW/BI project is business user acceptance of the deliverables to improve their decision-making. All of this makes testing and validation challenges much harder than in a traditional application. In our experience, the effort to acquire, integrate and provide high quality data has always been more demanding than anticipated.

To tackle this challenge, the recommendation from a technical architecture point of view is to **seriously invest in testing automation**. This means

- Use testing tools

- Automate the test environment to run both unit tests and system test suites

- Log results, compare against known correct results, and publish reports on these results

- Invest in performance and load testing tools

- Use a bug tracking system and use BI reports and metrics to track deployment readiness as well as user reported bugs

Details on the necessary testing that needs to be carried out for alpha and beta testing of the DW/BI solution are summarized in section 5.3.6. And, as in the previous section, the deployment activities centered on dimensional model, data cleansing, and performance and security topics are to be found in these chapters corresponding deployment sections.

The remaining aspects not developed in this document are related to lifecycle processes: backing up, recovering, physical database maintenance and automated system operation. These aspects are covered in [1], chapter 13, pages 567 - 573.

4.4 Enhancements to the reference publication

4.4.1 Technical Architecture in our own experience

The following are the key decisions that must be made while crafting the ETL architecture and the presentation server for a dimensional data warehouse. These decisions have significant impacts on the upfront and ongoing cost and complexity of the ETL solution and, ultimately, on the success of the overall BI/DW solution. General advice and best practices involved in taking them are highlighted in **boldface**. Needless to say, in many aspects we coincide with Kimball, but we have strived to color our presentation with our own experience.

4.4.1.1 ETL: build versus buy

One of the earliest and most crucial decisions you must make is whether to hand code your ETL system or use an ETL tool. **In today's environment and state of available technology, most organizations should use a vendor-supplied ETL tool**. There are several reasons for this:

1. Most ETL tools provide reusable code in the form of ready to use transformations (data extractions, selections, projections, joins, aggregations, data cleansing, matching and consolidation, loaders), that you can chain together in a data pipeline.

2. Most tools supplement these transformations with an implementation of subsystems presented in chapter 9 of [1], which we summarize in section 5.3.5 below. In particular, profiling, change data capture, data quality subsystem, slowly changing dimensions, and surrogate key generation.

3. ETL tools generate clearer and auto-documented code, which is therefore easier to maintain and to extend.

4. The maturity of these engines is now such that performance of the generated ETL code is on par to what can be expected out of custom code.

Technical issues aside, you shouldn't go off in a direction that your employees and managers find unfamiliar without seriously considering the decision's long-term implications. You may also be faced with the decision to have to use legacy licenses, although in many cases this requirement is one you can live with. Finally, if you do end up buying an ETL tool, be aware that payback in improved productivity will only arrive after the first iteration, as it takes as much time to set them up and learn how to use them as they save in the first round..

4.4.1.2 Which processing pattern: ETL or ELT?

As the order of the letters in the ETL acronym implies, in an ETL tool, the needed transformations to integrate and cleanse data are done prior to loading the data in the target engine, typically in a middleware engine that comes as part of the tool. A recent, alternative popular option is ELT, which leverages the power of the target system as the place to do the transformation, in particular when there are large amounts of data to load and transform.

This trend started becoming popular with ETL engines a few years ago, driven by performance reasons when dealing with data volumes: they can now operate in ELT mode by generating code that is "pushed down" to make it execute directly on the target engine, whether the target is a traditional database or data warehouse engine, a cloud data warehouse or a big data Hadoop cluster. This is the case of advanced tools such as Informatica PowerCenter, IBM DataStage, SAP Data Services, so this capability has now become mainstream.

We have also witnessed the emergence of a market for ELT add-ins to database engines. The difference with the ETL tool category is that there is no middleware ETL engine, just a target engine code generation play. In this last category, the main value add beyond performance is to provide an environment that helps with the complexity of transformations, at a reduced cost. Talend Open Studio, Attunity Compose and WhereScape RED are examples of this market category. Finally, we observe that the ELT trend has also become popular with traditional databases: both Oracle and SAP have now data integration and cleansing options working in ELT mode on their flagship databases. These vendors advocate that fast loading is not just about flipping some parameters around and transformation is not just about simple SQL code generation: complex data pipelines have to be mapped to complex scripts and/or stored procedures to get them to perform effectively, so they believe that they are the best positioned to effectively develop these products.

www.persistent.com

So, is ELT better than ETL? Not necessarily. There are scenarios where the ETL approach can provide better performance. For instance, if you are streaming data in real time (from messages, database changes or sensor data), and your data pipelines have multiple steps that can be processed in main memory without requiring any reading (e.g., data lookups) nor writing (e.g., no generation of history records) to the database, the time to process the data while it moves through takes little relative to the time required to read from or write to disk. Even complex calculations performed in memory take a negligible performance hit. Furthermore, if the data can be split without affecting the computation, parallel execution of pipelines, which is possible in many ETL tools, may further yield significant performance gains. Another scenario that works better with ETL is when there are several databases, data warehouses or data marts to be fed after the transformations, or the transformations are so complex that cannot be translated into an ELT code.

On the other hand, all things being equal, the reason why ELT has become a valid and important architectural pattern in data integration is twofold: (i) it is preferable to have fewer moving parts: **by reducing the data movement through processing engines you might get better performance** and this translates also into cost and risk advantages in terms of money, development time and maintenance; and (ii) **using all the horsepower of your data engine to load and process your data will also help with performance.**

A related decision is the following:

4.4.1.3 Should we stage the data during ETL or not?

As mentioned in the point above, ETL tools can establish a direct connection to the source database, extract and stream the data through the ETL tool to apply any required transformation in memory, and finally write it, only once, into the target data warehouse table. However, despite the performance advantages, this may not be the best approach. **There are several reasons an organization might decide to physically stage the data** (i.e., write it to disk) **during the ETL process**:

- The organization may need to stage the data immediately after extract for archival purposes — possibly to meet compliance and audit requirements.

- A recovery/restart point is desired in the event the ETL job fails in midstream — potentially due to a break in the connection between the source and ETL environment.

- Long running ETL processes may open a connection to the source system that create problems with database locks and that stresses the transaction system.

4.4.1.4 Which Change Data Capture (CDC) mechanisms should we choose?

CDC is the capability of being able to isolate the relevant changes to the source data since the last data warehouse load. The most relevant CDC mechanisms are reviewed in [1], pages 376-378 and summarized in section 5.3.5 below. Finding the most comprehensive strategy can be elusive and you must clearly evaluate your strategy for each data source. The good news is that many databases are providing CDC capabilities out of the box. Our experience with them, which we justify in more detail in sections 5.4.1 and 7.2.4, is that **using the CDC capabilities from the source databases is worth the effort.**

4.4.1.5 What to do with data of poor quality?

Business users are aware that data quality is a serious and expensive problem. Thus, most organizations are likely to support initiatives to improve data quality. But most users probably have no idea where data quality problems originate or what should be done to improve data quality. The best solution is to have the data captured accurately in the first place. There are three choices if poor data quality is found:

1. Discard the offending records

2. Sending the offending record(s) to a suspense file for later processing

3. Tag the data with an error condition and pass it through the next step in the pipeline

The third approach is recommended. Bad fact table data can be tagged with an audit dimension record ID that describes the overall data quality condition of the offending fact row. 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 field itself. Both types of bad data can also be eliminated from query results if desired. We discuss this topic further in section 6.2.4. The other two choices either introduce distortion in the data or may compromise integrity of the data, as records are missing.

In big data environments, this topic becomes more of an open question than in a traditional data warehouse, as we will see in section 6.3.3.1 below.

4.4.1.6 When is it important to capture and integrate metadata?

The list of requirements might include one or several of the following functionality:

- **Impact analysis** (e.g., which views and reports would be impacted in the production database upon new changes being prepared in development of the next version)

- **Lineage analysis** (e.g., where does the data from this report come from, which transformations has it gone through), and

- Automating updates / additions to the target presentation server model upon corresponding changes in the source model,

In any of these cases, there needs to be a way to integrate the various metadata repositories from the off-the-shelf selected tools, which may include a modeling tool, an ETL tool, the target database and a BI tool. The DW/BI architecture must address this matter. Metadata management tools are starting to become available as extensions of an ETL and/or a Data Quality suite and enable features such as the above. However, a single source DW/BI technology provider is the best hope to get a single, integrated metadata repository. Otherwise, standards such as CWM for exchanging metadata among different systems are now mature and being used for these matters, and some tools based on this standard can read and write to the major DW/BI vendor metadata repositories.

www.persistent.com

4.4.1.7 Should I consider buying an MPP database, or will an SMP database do the job well?

Please refer to the glossary (chapter 9) for a brief explanation of the terms MPP (massively parallel processor) and SMP (Symmetric Multi-Processing).

Although PSL's customers generally have already decided on the database engine they will use for their DW/BI project, we have seen recently opportunities (see 3rd solution of section 7.3.1.2) where we have been asked to study the matter and make a recommendation, so we believe it is pertinent to review this aspect in this section.

MPP and SMP database systems have been measured on a well-known decision support benchmark, namely TPC-H[13] .Unfortunately, TPC-H[13] has received multiple criticisms, and some of the most popular vendors (Teradata, IBM Netezza, HP Vertica, Pivotal Greenplum) have withdrawn from it. Only a handful of DBMS vendors continue measuring their systems, among which there are Microsoft (although not for their MPP database), Oracle, Actian, Exasol, and Sybase IQ.

Nevertheless, the only MPP database still officially measured by TPC-H (Exasol, a little known German DBMS provider, who markets an in-memory, MPP database) performs better than the other SMP systems, both in performance and in price/performance. Other supporting evidence is that, in previous years, results for vendors with both SMP and MPP products (IBM DB2, Oracle 11g/12g) were better for their MPP products. Finally, we have come across products in the popular vendor MPP list and they indeed perform well in the high end when they are tuned appropriately. Some results and further explanations can be found on section 7.2.2 and 7.3.3.1 below.

On the other hand, SMP Databases have their share of advantages with respect to MPP: they are relatively cheaper, and they are easier to manage and configure. So, for relatively small to medium database sizes (in the order of a few terabytes and below), DBMSs working on SMP machines remain good candidates. However, in SMP architectures, the use of a single shared communication channel to communicate between CPU, memory and I/O becomes a bottleneck to system performance as the number of CPUs and their clock speeds increase[14].

4.4.1.8 Should I consider buying an OLAP server?

As we will discuss in chapter 5.3.3 below, the single most dramatic way to affect performance in a large data warehouse is to provide a proper set of aggregate records that coexist with atomic fact records. OLAP servers have been designed from the ground up to natively support the dimensional model. However, the choice between deploying relational aggregation tables or OLAP dimensional cubes is not necessarily easy. It is a multi-faceted decision. **It is a mistake to consider it as a tactical choice to be executed at the very end of the data warehouse development** only because aggregation is the last step when building the warehouse.

OLAP cube technology has the following advantages (over relational):

- They have far superior analytic capabilities. We will review this aspect in section 5.3.3 below.

- When the cubes are designed correctly, they deliver much better performance than relational aggregations, with less tuning effort.

- Security is more powerful on OLAP thanks to the parent-child semantics implicit in access languages.

---

[13] A new benchmark is available for decision support systems: TPC-DS, which is designed to compare both traditional warehousing and big data solutions. However, there has been no official results submission since it became a standard in 2012.
[14] 64 processors seem to be the limit (and the price tag is in the 5-million-dollar range).

www.persistent.com

Because of these significant advantages, some vertical markets such as finance have developed advanced solutions based on OLAP technology. On the other hand, the list of disadvantages includes:

- Although Microsoft's MDX is the closest standard access language, it is not widely implemented. Thus, OLAP is non-standard, non-portable technology. Also, development expertise is fragmented by vendor; there is much less expertise in the field as compared with SQL.

- OLAP is sensitive with respect to some update types (e.g., to type 1 SCD change, see 5.3.3).

- For reasons explained in 5.3.3, OLAP cube technology does not generally replace relational technology in a warehouse. Thus, adopting OLAP cube technologies may add to the total cost of the solution, especially if the relational and OLAP technologies are bought from different vendors.

At one end of the spectrum, if you have very advanced analytics needs and a need for predictable, high performance complex queries, and/or if you are in an industry where you can purchase off-the-shelf solutions that work best on OLAP, and you are confident that you can hire expert developers, consider buying an OLAP server. At the other end, if you want to establish commonality across your DW/BI projects around standard database languages without being tied to any vendor, then consider a dimensional relational approach. In between these two extremes, you must take a closer look to the relative pros and cons of each solution for your case.

4.4.1.9 Aggregate Navigation

Aggregate navigation is the part of a query rewriting service that recognizes that a query can be satisfied by an available aggregate table or cube rather than summing up detail records on the fly. The aggregate navigators receive a user query based on the atomic level dimensional model metadata and shields the user application from needing to know if an aggregate is present (just as with DBMS indexes). At the implementation level, there are a range of technologies to provide aggregate navigator functionality, including: OLAP engines, relational materialized views, or BI tools.

Kimball (see [1], page 354) argues that it is preferable to use aggregate navigation from the presentation layer supporting systems (either OLAP or relational engines), rather than using one from the BI tool, the reason being that in the latter case, the benefit would be restricted to a single front-end BI tool. In our experience, there is a case however to be made to move aggregate navigation up the chain in the following cases:

a. When aggregate tables are implemented in straight relational DBMS tables, a quite common case, rather than in materialized views or ROLAP systems that understand dimensional semantics on top of relational.

b. When all BI tools available implement this functionality (which is becoming mainstream).

c. In the case of mixed OLAP – relational implementations, where some aggregates are built in relational implementations and others in OLAP, and the BI tool can recognize which aggregate to use based on the incoming query –this needs, however, a sophisticated BI semantic layer that abstracts over relational and OLAP models.

4.4.1.10 What to look for in BI tools/applications?

The front room is the public face of the DW/BI system; it's what business users see and work with day-to-day. There is a broad range of BI applications supported by BI management services in the front room, including ad hoc query and reporting BI tools, dashboarding and scorecarding tools, and more powerful analytic or mining/modeling applications.

Selecting a BI tool may be a complex process: there are many vendors offering products, none of them solving completely all requirements (there are very few that combine ad hoc query, enterprise reporting and dashboarding/scorecarding, for instance), all of them with different strengths and weaknesses (not all of them work well on OLAP cubes or provide a multi-engine BI semantic layer abstracting a presentation server that can, in fact, be a logical, federated service), and anyway there is a range of different access needs which no tool meets completely. A case in point is the recent emergence of new self-service BI systems, see section 3.3 below: analysts need data to solve more and more business questions which the existing reports and dashboards can't solve.

This means that the nuances of selecting a BI tool should pay special attention to the requirements: be as comprehensive as possible of all the types of calculations; include all critical functionality; obtain all the education you can before identifying the product candidates, and spend time testing the top candidate systems, involving business users to understand their perspective, and get their support for the final choice.

4.4.2 Cloud Deployments

Most of the technical architecture material for cloud deployments has already been introduced in section 3.1.

For PSL engagements in this area, **the piece of advice we can give is to first make sure to understand what your customer's goals are**. Most of the time they would have made the decision about what type of cloud they need, and even which vendor. However, requirements may change over time; sometimes there are conflicting requirements, and sometimes they want PSL to help with the decision, so it is always good to know the list of cloud data warehouse "advertised benefits" for each type of cloud (review section 3.1.1 to this end), and contrast it to the customer's needs, to be in a good position to advise the customer.

Also make, sure you understand the challenges around moving and integrating data in a cloud data warehouse, and on connecting cloud and on-premises applications, which were also developed in sections 3.1.2 and 3.1.3.

4.4.3 Big Data

4.4.3.1 Reference Architectures

Most Big Data projects use variations of some existing Big Data reference architecture provided by data platforms. Understanding the high-level view of this reference architecture provides a good background for mapping the data flows with components required and how it complements existing analytics, DW systems. This architecture is not a fixed, one-size-fits-all approach. Each component of the architecture has at least several alternatives with its own advantages and disadvantages for a particular workload and type of the data. **It is recommended that architects start with a subset of the patterns in this architecture, and as they realize value for gaining insight to key business outcomes they expand the breadth of the use by adding in more components or tools.**

www.persistent.com

We recommend deploying components from the reference architecture that address below needs across different data projects in Big Data setup:

- Data Models, Structures, Types

- Data Lifecycle Management

- Data transformations, Provenance, Curation, Archiving

- Target use, presentation, visualization

- Cluster size or Infrastructure - Storage, Compute, and Network

- Data security in-rest, in-move, trusted computational environments

4.4.3.2 Hadoop and MPP

This section addresses the relation between Hadoop-based big data architectures and MPP databases, as there is indeed some relation between these concepts –and quite some confusion.

In Hadoop, there is some notion of scale-out by adding servers to a Hadoop cluster. On the other hand, MapReduce, the first distributed processing that appeared does allow (but does not require) all its computational tasks to run in parallel.

So, this bears the question, is Hadoop MPP? There is no simple answer or, better yet, the initial answer was no, and it now progressing towards a "maybe", depending on what Hadoop solution you pick. Indeed, Hadoop is not a single technology, it is an ecosystem of related projects. The question needs to be asked, therefore, at each level of the stack. Recent solutions such as Impala and HAWQ are MPP execution engines on top of Hadoop working with the data stored in HDFS. SparkSQL is another execution engine trying to get the best of both MapReduce and MPP-over-Hadoop approaches, and having its own drawbacks.

As for the question, **"Should I choose an MPP solution or Hadoop-based solution?"** the state of the technology today is that with these sophisticated Hadoop execution engines, you can get queries to return in a decent amount of time these days on huge datasets, but Hadoop cannot be used as a complete replacement of the traditional enterprise data warehouse. On typical data warehousing queries, we are still talking about response time differences of about one order of magnitude of difference, sometimes even more.

4.4.4 Self-service and agility

This section will not delve into self-service technology and products, as this has been the focus of section 3.3 above. We will only add that you should make sure you involve business users to select their tools, especially those that are self-service.

We consider here the impact that agile methodologies could have from a technical architecture point of view on the development of a warehousing/analytics solution. Agile methodologies should not be limited to BI; rather, they should be directed at all layers of the data warehouse, particularly the database and ETL design, which typically make or break a data warehouse project. To apply successfully an agile methodology on a data warehouse project, several practical measures can be implemented in concert with the methodology itself. These include:

www.persistent.com

- **Adopting data modeling and ETL tools** that improve developer productivity (see sections 4.4.1.1 and 5.4.1).

- Placing a **heavy emphasis on repeatable, automated testing** that implies a test automation framework –we already commented on this in section 4.3.4 above.

- **Investing in emerging technologies such as BI SaaS or cloud data warehouse may also help in developing new reporting capacity and capabilities faster** without the footprint of setting up nd managing an on-premises data warehouse environment (see the discussion in section 3.1.1)

- **Selectively repurposing the staging area from a hands-off area to a publicly available resource to business users.** This area represents a partial view of source data in a raw form. Typically, business users cannot understand the raw source data, and in this area performance is not reliable. However, the upside to keeping a full view of the raw source data in the staging database is to keep all the data deemed relevant for analysis without filtering out anything and even without having completed all the details of the data model (you will have recognized here the same principle animating data lakes, see section 3.2.2). That way, technically savvy users can access the raw source data through ad hoc queries against the staging database and allow them to validate initial business rules, which usually helps uncover new business logic that impacts the data integration and data quality endeavors. Reaching these crucial validation points early in the design and development stage can help avoid scrap and redesign.

5. Dimensional data modeling

5.1 Dimension Data Modeling Overview

In [1], Kimball describes Dimensional modeling as "a logical design technique for structuring data so that it is intuitive to business users and delivers fast query performance. It divides the world into measurements, or facts, and context, or dimensions[15] .

- Facts: The measurements of the business are usually numerical and are referred to as *facts.*

- Dimensions: The context that describes the "who, what, when, where, why, and how" of the measurement are the dimensions.

Every dimensional model is composed of one table with a multi-part key, called the fact table, and a set of smaller tables called *dimension* tables. Each dimension table has a single part primary key that corresponds exactly to one of the components of the multipart key in the fact table.

Dimensional modeling is arguably the best approach to support the twin goals of understandability and query performance in BI / analytic environments.

---

[15] A historical footnote: the terms "dimension" and "fact" originated from developments conducted jointly by General Mills and Dartmoth University in the late 1960s. Data marts were defined as dimensional models describing a single business process. Nielsen was the first vendor to provide dimensional marts for retail sales in the late 70s. The original terms "conformed dimensions" and "conformed facts" were described by Nielsen Marketing Research to Ralph Kimball in 1984. Finally, the idea that a data warehouse can be built incrementally from a series of data marts with conformed dimensions was fully described by Ralph Kimball in a DBMS magazine article in August 1996.

5.2 Brief Summary

Dimension models are usually *star schemas* or *snowflake schemas.* These are unlike the 3NF models which are mostly used for transactional systems. Star schemas consist of a fact table surrounded by multiple dimension tables. The origin of the term star schemas has to do with the visual layout of the high-level schema diagram. In a star schema dimensions are de-normalized to avoid capturing too many minuscule relationships in separate tables. A star schema is generally recommended in the database presentation layer to satisfy performance needs as well as to service reports/dashboards.

Contrasted with the star schema model, we have the snowflake model. This seeks to model dimensions into separate tables, as with a 3NF model, based on redundant attributes, thereby modeling hierarchical levels within dimensions in their own table. Snowflaking generally compromises performance and understandability. However, under certain conditions which we discuss in section 5.3.3 below, this approach may be acceptable.

The other extreme is a fully de-normalized schema (or the spreadsheet model) that merges dimension attributes with fact table attributes in a single table. Though often talked about with disdain, is also a model that could be used in cases of databases that don't have stable join optimizer plans or those unable to perform them by design. Indeed, this pattern is best from performance as there are no or minimal joins or indexes lookups. However, it presents problems when dealing with historization of dimensions with respect to attribute value changes (see sections 5.3.4 and 7.2.3 below for more details).

When we look at the schema of the DW, we see tables of the following kind:

- Transaction Fact tables: These are facts where the grain is one row per transaction. These are the most basic and common facts (There are 2 other types of facts – *Periodic Snapshot Fact* tables, and *Accumulating Snapshot fact tables*, and we shall discuss them later).

- Dimension tables (who, what, when, where, how, why): Key questions related to a business event are answered by means of these dimension tables using their attributes, which can be used to filter data values in the where clause, slice/dice in the group-by clause or labeling results, in the select clause.

- Bridge tables: They help resolve m:n relationships with dimensions, while keeping the star schema intact (see section 5.3.3 below).

- Control tables: These are for the sake of recording operational information (permissions, etc.)

- Report tables: These table record summaries, and are based off facts and dimensions.

Conforming dimensions are the master data of the DW/BI environment. Common dimensional attributes for different departmental databases are modeled, making sure that the data from these sources refer to the same concept and hold values of the same domain; then they are then shared by multiple dimensional models to enable enterprise integration and ensure consistency.

Conforming facts are obtained by making agreements on common business metrics across these data sources so these numbers can be compared by formulae in the same analysis.

5.3 Best Practices

This section lists some best practices from the reference book, from a dimensional data modeling perspective. They appear in the sequel in **boldface.**

5.3.1 Dimension Data Modeling - Project definition stage

When the project is being launched and the core team is lined up, in addition to the classic roles of project manager, data architect, ETL developer, BI developer, etc., a best practice is to **make sure there is a Data Steward** ([1], p. 35). A common curse is departmental data silos, where every department has their own definition and interpretation of data entities, and transactional events, no one's data ties to anyone else, and the result is anarchy. Data stewards should be empowered to establish, publish and enforce common definitions, rules and permissible values for data for all information that is of cross-organizational interest in the enterprise. It is important to make sure data stewards work with both business users and IT team members, and are well supported, as this is a very politically challenging role. Down the line, together with the QA team, they will also identify data quality errors and drive them to resolution (which may imply revisiting the definitions and the rules).

Make sure you **drive priorities based on business goals, and that they are balanced with delivery feasibility assurance from IT architects. Set up realistic expectations with all stake-holders.**

5.3.2 Dimension Data Modeling - Requirements Interview

Document the high-level Enterprise Data Warehouse in the preliminary EDW bus matrix (see section 4.1) as an outcome of the interview process, as this practice helps come up with common dimensions that apply to several business processes. **The best practice of designing conformed dimensions based on the bus matrix** is of immense importance, as it serves as a data architecture blueprint to ensure that the DW/BI data can be integrated and extended across the organization over time.

5.3.3 Dimension Data Modeling - Logical

These are the logical dimensional data modeling best practices from the reference book [1].

- **Derive dimensional models from workshops with business users and IT representatives** rather than from an isolated designer sitting in an ivory tower. Engaging subject matter experts from the business is critical to designing appropriate dimensional models.

- **Creation of durable Surrogate/Supernatural Key** for item keys that are less likely to change but, surprisingly, do change (ISBN 13 is an example). Surrogate keys are created for primary keys in dimensions. This is a useful concept, even more so in the world of rapid data growth/collection, and mergers and acquisitions. See also the discussion on subsystems 10 and 14 in the section 5.3.5 below.

- In the case of dimension hierarchies, **when the conditions below are met, a snowflake model may be evaluated as a possible fit. Otherwise, the recommendation is to prefer de-normalized dimensions, i.e., a star model.** The discussion in [1], pages 268 and 339 about dimension hierarchies and outriggers is excellent for the practitioner and very applicable. We often encounter fixed hierarchies in the form of time-date, and geography (city, county, district, province, state, or region). Snowflake models show each hierarchical level separated into its own table, and this is acceptable when either

- Dimensions are large, as there may be space savings,

- Slicing-dicing by all possible hierarchy attributes is needed (in addition, some BI tools work better with a snowflaked design), and/or

- There is a need for other fact tables at a grain corresponding to higher hierarchy levels (as they can hook into the appropriate level of the snowflake structure; besides, surrogate keys at higher hierarchy level may be used for aggregating fact data with summary tables or with OLAP cubes–these keys enjoy the same advantages related to stability under changes just discussed in the previous point).

- The hierarchy is ragged, i.e., where the number of levels is variable and of the same type of data (as in organization structures or in a parts hierarchy). In this case, a bridge table (a form of snowflaking) works best, if you need to both restrict facts based on dimension data and summarize them based on the hierarchical recursive relationship (see [1], page 269).

- Another exception is what is called an "outrigger dimension", which is a dimension table joined to another dimension table[16] , i.e., another form of snowflaking. A discussion when it makes sense to use outriggers can be read in [1], page 267.

- When a fact measurement is associated with multiple values of the same dimension (even at the lowest level of granularity), it is necessary to have **a bridge table to link the fact and the dimension**. Examples of open-ended multiple values, such as a variable number of diagnoses for the same hospital bill line item can't be resolved directly on the fact table.

- To satisfy lookups and joins, it is recommended to **have the dimension columns of the Fact table declared as NOT NULL. This also means that each dimension should have a Null value as a row**. Generally, a row with key -1, and value as "undefined" or "unknown" is inserted in the Dimension table, and the -1 is the corresponding value in the related Fact table record, to make sure the lookup/join from fact to dimension is correctly satisfied. NULLs are also not very intuitive for SQL summary and ranking operations.

- Historization/Slowly Changing Dimension: Kimball [1], page 257 also has a great and educative discussion on SCD dimensions. There are multiple SCDs, **the most common of them is the Type 2**, which provides both the newer and older entry, with an indicator for which is the most current record, and the begin/end validity dates. See also the discussion on subsystem 9 in the section 5.3.5 below, as well as the discussion about mini-dimensions in subsystem 12.

- Special Types of Fact tables: These two kinds of facts have always been constructed where needed, in addition to the basic transaction fact, in response to a report/query requirement.

  - Periodic Snapshot Fact Tables: **At the periodicity of a day, week or month, a snapshot is captured for the fact table, with regular repeating measurement or set of measurements**. This is common in the financial industry for monthly balances, in ERP and retail systems for inventory balances, and in the travel industry for bookings. Both *en-masse*, end of period loads (adding rows per period) and continuous, rolling loads (updating the current period rows) may be implemented.

  - Accumulating Snapshot Fact Tables: For processes with many states with a well-defined beginning and end (e.g., added to cart, paid, shipped, delivered, returned), the lowest grain transaction fact would have multiple records with different dates for the same order. **The accumulating snapshot fact has the discrete states listed with dates** to enable queries like average interval between delivery dates and return dates.

[16] For instance, a hire date on employees, or a product introduction date, and the business needs to slice and dice by non-calendar date attributes (in this case, a virtual outrigger is to be set up on a date dimension table, as it plays a different role).

www.persistent.com

- **Aggregations: relational or OLAP?** Aggregates can have a very significant effect on performance, in some cases speeding queries by a factor of 100 or even 1000. However, deciding what to aggregate is not necessarily easy. User input is critical, as well a means to monitor slow running queries. And, as mentioned in chapter 4, the choice between deploying relational aggregation tables or OLAP cubes is not easy.

  OLAP cube technology have the following advantages over relational:

  - They have far superior analytic capabilities. There are several aspects to this:

    - The definition of calculations, both calculated members and measures, defined on robust query languages such as MDX, is very powerful.

    - The management of hierarchies is also advanced: for instance, parent-child, ragged dimensional hierarchies are traversed and used to aggregate measures with powerful and elegant semantics (whereas this is painful to do in SQL, as mentioned at the beginning of this section).

    - The management of aggregation rules for semi-additive measures, such as inventory balances, and non-additive measures, such as ratios, is much easier to specify (in relational this must be dealt directly in the aggregation code).

  - Cubes deliver better and more predictable performance than relational aggregations, with less tuning effort. With relational implementations, determining the optimal set of aggregations to implement is a difficult problem. OLAP databases provide more assistance than relational technologies in this area –even though it is still true that the performance and scalability of OLAP databases varies widely from one product to another.

  - As mentioned already, security is more powerful on OLAP as it meshes well with parent-child semantics.

The list of disadvantages includes:

- OLAP is non-standard, non-portable technology, and development expertise is scarcer as compared with SQL and fragmented by vendor.

- A Type 1 SCD change to a dimension may cause all the cubes using that dimension to rebuild, and this may take a very long time. Aggregates defined on a Type 1 dimension attribute should be allowed only under special circumstances (e.g., updates are rare, cubes depending on it very few and not too large).

Reasons why relational DBMSs won't be entirely replaced by OLAP engines in a DW include:

- Loading OLAP cubes from non-dimensional data schemas is difficult, as most OLAP systems do not directly address the thorny problems of conforming dimension and fact data, data cleansing and management of constraints such as referential integrity. For these reasons, Kimball recommends OLAP supported by underlying dimensional relational schemas.

- When the system is very large, detailed level transactional fact tables are kept in relational models and aggregate data is stored in OLAP cubes, because OLAP vendors may have size limits not present in relational implementations (although OLAP databases can scale to multiple terabytes these days).

- Finally, relational technology is more mature for data management aspects such as backup/restore and archiving.

www.persistent.com

5.3.4 Dimension Data Modeling - Physical

The following are the physical dimensional data modeling best practices from [1].

- Physical dimension and fact table management ([1], pages 262 – 271) is covered by subsystems 10 to 15 described below. The same physical dimension may play different logical roles in a dimensional model ([1], page 262). In this case, the illusion of independent dimension tables can be achieved through database views or aliases: strive to **name the columns playing the different roles using different names** (e.g., order date and shipping date) **through views**, but nevertheless build and administer a single physical table.

- **Consider adopting the fully de-normalized dimensional model in cases of databases not so performant for joins** (HPE Vertica), **or those unable to perform them** (NoSQLs – Cassandra, HBase –see section 5.2 above). This may also be applicable where the database join optimizer plan is not stable.

- Indexing strategy: There is a good discussion on indexing of Fact Tables and Dimension tables in [1], pages 345-50 as part of the "Designing the Physical Database and Planning for Performance". **Having an index plan, and then adjusting the indexes as needed** is a best practice, as it is not uncommon to see databases with 15 indexes per table. The index types (B-Tree, Clustered, Bitmapped, Index Organized Tables), the optimizations (Star Schema optimization, Cost based optimization and query plans), and the operations (Staging, Transforming, Loading) all have to be considering when creating the indexing plan.

- Partitioning: The advantages of partitioning when dealing with loading as well as with query performance are listed in [1], pages 359-60, in the section on Physical Storage Structure. **A partitioning plan should be drawn up considering performance of operations (load, archive, query), and table maintenance.** As mentioned below in section 5.4.2, partitioning is also useful in achieving multi-tenancy when shared schema approach is followed, and helps with security (see section 7.3.3.2)

5.3.5 Dimension Data Modeling - ETL Considerations

Chapters 9 and 10 of [1] introduce the most crucial part of any warehousing project, namely ETL. The former introduces a series of ETL subsystems (pp. 387 – 404), and the latter some best practices on how to develop dimension and fact table initial and incremental loading (pp. 437 – 468). In what follows we present some distilled notes of what we consider best practices on both the subsystems and the development techniques. We start by the Chapter 9 subsystems.

- Subsystem 2: Change Data Capture (CDC) system. This is the capability of being able to isolate the relevant changes to the source data since the last data warehouse load. There are several strategies to compute the delta record set; the most popular ones are (i) database transaction log read, (ii) trigger-based snapshot capture, and (iii) timestamp (a.k.a. audit) columns in each source table, with the ETL code finding the delta record set with time-stamp based queries on these tables. The two first ones are more reliable since the underlying data is generated by automated means. When timestamp columns are populated by anything other than automated triggers, you must pay special attention to whether they are a reliable source to indicate change.

- Subsystem 9: SCD (Slowly Changing Dimensions) Manager: **Having a specialized/centralized SCD Manager** that has logic to deal with SCD Type 1 (Overwrite), Type 2 (New row, with older row having a "valid until date"), Type 3 (new descriptive attribute added to dimension) changes or a Hybrid SCD approach **is a definite best practice.**

- Subsystem 10: Surrogate Key Generator: The ability to **generate surrogate keys** independently for each dimension, independent of database instance, independent of the operational key, with the ability to serve distributed clients, in a non-bottlenecked and scalable manner **is an often overlooked, but very crucial practice for ETL performance and portability.**

- Subsystem 11: Hierarchy Manager: For intermediate tables and **ETL staging tables storing dimensions with hierarchies, often normalized structures work best**, to have the database assist in pre-verifying the many-to-one hierarchy relationships through referential integrity, especially if the data comes from an informal source (see pages 393 and 442). However, as mentioned in section 5.3.3 above, the same discussion about dimensions with hierarchies at the presentation layer points to a different conclusion: prefer de-normalization, except when meeting several of the conditions mentioned above.

- Subsystem 12: Special Dimensions Manager: This ETL subsystem deals with dimensions as the ones presented below. **The best practice in this case is to manage them in a special way and not just like the normal dimensions.**

  - Date/Time: Dates and times over last few decades, and next few decades are stored here, and with consideration to financial reporting years of the organization. These dimensions generally don't come from a managed source and are typically created in spreadsheets and imported.

  - "Junk" dimensions ([1], page 263): These are dimensions recommended to be built from test and miscellaneous flags left over in the fact table, and that are not naturally members of any existing dimension. Rows should be created either based on number of rows known in advance, or as-and- when fact table inputs rows arrive.

  - Mini-dimensions: These are recommended when it is required to track changes to frequently changing dimension attributes in a very large (wide) dimension and the SCD Type 2 technique is too costly in terms of size. The solution is to **break these dimension attributes in their own separate dimension table**, called a mini-dimension (see [1], page 259).

  - Shrunken dimensions: These consist of conformed dimensions that are a subset of rows and/or columns of a base dimension, and built entirely from its base dimension rather than from source data to assure consistency.

  - Small static dimensions: These are dimensions built without a real outside source usually for small lookups (e.g., time zones, full names of months of the year).

  - User maintained dimensions: These dimensions are created by the lines of business and are typically used for descriptions, groupings and hierarchies for reporting and analysis. Examples are organization chart structures, and lines of business sales hierarchies. As such, **it is recommended to have the ownership of the data be with the business team,** and the ETL team just imports data from an operational/front-end system with the appropriate defaults.

- Subsystem 13: Fact Table Builders: The ETL considerations for the three kinds of fact tables are listed below. **All three table types may coexist, as each allows meeting different needs.**

  - Transaction Grain Fact Table Loader: After the initial load, records are fed into the transaction fact table by using the CDC (Change Data Capture) system, and the data is loaded with the proper surrogate foreign keys in a process described in subsystem 14 below, thereby guaranteeing referential integrity. Tables are usually **partitioned by transaction date** (for database administration and performance), and columns related to ETL job sequence ID, loading timestamp and source lineage information. Late arriving fact data should be processed along the lines of subsystem 16 (see below).

www.persistent.com

- Periodic Snapshot Loader: An ETL fact table builder can be run at the periodicity of a day, week or month, to capture a snapshot to update this fact table.

- Accumulating Snapshot Loader: An ETL Fact table builder runs at intervals to gather the state transitions and summarize them into the Accumulating Snapshot Fact.

- Subsystem 14: Surrogate Key Pipeline: This is the process that exchanges the natural keys of each dimension by the surrogate keys and handles any referential integrity error. Dimension table processing must be complete before the fact data enters the surrogate key pipeline, for both the initial (historic) load and the incremental load. **Mapping tables should be retained in the ETL pipeline to get the corresponding surrogate keys for natural keys.** The ETL should have logic to insert into the dimension as well as the mapping tables, in case a new natural key is encountered.

- Subsystem 15: Multi-Values Dimension Bridge Table Builder: **Construction of the bridge table should happen during the during the ETL pipeline**. Additional complications arise as these dimensions may also be SCD Type 2.

- Subsystem 16: Late Arriving Data Handler: **The ETL system needs to take care of late arriving data, either dimension or fact records**. In the former case, if the business can live without facts that have no dimension data, one strategy is to park aside (into another area), fact data arriving before its dimension data. If reports with all fact data are needed, the facts could be loaded with dummy dimension placeholder records with their definitive surrogate key initially. These dimension placeholders are later updated when its data arrives. Additional consideration is needed when the dimension is slowly changing and updates arrive late, as the surrogate keys in the fact table point to the wrong dimension data, and for summary (aggregate) tables based on now obsolete dimension data[17]. If the Facts arrive late, expensive searches are needed to backtrack and adjust all the summaries, semi-additive facts, as well as locating which dimension surrogate keys were in effect at that time. During this time window, the reports will not be always accurate.

- Subsystem 17: Dimension Manager System: The set of dimensions should be tightly controlled by a Dimension manager interacting with other subsystems managing dimensions (9 to 12, 15, 16) The difficulty increases with multi-system EDWs and distributed environments where dimension tables are replicated. In this case, **a best practice is to have this subsystem replicate dimensions to the Fact Provider Systems (see below) with a version number attached to each dimension row.** The version number is very useful in case of drill across queries spanning systems, to ensure consistency.

- Subsystem 18: **Creation of a Fact Provider system** is a best practice, as it is responsible for replacing/recalculating surrogate keys, accumulating snapshots, and aggregations using some of the subsystems introduced above, plus subsystem 19. In distributed, drill-across query environments, it receives conformed dimensions released by the dimension manager.

- Subsystem 19: Aggregate Builder: This applies in general to environments that have not selected OLAP engines in their technical architecture (see section 5.3.3 as well as the point below). **ETL must make sure that the aggregates are consistent with the base data**. There are several possible implementations that range from relying on materialized view DBMS technology, to explicit ETL code implementation of this subsystem. The Fact Provider System usually triggers the Aggregate Builder based on changes to Fact table data.

---

[17] For instance, a sales by city aggregate for last month will be wrong when notice of a customer moving two months ago arrives.

- Subsystem 20: OLAP Cube Builder. This applies to environments that have selected OLAP engines as their primary user access presentation server in their technical architecture (see section <u>5.3.3</u> above)[18] . **At the very end of the ETL processing, the OLAP cubes get loaded**. However, as mentioned above, they are sensitive to the SCD nature of dimensions. Type 1 and Type 3 SCDs would cause large scale reloading/rebuilding of OLAP data (the same is true with aggregate tables just described).

- Subsystem 21: Data Propagation Manager: **The ETL system is the platform that now handles the distribution of integrated and conformed data of the warehouse to other systems, through data extracts** (subsuming the former EAI systems). Inputs from the warehouse feed to external entities (customers, regulatory bodies, partners, vendors), and for corporate analytical systems for data mining and algorithmic analysis. Some amount of transformation may be required from facts and dimensions into the formats required by each system.

Below are some best practices related to the development of dimensional modeling ([<u>1</u>], Chapter 10):

- **Use a data profiling tool** (or some queries that can be used to perform Data Profiling). Using a data profiling tool helps uncover the actual state of quality of the data (usually, not expressed via database metadata), understanding data issues, and sometimes, even discovering relationships and derivation rules. This is discussed more widely in section <u>6.2</u> below. The data profile is an input to the ETL transformations and logic, and guides the data quality effort.

- Step 5: This **checklist of steps to follow for dimension tables** is about the initial population of dimension data, including data cleansing and match/consolidation, especially when populating from different sources (this is developed in section <u>6.2.4</u> below), Type 1 / Type 2 transformations, validating relationships, surrogate key generation and assignment, techniques for fast loading, and loading Date and other special dimensions introduced above.

- Step 6: Perform the Fact Table Historical (initial) Load. Extracting the right range of facts, with the right date filter criteria is important to start the Fact table loading. Audit statistics with counts and sums gathered in these steps are important to perform validation and verification later. The fact records may contain derived, calculated facts, although in many cases it is more efficient to calculate them in an OLAP cube or a view rather than in the physical table. **Loading should be done by disabling constraints, dropping indexes, fast loading, and then enabling constraints and re-creating indexes.**

- Step 7: Dimension Table Incremental Processing has two basic strategies: extract full load at the source and compute the delta at the target, or extract only new and changed records at the source. The first strategy applies when the source system has no capability to identify the new, changed and deleted data; **the second strategy should be preferred, as it minimizes data transfer loads and expensive comparison operations.** As with initial loads, the dimension data should be cleaned and transformed prior to identifying new and changed rows. Techniques for identifying changes in the rows for Type 1 and Type 2 SCD processing if the dimension has many rows and columns is presented in pg 458.

- Step 8: Fact Table Incremental Processing. **Automatic auditing, error handling** (including persistence in an error event schema and verifying quality measures, as mentioned in section <u>6.2.4</u> below) **and enabling restarts after catastrophic errors in case the incremental load fails, are key in this step.** Loading of the transactional fact table should be straightforward; however loading the accumulating snapshot fact table could be complex. Loading could be improved by trickle load or parallel batch load operation, as suited to the database.

---

[18] In this case, there relational aggregation tables would not be needed (except in mixed OLAP/relational environments).

www.persistent.com

- Step 9: Aggregate Table and OLAP Loads. Aggregation may be managed by technology such as relational materialized views or OLAP cubes, in which case most of the maintenance work is done automatically after initial loading. Incremental processing refreshing the aggregates is possible even if aggregates are maintained by ETL code (you can be confident this is the case in OLAP and materialized view technology); however, a major change of a dimension attribute, such as a Type 1 SCD attribute change, may require reprocessing the aggregate table; this will be the case no matter how aggregates are implemented, if by ETL code or through database technology.

- Step 10: ETL System Operation and Automation.

  - Jobs are scheduled for a specific time, or may be triggered based on polling. **A clear workflow should be built for steps requiring order such as dimension management before facts, aggregate refresh after incremental fact processing, OLAP rebuild.**

  - Concentrate on **ETL testing** ([1], p. 455). First, use a small dataset to unit-test both fact table loading and dimension loading before throwing the full set of historic data at it. Direct error records to a file, investigate what went wrong, fix the code and rerun the test. When getting your first production load complete, with the right record count, don't cry victory too soon: confirm that the data is accurate at the target (again, a profiling tool is your best alternative). When initial loads are confirmed accurate, move on to test the incremental load processes ([1], p. 467). This is often neglected, is very unglamorous, **but these test scripts make the difference between a successful ETL project and an unsuccessful one.**

  - A point that deserves particular attention is how to **handle unpredictable errors**, and how to **recover automatically from them in a transaction-consistent manner**, especially given the fact that these errors typically occur in a "lights-out" environment. Many ETL products provide tools and features to enable you to build a robust system under all possible failures, but doing so is more complicated than it seems. **Alerts on unpredictable errors should be raised and audited.**

  - **Once any unpredictable error occurs and ETL is re-run, it should not result into any data loss or inflation;** i.e. the ETL queries or workflows must be coded to handle this. This is very important as the DW is supposed to provide single version of the truth.

  - There are several supporting database tasks necessary for ETL operations, e.g., table space management, periodic index maintenance, and database backups. **These tasks should be automated within the ETL environment to allow for end-to-end operations testing** (see next section), preferably by using database system metadata.

  - Finally, strive to **develop and test an automated operation as complete as possible** for ETL processes: the ideal ETL operation runs the regular load processes in a lights-out manner, without human intervention. More on this on the section below.

5.3.6 Dimension Data Modeling - Deployment Considerations

The deployment stage should go through the traditional rigors of alpha and beta testing prior to general availability and rollout (and this is true of any subsequent delivery after the initial release of the data warehouse).

www.persistent.com

- The alpha period is the data warehouse team's first opportunity to conduct an end-to-end system test before going live containing the following items:

  - Hardware and software installation including initial system and user configuration.

  - An automated ETL system test suite composed of test scripts containing (i) data extraction against a known, static set of data (containing both correct and imperfect data), (ii) initial loading tests, (iii) incremental data processing tests, and (iv) queries contained in predefined reports. **Build an application, the regression test suite, which compares and logs the result of all the runs from the test suite against known correct results**. Make sure you run it always before releasing your ETL and BI system.

  - **Data quality assurance testing.** This is described in section 6.2.5 below.

  - **Operations process testing.** This consists of the following steps.

    - Verify end-to-end operations first using the static regression testing dataset. Make sure that the imperfect data generates predictable exceptions and associated management. For instance, when receiving a late arriving dimension, test the handling of SCDs associated with such late arriving dimensions, recalculations, and re-aggregations; furthermore, make sure these don't halt processing.

    - Verify that jobs start when they should, that they run correctly under normal operation, and that standard reports are built on schedule.

    - Then, move on to alternative datasets allowing testing for handling unusual events (e.g., receiving data that is garbled, or has unbelievably high row counts). Test the procedures handling these events –this includes invoking database tasks such as table space management, as discussed in the section above.

    - Finally, move on to injecting unpredictable errors such as a network failure or a power outage, and test that the restart capabilities of the system work as they should.

  - **Live Testing**. Once the historical data is accurate and the operational processes are clean, real data testing may start. In the test environment, point to the same operational sources you will use in production. Run the live test long enough so that you can see real world patterns in the data and operational environment. If there is an operational cycle that may affect the system, such as a monthly closing process, make sure the tests span a complete cycle.

  - **Performance testing**. This is described in section 7.2.5 below.

- In alpha-testing generally no access to business users is given. This changes in beta testing, where the main goals are (i) to conduct an end-to-end user test going through the same items of alpha testing **and adding items such as end user application quality, training materials, support infrastructure and user communication,** and (ii) to work out any remaining glitches. The data warehouse team should also use the beta period to verify the necessary back-room operational processes before the user population expands, including back-up and archiving procedures, disaster recovery, stress testing, and performance monitoring, as described in chapter 13 of [1].

System deployment in production starts by passing the complete set of tests on a test system that is as similar as the production system as possible. Then, move the software artifacts from the test repository to the production repository. **The more automated and parameterized this can be, the better – and this deployment move is something else to test**. And perform a final automated test on the production system before letting people in. Deployment is much harder on an existing system in production. Pay attention to the chain dependencies: reports depend on views, which depend on data warehouse tables, which depend on ETL data pipelines, which depend on data sources. Changing any of these may break something down the line, so a system that maintains dependencies and allows impacts to be dealt with individually is critical.

Other deployment considerations for dimensional data warehouses are the following:

- From a dimension data perspective, **having a single deployment platform for MDM, Identity data, Customer 360 and BI/Reporting is the recommended approach.**

- **Storage management is another key component of the Deployment**, as the growth of data under analysis, can affect the performance (response times), and may force storage reconfigurations.

- The **deployment should facilitate behind the scenes housekeeping** - ETL metadata, availability, BCP/DR (Backup Continuity Planning/Disaster Recovery), Performance, Security, Archival, Staging Management.

5.4 Enhancements to the reference publication

5.4.1 Dimension Modeling in our own experience

- **High Level Model Diagram is important for anchoring discussion**. The creation of a high-level model diagram (Kimball, [1] figure 7-3) helps all the stakeholders including business users, in discussions. Diagrams usually elicit better feedback and discussion. This is even more important with distributed teams. The visual anchoring of the conversations as well as the visual recall provided by the diagram are of great value. To this effect, the advice about not radically altering the model (or even the relative positions within the high-level diagram) is very relevant.

- **Automatic Change Data Capture (CDC) mechanisms at the source are key for keeping the data warehouse up to date in the presence of hard deletes at the source.** Indeed, if the source deletes rows physically, only database logs or database triggers that generate data as a side effect of the deletion, with the correct operation code (DELETE), will provide the ETL layer with the necessary information to keep the target database up to date without having to compare full table contents from the source and the target during incremental loads, which will be a performance killer when tables are large. Also, as further discussed in the point below, **CDC mechanisms based on database transaction logs or triggers can be used for pushing changes in real time to the data warehouse** (as opposed to pulling them through queries). Section 7.2.4 compares these mechanisms from a performance point of view.

- **Creation of a "data highway" is often needed. Kimball highlights the need for data traveling at various speeds / in different lanes:**
  - Raw-source (immediate) - CEP, alerts, fraud
  - Real time (seconds) - ad selection, sports, stock market, IoT, systems monitoring
  - Business Activity (minutes) - Trouble Tickets, Workflows, Mobile App Dashboards
  - Top Line (24 hours) – Tactical Reporting, Overview status dashboards
  - EDW (daily, periodic, yearly) – Historical analysis, analytics, all reporting

Variations of the concept of data highway, are the Lambda architecture (Speed layer, Batch layer and Serving Layer), or the Kappa architecture (Streaming layer and Serving layer).

We see a demand for similar information delivery, and to satisfy this demand, a CDC database transaction based mechanism like Oracle GoldenGate, or a framework like Apache Spark is used. GoldenGate or Apache Storm fits in with the existing enterprise to create the "higher speed lanes", while not affecting any other existing operations (like batch Informatica or DataStage jobs), while Apache Spark has the advantage that it operates on both streaming and batch data.

- **Create a detailed "interface agreement" for feed based extraction.** The data modeler needs to understand, profile and document every data source. A large portion of data sources are feeds coming from the SoRs (Systems of Records). In large financial institutions, or airline companies (with a lot of legacy), these are often mainframes. For each of these systems that supply a feed, it is important to create a common understanding, and this understanding is documented in an "interface agreement" that both producer and consumer agree on. In large enterprises, other Lines of Business or business entities that are the source of data to the warehouse are almost like external organizations. This document is jointly created by the data modeler and the ETL architect.

  The agreement would detail the following information:

  - Format of the data (fixed width, separated – with separator character, block-style layout
  - Field names and expected data characteristics for fields (domains, ranges)
  - Incremental or full feed
  - File encoding (ASCII/Unicode, EBCDIC)
  - Transmission mechanism (Secure FTP, Shared Folder) and Transmission Frequency
  - Re-transmission policy
  - File naming convention (with overwrite-if-duplicate logic)
  - Any markers to indicate file is ready for consuming (a polling agent can poll for a filename.complete file in the shared folder)
  - Any markers to indicate that "no data available" for that time period (maybe a 0 byte file is made available, or maybe filename.empty file is created in the shared folder)
  - Any actions to be taken once file is consumed (archiving)

- **Use Modeling tools to the maximum, but only till they hit their limitations**.
  Some observations about usage of modeling tools are

  - In the organizations we have worked with, the preferred diagramming tool to be used by data modeler is either CA ERWin, MS-Visio, or Embarcadero/Idera ER/Studio.

  - When supported by the tool, these organizations have a repository of data models, and well defined (tool-enforced) naming conventions and practices.

  - The reverse engineering capabilities generating physical models from database schemas are largely unused, partly due to the version issues, but also due to the fear of "messing" with the original design/layout.

  - Very often, the tool versions used are the older versions

www.persistent.com

As a practice, **we would continue to recommend usage of modeling tools**; however, in case limits are encountered, it may be better to stay away from physical modeling. Either the versions cannot model the newer physical constructs in the database (like newer data types), or they are not equipped to deal with newer databases (like HPE Vertica).

- **Shared Data Architect/Modeler and ETL Architect roles.** In case of smaller projects or departmental data marts, **ETL architect and Data Architect roles are often played by the same person**. However, **a recommendation is for the person to recognize this dual role.** When executing an Agile project, the Data Architect/Modeler is usually 1-2 sprints ahead of the ETL architect, the tasks should reflect it. Another recommendation in this case is that since the "producer" and "consumer" of the model is the same person, **another set of eyes to review the work is useful.** Especially the BI Architect (responsible for the work in BI tools) is usually the consumer of the work of both the Data Architect and ETL Architect, with focus on the performance of the front-end visualizations and reports. This person can be a reviewer of the work.

- **Design the presentation layer database structures to suit the report refresh intervals, taking into consideration different reporting patterns**

  In our experience we have seen reports of the following kinds:

  - Existence: Did audience of certain demographic attend an event?

  - Comparison: How does the revenue for a particular demographic compare to another demographic?

  - Ratios/rankings/clusters: How does the ratio of a particular month rank with annual?

  - Statistical: Top N earning events

  - Point-in-time Target/Order fill chart: For a particular date in the future, what is the order fill chart as of today?

  - Rate/Velocity: What is the rate of booking as compared to rate of booking last year?

  - Time Trend (MoM, YoY): What are the Year on Year audience number?

  - Correlation: Increase/Decrease over a (non-date) dimension, usually numeric or scaled dimension. (For example: age, distance).

  - CrossTab: A plot of values of one dimension over the discrete values of another dimension

  Some of these complex reports require a presentation layer that has to be refreshed fully, not just incrementally. These full refreshes render a dashboard or report unusable for the duration of the refresh. To remedy this, at times, it is required to generate two sets of tables, one from which the reporting is taking place, and the other being refreshed. Sometimes this may be the key to deciding if an OLAP layer is needed or the presentation layer tables can be created through ETL.

www.persistent.com

5.4.2 Cloud Deployments

- **Kimball recommends using the public cloud in the prototyping phase and then moving to the private cloud as things mature, or the return-on-investment is proven**. Given the capital expenditure in provisioning the servers and tools, this is a very practical suggestion, after considering the aspects of data security and PII (personally identifiable data). In the context of the cloud, the physical data model and ETL may be tailored to the infrastructure available in the cloud. The public cloud may use products such as SnapLogic and Amazon RedShift (or Google SQL and Cloud DataFlow), while the private cloud may have Informatica and Teradata, for instance.

- **Cluster based physical table design**. In a cluster of machines (in the private, or public cloud), **we would partition facts across the Data Layer, and replicate dimensions across the Query Layer**. For very large fact tables, we can partition across the Data Layer by time, geography, or tenant-id in a shared schema approach or other partition elimination (and selective) mechanisms. These fact tables would be **partitioned** across the storage layer. Contrasted with that, the dimension tables usually get **replicated** across the nodes of the cluster to service the Query. Usually, the dimensions get replicated in SSD storage or in-memory caches. This is a performance boosting strategy. Of importance is the key based on which the facts get partitioned.

- **Physical table design strategies for Cloud based DW products.** Cloud based DW products (like Amazon RedShift) force a **partitioning of Facts with distribution keys and fixed sort orders**. HPE Vertica also follows the same approach for projections. In both these systems, the query patterns dictate the physical storage.

- Cloud ETL comes with challenges related to the following points below. **Having a clear architecture, strategy and cost estimates help here.**

  - Initial data population and the transfer/bandwidth costs.

  - Dealing with data characteristics (rapidly changing data, streaming data, unchanging data).

  - Tweaking the performance of cloud data warehouse nodes.

  - Bandwidth issues between nodes, and communication costs.

  - Pay-per-use costs association with Transformations.

  - Comprehensive and unified cloud system administration and data lifecycle management strategy.

- Cloud deployment of a warehouse leads to flexibility in scheduling workloads and having machines mapped to the workload. The Compute and Storage can be separated and provisioned independently. An example is SnowFlake.

www.persistent.com

5.4.3 Big Data

In this section, we can see what best practices we can apply from Dimensional Modeling to the Big Data space.

Big Data has forced a new way to thinking about

- A world extending beyond RDBMS
- Analytics, beyond slice/dice/filtering
- How to eliminate high latency insights.

Dimensional thinking can be applied to Big Data. For example, analysis of a Twitter firehose or an IoT application data stream is valuable when provided with context. The dimensions are what provide the context. Prior Event, Weather, Who (person), What (Condition, Topic), Where (Location), When (Time), are all examples. Dimensionalizing should be automated to keep pace with the scale and velocity of data.

In Big Data environments, as mentioned in section 3.2.2 above, modeling is done after the data is ingested. It is not schema-on-write, but the schema is modeled to suit data access patterns.

The diversity of data sources (usually external or not controlled by the Enterprise IT) means that conforming the data by dimensions (keys, definitions, granularity, attributed) is even more critical to the integration pipeline.

- Generation of durable surrogate keys is applicable to Big Data
- Time variance tracking (via SCDs) is pertinent.
- Both Structured and Unstructured data will be integrated. Attributes of a medical examination fact table will be linked to multimedia data (image, scans) and text (physician annotations).

Changing role of Data Marts and ODSs (Operational Data Store). We have seen how the role of a data mart in any organizations data ecosystem changes over time. Data Marts usually start out as an initial proof-of-concept to gauge the return-on-investment for analytics investments. In some organizations, they supply curated "departmental" data to the EDW (Enterprise Data Warehouse). With the advent of a Big Data System into the data ecosystem, we now see Data Marts as a downstream system from the Hadoop Big Data System. **Data Marts now play the role of a "serving layer" or presentation layer**. In some cases, data marts are being done away with. Also, with the "unlimited" storage provided by Hadoop based Big Data Systems and relief in terms of database licensing costs, there are instances where ODSs are now being retired (especially those that have no operational reports built on them). The data quality checks on the lowest level of transaction granularity are now performed during the ingest phase, or after the ingestion. As discussed in section 6.3.3.2 below, in case of data lakes, the data quality checks are done after the ingestion.

As mentioned in section 3.2.1 above, **the tendency is to perform data transformations in the Hadoop Layer** (no longer is there a need for a separate Staging Databases + Staging DB Servers, or extensive investments in ETL tools, coupled with expensive database licenses).

OLAP style multidimensional analysis directly on big data. One of the use cases for Hadoop was to prepare data for multi-dimensional analysis. Up to recently, data had to be extracted to a traditional downstream data warehouse or OLAP engine (from players such as Microsoft, Oracle, IBM, SAP, etc.). However, Hadoop and its ecosystem tools are now increasingly capable of addressing use cases that go well beyond distributed batch processing, and these now include OLAP style analytics. Indeed, several recent open source multidimensional analysis capabilities that run "on the cluster" are putting pressure on the traditional OLAP engine players.

www.persistent.com

- **Apache Kylin** [28] is an open source analytics engine designed to provide an SQL interface and OLAP style multi-dimensional analysis on Hadoop large datasets. It was originally developed by eBay and is now an Apache project. Kylin builds MOLAP cubes starting from a tabular star schema warehouse model in Hadoop Hive (version 1.5 allows Spark and Kafka sources) using a MapReduce engine that produces a cube stored in HBase. You can then use standard SQL to query the cube as if you were querying the star schema. From the outside, clients can connect through ODBC/JDBC; there is a connection with Tableau where this tool can generate SQL queries automatically from the query generator, as well as with Zeppelin, Excel and Microsoft Power BI.

- **Druid** [29] is another open source option designed for OLAP queries on fast arriving event data; it leverages its own (non-Hadoop) technologies based on a column-oriented storage layout, a distributed, shared-nothing architecture, and an advanced indexing structure. Druid is suitable for real time analysis and has good integration with Kafka (the realtime capability of Kylin is still under development); However, it has its own query API and language, is limited with respect to joins, does not support SQL and does not support integration with BI tools.

- **Apache Lens** [30] is another open source solution for Hadoop. It is a more traditional ROLAP solution based on Hive. A cube model can be built on a star or snowflake Hive model with optional aggregated tables, and can be queried with OLAP cube QL (a logical subset of HiveQL). The system implements aggregate navigation (see section 4.4.1.9): the input query is rewritten taking into account aggregated tables.

- **Atscale** [31] is a commercial solution that turns a Hadoop cluster into scale-out OLAP server. As Apache Lens, it allows to overlay an OLAP model with dimensional hierarchies on top of SQL on Hadoop engines and builds and maintains a mix of virtual BI cube definitions and aggregate tables. It supports any BI tool that can talk SQL or MDX, and works out-of-the-box with the leading SQL-on-Hadoop engines, such as Impala, SparkSQL, or Hive. It can also run on top of Altiscale data cloud, a Hadoop as a service solution (see section 3.2.3), turning it into an OLAP as a service solution.

Query performance, at the cost of (bounded) accuracy. The Kimball data model has emphasized understandability and performance. However, to cope with the growth in data, there is a growing need for returning data in finite and predictable time. We are running into barriers imposed by physics (storage blocks accessed by the query, memory accessed, network round trips made), see [23]. **Sampling or micro-querying is now being explored as a solution to returning results in a finite bounded time.** The cost to pay is accuracy. However, advances in research and newer database products (like BlinkDB [27]) point to a minimal loss of accuracy – Bounded errors and Bounded latency. In these systems, a sample model is also retained as a representative of the full data. In essence, we now deal with 3 data models (the logical, the physical, and the sampled).

5.4.4 NoSQL

With the rise in NoSQL Technologies, the Data Warehousing space has seen many changes. Here we track the Dimension Modeling related practices that we see, and use in the field, trying to map dimension models to the NoSQL models.

www.persistent.com

- **Multidimensional schema to NoSQL - Column Family.** The basis for the logical model in NoSQL systems is unchanged. However, the physical model for column family based systems like Cassandra and HBase will differ. With these systems, usually a single table for the Fact will contain a de-normalized structure. The Dimensions attributes become part of column families of the Fact table. This is the way to achieve join-based filtering for these databases. In addition to being present in the de-normalized structure, the dimensions are also separately retained as tables. Performance of equality driven queries on dimensions and range based queries on dimensions need to be carefully evaluated.

  This approach works well in case of few dimensions. **A NoSQL system is applicable in multiple business domains like network access, perimeter/physical security, and data related to audio-video analysis.**

- **Multidimensional schema to NoSQL - Document Oriented.** In case of document oriented NoSQL databases, new "dimensional attributes" (who, why, what, where, when) get added as JSON elements, to the main "document" fact table. The "document" fact table stores the "how" (many, much, often). We don't see document databases playing the role of a data warehouse system, as often.

5.4.5 Self-Service and Agility

Self Service Dimension Modeling and ETL seem a little distant currently. Some of the difficult problems in this field relate to inferring a relationship, and then cascading maintenance of updates or deletes automatically. Tracking history and propagating changes, or merging keys related to entities (and their associated measures and results) are all difficult to achieve. In section 6.3.4 we show an example of the limitations of today's self-service data preparation tools when it comes to maintaining relationships across datasets (e.g., referential integrity). When data is deleted or updated, the relationships with other datasets typically need to be maintained by hand.

Agile software development methods can be applied to dimensional modeling in the following areas:

- During Sprint Planning. **The Bus Matrix is an effective framework to help come up with the backlog and plan the sprints.** Clusters of the common dimensions (X axis of the bus matrix), or dimensions in order of importance (in terms of priority/criticality of business processes) could be a guiding principle for selecting activities per sprint. Make sure you involve business users. In addition, experience has shown that the first iteration should deliver the maximum number of key dimensions and facts. This ensures that the designs for the data model and BI semantic layer are validated with actual data and reports early, minimizing refactoring later in the project.

- During Development. **Agile BI/DW & ETL development can take place provided the Data Modeler, Data Architect and ETL Architect all work in separate sprints**, with the modeler working at least 1-2 sprints in advance of the ETL Architect. Some of this is because the Modeling activity involves a lot of discussions and meetings, it is not always possible to complete this activity in the first few days of the sprint, and it is better to dedicate a full sprint advantage to modeling. The Data Architect must (in conjunction with the Data Modeler) design the physical schema and carry out some scripting/tests to validate the structures. The ETL Architect can then take over in the following sprint.

- After Deployment:
  - In our experience, the work carried out after deployment largely relates to the performance tuning of the presentation layer to address issues related to the BI layer.
  - Other than the performance, the addition of newer attributes is the other big set of changes, post deployment.
  - Major requirements about historization or changes to the bus matrix can be driven the same way in a series of development sprints.

www.persistent.com

Agile development methodology is now influencing ETL data warehouse development by moving towards Script/API based development at the expense of ETL technical UIs.

- The ETL UIs are good for starting out; however, the inadequacies come to the fore very quickly, when dealing with performance. On the plus side, the reusability and API provide a quick start mechanism, and provide a factory approach for dealing with Data Integration. The UIs serve to hide the details, but over time, force the users (not just power users) to learn the details, and workarounds. Sometimes, the workarounds become the go-to path of usage.

- Agile development methodology is now influencing ETL, with the BigData ETL world moving towards Domain Structured Languages, a.k.a. DSL [36] in a big way. We see emergence of Cascading, Apache Beam (Google Cloud Data Flow), Pig and PySpark. These can be viewed as Software Development Kit (SDK/API) or programming model, for high-level abstractions and logical constructs in a programming language matter — thus simplifying the overall code. Data flows built with DSLs can be visualized graphically and used for monitoring and troubleshooting the flows: there is no inherent contradiction between API based development and visual interfaces.

6 Data Quality

6.1 Brief Summary

A few decades ago data and data management software were not considered to be an asset. Everyone agreed on their importance, but they were not considered concrete goods. Nowadays, this view has changed so much that the exact opposite view is starting to prevail: not only data and software are an important part of an organization's assets, but an increasing number of organizations are using data as a competitive advantage. New disciplines have emerged to treat data or, more generally, information (to include content such as text documents, images and such) as an asset: data governance is a new a corporate-wide discipline involving the management of people, technology, and processes to deliver information-driven projects on time, which will provide to its users the right information at the right time.

Now, how can organizations evaluate their information to see if it is "right" and take corrective actions if it is not? First and foremost, by evaluating its quality. When the quality of data is low, it is either not used (e.g., point of sales leads on prospective customers with incomplete contact information), or leads to incorrect decisions (e.g., cost/benefit analysis on product profitability with inaccurate data), or even tangible money loss (e.g., erroneously lowering the price of an item, or incurring in shipping extra costs because of products won't fit into a lorry as product sizes were wrong).

Data quality refers to the level of quality of data. There are multiple definitions of this term. The seminal work by Tom Redman [2], summarized in [3], initially motivated and shaped the discussion. Redman initially identified 27 distinct dimensions within three general categories: data model, data values, and data representation. Over the years these have been refined to a more manageable number. If the ISO 9000 definition of quality from 2015 is applied, data quality can be defined as the degree to which a set of characteristics of data fulfills requirements for a specific use in areas such as operations, decision making or planning. These generally agreed characteristics are:

www.persistent.com

- **Completeness,** defined as expected comprehensiveness, generally the proportion of missing item values among those expected as mandatory;

- **Validity,** also called conformity, the degree to which data follows the specification of its value domain (data type, size, range, and format).

- **Consistency,** which is the degree to which no conflicting versions of the same data item(s) appear in different places.

- **Accuracy,** the degree to which data correctly describes the real world object or event being described (besides numerical and ordinal data, this includes also typographical errors or misspellings in string data).

- **Timeliness,** also called currency, is the degree to which data represents reality from the required point in time.

- **Integrity,** the degree to which the data captures the semantic business rules governing the components of the data model (for instance, functional dependencies among data items, primary key / foreign key relationships among data items of different data sets, and more generally, business rules governing the values of different records between the same or different data sets).

A typical cause of low quality (a.k.a. dirty), data is the collection of data by end users through data capture forms (data may be mistyped, or entered in the wrong field). Input masks can address some of these problems, but most probably won't correct them all: for instance, there may not be formal standards in the organization on how to enter qualitative attributes such as ordinal rating scales (e.g., very high / high / medium / low / very low), so data may become inaccurate or even inconsistent, i.e., there may be significant variations across time and type of qualified item for this attribute. Another example: information may be deliberately distorted at data entry: for instance, entering a fictional name or identifier in a person data entry screen. It is not uncommon to find that this is caused by a broken business process. Overloaded codes are another curse of an era where saving bits was important; fields with codes or comments used for unofficial or undocumented purposes are yet another. The list is long.

Validation rules may catch some of these errors, and other technology services improving data quality can also help, but will certainly not capture all possible data quality issues. As discussed throughout this chapter, ensuring that data is clean (or at least, clean enough[19]) is the result of data governance, which implies an organizational commitment to a continuous quality improvement process.

The glossary in chapter 9 below serves as a crash introduction to the vocabulary of data quality. In particular, we talk about data cleansing (or cleaning), and it is important to clarify its meaning with respect to a crucial aspect of data quality, namely, to define validity of data, i.e., agreement on definitions of data elements and their shared value domain, to strive for enterprise-wide consistency and accuracy of data elements. This is the role of data stewards, about whom we talked in the previous section. As mentioned in chapter 5, the secret sauce for integrating data from different departments using a dimensional data model is to strive for standardized, conformed dimensions and conformed facts. Lack of conformity of dimensions or facts becomes certainly a data quality problem, under our definition. Data cleansing may help in determining and cleansing dimension attribute values to be shared.

---

[19] According to [3], the optimal level of data maintenance is not to achieve perfect data, but only a level where the costs of the work to clean the data do not exceed savings from the costs inflicted by poor quality data.

6.2 Best Practices

6.2.1 Data quality at project definition stage

At the time that an analytics initiative is being evaluated to become a project, the first recommended best practice is to **perform a readiness check of your organization for the project to have a chance of succeeding.** In [1], p. 16, Kimball convincingly points out that the potential issues with the data needed to support the business motivation for the initiative is the single technical feasibility factor that may be a deal breaker for the readiness of an organization to launch the project (the other two factors being strong business sponsorship and a compelling business reason for the project).

It is important to say that Kimball's strong stance assumes all along that analytics projects rely on a well-defined, well-structured data warehouse (or a data mart, which conceptually is a subset of a warehouse) on which to build business intelligence applications. This is not the case with data lakes, where the primary motivation is not to lose any data related to a business process as it might be important for analysis further down the road. We will further develop this topic in section 6.3.3.2 below.

Readiness to proceed on this data feasibility aspect then translates into readiness of the candidate data sources. Another best practice at this stage is to **perform a quick assessment to disqualify early a candidate data source from the quality point of view** ([1], p. 16). There are several causes for disqualifying data sources: (i) the required data is not yet collected, or (ii) it is collected but not at the level of detail required, or (iii) it has severe data quality problems (data values are incomplete, inaccurate, inconsistent, duplicated, obsolete, etc.).

Depending on the seriousness of the issues, it may be a challenge to construct a data warehouse with the available data sources. If the issues are very serious and many data sources are disqualified, it may be wiser to defer this project until the IT department closes the data feasibility gaps, and consider another business initiative with less data feasibility problems.

When the project is being launched and the core team is lined up, in addition to the classic roles of project manager, data architect, ETL developer, BI developer, etc., a best practice we already mentioned is to **make sure there is a data steward** ([1], p. 35). He/she should work with business experts to address data quality issues and make sure that IT developers understand them if they are to implement the corresponding validation and cleansing rules[20] .

As [1] points out (in p. 125), master data managed by MDM systems are starting to get traction because of their support for transactional systems: they are positioned to be the authoritative source in an enterprise to reconcile different sources for the same attribute, such as customer name or product price. Errors in master data can have significant costs (e.g. an incorrect priced product may imply that money is lost); MDM systems fix this kind of problems. Our experience on the subject (see section 6.3.1) is aligned with this view, and MDM systems are great news for analytics projects: it makes the integration problem much simpler and they solve the integration problem in the source systems that created the problem. **If the organization has a serious customer, vendor or product integration problem, the recommendation is to start lobbying for a master data management (MDM)** system rather than continuously trying to fix the problem in the EDW ETL system.

---

[20] This actually depends on available technology, which could be difficult to use by data stewards, or not –more on this in section 6.3.4 below.

Finally, it is highly recommended to influence management about **establishing a corporate data governance program** ([1], p. 56) to commit to a continuous data quality improvement process that transcends departmental level organizations and works at the enterprise level. Technology is an enabler, but it does not fix quality problems in an organization. In [1], page 381, Kimball proposes an interesting 9-step data governance program template for any organization that wants to address and build data quality as part of its culture (he calls it information governance, but the two terms are basically synonymous). Michael Hammer in his famous reengineering book [19] points to several case studies where improvements in information technology and, in particular, in the quality of data involved in key business processes, was credited as an essential enabler of spectacular gains in productivity in well-known corporations (in his own words: "seemingly small data quality issues are, in reality, important indications of broken business processes").

6.2.2 Data Quality at Requirements Definition Stage

At this stage, it is recommended to have the data steward **have a first "dig into the data"** ([1], pp. 95, 99) to better understand the underlying data sources, starting with the primary data source for the project at hand. It is suggested to talk to the owners of the core operational system of the project, as well as with the database administrator and the data modeler. The goal of this data audit at this early stage is to perform a strategic, light assessment on the data to determine its suitability for inclusion in the data warehouse and provide an early go/no go decision. It is far better to disqualify a data source at this juncture, even if the consequence is a major disappointment for your business sponsor and users, rather than coming to this realization during the ETL development effort.

This exploration journey will be **greatly facilitated by a profiling tool,** rather than hand coding all your queries (e.g., SELECT DISTINCT on a database column). Profiling should continue as requirements are getting extracted.

6.2.3 Data Quality at Design Stage

The data quality related activities at this stage are still driven by a **deeper, tactical profiling effort** of the formal data sources, i.e., those maintained by IT, and the informal sources, coming from the lines of business or which are external to the organization. The first step in this process is to understand all the sources that are candidates for populating the target model; the second, is to evaluate each, and determine each data source ([1], p. 307). The outcome (p. 308) includes the following:

- A basic "Go/ No Go" decision for each data source

- Data quality issues that must be corrected at the source systems before the project can proceed

- Data quality issues that can be corrected in the ETL processing flow after extraction – sort out standardization, validation, cleansing and matching needs.

- Unanticipated business rules, hierarchical structures and foreign key / primary key relationships.

From this analysis, a decision needs to be taken regarding the best source to populate the dimensional model. A criterion for choice in case of two or possible feeds for the data include accessibility and data accuracy, as explained in page 308 of [1].

Data stewards should **try in obtaining and validating optional data**, which systems and data source owners are happy to leave unfilled (p. 321).

www.persistent.com

The integrity constraints that should be added to the target model is discussed in p. 330 (NULL constraints), primary keys and foreign keys constraints (p. 332 – 334). In a nutshell, it is recommended to **declare dimension columns to be NOT NULL** (but not necessarily for fact tables), **use surrogate keys for primary keys in dimensions** (but not necessarily for fact tables) and, if possible, **declare and enforce foreign keys between facts and dimension tables** (except if it becomes too heavy for the time allotted).

6.2.4 Data Quality at Development Stage

At this stage, the key requirements (p. 381) are to **develop a system that can load valid data, correcting invalid data and tagging invalid data it can't correct** (e.g., there may be no default rules for missing data in all cases); and then, **highlighting and explaining** the modifications related to standardizations and rules, as well as the data that was rejected, and why. Quality screens, also called validation rules, are the heart of the data quality architecture. Single column, multiple column and business rules and/or queries testing that integrity constraints are valid are typical examples of validation rules. Profiling the data will help determine the necessary validation rules and the ensuing cleansing and matching rules correcting invalid and/or duplicate data.

ETL tools vary a lot in their built-in support for data cleaning capabilities, but allow the user to specify cleaning functionality via a proprietary rule languages and/or transformations supported by a graphical user interface to build data pipelines. Validation rules generally must be coded with column expression editors and hand-coded functions where they can determine characteristics of incoming data with the help of aggregation functions. Cleansing can be done using a programming library that helps with type conversions, string, arithmetic, scientific functions and the like. Some tools do provide high level transform to cleanse specific domains (e.g., addresses, names, phones). Matching is generally done through joins, string matching functions including fuzzy and wildcard matching and, in some tools, high level transforms for specifying matching policies based on several fields and survivorship rules.

Rejecting invalid data is a topic of much debate. Some tips about what to do with such data appear in p. 383, "responding to quality events", and the overall recommendation is to **accept as much data as possible in the warehouse, tagging fact and dimension data with a special audit dimension**, as illustrated in figure 9-2, p. 385, so that data that is not quite valid can be corrected down the line, and can also be eliminated from query results if desired. Statuses such as normal, anomalous, outlier, and impossible may be used. Also, as we discuss in the big data section below, as a minimum, data cleansing rules should **differentiate numeric, missing fact values between "cannot exist" and "exists but is unknown"**. Ultimately, this is quite dependent on the type of data and on the type of analysis that will be performed once the data is accepted in the warehouse. As we will see in section 6.3.3.1 below, what to reject and even what type of data quality is necessary in big data repositories such as data lakes is more of an open question than in a traditional data warehouse, as use cases are more varied, queries to be asked to the data are figured out after it is ingested, and analytics workloads tend to be more advanced.

As [1], p. 383 suggests, it is recommended to **persist audit data for each ETL pipeline run, along with detailed errors in a specific error event schema**. Not only it will provide the necessary explanations for what happened during an ETL pipeline process, it will also be the **basis for making data quality auditable and being monitored** over time. Make sure it has detailed structures to persist errors at a record level, and that audit data containing measures of data quality for the pipelines being run, along with checksums tying back to data sources to cross-check the integrity of the data ([1], pp. 448 and 453). The tie back to the source data systems is important because it contributes to establish the credibility of the warehouse.

A data target with duplicate data produces inaccurate results and is frequently inconsistent; ETL tools can help with this operation (see chapter 9 and [1], p. 383). There is very seldom a universal column that makes the matching operation easy; the only clues are generally on columns carrying similar data. Before pair-wise comparisons are performed among all records in the dataset (an operation which has O(n2) complexity), some systems allow to specify a discriminating column or set of columns (e.g., such as zipCode for matching vendor data), and only records with the same values on that or those fields will be compared (this is called blocking, and in big data systems blocking is receiving a lot of attention –see section 6.3.3.1 below). Then, comparisons using fuzzy matches are frequent on some data elements such as names or addresses, as similar but different spellings or conventions are used; other, such as phone numbers, need exact matches. Survivorship is the process of combining a set of matching records into a unified, single record.

6.2.5 Data Quality at Deployment Stage

We concentrate in this section on **data quality assurance testing** ([1], p. 546-547) among the various others to be performed at the alpha and beta testing of deployment stage, which is basically a process that makes sure that the contents that are loaded (or to be loaded) in the data warehouse are correct. The audit information captured by the flows may tell us we have the right number of rows, and referential integrity checking may tell us everything matches up. But correctness can only be verified by running a set of queries or reports from the source system(s), running the corresponding queries or reports from the data warehouse, and comparing results.

When there is a single source, determining the corresponding queries and reports is much easier than when integrating data from several sources. When there are several sources and they are large, the difficult part is determining sufficient coverage through queries and in determining the corresponding warehouse queries, including complex calculations needed by the downstream BI applications. **Engage business users and the source systems owners to help create a solid set of data checks**. When there are differences that can't easily be explained (existing reports may have long been in error) document them and get help from an internal audit team.

During this last phase, data assurance testing may be done at three levels:

(i)   At dimension or fact initial load test level, with a representative subset of real data

(ii)   At the primary test dataset used for end-to-end system testing (usually a small dataset), and

(iii) At the real, live data level, both the initial load and the incremental data loads, as mentioned in section 5.3.6 above.

Make sure that automation is done at least for the two first levels, saving a log of the results of every run and comparing test results against known correct results. It is also recommended to divide automated tests into a basic acceptance test, to run very frequently to test that builds are correct, a regression test suite to make sure that functional changes on a new build does not break anything, and a complete test suite covering all branches of the ETL processing.

Go-live is described in section 5.3.6 above. After the system goes live it is possible that some data quality errors might slip through the net, even if you thought data quality assurance had weeded them all out.

As you define your warehouse support strategy, it is wise to **assign resources for data reconciliation** shortly after deployment. These analysts require a broad set of skills: they will need access to the data warehouse, to the source systems or production reports, and to have a solid understanding of the data transformation process.

6.3 Enhancements to the reference publication

6.3.1   Our own experience

We begin this section with our experience on integrating datasets which will become dimensions in the target dimensional schema.

- **Matching party data from different sources is fundamental for data integration**. By "party data" we mean people names, organization names, addresses, e-mail and phones –the list is not exhaustive. Customer, employee and vendor data fall in this category. It is very frequently the case that party data comes from different data sources, both within and outside the organization (some of which is known to be of dubious quality, such as retail point of sale data [18]) , and need to be merged in the warehouse. There is no common join key that makes this operation easy. So what is generally done is to parse party data in their component elements, correct mistakes and fill in missing values (for instance, correct mistyped city names, fill in zip codes if there are enough address data elements available), and then perform matching on these parsed-out component elements.

- **The above discussion on parsing and cleansing leads us to the following observation.** Domain-oriented validation and cleansing rules generally work better than generic ones. This has been observed as well in a recently published comparative experiment [17]. Party data elements (e.g., City, a component of address data) is an example of such a domain. But this is also true of other domains as well. The way this generally works is through identification of columns as belonging to these domains, and through access to dictionaries or knowledge bases that validate the values in the column and/or provide the way to correct frequent mistakes (we have also seen recently a push towards using machine learning techniques –more on this below). The reason why so much effort has been put in detecting, cleansing and matching party data (including for instance country and language-specific variations which are frequent in industrial-strength data cleansing tools) is because of our first observation above.

- **Managed master data is great news for analytics projects.** In our experience, organizations as a whole are finally starting to get interested in data quality. An area that gathered a lot of traction recently is master data. As mentioned above, the main reason for this is because transactional systems are based on master data, and MDM provides them with high quality master data (as explained in chapter 9, MDM systems are consumers of data quality software).   Master data managed by MDM systems are great news for analytics projects for two reasons.

    1. A lot of work in the cleansing and conforming stage of the ETL system centers on creating exactly this single, conformed version of customers or products, so this work is avoided when the master data repository of the MDM system is sourced into the data warehouse.

    2. MDM systems allow solving the quality problem at the source, rather than continuously keep fixing the same problems over and over again when moving data over to a data warehouse.

www.persistent.com

Other aspects on our experience with data quality tools include the following:

- **On profiling tools:** Most profiling tools have user interfaces where users can manually specify columns to inspect. In large enterprises, the data to profile is large, and manual profiling would be very impractical. For relationship profiling on columns of two tables, tools traditionally allow for pairwise column dependency profiling. Architects need to look for solutions that allow to **automate profiling of across all columns and on entire schemas**. We have also found that it's best to look for a data profiling solution that enables you to construct data validation and correction routines directly from the profiling reports. This will help you combine data inspection and correction phases, which will **streamline your overall data quality development process**. We will look again at this aspect in self-service data preparation systems below, as business users are those who would benefit more of streamlining.

- **Data mining, statistics and machine learning technology is being increasingly applied to address data quality problems**. We devote a subsection (see 6.3.3.3 below) of the big data quality section on this subject, even though the subject is not strictly specific to big data.

- **Data governance aspects such as measuring and monitoring quality is often neglected**. Customers want to easily drill and navigate to specific areas of the warehouse where quality is being monitored and find out about quality measures. Our experience is that this area is often overlooked and, when addressed, it is done in a separate product, which brings usability and productivity problems. On the other hand, it is true that measuring and monitoring quality is the natural role of a data steward, who is a less technical user, and this explains why data quality governance may be built on a separate product offer[21]. **Data governance should also include review steps for automatic data quality actions**. People do not always trust changes proposed by automatic data quality actions in production systems. This is a healthy attitude, as experience today shows that results of tools in real world datasets are far from perfect. This is another of the duties of data stewards.

- If quality of streaming data and "static" data needs to occur in the project, you should **favor tools or use a streaming-oriented architecture that allows operating on both types of data**. This favors reuse of existing pipelines on either type of data, and avoids having to develop explicit batch-oriented "delta" loads, by applying the same changes done through an initial load to data being streamed in as it changes in the sources. In our experience, attention needs to be paid to the degree to which these architectures are robust with respect to schema changes at the data source. Indeed, we have seen that upon source schema changes, pipelines frequently break.

- Last, I'd like to comment on the relationship of between integrity rule violation and data quality. In theory, developers could rely on constraints on the target DBMS set up by the project's database designer (e.g., key constraints, foreign key / primary key constraints, NOT NULL, and the like). In practice, **developers have to go through the trouble of developing validation rules on the ETL tool**, which sounds like duplicate work –even though these days ELT processing mode has become mainstream, as mentioned above in section 3. The reason is that database engines allow to catch constraint violations with proprietary SQL syntax (even though the SQL-99 standard has comprehensive support for integrity constraints), and ETL engines don't go into the trouble of generating specific code for each engine[22].

---

[21] This separate product may be a self-service data preparation product, which we will comment about in section 6.3.4 below.
[22] Personally, after having working first in the database field and later in data integration, it came to me as a surprise that I never saw an ETL tool allow developers to provide treatment for rows violating a DBMS constraint.

6.3.2   Cloud Deployments

As mentioned in section 3.1.2 above, deployment to the cloud does not change the fundamental data management processes: this includes of course data quality and data governance, which still need to be applied to information that is warehoused in the cloud.

Data quality transforms on cloud integration tools are conceptually the same as their on-premises counterparts. However, be attentive to the fact that, even within the same vendor, cloud tools do not always offer the same capabilities as on premise tools. One area is transformations; another is workflow management. As an example, on the former, per our experience, this is particularly true for cleansing and matching transforms: cloud integration tools are generally simpler. This simplicity is sometimes beneficial in terms of ease of use, as cloud tools are more modern and have benefited from the on premise tools maturing process, but some tools may lack functionality.

6.3.2.1   Cloud Application Integration

As mentioned in section 3.1.3 above, IT departments are now expecting that hybrid system landscapes connecting cloud and on premise applications will become the standard way in which they will manage the organization's IT assets in the future. We also mentioned that our experience is that the best architecture concept to handle this list of requirements is a data governance and integration platform avoiding point-to-point interfaces between applications.

To provide governance and high data quality, this platform converts data from applications to a global model, reducing the O(n2) possible mappings between applications to 2N mappings. It must store reference data comprised of (i) basic identities of master data elements of the connected systems/applications, (ii) reference metadata as for capturing domain mappings, i.e., the relationships between valid values from source applications to the conformed dimension and fact values, and (iii) retrieve the metadata models of the connected systems and their subsequent transformation flows (which are to be modeled through the platform). It may also use enrichment services (see next section) to complete data elements.

6.3.2.2 Data quality services

Cloud architectures allow developers to take advantage of real-time web services so that data transformations for improving data quality can be performed as a service. Common as-a-service use cases include:

1. Several vendors now expose public cloud services to perform data validation for customer, vendor or employee master data, for example, address cleansing and enhancing (correcting or enriching empty data elements, for instance unknown postal codes).

2. There are vendors that allow to enrich specific types of data: Axciom for individual customer data and Dun & Bradstreet dataset for organization data.

3. Geocoding and reverse geocoding are also enrichment services available from vendors.

4. Another private cloud or on premise well known use case is for organizations to expose a service to perform matching against their master data to ensure that they are preventing duplicates.

These services can be bundled with cloud integration products, cloud based platforms, or attained from REST client apps through simple URLs. This is particularly useful for a wide number of applications that want to provide real-time data quality at record point of entry, locate one or several places or addresses on a map, turn geographic coordinates into a place name or address, or consolidate master data through batch cleansing and matching services.

6.3.3   Big Data

In this section, we first present data quality overall role and techniques in big data repositories and then, more specifically, in data lakes (which are big data repositories with discovery and governance tools for post-ingestion data quality). We then devote time to present a bird's-eye view of machine learning techniques which are becoming popular in big data environments (even though some of these techniques predate the era of big data) to assess quality and help with the data improvement process.

6.3.3.1   How much data quality is needed for big data?

Traditional data warehouses treated dirty data as something to be avoided through purging, cleansing and / or reconciling, record by record, on the premise that data warehouses need strict data quality standards and avoid double counting. It is by no means clear that this is still the case in big data repositories such as Hadoop. For content such as log files, it will neither be practical nor worthwhile to impose a structural standard, given the high variety of such types of data structures and sources. Volume and velocity, the other 2 'V's in big data, also make this proposition impractical.

As the reader would expect, the value of high quality big data is driven by the nature of the applications or the type of analytics that might be running on top of the big data repository. In what follows we examine some analytical applications that may use big data repository content.

- For transactional applications that carry regulatory compliance requirements and/or require strict audit trails, data quality is as important as in a data warehouse. These applications generally refer to transaction data related to key entities like customers or products, and may leverage traditional data quality technology as long as it scales to meet the needs of massive volume. Attention needs to be paid to the following aspects:

    - Given the massive amounts of data involved, data movement into a middleware ETL engine should be avoided. Consider ETL engines that execute within the big data environment.

    - Consider the experience gathered in optimizing costly data quality algorithms, for instance blocking methods in record matching and de-duplicating. We talked about blocking in section 6.3.1 above, and we have seen recently a lot of interest in adapting blocking methods to Hadoop distributed architectures and to perform schema agnostic blocking (i.e., which don't rely on a common set of attributes, which is frequently the case in semi-structured or unstructured data).

    - Prefer tools that provide high level GUIs. Programming transforms in lower level interfaces such as MapReduce or Pig take longer to develop, are hard to read and reuse, and are less extensible.

www.persistent.com

- Hadoop HDFS does not support updates. Reusing data quality approaches that rely on update in place simply won't work. One approach is to apply data cleansing to data that is flowing into or out of file systems that don't support updates. Applying basic quality checks on the incoming data is recommended as typically incremental data will be much less in size as compared to the full data set.

- Exception to the above statement include applications that may work on transaction data but are detecting fraud, or any other kind of risk. Outlier data or unusual transactions should not be cleansed away. In this case, the adequate treatment is to determine whether the outlier data correspond in fact to anomalies –more on this on the section 6.3.3.3 below.

- An application may still deal with transactional data, but the high variability of the data it needs to integrate may imply that a domain can no longer be simply modeled through integrity constraints, as there may be data that does not comply with the constraints but are actually legitimate usable data. In this case, cleaning the supposedly faulty data would in fact induce distortion in the data. Several recent proposals use machine learning to learn constraints from the data, as explained in the section 6.3.3.3 below.

- For web applications relying on click-streaming data or for ad placement applications, if the data is not 100% correct this might not be crucial, especially if the application or analysis is trying to convey a big picture of the situation. For example, when looking for patterns such as at which point users leave a site, or which path is more likely to result in a purchase, on a massive amount of clickstream data, outliers most likely won't not impact the overall conclusion. In this case, pattern chasing is more of an analytics process, as opposed to a data quality process[23].

- Sensor data is not generated by humans, but it could still be missing, uncertain or invalid. Applications such as environmental monitoring and need to have indicators on the quality of sensor data. Profiling and/or validation on this data to detect deviations from the norm are important as they may indicate a problem with sensor infrastructure, misplacement, climate conditions, etc. This is another area where machine learning technology can help.

- If data mining analysis is likely to be performed down the line on numerical data, missing values on data elements that should not be missing are often problematic. As a minimum, the data quality processing should distinguish between "cannot exist" and "exists but is unknown" and tag the fact data accordingly. This is a familiar problem in the medical and social sciences, where subjects responding to a questionnaire may be unwilling or unable to respond to some items, or may fail to complete sections of the questionnaire due to lack of time or interest. For this reason, it may be a good idea to leave the statistical estimation, down the line, to a special purpose data mining application for missing values of the kind "exists but is unknown", as opposed to rejecting rows with these missing values, which would introduce distortion in the data. More on this in section 6.3.3.3 below.

- Applications on social media data will bring data quality considerations at two levels: entity matching and data quality related to text data.

  - Entity matching, a.k.a. entity resolution, is important when datasets built with data from social sites may contain multiple different references to the same underlying entity (person). This is like the problem of data matching and consolidation (see chapter 9), as entity resolution makes use of attribute similarities to identify potential duplicates, but also may make use of the social context, or "who's connected to who," which can provide useful information for qualifying or disqualifying matches.

---

[23] Data quality may be centered on other aspect, namely, relevance: it's about choosing from the population generating the click stream only those that are relevant to your analysis (e.g., select only actual customers accessing your site).

- Understanding text data through NLP (Natural Language Processing) is a broad area that is receiving renewed attention: for instance, sentiment analysis algorithms are very popular these days, as online opinion has turned into a kind of virtual currency for businesses looking to market their products and manage their reputations. Data quality challenges related to text include identifying misspelled words, managing synonym lists, identifying abbreviations, taking into account industry-specific terminology (e.g., the word "stock" may mean very different things depending on the industry), leveraging context to filter out noise in textual data (e.g., relating to a company name: differentiating among Amazon the company, Amazon the river and Amazon the female warrior) and to attach correct meaning.

  Interestingly, sometimes these two topics are connected: a frequent problem is to identify whether a reference in a social site (e.g., a Facebook identifier of a person) corresponds to an entity (e.g., a customer) known in an organization's big data repository or CRM database. In this case, extracting information from the text can help entity matching: for instance, a product name in a posting talking about the author's experience with the product is a candidate for extraction to help match the author to a customer if there is transactional information in the repository reflecting the purchase.

- Applications dealing with continuous streams of data, e.g., measurements, or real-time customer engagement systems, should check this incoming data for quality in real time. If rate changes are very high, data quality techniques such as outlier detection and record matching won't work and then users may obtain outdated and invalid information. On the first topic, data errors such as switching from packets/second to bytes/second in a measurement feed may cause significant changes in distributions in the data being streamed in. Standard outlier techniques must be made considerably more flexible to account for the variability in data feeds during normal operation, so as to avoid raising unnecessary alerts. The system reported in [21] builds simple statistical models over the most recently seen data (identified by a sliding window) to predict future trends and identify outliers as significant deviation from the predictions. To ensure statistical robustness, these models are built over time-interval aggregated data rather than point-wise data. On the topic of record matching, when data sources are continuously evolving at speed, applying record matching from scratch for each update becomes unaffordable. In [22] incremental clustering techniques have been proposed to deal with change streams including not only inserting a record, but also deleting or changing an existing record.

As with warehousing, profiling tools are very important to understand data. However, with big data, it is not always straightforward to understand its content, as it is stored in raw form: for instance, the numbers you want to profile may have to be obtained after some extraction transform. Data volume may also become an issue, so methods to perform sampling and summarizing may have to be put into play. And then it becomes crucial to discover relationships among datasets, as in big data repositories there are frequently no schemas, as well as correlations among attributes. Data scientists may then decide to select attributes (features, in their parlance) for their machine learning tasks (e.g., classification or anomaly detection) where they try to come up with some hypothesis that sheds light in identifying a real business opportunity. Profiling, again, is crucial to this selection activity.

www.persistent.com

A transversal topic is when to worry about improving data quality in the analytical lifecycle of a big data repository. When this comes after datasets have been placed in the repository, after users have profiled a subset of the data, and after a business opportunity has been detected that may use a well-defined subset of the repository, then we are talking about data lakes, and we discuss this matter in our next section. At the other end of the spectrum, when data is cleansed immediately after it has been placed in the repository (or not stored in raw format, but in a format ready to be processed), it is generally the case that the questions to be asked to the data are known, and the data is made to comply to a structured schema managed both at the logical and physical level for optimal performance of these questions/queries. We are in this case talking about a big data repository that behaves very much like a data warehouse and for which the best practices from section 6.2 apply.

6.3.3.2 Data quality in a Data Lake

Data lakes were introduced in section 3.2.1 above. A data lake is not just a big data repository such as one built with Hadoop HDFS: it has added capabilities such as dataset discovery, data quality, security and data lineage to deal with data at scale. In a data lake, ALL data of ANY type, regardless of quality, that is related to a business process is stored in raw form: it is not directly consumable. Data modeling, data cleansing and integration of data from several sources, which is needed to make the data consumable, if and when it occurs, is done late, i.e., when the right business opportunity is identified, and the questions that the analysis will be driving towards become clear.

Now, when this does occur, the quality of the data becomes a central concern: in order to get any value out of the data, consumers need to make sure that the data has been checked and processed to improve its quality, so that consumers can have any confidence in the accuracy of the analytics results. Clean data from these (henceforth, curated) datasets is then made to conform the data to one (or several) structured schemas in preparation for the anticipated analysis questions. Note that this also means that, if data quality processes are not in place in the data lake, consumers might inadvertently consume bad quality data from the lake, and get wrong insights.

This motivates the following choices:

- The liberal choice: data consumers should be able to tell by themselves the quality of the datasets they discover before they explore them (through profiling or inspecting quality metadata –see below).

- The cautious choice: business analysts should only be able to discover curated datasets (those already processed for data quality). Data scientists are technical enough to assess and improve the quality of datasets.

- The traditional choice: the platform for most business users would not be the data lake, but a data mart or a data warehouse that is fed with data from the lake. The lake in this case was used by data scientists and some power business analysts to identify some new insights on a collection of related datasets. The data mart can be populated with curated datasets from the lake; or alternatively, the data quality processes are applied by ETL engines to selected, non-curated datasets feeding the data mart. This choice may be motivated by the experience of an organization with a given ETL tool, or BI tool that may work only with the data warehouse.

www.persistent.com

All datasets in the lake should have an associated "**quality metadata" tag**, and set by default to an uncertain value. The data custodian (and eventually the data scientists) can assess the content and the quality of the data by using profiling tools or machine learning algorithms, and adjust the value of the quality tag accordingly. He/she can then meet with data producers to agree on standardization of data elements, as well as the minimal business validation rules that need to be put in place for the data to be deemed consumable.

Once this is done, depending on the nature of the application (see the above section), the data custodian can define the appropriate data quality processes. Data quality for applications working with transactional data in the lake would be as follows: (i) the custodian or the data scientist defines transformation flows needed to cleanse the data, directly within the lake, for it to meet agreed data standardizations and the necessary business rules, (ii) s/he would then define the validation and the cleansing rules, as well as (iii) the data matching and consolidation rules to remove duplicates and improve data accuracy in the lake. But then again, as mentioned in the previous section, clickstream analysis applications, fraud detection applications or applications based on sensor or social data may call for different data quality processes.

As for the selection of features for a machine learning task, custodians or data scientists preparing the data may be surprised of the pervasiveness of profiling tasks during the data cleansing stage while working on big data. This is more the case than in traditional data cleansing for populating data warehouses, perhaps because the availability of all the data in the lake prompts for more interactive workloads. Also, these days, data preparation on big data sets including data quality is becoming possible by less technically skilled people, for instance business analysts: we will look at this topic in section 6.3.4 below.

As datasets become curated, the value of the quality tags can be raised. Curated datasets may get permanently refreshed from their original source, so it is appropriate to apply their corresponding validation rules to these datasets, monitoring the results of validation rules over time, and adjusting the quality metadata tags accordingly.

6.3.3.3 Data mining & machine learning techniques relevant to quality on big data

Besides being used to learn to accomplish a wide variety of business problems such as image recognition, product recommendation, and online advertising, machine learning allows the use of very raw forms of data in order to build higher-level understanding of data automatically, which is essential for improving the quality of data. The following is a very quick summary, not meant to be exhaustive neither, of the areas where machine learning has been applied successfully to such end.

Data dependency profiling. Learning dependencies among table columns directly from the data, such as functional and inclusion dependencies among columns [11], as well as correlations and soft functional dependencies is a popular aspect when profiling the data. The general complexity of the problem is O(2m) for m columns, but algorithms have been maturing for over 20 years now, and new algorithms [12] can realistically be applied to datasets with a few million rows and tens of attributes.

Missing values. One important aspect of data quality is the proportion of missing data values. Many data mining algorithms handle missing data in a rather ad-hoc way, or simply ignore the problem. Until relatively recently, the only methods widely available for analyzing incomplete data focused on removing the missing values, either by ignoring subjects with incomplete information or by substituting plausible values (e.g. mean values) for the missing items.

These ad hoc methods, though simple to implement, have serious drawbacks: deletion of the offending records may bias the results if the subjects who provide complete data are unrepresentative of the entire sample, and simple mean substitution may seriously dampen relationships among variables (columns). A robust method, multiple imputation (MI) [20], has gained traction among statisticians. In this method, each missing value is replaced by a set of m > 1 plausible values drawn from their predictive distribution. The variation among the m imputations reflects the uncertainty with which the missing values can be predicted from the observed ones. After performing MI there are m apparently complete datasets, each of which can be analyzed by complete-data methods. After performing identical analyses on each of the m datasets, the results (estimates and standard errors) are combined using simple rules to produce overall estimates and standard errors that reflect missing-data uncertainty. It turns out that in many applications, just 3–5 imputations are sufficient to obtain excellent results. General-purpose MI software for incomplete multivariate data is now available[24] .

From integrity constraints to unsupervised anomaly detection. In traditional ("small data") systems, integrity constraints played a major role in encapsulating the knowledge of a given domain. To a certain degree, this is a successful approach for ensuring quality of data in the domain when the latter is well understood and is static. In the big data realm, however, big variety introduces so much variability in the data that this approach no longer works well: it may be the case that subsets of seemingly noncompliant data are actually legitimate usable data: they are anomalies, but may be acceptable ones.

Recent investigations have reported using machine learning to detect these anomalies, i.e., these integrity constraints violations, from the data. In a first pass, traditional constraints such as functional and inclusion dependencies are declared (or they can also be learned from the data, as mentioned above), and in a second pass those violations are re-examined through special unsupervised techniques [14], [15]. A human domain expert can then look at candidate explanations for these supposed glitches and decide to incorporate them as new constraints, or decide to cleanse the data away. For instance, in [15], an HR database example is given where, upon a violation of the unicity constraint of phone numbers among employees, the learner reveals that the faulty records correspond to employees with "new hire" status and have been given the same phone number as their supervisor. After analysis, the HR user ends up accepting this revised constraint (e.g., because s/he knowns that "new hire" is a transient status), which becomes a conditional functional dependency.

Supervised anomaly detection. Assessment of the quality of sensor data is an example of domain where unsupervised outlier detection methods as the ones described above don't work in general. Indeed, the quality of sensor data is indicated by discrete quality flags (normally assigned by domain experts) that indicate the level of uncertainty associated with a sensor reading. Depending on the problem under consideration, the level of uncertainty is different. Supervised classification is thus a feasible alternative, even though it has the problem that data of dubious quality exists in a representative set of labelled data with very small frequency. Proposals such as [13] solve this problem through cluster oriented sampling and training multiple classifiers to improve the overall classification accuracy.

Matching / entity resolution. Traditional research and practice from the database field emphasizes relatively simple and fast duplicate detection techniques that can be applied to databases with millions of records. Such techniques typically rely on domain knowledge (people names, addresses, organization names) or on generic distance metrics[25] to match records, and emphasize efficiency over effectiveness. There is another set of techniques coming from research in AI and statistics that aims to develop more sophisticated matching techniques that rely on probabilistic models. The development of new classification techniques in the machine learning community prompted the development of supervised learning systems for entity matching. However, this requires a large number of training examples. While it is easy to create a large number of training pairs that are either clearly non-duplicates or clearly duplicates, it is very difficult to generate ambiguous cases that would help create a highly accurate classifier. Based on this observation, some duplicate detection systems used active learning[26] techniques [16] to automatically locate such ambiguous pairs.

Understanding meaning in text and images. There exists by now a large number of ML techniques allowing to determine the semantics and meaning of textual data, as well as images and video. Leveraging these tools and techniques are key to deriving the value from unstructured big data for a number of different tasks (text or image classification, entity extraction from text or images, degree of correlation of textual or image content, etc.). Each of these tasks have techniques that are preferred by experts in the field: Naïve Bayes for text classification, Neural Networks or SVM for image classification, even though experts warn you about making hard rules such as these ones.

6.3.4   Self-Service Tools

As already mentioned in section 3.3, self-service data preparation tools provide many capabilities that go beyond those of most data discovery tools for discovering, combining data and improving its quality. Their main strong points related to data quality improvement are the following:

- Intelligent profiling powered by data mining and machine learning algorithms that visually highlight the structure, distribution, anomalies and repetitive patterns in data.

- Data driven, interactive user experiences that lower the technical skill barrier for most business users. These systems provide automatic suggestions for cleansing, enriching and matching duplicate records and its associated survivorship procedure[27]. They show the results of these operations interactively through dashboards and drill-down capabilities, and allow for undoing/redoing operations interactively as well, to make it easier to tune the parameters controlling data quality operations. This is a real change from the technical user experience where, to run for instance matching and survivorship, users need to define configuration rules with little or no feedback via example data, define and run jobs that execute asynchronously, and their results need to be manually inspected through data browsing or SQL query execution.

---

[25] Rule-based approaches can be considered as distance-based techniques, where the distance of two records is either 0 or 1.
[26] Unlike an "ordinary" learner that is trained using a static training set, an "active" learner actively picks subsets of instances from unlabeled data, which, when labeled, will provide the highest information gain to the learner.
[27] For instance, the tool may suggest to run deduplication on datasets where it has not been run. When deduplication is selected, it incorporates in its workflow a suggestion to perform cleansing if it has not been done, as it makes duplicates easier to eliminate. Also, survivorship rules take into account the existence of fields in the datasets to suggest frequent rules that may use those fields: take the most recent record in match set, or take the record from source A as opposed to source B.

- Some tools are delving into data quality governance, allowing for users with a data steward profile to define rules to define validity for data elements on data domains, cleansing invalid data, and monitoring data quality on selected datasets over time. More on this below.

- Some tools allow for IT to enforce governance from two points of view:

  1. The system resources being used, imposing limits on sizes and time spans of user-generated content (transformations and resulting data model and instances), and

  2. Promotion processes for business user-generated content to a pilot phase and then to production, each phase providing more governance and quality checks. Importantly, the tool may allow the IT architect to retrieve the transformations done by the business user in the technical user interface he needs to complete the job: for instance, to improve performance on large volumes, the IT user needs a more technical, traditional interface.

This space is very young, and the state of the art of the existing products is being challenged (and rapidly making progress) in a number of areas:

The first area is in **Data steward input**. Customers are asking for explicit approval steps for data steward users which may know better the quality of their data than the tools in some cases. Indeed, the cleansing done, the matches found and the survivorship performed by the system is, in general, suboptimal[28].

Let's illustrate these points with a customer deduplication example. A match workflow may suggest the user to run cleansing / standardization. Once names and addresses have been cleaned and put into standardized formats, matching will detect that what appeared to be two customers at first, it may turn out to be one. Perhaps one entry has a PO box address and the other has a street address, but the rest of the data indicates that it may be the same customer. The system might suggest that the two records are in fact duplicates, but this may turn out not to be the case. And even if a data steward user does approve the suggestion, at the survivorship phase, it is not clear which address to pick, and the default survivorship rule might not be the right choice. The tool should let the data steward review and ultimately decide about duplicate set suggestions and about surviving address values.

A second area is **relationships across datasets** (e.g., referential integrity). Simply put, when data is deleted or updated, the relationships with other datasets typically need to be maintained by hand.

To illustrate this point, let's continue our example and assume that our user wants to combine this clean and de-duplicated customer dataset with a sales dataset, which will very likely have entries for the two merged customers using the two previous different keys. After the match, one of the keys is logically invalid: all sales records that exist with the invalid key would have to be corrected to maintain referential integrity and/or to get correct results, and this, in today's systems, has to be done by hand. Otherwise, several bad things may happen: (a) if there is referential integrity enforced by the DBMS among the two tables, a deletion of the record with the invalid key may not go through (and our user might not understand why the deduplication operation fails); (b) if it is a soft delete, or (c) if there is no referential integrity enforcement among these two datasets, then sales from the customer record with the invalidated key will not be taken into account, and results will be wrong.

---

[28] Recent evidence on this aspect is reported in [17]: even specialized, domain specific tools achieve precision (percentage of correct error detections) and recall (percentage of real errors detected) numbers in the 60 – 70% range on average.

As the reader will have understood by now, self-service systems don't have the notion of a data model capturing relationships among several datasets is being built: they work a dataset at a time. They have yet to capture more model semantics, such as dimensional models, or at least foreign key/ primary key relationships among datasets. ETLs can be programmed to produce either one of these through technicalities such as surrogate key generation, automated mapping of source keys to surrogates, and replacement of (both the valid and the invalid) source keys with the single surrogate for matched records, in all fact tables linked to the customer table. Needless to say, this remains out of the reach for most business users: only technically savvy users would do this right (and on a more technical interface). Simpler manual solutions could be proposed through specific workflows, such as replacing the invalid keys from non-surviving records by those from surviving records in other datasets as indicated by the user, but this goes beyond the state of the art of today's tools.

Third, which is related to the previous example, users could benefit greatly from **two-way content authoring between IT and business users**. This means that not only IT should be able to take content authored by business users to operationalize it (rather than redoing everything the business user did from scratch), for example, to deal with data modeling aspects as the previous example shows, or with incremental updates to the original datasets, but the converse should be true as well: when the content is in pilot phase or in production, and when functional requirements impacting the former data preparation change, when new data elements in the sources become available, or when new datasets could help in improving insights, the business users would like to work on their content again (rather than redoing everything they and the IT user did from scratch).

Finally, a fourth area is **support for a broader set of domains**, as well as in **support for user-defined domains** (a.k.a., business types), i.e., the ability for advanced business users and data stewards to define validation, cleansing and de-duplication rules for their own data domains in an easy and data-driven manner. Self-service data preparation systems have here an opportunity to dramatically increase the usefulness of their systems with the introduction of these domains. To mention a few:

I) The ability to involve profiling during the definition of a data domain would streamline the development of the validation, cleansing and matching rules of the domain. We already commented on this in section 6.3.1, but we believe that it is in self-service systems where this functionality is the most cost-effective. Profiling a column would immediately tell the usual patterns and/or frequent erroneous values of a domain under definition, and enables the user to construct data validation, correction and matching rules directly from these patterns for the domain.

ii) The ability to automatically detect columns as being of a certain domain (which is a problem that is being framed as a machine learning classification problem), will improve the automated suggestion capabilities and subsequent execution of data validation and cleansing of data in these columns;

iii) The ability to use domains in keyword search will improve the discovering power of the system to find datasets containing instances of these domains;

iv) The ability to use columns assigned to a domain on operations such as joins may make these operations much more robust, as the automated application of validation/cleansing operations on domain data will produce more and better quality join results. A simple, pervasive example is on date/time data, which may appear in different formats on two dataset columns and not be directly joinable. The application of the right cleansing / standardization rules will convert the data into joinable values. Again, the system may make smart suggestions to help the user given the data instances at hand.

www.persistent.com

7 Non-functional aspects

7.1 Brief Summary

Performance is critical to the functioning of the business at ease with speed and agility. The two main measures of performance are

- Response time of queries to the DW/BI system, and

- Throughput, i.e., a count of the number of queries the DW/BI system can process within a defined time interval.

A related but **similar** topic that we will be addressing under this section is **scalability**, which corresponds to the ability to overcome performance limits by adding resources.

Similarly, Security is a big, esoteric and complex topic which, if not acted upon, can result in range of potential problems. The wide variety of data speeding through the enterprise and moving into and out of the data warehouse requires unprecedented levels of protection. An end-to-end security strategy is required for the data warehouse environment. A data warehouse environment consists of much more than just a database. The entire environment ranges from components extracting data from operational systems, moving this data to the data warehouse, distributing the data to other analytic platforms, and finally, distributing it to the end business user. In today's highly distributed, complex landscapes, the environment spans multiple servers, applications and systems.

This chapter focuses on key sub-areas as listed below:

I. Performance and scalability of various components in the DW/BI (space as well as speed); Sizing and production deployment

II. Security – Security aspects of the DW/BI system; Auditing, Error logging and data life cycle.

7.2 Best Practices

7.2.1 Performance at Requirements and planning stage

- Interview the business process owners to get an understanding of the business requirements ([1] page 72, 83-86) and **gather detailed information from the performance point of view on each requirement.** Some of the questions could be: how many users will be using the system, what is the mix of users based on workload, what are the data query patterns, how would the system be used by these users (on Web or Smart Phones/Tablets), what is the current data volume and anticipated data growth in next 6-months or 2 years, what are the SLAs anticipated for end-to-end data movement vs SLA required at each component level, what kind of hardware is planned in the production, will the hardware be shared across departments, will it be deployed on cloud and if yes, are there any budget constraints, is time for performance tuning or dedicated personnel for testing and optimization accounted for in the plan etc.

- Requirements can never be frozen and may evolve, or change, or drop as the design and implementation progresses. Thus, managing the expectations from performance is a continuous process by not over-selling,

www.persistent.com

and **driving or building a culture of plan-test-measure-optimize**. Documenting the performance success criteria per component of the BI system and prioritization of reports critical to the business is very important and can go a long way in minimizing the potential risks.

- Assess current performance needs as well as future scalability and concurrency (for multi-tenant DWs) requirements, their cost implications, and come up with sizing estimates for space based on factors like row length, number of tables, number of years of history, index structures, de-normalization applied, temp/sort space, number of aggregate tables, number of stage tables, metadata and configuration tables, log/error tables, etc. ([1] page 340, 360). **In general, DW takes up three or four times as much space as that of OLTP space.**

7.2.2 Performance at Design and Technical architecture stage

- **Identify the components in the entire data flow or pipeline right from source to consumption, which need to be tested or optimized for performance** ([1] pages 336,363). Typically, this includes:

  - Check source systems involved and their connectivity methods, and strive finding the optimal way to get data in. For instance, does this connector allow batching to increase the throughput, does it allow parallel reads? If not, can it allow bulk reads and what will be the impact on the source system? If the source system is message queue, check for data retention or caching knobs.

  - ETL system: choose an ETL tool which allow parallel processing of data without any data loss, or duplication (like the potential for duplicate surrogate keys), or any other implication in case of error events. Parallel processing both at job and at data level is preferred. Job-level parallel processing means multiple jobs can be run by a master ETL process or job without causing any disruptions and utilizing multi-core hardware. On the other hand, for processing huge amounts of data, data-level parallel processing is a must. Data should be partitioned logically, on certain keys, say date range, and processed by different threads in parallel. The parallel data processing may also be referred to as grid processing.

  - Reporting tool: Understand the architecture and the various rendering layers of the reporting and visualization tools to get an understanding on the time spent between layers, from data request to final response. Choose a tool having minimal overhead at Model-View-Controller and presentation layer (of the order of 100 to 30 milliseconds is reasonably good)

- **Database infrastructure**. Section 4.4.1.7 asked the question about choosing database technology for a traditional data warehouse between MPP and SMP technologies. As this is a question mainly about performance, in this section we present an overview of the public measurements that have been done a well-known decision support benchmark, namely TPC-H [25], backing up, along with our own experience in the matter, the recommendations that were given in chapter 4 above. TPC-H is a warehousing benchmark that defines 22 relatively complex queries on top of a very simple database schema (just 8 tables) modeling the activity of a wholesale supplier. Different database sizes can be generated using scaling factors, from 1 GB to 100 TB. Two performance metrics are defined by the benchmark: (i) TPC-H Composite Query-per-Hour Performance Metric (QphH@Size), which reflects several aspects of the capability of the system to process queries in an hour, and (ii) The TPC-H Price/Performance metric, expressed as dollars per QphH@Size.

www.persistent.com

Unfortunately, TPC-H has received multiple criticisms, from both MPP database vendors and analysts, and it is disappointing that some of the most popular vendors (Teradata, IBM Netezza, HP Vertica, Pivotal Greenplum) have basically shunned it. Only a handful of DBMS vendors continue measuring their systems, among which there are Microsoft (although not for their MPP database), Oracle, Actian, Exasol, and Sybase IQ. Nevertheless, MPP databases ("Clustered systems" in the TPC-H terminology) have been measured to be generally superior to traditional non-clustered systems, both in performance and in price/performance. Here is a snapshot of the results for the 3000 GB size:

3,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	7,808,386	.15 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXA Cluster OS 5.0	09/23/14	Y
2	Hewlett Packard Enterprise	HPE ProLiant DL580 Gen9 Actian Vector 5.0	2,140,307	.38 USD	NR	07/31/16	Actian Vector 5.0	Red Hat Enterprise Linux Server 7.2	06/02/16	N
3	CISCO	Cisco UCS C460 M4 Server	1,071,018	.60 USD	NR	06/01/16	Microsoft SQL Server 2016 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard Edition	05/14/16	N

You can see that the 'cluster' column of the first product is a 'Y' and the two others are 'N'. Details on each system can be obtained by clicking on the link under the System column. For instance, the Exasol database results were run on a Dell cluster of 28 bi-processor servers. The second line results, for the Actian database, were obtained on an HPE SMP machine with 4 multi-core processors. Here are the results for the 10 TB (10,000 GB) database size, in where MSFT SQL Server recently posted a world record in this TPC-H 10 TB "non-clustered" category[29]. The ratio in terms of QphH widens from about 8:2 to 10:1 with the size increase, an expected result.

10,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	10,133,244	.17 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXA Cluster OS 5.0	09/23/14	Y
2	CISCO	Cisco UCS C460 M4 Server	1,115,298	.87 USD	NR	11/28/16	Microsoft SQL Server 2016 Enterprise Edition	Microsoft Windows Server 2016 Standard Edition	11/28/16	N

- **Hardware infrastructure**. Assessing the server and network infrastructure required to support the desired performance and scalability is addressed in [1] pages 362 and 363. It is best to keep a factor of safety margin while planning for hardware by considering factors like back-up storage, multi-user concurrent load, unplanned shut-downs, network issues etc. Hard-disk, CPU cores and memory are predominant factors while arriving at optimal hardware needs.

www.persistent.com

[29] This was on a Lenovo x3850 X6 system that was configured with four Intel Xeon E7-8890 v4 processors at 2.20 GHz (4 processors/96 cores/192 threads), 6 TB of memory, and eight SanDisk ioMemory SX350 6.4TB PCIe flash adapters.

7.2.3 Performance at Dimensional DW Model design stage

- Design the DW model in such a way that it provides a balance between requirements of slice/dice and filters across dimensions and fact data, and the need for preserving history of changes ([1] page 262, 270, 332, 338, 339). We review the three patterns introduced in section 5.2 from a performance point of view hereunder:

  - Star Schemas allows for history to be tracked individually on dimension tables, as well as slicing/dicing by dimensions. De-normalization of dimensions is done to avoid capturing too many minuscule relationships. This is generally the most efficient from a performance angle.

  - Snow-flake model (normalized or 3NF) works best if you want to slice-dice by all possible relationships. Other condition where this may apply is in the case of ragged dimension hierarchies and outrigger dimensions, as explained in section 5.3.3. However, due to too many joins, the database view complexity goes up and performance is slower. If this is really a requirement, then assess the performance impact before designing, as the space saving may not be substantial (between 1 to 3% as compared to de-normalization – [1] page 267).

  - Fully de-normalized fact table with dimension attributes and no dimension tables (degenerate dimensions) – this may be the most simplistic design approach but may be difficult to maintain in the long run. If any attribute value changes (SCD Type-I or SCD Type-II), it may warrant updating the whole fact table which is costly from performance point. Also, if there is a need to show a filter in reports to choose values from, then it would require creating additional views on the fact table with DISTINCT clause which is inefficient. If slicing/dicing by dimensions is not a key requirement, then this pattern is best from performance as there are no or minimal joins or indexes lookups. Degenerate dimensions are normal and helpful too.

- Design aggregate tables to speed-up dashboards and score-cards which only show summary data. The aggregations are used to delivery fast query responses. Listed below are 3 aggregation patterns deriving from [1]. For purpose of discussion, we shall use a dimension model with Attendance as the main fact, and the Dimensions as Date, Promotion, Showing, Theatre, and Movie.

  - Lost dimensions/ Summaries of Fact tables are created by only using required dimensions. Data from other dimensions is "lost". For instance, a daily sum of earnings by theatre is a case of lost dimensions as the dimensions Promotions, Showing, and Movie do not matter.

  - Rolled up dimension: Aggregations is often used with dimensions that are hierarchical in the business world. Some common aggregation dimensions would be date and geographies. However, some business specific aggregations follow dimensions of fixed hierarchies. These dimensions are also called a Shrunken Rollup dimension.

  - Collapsed dimension (de-normalized): Columns that are part of dimensions usually get reproduced as is in the de-normalized structure that has the facts and the dimension attributes.

- Surrogate integer keys are tight and more efficient than alpha-numeric natural keys or GUIDs. They result into **faster performance due to efficient joins, smaller indices and more fact rows per bloc**k ([1] page 255, 332).

- Matrix tables with no measures or fact-less fact tables: multiple column keys to unify a relation between multiple dimensions can be chosen as a pattern for some subject areas to model m:m relationships.

- **Avoid overly generalized columns with key filters**, e.g. DimAddress as a generic table, with a key to filter on an employee, vendor, customer, and contract staff etc. As there is virtually no overlap in these physical entities, it just complicates the index structure unnecessarily.

7.2.4 Performance at Implementation, physical design stage

- **Avoid multiple columns for hierarchical dimensions** ([1], page 268), e.g. instead of having product, product category, line of business as separate columns, we could have a single column named product and indicate the granularity. This results into less and simpler indices.

- **Date dimension should have integer surrogate keys** instead of format YYYYMMDD ([1], page 254), as the PK since it is 8 bytes, we would be wasting 4-bytes storage per row or index.

- Design initial index plan for all dimensions and facts ([1], page 344). **Iteratively improve upon it as per reporting requirements or data access patterns.**

- For good query performance, it is important that **key columns be the most efficient data type for join performance**, usually an integer but a fixed-length CHAR might be relatively efficient for low-cardinality columns ([1] page 268).

- Section 5.3.5 presented the CDC subsystem to capture the changed source data. Generally speaking, timestamp columns and database transaction log reads have less impact at the source as compared to database trigger mechanisms that generate new rows on logging tables holding the delta records, as the ETL imposes read-only operations as opposed to writes (triggers) followed by reads (ETL) to compute the delta record set.

- For data movement performance -
  - Avoid using transforms which does row-by-row processing/fetch/load.
  - Utilize ELT transforms to push transforms to DW
  - Develop and test jobs for parallel processing
  - Test ETL for near real-time loads, if it is a key requirement
  - Drop indexes before fact loads
  - Avoid fact updates as they are slow and prefer DELETE-INSERT. It will result into more fragmentation of data but rebuild the indices post each load.
  - Create stage tables for more flexibility, manageability and better ETL performance

- For reporting/BI tool performance -
  - Apply appropriate filters to hit the indexes
  - Avoid report level calculations, grouping, applying business rules, null checks etc.

- Data Warehouse -
  - Implement indexes as per data access requirements. Prefer to use covering indexes.
  - Build a script to rebuild indexes periodically and collect DB statistics (defragmentation)

www.persistent.com

- Leverage star schema optimizations at the DW as most DW support this by creating appropriate indexes to filter individual dimensions first and then apply it to indexes on fact table to read relevant data rows

7.2.5 Performance at Deployment and support stage

- Test for performance – create a test plan for performance use cases to be validated, decide on the data set size, load, number of users, hardware to run on and what metrics to collect. It is best to test both ETL and BI tools for desired loads and scalability. If possible, script the use cases in an automation tool. Apply optimizations before running any tests like clearly the DB or BI tool cache, defragmenting or collecting the stats.

- Multi-user benchmarking – it is good to test the system under realistic workloads by simulating multiple concurrent users accessing the data as well as loads happening simultaneously for near real-time systems.

- Monitor and profile the long-running queries and measure performance as data grows. By doing so, assess need for additional hardware.

- Aggregation tables should be considered a performance-tuning vehicle. Build some aggregates, conduct some performance tests, and then revise which aggregates you will use in production. From there, you need to monitor aggregate use over time and make adjustments accordingly.

- Purge historical data after retention period from aggregation tables.

7.2.6 Security at Requirements and planning stage

- The DW/BI system must publish data to those who need to see it, while simultaneously protecting the data ([1] page 83). Security cannot be an after-thought and should be discussed right in from the beginning with business users and how each piece of the data will be accessed and by whom. **Making a matrix sheet, with classification of data attributes as public, sensitive, private and roles/users access permissions is considered very handy while implementing data views and reports.**

- Document the real needs to secure data and do not go over-board as the DW/BI system is successful only if people can access it. There is also cost and performance overhead attached to each layer of security we add. Cost of deploying and maintaining security solutions should also be factored in. It is recommended to have authorized access by striking a good balance between no security (all data is open and available) vs. extremely secured system (most data is sensitive).

- Secure from threats to business continuance ([1] page 174-177). Other vulnerabilities that may affect ability to conduct the business include: denial of service attack on the report server or database with huge number of false requests or virus or Trojan horses halting the server, inability to reconstruct the consistent snapshot of the software (ETL, Reports) from code due to loss of backups or corrupt files or simple oversight. Many of these can be controlled with limited access on the network using routers, firewalls and security-specific devices that monitor for malware behavior and mitigate their effects.

7.2.7 Security at Design and Technical architecture stage

- Decide on ETL users and required minimum permissions to run the ETL jobs, it should not be with system administrator or DBA permissions ([1] page 144-145).

- Design approach to secure the data coming out of the DW either in the form of exported reports, or DB backups or log files or tools that DBA use to browse the data. Have an audit log mechanism in place to log all data access events. Data going out of DW system periphery should either be encrypted or masked using a suitable algorithm.

- Network and Hardware security -

  - Restrict login access such that only BI system administrators log in to the servers running the DW/BI components and other network services.

  - Restrict network (TCP/IP) access by ensuring the proper network components and protecting the contents of the envelope via encryption, not tinkering with the IP addresses posted on the outer side of the envelope.

  - Deploy packet filtering router to reject connection attempts from unknown hosts or spoofed as insiders or sniffing

  - Ensure that the data repositories, code, build system, backup files are secure

  - Keep up to date with security patches of all software components including OS

- Use a directory server using LDAP standard for communicating, authenticating, and authorizing all users of the DW. This also provides single point of administration for enterprise policies and DBAs do not have to set up low-level SQL GRANTs/REVOKEs to implement security, thus defeating the flexibility

- Secure contents (not the addressing scheme) of distributed communicating systems like message queues, enterprise bus, distributed grid systems like Hadoop. The following elements that encrypt communications at some level are:
  - Private and Public key encryption
  - Trusted certificate authorities
  - Secure sockets layer (SSN)
  - Virtual private networks (VPN)
  - Kerberos authentication

7.2.8 Security at Implementation, physical design stage

- Secure the development environment ([1] page 210), as the data on development and QA servers is typically drawn from production sites and is sensitive, thus create an access control policy on dev/QA servers. Ideally, these servers should have similar security enforced as that of production servers but we may not want to lock down the system so tightly that developers cannot function at ease.

- Design and create metadata tables related to application security, scripts to create users/groups and relevant GRANT/REVOKE permissions on objects to groups ([1] page 211-220). Authenticate the users at BI system layer as well as DW layer. Implement row-level and column-level security, by utilizing these metadata tables and DB roles to controls what is being displayed in a DW views.

- Build a data sensitivity matrix like who can see raw data vs. aggregated data or who can see certain columns. Also think of inferred values aspects e.g. the column may be secured but the users are able to infer the values, for example, either from other columns or by taking count of values

- Never provide access to BI system to query base tables, instead all access to be granted through DW views

www.persistent.com

- Mask sensitive data elements like ID numbers, name, compensation, revenue or any other personally identifiable information (PII) during the ETL process. Most database engines can encrypt the data at column level.

- Configure BI tools with a single administrative ID for access to the DW and limit the user access to that tool. This way all read access is controlled via one ID. This is also sometimes referred as reporting service account.

- Configure BI tool to implement project and folder-level security to limit access to reports and ad-hoc templates.

- Unrestricted data – accessible to all users who are authorized to access BI or DW database, under reporting service account, no special permissions to be given

- Restricted data – restricted reports don't require a security layer within the underlying database and can be handled at BI system layer.

- Filtered data – the data presented is different to each user based on the row-level and column-level access control and cannot be cached as the report is dynamic in nature. Connect user's ID with data elements such as department, business unit and columns thereof. Recommended way to do is to have the source query written as a stored procedure or function accepting user's identification claim as a input parameter or use row-level security in database. This is expensive to develop and maintain (in cases like change in user privileges or business unit mergers), so use this data security pattern with utmost care.

- Ad Hoc access – here users leave the boundaries of predefined reports and authorization is recommended to be controlled at database layer. This is done by providing access to views, tables and creating security related metadata tables (user ID and list of dimension row keys where user have access) which can be joined with fact table to filter the data. Implement an analytic server or OLAP cube for more rich yet secured ad hoc access.

- External user access – it is recommended not to provide access to external users to DW/BI system and follow a subscription-based approach. Provide data access via a REST API layer or e-mail the reports to subscribed users.

7.2.9 Security at Deployment and support stage

- A solid backup and recovery system is a key to any security plan. In case of disk crashes, ETL should be designed in such a way that it can be run in historical mode to source the data again or restore the data from backup stage or DW tables. Most database engines provide good backup and restore mechanisms. However, restoration may happen to a point in time where backup was taken and it is imperative to ensure there is no data loss or inflation, after re-run of ETL for the duration of data loss and few simple data integrity checks would go a long way.

- There are various hardware, database, network and application components in a DW/BI system distributed across the enterprise and even beyond its boundaries. This means security is a distributed problem. We recommend identify someone in the BI team as a security manager who can co-ordinate with architecture and security team to secure all possible components to implement a robust security strategy and address vulnerabilities, on a continuous basis.

- Secure physical assets – these are servers, build machines, source control systems, workstations, virtual machines and laptops involved in the implementation of BI system. Most modern servers/systems provide some level of access control.

www.persistent.com

- Secure software assets – decide who will have access to which software like modeling tools should only be accesses by designers, ETL tool and jobs by ETL team etc. It is recommended to load each software on a dedicated machine or virtual machine as it becomes easier to track and provide privileges and access can be given to a dedicated group than individual users.

- Monitor system usage, ensure compliance and identify unusual access patterns or events like large dataset query or failed attempts to access sensitive data without permission and investigate them. It does no good to simply monitor the events. Some events to monitor are:
  - User connections, login/logout date/time
  - Failed attempts to connect/reconnect
  - Data access activity, especially ad hoc queries
  - Database transactions or DBA related activities

- Every new component or upgrade, user group changes, system changes need to be examined from security angle to make sure it is not compromised at any point. Many organizations require a sign-off from a security manager as part of the deployment process.

7.3 Enhancements to the reference publication

7.3.1 Performance optimization based on our own experience

The first section below lists some performance optimization practices that have worked well in our projects, some of which are summarized in the second section below.

7.3.1.1 Best practices

1. **Performance should never be an afterthought.** In projects where performance and scalability requirements are clear, make sure that performance is addressed from the beginning in the overall design. If performance requirements are not clear, make sure this situation is corrected.

2. **At the same time, don't optimize too soon**. There is no contradiction with the above point if performance is measured at each step and optimizations are provided based on these measures.

3. **Plan for performance testing with real data before going live**, both getting the data in (via ETL) and out (via queries from reports and dashboards). Confirm your performance SLAs with real data. Also preferably perform this testing in an isolated staging area different than development, with similar workload characteristics simulated to at least 50% of peak load or average load anticipated in production and on a hardware similar to that of production but on a reduced scale.

4. **Leave automated performance tests behind to make sure that performance is measured on every cycle.** For products or solutions which are being developed continuously, these tests can be part of periodic or regular sprint activity carried on stable builds. Include canned queries from reports and dashboards, as well as ad-hoc queries. Performance testing of the latter may be challenging as, by definition, ad-hoc means… ad-hoc. You can (i) re-examine the business requirements to come up with your own queries; (ii) observe how initial business user testers query the system, or (iii) look at the system logs for ad-hoc query sessions if the system is in production. A good performance and scalability test suite should contain multi-user sessions and, for each session, think time between sessions. The workload should be realistic matching

www.persistent.com

to what is anticipated in production with mix of concurrent users accessing different types of reports with varying query workloads or resource needs. For example - for a Retail data warehouse, the workload could be – 5 Regional managers looking at inventory and sales snapshots of aggregated data, week-on-week performance while 25 category managers looking at detailed sales records for the last day and inventory forecast needs for next week.

5. D**evelop a plan to conduct ongoing performance monitoring regularly.** Collect system and database statistics to understand how system usage and performance may be changing over time to access if future data growth can be handled with the hardware or software deployed

6. **In general, prefer processing as close to the database as opposed to techniques outside of the database.** At the same time, prefer using high-level tools for faster development cycles. The sweet spot are those tools that are high level, that contain the functionality you need, and that generate efficient code that runs close to the data.

7.3.1.2 Specific solutions

1. **ETL and BI Performance - Real-time insights for Network & Subscriber Intelligence**

- Requirements/Challenges

  a. PSL was tasked to develop an analytics solution for network assurance, which was deployed for one of the biggest Telecom carriers in USA.

  b. The product features were:
     i. Real-time monitoring and reporting of mobile and wireline network performance, service performance and customer experience
     ii. Troubleshooting of mobile and wireline network performance, service performance and customer experience problems

  c. The input data was call data records (CDRs) and data detail records (DDRs) collected in real-time across disperse networks and collected into a centralized server. The challenges were: collect a very large (16 billion+ records per day) data set, load it into a data warehouse, and compute and display real-time statistics about network and subscribers on a chunk of this data set, with an end-to-end SLA of 5-mins delay, with reports to be displayed in less than 20 seconds.

- Solution

  a. PSL designed a data pipeline to meet above SLAs and throughput requirements.

  b. **Data Collection** - As the data moved to the central server, there were configurable polling agents which diverted the traffic to a distributed cluster running a set of java agents to process the data. The uncompressed data volume per 5-min was ~50 million records from all networks i.e. 16 billion records/day

  c. **Data enriching and raw data load** - The customer used a cluster of 40 blade servers to run the ETL system. The enrichment processes (java agents) ran on each of the cluster servers to pre-process data through 112 processes running in parallel, piping multiple input data files into a single one and compressing it. These agents loaded files into dynamically created staging tables of a data warehouse implemented on a MPP database Teradata Server (v12, 4-node, 180 AMPs), along with logs of what files

www.persistent.com

were loaded, how many bytes, number of rows etc. PSL used Teradata's FASTLOAD (bulk data loading) utility to process this large amount of data.

d. **Data Processing and target load** - On the Teradata server, an E-L-T approach was followed, which ran multiple stored procedures to create aggregation tables per 5-min window to cut down on the data volume. The stored procedures also handled late-arriving data and re-build of aggregate tables. A Kimball-style dimensional data model was followed along with multi-level partitioning techniques and indexing to speed-up reporting queries. Aggregate tables were created with various grains (hourly, daily, with storage up to 365 days). PSL achieved fresh data load rates of 1.5 to 2 TB/hour with this configuration. The data warehouse size was 90 TB/week and 5-min aggregation data was expired every week with daily/weekly aggregates stored which were expired after 365 days.

e. **Data Visualizations** - SAP Business Objects (3.x, 4.x) was used with metadata model created along with aggregate-awareness feature enabled. Aggregated join indexes were used to further speed-up queries by reducing the IO on database side. Dashboards were refreshed every 5-mins and the response time for most reports were in 10-15 seconds range.

2. **ETL and BI Performance – Workforce Analytics**

- Requirements/Challenges

a. PSL was tasked to develop a multi-tenant analytics HR workforce solution including a Time & Attendance, and a Scheduling and absence management products for customers from multiple industry verticals like Healthcare, Retail, and Manufacturing.

b. The product features were:
   i. Real-time visibility into workforce absence and time/attendance tracking
   ii. Guided discovery to find outliers, patterns, trends, and relationships
   iii. Labor-cost optimization

c. The input data was time/attendance and scheduling data coming from both a MS SQL Server and an Oracle-based OLTP system (multiple DB versions to be supported), and the target could be MS SQL Server or Oracle (again, multiple DB versions to be supported). The solution was hosted on Amazon AWS public cloud or could also be deployed on premise.

d. The largest data volume handled was 2.5 TB for a retailer in USA

e. The ETL performance requirement was to process the incremental data in near real-time with a configurable delay (2 to 5-minutes) and load the historical data for past 2 years within 4 hours. There were a lot of ETL calculations and business rules involved, which slowed down the end-to-end ETL process. There was also a need to show reports to be sliced-diced by 45 pre-defined time-period ranges and custom groupings. Response-time performance refreshing reports was very critical (< 10 sec) on the web as well as on mobile devices (Tablets, iPhones).

- **Solution**

a. PSL used date/timestamp-based strategy for change data capture on the data source tables.

b. PSL developed the ETL in Talend Enterprise Studio (5.5) with E-L-T approach to push the queries to the database to execute queries natively and thus save time.

www.persistent.com

c. Techniques like dataflow parallelism for unrelated dimensions were used within ETL jobs to speed data loading. This did pose problems, e.g., surrogate key generation within DB using SQL Server or Oracle Sequences could not be used with IDENTITY columns. However, with tMap (a join component in Talend) and global variables this was resolved.

d. Load processing of the FACT data chunks was done in parallel.

e. Data was sourced with Talend's JDBC components in batches with a database fetch size=50000 to 100000, to optimize the process time.

f. Stored procedure-based ETL was profiled for each query using database query profilers and optimizations were applied like loading the data into temporary stage tables, dropping and rebuilding indexes post-load and loading data in chunk sizes of 200000 - 250000 records.

g. Data pages and indexes became fragmented due to continuous loads (inserts/updates), so periodic rebuild of database statistics and indexes was done using scripts. A post-load re-build indexing job was run in parallel across many processors available on the machine.

h. FACT and dimension tables were created on separate file groups to speed up the disk IOs for dimensional query performance.

i. It was recommended to use a dedicated server for ETL, as Talend is a memory-centric engine, along with JVM heap-space and other factors tuned.

j. Data model was designed in such a way that each business area / process was modeled in a single star schema.

k. For large fact tables, partitioning was used with an appropriate index scheme to cover the report queries.

l. PSL also proved that system is linearly scalable with additional hardware as the mixed query workload was doubled.

m. Eventually, the team followed an approach of "Monitor-Analyze-Optimize" blended within sprint cycles, to collect system and DB statistics such as query counts, avg. SQL executions per second, avg. execution time taken per query, max execution time, avg. IOPs, avg. active sessions, etc. to identify if system was deteriorating over time.

n. **Data Visualizations** – Microstrategy (9.x) was used view dashboards and reports along with Ad Hoc views. Report queries were optimized using de-normalized dimensional model and aggregate tables. For a few dashboards, Microstrategy's in-memory cube was used to speed up dashboards on Mobile devices.

3. **BI Performance – School Information Management System**

- Requirements/Challenges

a. PSL was tasked to develop a cloud-based analytics solution for School Information management.

b. The reporting requirements were a mix of transactional reporting along with analytical reporting. There were a large number of entities which were having 1: m relationships and needed to be tracked for each change (Slowly Changing Dimensions). In general, there were 50+ dimension tables by which data had to be sliced-diced along with pre-defined date ranges.

c. Performance of reports and dashboard (less than 3 seconds for transactional reports, about twice as much for analytic reports) was also critical due to nature of business and all data to be reflected in near real-time (15-minute delays)

d. Data volume was close to 15 TB per 1000 tenants (schools), the public cloud provider was Microsoft Azure.

- Solution

  a. Customer initially had decided to build an Operational Data Store (ODS) type of data warehouse. The initial results for transactional reports were substantially above the requested performance requirements, due to the large number of joins between dimension tables in the normalized schema.

  b. PSL proposed not to build a separate ODS but follow a dimensional de-normalized model to serve both transactional reports on a particular effective date as well as analytic reports.

  c. PSL proposed to build aggregate tables along with Aggregate-aware feature in reporting tools like SAP BusinessObjects or IBM Cognos.

  d. PSL conducted a workshop to help customer select adequate database platforms and analytic tools that are designed for high performance such as MPP DBMSs, and also investigated in-memory databases, columnar databases, and DW appliances. Given that in their future plans they want to handle unstructured and semi-structured data, PSL also investigated big data stores such as Hadoop with MapReduce, Spark, as well as No-SQL databases.

  e. PSL recommended to use an MPP database engine with a shared schema approach, partitioning by tenant-id and suitable indexing scheme to make reports perform and keeping below requirements in mind
     - I. Analytic app with terabyte-size data mart
     - ii. Intense queries not following the dimensional pattern
     - iii. Linear scalability

  The customer has not made the final choice yet.

7.3.2 Security best practices based on our own experience

- Access control on DW data contained in dimensional model entities

  - This approach is most widely used and is common when the security provided by DB engines is sufficient to control the access[30]. This is preferred to the pre-defined security framework provided by BI tools on DB engines are proprietary in nature and can only be used for the tool in question.

  - The security approach to control the access on all dimensions, fact tables and summary tables via views is the simplest and the one we recommended most, because of its flexibility in assigning user roles (provided by BI tools or DB roles) to different sub-systems (or subject areas or data marts) of the DW. This process involves identification of all data objects that will be exposed to the end-users, classification of data tables and fields as per user roles and access levels.

[30]This might not be the case, as explained below.

www.persistent.com

- Column-based security – Most DB engines provide out-of-the-box column-level permissions to be set for a role or user or a group of users.

- Deriving access control policies based on OLTP or source data is not recommended and rather should be based on how users of DW will be accessing the data and for what purpose.

- Metadata-based security model

  - This approach is common when enterprises want to control the data access at a more fine-grain level or driven by applying certain business rules like organization hierarchy with

  - certain policies relevant to the business, where DB provided roles/groups are not sufficient. For example, in a workforce analytics, there can be contractor workers who are part of an organization hierarchy but certain labor account and employee related policies are also to be enforced simultaneously.

  - In this approach a custom layer is built, where list of objects, their permissions to various users, or groups, or roles, is configured.

  - While projecting the data out of the DW via views, views have to be joined with these custom metadata objects and, based on the logged user; the view is filtered to provide access to the allowed rows or columns.

  - The configuration of metadata can also be xml-driven or defined in a json file; however, it cannot be directly used in views and warrants a building a thin access layer on top of the DW.

  - Column-based – When DB provided column permissions are not enough and some business rules need to be enforced based on user login, then sensitive data can be either masked or projected via stored procedures or routines which project only the required columns where the users have access to.

- This is a cost-intensive and performance-overhead adaptation, so such requirements have to be considered carefully.

- OLAP / data cube security

  - OLAP is meant to provide quality inputs to the end-users in a reasonably quick time. Thus, growth in the number of users accessing the cube increases over a period of time and demands a good security model.

  - Most vendors provide framework to access control the OLAP cubes.

  - Since OLAP is exploratory analysis in nature, security controls may hinder the access to the data and therefore good auditing mechanism should be in place.

- Data Encryption

  - If the data in the DW is to be accessed by 3rd party systems (say web services or vendors), it must be transported with encryption enabled.

  - Encryption over the wire prevents malicious users from intercepting data they might not otherwise have access to.

- Ensure that security does not impact performance and intended functionality, especially real-time analytics. If it does so, trade-of between relaxing SLA requirements over security.

- Audit logging and control

  - Transactional databases don't generally store the history of changes and, as data in the data warehouse is held for much longer periods of time, it makes sense that data auditing is implemented in the warehouse. Data auditing also keeps track of access control data in terms of how many times an unauthorized user tried to access data. It is also useful to audit the logs recording denial of service events.

  - To implement a data auditing, the first step is to scope out auditing, i.e., identify datasets which are required to be audited. Doesn't push for auditing on every dataset as it not only require processing of data; it may also end up hampering the performance of the application. Identify business needs and then go about creating a list of datasets, rules (e.g. who can access it, legal retention requirement of 1 year) associated with it in some kind of repository. Define policies and identify data elements (like location of data, condition/status or actual value itself) whose change values need to be collected as part of the data auditing.

  - The next step is to categorize or tag datasets in terms of importance in the enterprise. While this will not help in searching or indexing, it does help in scoping the level of audit activity for each type of dataset.

  - Finally, for selected datasets to audit, log all data access and the value changes that might be sensitive to business as well as data export transactions from the data warehouse. Logging changes may be implemented in two ways: either by copying previous versions of dataset data elements before making changes, as in the traditional data warehouse slow changing dimensions, or by making a separate note of what changes have been made, through DBMS mechanisms such as triggers or specific CDC features, or auditing DBMS extensions.

  - Monitor that right users see the right data and detect security breaches, build audit-ready activity reports

  - It is better to build or identify existing data audit frameworks to collect the information about these data sets and support policies and rules required for auditing. Using the framework will help in automating some of these processes efficiently and risk-free. The framework may require a separate policy and rule engine to be built so that policies applied to data sets can be rationalized/prioritized automatically based on the risks and compliance fulfillment.

7.3.3 Cloud deployments

7.3.3.1 Performance

- **Data connectivity**
  Often, cloud providers will leverage APIs for data connectivity or provide pre-built connectors that work with heterogeneous systems or cloud services. These connectors will have certain characteristics (knobs) to optimize the fetch performance. Understand them well before implementing; if possible do a POC to gauge the performance. This also applies to ETL services such as data quality and data validation which are cloud-resident.

www.persistent.com

- **Data movements from on premise-to-cloud or cloud-to-cloud**
  Understand the data structure, volume, and update frequency. There are many ways, including secure FTP and WAN optimization. Some cloud vendors provide compression for big data that enterprises want to upload to the cloud.

  - Perform heavy-duty data export or import or ETL operations during night or off-peak period. Also perform parallel loads to utilize resources optimally.

  - With larger volumes of data, one approach is to put the initial data upload onto disk and send it to the cloud provider for upload on their internal network via sftp, and then use a VPN for ongoing differentiated loads. Some vendors also provide loading compressed data in the DW from storage services: e.g. loading files in a S3 bucket into Amazon Redshift DW.

  - Another approach is to use private network connection solutions, which allows organizations to directly connect to cloud providers rather than through connections established on the public Internet. In these kind of solutions, the customer should choose an interconnection provider that has a broad geographic reach, and place its equipment close to the provider's data centers from where direct connection to public and/or private cloud providers (Amazon, Azure, Google, etc.).

- **Query Performance**
  - Frequently monitor and profile the queries to identify long-running operations and run EXPLAIN to identify any bottle-necks. Outdated system statistics can severely degrade performance, so keep performance related system statistics up-to-date. A nightly or weekly statistics rebuild is all right, all depends upon the data load frequency. This is recommended to be performed only by designers or advanced users.

  - On MPP cloud databases, it is important to partition data in the cluster by appropriately defining distribution keys and sort keys, so that related data can be co-located on a node and is sorted. For example,

    - Using CUST_ID as the distribution key with a hash partitioning function will partition the table according to the value of the hashing function on the CUST_ID column. Associative access to the data with a specific attribute value can be directed to a single node, and sequential access can be served in parallel.

    - Using CREATED_DATE as the sort key will sort the data ascending such that date range queries response improves substantially.

  - While reading the data, project only the columns required instead of "SELECT * ", which may not hit indexes always and end up doing table scans.

  - Apply selective filters before join –filtration will significantly boost join performance, if we filter out irrelevant data as much as possible.

  - If possible, run one query at a time as multiple queries will demand resources and may slow down overall query execution or entire cluster.

  - Data transfer for cloud database to client – avoid transferring data directly from DB to the client. Instead, projected data can be loaded to a temporary table and exported to a file using command-line utilities.

www.persistent.com

7.3.3.2 Security

- If sensitive data such as medical records or financial information can't be put on a public cloud but the customer wants to take advantage of public cloud scalability, flexibility and fast ramp-up, consider implementing a hybrid cloud architecture. This way customer can slice sensitive data out from non-sensitive data and store the former in private storage (on premises or in a private cloud) and the latter in the public cloud. Customers can address the performance issues of this data split by locating their private infrastructure within a colocation facility close to the public cloud of their choice. As mentioned in the above section, consider using a network provider to further reduce latency.

- Cloud DW may contain data for multiple source tenants or customers. It is recommended to separate the multi-tenant DW systems by either

  - Sharing the schema among all tenants, partitioning a table by tenant-id and injecting filter clauses by tenant-id to each SQL request, or

  - Separate table for each tenant, or

  - Separate database schema for each tenant

  For stronger isolation guarantees, consider using Virtual Private Networks.

- Perform frequent operational readiness checks in terms of system surveillance and incident management for system failure, as well as incidents such as fire or other crises, business continuity plans, and how often and what kind of vulnerability assessments are to be performed.

- Manage user accounts – Ensuring that users have appropriate levels of permissions to access the resources they need, but no more than that, is an important part. These can be ensured by creating cloud accounts.

- Manage OS-level access to the Virtual Machine instances or images.

- Platform and application security: This should include intrusion detection; support for secure connectivity via VPN, SSL, and others; and role-based access controls.

- Data security: This includes provisions for controls to maintain the integrity of data and for secure transmission of data using encryption or other techniques.

- Security measures that rely on encryption do require public-private keys. In the cloud, as in an on-premises system, it is essential to keep keys secure. You can use existing processes to manage encryption keys in the cloud, or you can leverage server-side encryption with Cloud vendor provided key management and storage capabilities.

- Always change vendor-supplied defaults before creating new images or prior to deploying new applications

- Remove or disable unnecessary user accounts

- Enable only necessary and secure services, protocols, daemons, etc., as required for the functioning of the system. Disable all non-essential services, because they increase the security risk exposure for the instance, as well as the entire system.

- Disable or remove all unnecessary functionality, such as scripts, drivers, features, subsystems, EBS volumes, and unnecessary web servers.

www.persistent.com

- Cloud-based services, whether operating as high-level analytics services or foundational platform services, address some security capabilities while introducing new challenges. The service provider may be addressing platform and network security to a high degree of assurance but lack visibility into who has accessed what. In this cases, it is desirable to build an auditing framework (or re-use an existing one) on premise to control data access.

- Compliance: Compliance requirements can come from both internal and external sources and organizations might adhere to certain regulatory requirements or those imposed by customers or partners. Some typical data-related compliance requirements that might affect a cloud provider include: PCI DSS, HIPAA, SOX etc.

- Visibility: Extended governance requires visibility into cloud operations, including ETL, archiving, and the like. Cloud providers offer tools and protection strategies to (i) avoid problems that may occur during normal operations, as well as (ii) to support service-level agreements. These are summarized in the following table.

Concern	Protection strategy
Accidental information disclosure	Permissions File, partition, volume or application-level encryption
Data integrity compromise	Permissions Data integrity checks Backup / Restore Versioning
Accidental deletion	Permissions Backup Versioning
System, infrastructure, hardware or software availability	Backup / Restore Replication

7.3.4 Big data

The following best practices apply to the performance and security management of a big data environment.

7.3.4.1 Performance

- Hadoop is a flexible, general purpose environment for many forms of processing presented in section 3.2 above. The same data in Hadoop can be accessed and transformed with Hive, Pig, HBase, Spark and MapReduce (MR) code written in a variety of languages, even simultaneously. Choose the tooling that provides optimal performance for your use case as depicted below.

  - MR applies massive parallel computation to the data, but is a batch operation and is too slow for interactive workloads. Hive on MapReduce and Pig inherits from this problem.

  - Partitioning the data sets is the single key recommended best practice to speed up computations on data lakes

- Included below are some of the best practices to tune Apache Pig queries:

  - Explicitly declare types for the columns, to avoid lazy evaluation or type conversions downstream
  - Run pig jobs on Apache Tez mode – for optimized job flow, edge semantics and container reuse
  - Use optimal number of mappers/reducers (4:1), based on size of the input files e.g. if input file size=1.280 GB and Hadoop data split size=128 MB, then use 10 mappers and 2 reducers
  - Reorder JOINs properly to reduce I/O - Join with small/filtered set first
  - Project out columns early and often
  - Prefer DISTINCT over GROUP BY/GENERATE
  - Push up FILTER – filter early, filter often
  - Push up LIMIT– reduce number of rows
  - Push down FLATTEN
  - Enable LZO compression

- Newer frameworks are available to perform interactive data analysis over data on Hadoop, also referred as SQL-on-Hadoop. These frameworks are:

  - Hive running over Apache Tez engine, a DAG network engine with cost-based optimizer to reduce wait time.

  - Presto, from Facebook, a stage pipelining engine with in-memory data transfer with reduced or no IO.

  - Impala, from Cloudera, an MPP execution engine on top of Hadoop working with the data stored in HDFS.

  - SparkSQL, an in-memory execution engine trying to get the best of both MapReduce and MPP-over-Hadoop approaches.

  These engines do not leverage the MapReduce paradigm (except for SparkSQL) but utilize in-memory, distributed computations to speed-up queries. For a query having workload to calculate MAX, MIN, AVG (Query-1), STANDARD DEVIATION (Query-2) on a time-series data with 16.5 million records residing on Hadoop, here are the numbers on a 5-node cluster.

- Apache HBase, which also works on top of the Hadoop data nodes, provides fast data inserts and reads with very low latency to locate relevant rows. However, this engine is not recommended for analytics workloads. Some of the HBase performance tuning best practices are:

  - Control the number of Hfiles
  - HBase performs major compactions by time, tune it
  - Tune MemStore /cache sizing;Increase the JVM heap size for the region servers
  - Many cores are good as HBase is CPU intensive!
  - OS - Disable swap – or else JVM will respond poorly and will throw OS OOM
  - Tune JVM parameters like -XX:SurvivorRatio=4, -XX:MaxTenuringThreshold=0
  - Increase write buffer - hbase.client.write.buffer = 8388608, for better write performance
  - Enable Bloom Filters, save having to go to disk and improve read latencies

www.persistent.com

- Improving performance of computation engines like Spark

    - Recent versions of Spark [26] provide DataFrame and Spark SQL structures and are much faster than RDD APIs, as the optimizations happen as late as possible and even across functions with predicate push-down and/or column pruning.
    - Tests have revealed that in a clustered environment network performance improvement does not contribute much to the cluster performance and jobs become more bottlenecked by IO and CPU.
    - Poor data storage choices can degrade performance drastically like too small file sizes may require opening too many file handles vs. too big file will require more splits (64Mb to 1 GB may be optimal) or use of compression formats (minimize compressed file size) or use of data interchange formats (Avro, Parquet are common ones)

- ETL data processing – Apply filtering, cleansing, pruning, conforming, matching, joining, and diagnosing at the earliest touch points in the data pipe line possible.

- Streaming applications – Make use of continuous query language (CQL) or SQL-like structures provided by some of the libraries to perform computations on a smaller chunk of the data, the results of which can be consolidated by other process downstream.

7.3.4.2 Security

The use of big data technology offers such great opportunities in terms of cost reduction that organizations are looking to store and process both sensitive and non-sensitive data in big data repositories. However, big data presents several challenges to data security. Security and privacy challenges are magnified by the velocity, volume, and variety of big data.

Volume - Managing massive amounts of data increases the risk and magnitude of a potential data breach. Sensitive, personal data and confidential data can be exposed and violate compliance and data-security regulations. Many existing enterprise security products are licensed by user or by CPU. This model makes sense in traditional systems but is problematic in distributed big data systems because of the larger user base.

Variety - Understanding what data you have (or don't have) in any dataset is made more difficult when you must manage and enforce data access and usage of unstructured data from a variety of sources.

Velocity - The faster data arrives, the more difficult backup and restore operations become. Throughput is also challenging, because traditional security tools are designed and optimized for traditional system architectures and may not be equipped for the throughput required to keep up with big data velocity rates.

Security drives business value by enabling safe handling of regulated data, but new technologies are required. Traditional security tools designed and optimized for traditional system architectures may not be equipped for the throughput required to keep up with big data. Although existing and emerging security capabilities apply to big data, many organizations lack the tools to protect, monitor, and manage access to data processed at high rates and in a variety of formats.

The approach to securing big data varies by platform: Hadoop distributed file systems, NoSQL databases, and search services.

www.persistent.com

1. Hadoop is a framework for processing computationally intensive workloads –often batch processes – with large amounts of unstructured and structured data. The distributed nature of the technology makes it vulnerable, provides a broad attack surface and emphasizes the need for automated, validated, and consistent application of security capabilities.

   At the infrastructure level, Hadoop files can be protected via the network using a firewall that allows access to the Hadoop Name node only. Client communication to data nodes is prohibited. Data-at-rest encryption protects sensitive data and keeps at a minimum the disks out of audit scope. This encryption ensures no readable residual data remains when data is removed or copied and when the disks are decommissioned.

   Hadoop cluster management security involves token delegation and managing lifetime and renewable tokens. Data security controls outside Hadoop can be applied to:

   - Data inbound to Hadoop: Sensitive data such as personally identifiable information (PII) data may be chosen not to be stored in Hadoop, or encrypted, as mentioned above.

   - Data retrieved from Hadoop: Data in Hadoop should inherit all data warehouse controls that can enforce standard SQL security and data auditing capabilities. Most computation engines along with NoSQL databases like Cassandra, Couchbase or HBase and Search services like ElasticSearch provide row-level and column-level security with simple add-ons like Apache Ranger or Shield. Apache Accumulo (NoSQL DB) also provides full flexibility to control the data access at cell level in a column family.

   - Data discovery: Identifying whether sensitive data is present in Hadoop, where it is located, and then triggering appropriate data protection measures such as data masking, tokenization/encryption, etc. Compared to structured data, activities such as data discovery, identification of location and subsequent classification and protection of unstructured data, are much more challenging. To mitigate the risk of exposing the entire data to a few set of people, it can be categorized into projects or high-level directories in Hadoop and they can be given access to specific entities or columns. These people or data analysts can run tools (semi-automated or custom scripts) to gather stats and once the data is discovered, profiled and analyzed, further granular access control can be granted.

2. NoSQL databases (such as Cassandra) form a distributed columnar data platform designed for scalability and performance, often for online transaction processing. Cassandra nodes do not have distinct roles, requiring a well-planned, layered approach to evaluating and applying security controls.

3. Search services – With the rapid adoption of search-based tools to perform interactive and context-based searches, it is becoming easier to store, search, and analyze data. These distributed search engines allow to easily protecting the data with a username and password, while simplifying the security architecture. Advanced security features like encryption, role-based access control, IP filtering, and auditing are also available. Moreover, these tools also support plug-in based architecture for authentication and authorization and are easily extensible.

www.persistent.com

Conclusion

In this document, we have attempted to document a collection of best practices for acquiring, integrating, modeling and governing data from a growing number of sources for analytics opportunities. We picked the best reference publication available, Kimball's data warehousing lifecycle toolkit [1], and summarized it along what we believe are the four most important transversal topics: dimensional data modeling, data quality, performance and security, as well as technical architecture. Each of these topics is briefly presented in its own chapter, and followed by a summary of its treatment through the requirements gathering, design, development, and deployment phases in Kimball's lifecycle, serving as a high-level view of the reference document from the vantage point of the topic in question.

We have gone beyond Kimball's traditional data warehousing architecture by recognizing that the mission of analytics has expanded on three very important areas not covered by Kimball's reference publication: Cloud analytics deployments, Big Data and the use of self-service tools along with use of agile development techniques, which we believe will be key drivers for any Data Management endeavor today. Even though some of these tendencies are still evolving, we have attempted to grow the list of traditional data warehousing best practices with our own recent experience on these new areas, in addition to our experience in traditional warehousing projects.

Our key recommendations are:

- The true measure of success in a DW/BI project is business user acceptance of the deliverables to improve their decision-making. Thus, strive to collaborate with your business users at every step of the lifecycle.

- Leverage conformed dimensions as the basis for integration; this is still true independently of the deployment and the database technology. Understand and embrace dimensional design as the organizing theme for the data within the DW/BI environment.

- Think about performance, scalability, security, usability and other non-functional aspects right from requirements and design: these aspects cannot be treated as an after-thought. Make sure you understand the tradeoffs in the new cloud and big data environments.

- Pay special attention to data quality. Many data projects fail for lack of attention to this topic. The list of available technologies and services has expanded, so don't reinvent the wheel. But also, make sure you understand what is meant by high quality big data in such an environment.

- Embrace agile methodologies, but beware of extracting a limited source data or focusing on a narrowly-defined set of business requirements in a vacuum. The deliverable may be built quickly to claim success, but it should be leveraged by other groups and integrated with other analytics.

- Consider testing very seriously, invest in testing automation early and take advantage of agile methodologies to test early and frequently. Be flexible to experiment with self-service systems with your business users.

- Finally, remember that a DW/BI system evolves and grows after the first deliverable. Metadata management should help you down the line with data lineage and impact analysis functionality.

We hope that having access to Kimball's proven reference publication and having read this document's collection of best practices for both traditional, on premise data warehousing and cloud analytics and big data environments will help you in designing, building, maintaining and extending a successful analytics solution.

www.persistent.com

Glossary

We start by an explanation of data quality terms.

Data Profiling is the systematic exploration of source content and analysis of relationships of data elements in the content under examination. Profiling generally delivers statistics about the data that provide insight into the quality of data and help to identify data quality issues. Single columns are profiled to get an understanding of frequency distribution of different values, whether they are unique values, and use of each column. Embedded value dependencies such as those modeled by primary key / foreign key (PK/FK) constraints can be exposed in cross-columns analysis. Profiling should be performed several times and with varying intensity throughout the data warehouse developing process.

Data Conformance is the process of reaching agreement on common data definitions, representation and structure of data elements and values, and **Data Standardization** the formatting of values into consistent layouts (sometimes these terms are used interchangeably). Representations and layouts of values are based on industry standards, local standards (e.g., postal standards for addresses), and user-defined business rules (e.g., replacing variants of a term by a standard).

Data Validation is the execution of the set of rules that make sure that data is "clean". Some of these rules are those defined or implied by data standardization, but these rules may be more involved than that: they include key uniqueness constraints, PK/FK constraints, and may involve complex business formulas constraining the value of several fields. These rules return true or false Booleans, indicating if the data row or element verifies the criteria. A specific type of validation rules dealing with duplicate data are called out in a separate topic below.

Data Cleansing is the process of fixing dirty data to make it clean(er). This process covers filling (some) missing values, correcting mistakes in strings, applying transformations to values to meet the agreed data standardization, integrity constraints or other business rules that define when the quality of data is sufficient for an organization.

Data Matching and survivorship is the task of identifying and merging (a.k.a. consolidating) records that correspond to the same real world entities through different data values and formatting standards. This typically happens when bringing data from different data sources to a target source, but may also occur within the same data source. This task reduces data duplication and improves data accuracy and consistency in the target source.

Data Enrichment: the enhancement of the value of internally held data by appending related attributes from external sources (for example, consumer demographic attributes and geographic descriptors).

The following chart illustrates these steps with a simple example.

www.persistent.com
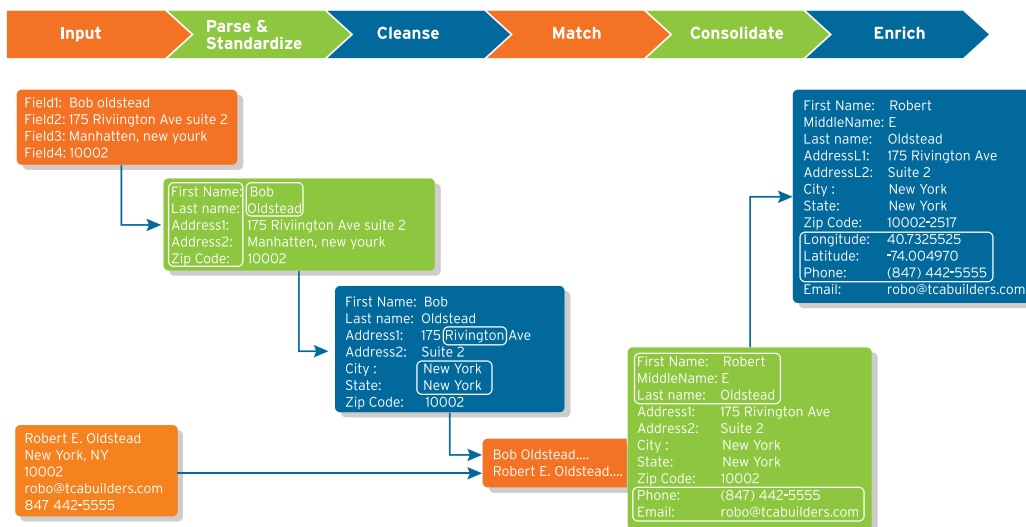
Analytic application. Prebuilt data access application that contains powerful analysis algorithms based on domain expertise (e.g. data mining algorithms), in addition to normal database queries.

BI applications. The value-add analytics within the DW/BI system, they include the entire range of data access methods, from ad-hoc queries, to standardized reports, to analytic applications.

Customer 360. An application allowing to combine customer data from the various (external) touch points that a customer may use to contact an organization and the internal data sources that that trace which products they purchase, how they receive service and support, etc., giving a complete picture of how they interact with the organization.

Data Silos. An enterprise has data silos when data is stored redundantly by an area of the organization, with each area mandating its own policies and processes. This leads to inconsistent data definitions, formats and data values, which makes it very hard to understand and use key business entities that are common across these silos. The first, classical version of an area generating a data silo corresponded to a local facility or a department within an enterprise. Then, ERP systems were introduced to help alleviate this problem (among several others). However, ERPs only deal with internal company data, and provide only partial management of customer data or supplier data: that is done by other packaged applications such as CRMs and SRMs do this. These generate today's modern version of data silos.

MPP Databases. The MPP acronym stands for "Massively Parallel Processing". These databases can be best described as providing a SQL interface and a relational database management system (RDBMS) running on a cluster of servers networked together by a high-speed interconnect, where the clusters form a Shared-Nothing architecture: i.e., each system has its own CPU, memory and disk which they don't share to any other server in the cluster. Through the database software and high-speed interconnects, the system functions as a whole and can scale as new servers are added to the cluster (this form of extending capacity is known as scale-out). This approach is used by MPP database systems like Teradata, Greenplum, Vertica, Netezza, ParAccel, and others. Why do MPP databases working on a shared nothing cluster work well for data warehouses? For mainly two reasons:

1. Relational queries are ideally suited to parallel execution; they are decomposed into uniform (relational algebra) operations applied to uniform streams of data. By partitioning data across disk storage units attached directly to each processor, an operator can often be split into many independent operators each working on a part of the data. This partitioned data and execution gives partitioned parallelism. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs. By streaming the output of one operator into the input of another operator, the two operators can work in series giving pipelined parallelism.

2. Shared nothing architectures scale well up to hundreds and even thousands of processors that do not interfere with one another. As we will see below, this does not happen with single machines with parallel processors (SMPs), where there is an interference effect.

A very good introduction to subject is the classic 1992 paper from David Dewitt and Jim Gray [24], from where we took these two paragraphs above, and which is still strikingly relevant.

OLAP, OLAP database, or engine. OLAP stands for "Online Analytical Processing" and is a set of principles that provide a framework for answering multi-dimensional, analytical queries. A OLAP database or engine is one that organizes data natively per a dimensional model, in cubes (as opposed to relational tables) where data (measures) are categorized by dimensions. OLAP cubes are often pre- aggregated across dimensions to answer multi-dimensional, analytical queries swiftly and predictably.

SMP Databases. Traditional databases work well on small to medium database sizes (up to a few tens of terabytes) on Symmetric Multi-Processing (SMP) machines, which are tightly coupled multiprocessor systems where processors can run in parallel, are connected using a common bus, are managed by a single operating system, and share I/O devices and memory resources. SMPs are rather of the scale-up sort, where additional capacity is obtained by getting a bigger machine. These days, SMPs come with 4 up to 64 processors.

www.persistent.com

Data Lake. This term refers to an emerging approach to extracting and placing big data for analytics, along with corporate data, in a Hadoop cluster which several components are layered in order to effectively enable data scientists and business analysts to extract analytical value out of the data. One of the primary motivations of a data lake is not to lose any data that may be relevant for analysis, now or at some point in the future. Data lakes thus store all the data deemed relevant for analysis, and ingests it in raw form, without conforming to any overarching data model.

Data Mart. In the Kimball reference book this term now means "business process dimensional model". In the first version, the term "data mart" was intended to refer to "a logical subset of the complete data warehouse", generally restricted to a single business process or to a group of related business processes targeted toward a business group. The reason for the change, in their own words, was because "the term has been marginalized by others to mean summarized departmental, independent non-architected datasets".

DW/BI System. The complete end-to-end data warehouse and business intelligence system.

Data Warehouse, DW or EDW. This is the queryable data in the DW/BI system, the largest possible union of presentation server data. A data warehouse is made up of the union of all its business process dimensional models.

ETL System. Extraction, transformation and loading system consisting of a set of processes by which the operational source data is prepared for a data warehouse.

Master Data Management. Master data are defined as the basic characteristics of key business entities, such as customers, products, employees, and suppliers of an organization. Master data management, or MDM, are systems designed to create, manage and hold master copies of these key business entities, and have been built in response to the proliferation of tables managed by different transactional systems, which often represent the same business entity multiple times. MDM systems support these transactional systems and have a way to reconcile different sources of data for attributes of the same real world business entity. MDM systems typically consume data quality software through specific APIs to make sure the data they hold are clean.

There are two main scenarios supported by MDM systems: (I) as a central hub to create master data, which then replicates to applications, and (ii) a master data consolidation application. In both scenarios, MDM systems help enormously in removing data silos. MDM systems provide a bridge to move master data in silos to a single source of truth, managed by the MDM system.

References

[1] Ralph Kimball, Margy Ross, Warren Thornthwaite, Joy Mundy and Bob Becker, The Data Warehouse Lifecycle Toolkit, 2nd edition, Wiley, 2008. http://www.kimballgroup.com/data-warehouse-business-intelligence-resources/books/data-warehouse-dw-lifecycle-toolkit/

[2] Redman, T.C. (1997). Data Quality for the Information Age. Artech House Computer Science Library, Norwood, MA, USA, 1997.

[3] http://booksite.elsevier.com/9780123970336/downloads/Sebastian-Coleman_Appendix%20B.pdf

[4] Haug, A., Zachariassen, F. & van Liempd, D. The costs of poor data quality. Journal of Industrial Engineering and Management, 2011 – 4(2): 168-193.

[5] Fernando Velez, Sunil Agrawal, Data Lakes: Discovering, governing and transforming raw data into strategic data assets, Persistent Systems White Paper, August 2016,
https://www.persistent.com/wp-content/uploads/2016/05/Data-Lakes-Whitepaper.pdf

[6] Gartner, p.6, Private Cloud Matures, Hybrid Cloud is Next, Gartner G00255302, Sept 6, 2013,
https://www.gartner.com/doc/2585915/private-cloud-matures-hybrid-cloud

[7] https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/

[8] https://www.gartner.com/doc/3000017/market-guide-selfservice-data-preparation

[9] Kimball Design Tips - http://www.kimballgroup.com/category/articles-design-tips/

[10] Xing Luna Dong, Divesh Srivastava Big Data Integration, Morgan & Claybook Publishers, Feb 2015 https://books.google.co.in/books/about/Big_Data_Integration.html?id=p7d1BwAAQBAJ&redir_esc=y. An ICDE tutorial was initially published in 2013 and is part of the VLDB endowment: http://www.vldb.org/pvldb/vol6/p1188-srivastava.pdf

[11] Flach, P. and Savnik I., Database dependency discovery: a machine learning approach, Journal of AI Com., 12(3):139–160, 1999

[12] Z Abedjan et.al., DFD: Efficient Functional Dependency Discovery, CIKM 2014

[13] A. Rahman, A Novel Machine Learning Approach Toward Quality Assessment of Sensor Data IEEE Sensors Journal, Vol 14(4), April 2014.

[14], J. Freire et.al., Data Polygamy: The Many-Many Relationships among Urban Spatio-Temporal Data Sets SIGMOD 2016

[15] T. Dasu, J. M. Loh, and D. Srivastava. Empirical glitch explanations. In KDD, pages 572–581, 2014.

[16] D.A. Cohn, L. Atlas, and R.E. Ladner, Improving Generalization with Active Learning, Machine Learning, vol. 15, no. 2, pp. 201- 221, 1994.

[17] Z Abedjan et.al., Detecting Data Errors: Where are we and what needs to be done?, VLDB 2016

[18] Raman, A. Retail-data quality: evidence, causes, costs, and fixes (2000). Technology in Society, 22, 97–109. dx.doi.org/10.1016/S0160-791X(99)00037-8

www.persistent.com

[19] M. Hammer and J. Champy, Reingeneering the corporation: A Manifesto for Business Revolution. New York, Harper Collins, 1993.

[20] D.B. Rubin. Multiple imputation after 18+ years. Journal of the American Statistical Association, 91:473—489, 1996.

[21] T. Dasu et. al., FIT to Monitor Feed Quality, VLDB 2015, http://www.vldb.org/pvldb/vol8/p1728-dasu.pdf.

[22] A Gruenheid et. al., Incremental Record Linkage, VLDB 2014, http://www.vldb.org/pvldb/vol7/p697-gruenheid.pdf.

[23] Jeff Dean, Numbers you should know, slides 14 to 19 in presentation https://static.googleusercontent.com/media/research.google.com/en//people/jeff/stanford-295-talk.pdf

[24] David Dewitt and Jim Gray, Parallel Database Systems: the future of High Performance Database Systems, CACM June 1992 Vol 35 No 6, http://people.eecs.berkeley.edu/~brewer/cs262/5-dewittgray92.pdf.

[25] TPC-H: an ad-hoc, decision support benchmark, http://www.tpc.org/tpch/

[26] Beyond SQL: Speeding up Spark with DataFrames http://www.slideshare.net/databricks/spark-sqlsse2015public

[27] S. Agarwal, B. Mozafari, A.Panda, H. Milner, S. Madden, I. Stoica, BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data, ACM Eurosys 2013, Prague, Czech Republic.

[28] http://kylin.apache.org/

[29] http://druid.io/druid.html

[30] https://lens.apache.org/

[31] http://www.atscale.com/

[32] https://nifi.apache.org/

[33] https://storm.apache.org/

[34] https://kafka.apache.org/

[35] http://airbnb.io/projects/airflow/

[36] Martin Fowler and Rebecca Parsons. Domain-Specific Languages. Addison-Wesley, 2011.

www.persistent.com

PERSISTENT

About Persistent Systems

Persistent Systems (BSE & NSE: PERSISTENT) builds software that drives our customers' business; enterprises and software product companies with software at the core of their digital transformation. For more information, please visit: www.persistent.com

India
Persistent Systems Limited
Bhageerath, 402,
Senapati Bapat Road
Pune 411016.
Tel: +91 (20) 6703 0000
Fax: +91 (20) 6703 0009

USA
Persistent Systems, Inc.
2055 Laurelwood Road, Suite 210
Santa Clara, CA 95054
Tel: +1 (408) 216 7010
Fax: +1 (408) 451 9177
Email: info@persistent.com

www.persistent.com