Designing a Logical Data Warehouse

A Technical Whitepaper

Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

February 2016

Sponsored by

redhat
Table of Contents

1 Management Summary 1
2 The Classic Data Warehouse Architecture 2
3 The Logical Data Warehouse Architecture 5
4 Four Layers of Views 6
5 Migrating to a Logical Data Warehouse Architecture 7
   Step 1: Import Source Systems 8
   Step 2: Redirect Data Consumers 8
   Step 3: Remove Derived Data 9
   Step 4: Migrate ETL Logic to Views 10
   Step 5: Remove Obsolete Data Mart Tables 11
   Step 6: Identity Common View Definitions 11
   Step 7: Define Enterprise Data Views 12
   Step 8: Make Room for Self-Service Usage 13
   Step 9: Optimize Performance 14
   Step 10: Access Source Systems 14

6 Implementing a Logical Data Warehouse Architecture in a Greenfield Situation 15
   Step 1: Developing Staging and Data Repository 15
   Step 2: Import Source Systems 16
   Step 3: Define Enterprise Data Layer Views 17
   Step 4: Define Data Consumption Layer Views 17
   Step 5: Develop Reports 19
   Step 6: Define Shared Specifications Layer Views 19
   Step 7: Define Areas for Different Forms of Usage 19
   Step 8: Optimize Performance 20

7 Closing Remarks 20
8 Overview of Red Hat JBoss Data Virtualization 22
   About the Author Rick F. van der Lans 24
   About Red Hat, Inc. 24
1 Management Summary

The classic data warehouse architecture has served many organizations well over the last 25+ years. What’s typical about this architecture is that it consists of a chain of databases, such as a staging area, a central data warehouse, and several data marts, and ETL (Extract Transform Load) programs are used to push data through the chain from one database to another. But as with so many things, the business requirements have changed over time. What was acceptable 15 to 25 years ago, isn’t anymore. Currently, the classic data warehouse architecture has the following drawbacks:

- Limited flexibility
- Limited support for operational business intelligence
- Complex incorporation of big data
- Restricted self-service BI
- Non-trivial support for Bi-modal BI
- Complex import of external data
- Duplication of data
- Diminished data quality

Lately, many new technologies have become available to develop more agile architectures, and one of them is data virtualization technology. This whitepaper describes the design of a logical data warehouse architecture with Red Hat JBoss Data Virtualization Server (JDV), which is based on the community project Teiid. The logical data warehouse architecture is a more agile architecture. It allows BI specialists to develop new reports and change existing ones more quickly; it’s easier to include new data sources, such as big data and external data; and allowing users to see operational data is less difficult to implement, because in the logical data warehouse architecture no chain of databases is developed, or in other words, less duplication of data is required.

Two approaches to develop a logical data warehouse architecture with JDV are described in this whitepaper. The first approach deals with the migration of an existing classic data warehouse architecture to a logical data warehouse architecture. This evolutionary approach helps organizations to migrate step by step without little to no disturbance to the current BI workload. The second approach focuses on how to setup a logical data warehouse architecture in a greenfield situation.

Note, readers are recommended to read the whitepaper Designing a Data Virtualization Environment; A Step-By-Step Approach as well, because it describes and explains in general how to design and develop data virtualization environments based on JDV.

---

The Classic Data Warehouse Architecture

Introduction — The data warehouse is not a new concept. Most organizations have developed a data warehouse to support their reporting and analytics. Several definitions exist, and a very popular one comes from Bill Inmon: “A data warehouse is a subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management’s decision-making process.”

Generally, a data warehouse doesn’t stand in isolation, but is a database within a larger chain or network of databases all together forming a data warehouse architecture; see Figure 1. Other databases commonly making up this architecture are a staging area, an operational data store, and several data marts. The data processed by this architecture comes from multiple source systems. ETL (Extract Transform Load) programs are used to copy data from one database to another. The data marts are the databases accessed by users to satisfy their information and reporting needs.

This architecture has served many organizations well over the last 25+ years. For a long time, the classic data warehouse architecture was the right one based on the state of hardware and software technology. But is it still the right one given all the new business requirements and new available technologies? For some organizations this may still be the case, but for many the classic data warehouse architecture has the following drawbacks and limitations:

Limited Flexibility — Due to the fact that the architecture consists of a chain of interrelated databases and programs, its flexibility is somewhat limited. Even the simplest change of a report (such as adding a new column or dimension) can lead to several changes throughout the entire architecture; it can lead to changes of the tables in the data marts and the data warehouse itself and also to changes in the ETL programs. Also, if the definitions of business concepts alter, then it’s all hands on deck to implement the required changes throughout the entire architecture. Adding a new source system can also be a major undertaking, because it leads to modifications of the ETL programs and the existing database structures.

Limited Support for Operational Business Intelligence — More and more users need to make decisions on data that is 100% up to date or close to it. They need to see operational data, not yesterday’s data. Especially the workforce and operational managers need 100% up-to-date data. In the classic data warehouse architecture, data has to be copied three or four times from one database to another in order to get from the source systems to the reports. This intricate copying process is not completed in just a few microseconds; the chain is just too long. Supporting operational business intelligence involves a major redesign of the classic architecture. It has to be simplified by removing data stores and minimizing the number of copy steps.

Complex Incorporation of Big Data — The need to incorporate data from big data systems, such as sensor data and social media data, in the classic architecture is rapidly increasing. But how must big data be incorporated in the classic data warehouse architecture? Big data systems can’t be treated as ordinary source systems. Imagine that all the new data from a big data system must be periodically copied to the staging area, onwards to the data warehouse, and finally aggregated versions of the big data must be copied to several data marts. This approach is not recommended for the following reasons:

- High data storage costs for storing multiple copies of big data
- Transmitting big data from one database to another can take a long time
- Loading big data several times can be time-consuming
- Enlarging existing databases with big data can lead to query performance degradation
- Backup and recovery can become slow and resource intensive

Big data systems have to be processed differently. They can’t be treated as ordinary source systems.

Restricted Self-Service BI — The last years many self-service BI tools, such as QlikView, Tableau, and Tibco’s Spotfire, have become available. Their intuitive, graphical, and easy-to-use interfaces make it possible for business users and decision makers with minimal or no IT background to develop their own reports and to analyze the data in any way they want. These tools give them the freedom to analyze.

The value of self-service BI to business users has been heralded over and over again. But not everything is entirely rosy. Initially, all these tools look easy to use, but a study by Wayne Eckerson7 shows some drawbacks. To quote the report: 64% of the organizations struggles with self-service BI, giving their self-service BI initiatives a grade of “average” or lower, with 29% rating self-service BI as “fair” or “poor.” Clearly, deploying self-service BI tools is not without problems.

In a way, this form of self-service usage clashes with the more traditional forms of data usage for which IT specialists have initially developed the classic data warehouse architecture. Commonly, self-service users are only allowed to access one or two data marts. But when IT confines them to use a particular data mart, their self-serviceness is still restricted by what data is included in this data mart, how the data is defined, and which relationships are implemented. They are still not fully free to analyze whatever they want.

Non-Trivial Support for Bi-modal BI — The Bi-modal concept4 has been introduced by Gartner. Bi-modal refers to the two modes of IT development. Mode 1 is the classic form of IT development where every system must be reliable, predictable, and safe. Systems must be formally tested, governed, managed, and they must be auditable, and so on. Mode 2 relates to the more agile development styles that focus more on speed and agility. This same distinction can be made for the development of reports and analytics. Some reports must be reliable, predictable, tested, governed, reproducible, and so on. This is typical mode 1 development, whereas self-service BI is clearly an example of mode 2 development, because with self-service BI it’s all about fast development and agility.

The challenge is to unite these two forms of development. First, results coming from reports developed in mode 1 must be consistent with the results of reports developed using a mode 2 approach. Second,

---

1 W. Eckerson, The Promise of Self-Service BI, April 2013, see http://insideanalysis.com/2013/04/the-promise-of-self-service-bi
reports developed in mode 2 may must be migrated to a mode 1 environment; this is sometimes called industrialization. Practice has shown that this is not evident with a classic data warehouse architecture. In most cases these two modes remain independent of each other.

Organizations have to support both forms of development. In a Gartner survey\(^5\), 45% of CIOs state they currently have a mode 2 operation and by 2017 they predict that 75% of IT organizations will have a Bi-modal capability, meaning they support both modes.

**Complex Import of External Data** — More and more business analysts and data scientists don’t restrict themselves anymore to internally produced data stored in the source systems managed by IT. For their analysis they use all the data they can lay their hands on and that includes external data sources. Especially tech-savvy analysts obtain data from the internet (such as research results), access social media data, analyze open data and public data, get files with analysis results from colleagues, and so on. They mix this external data with internal data to get the most complete and accurate business insights.

Every day more files with open data become publicly available. Examples are data sets containing weather-related data, medical data, pollution data, socio-demographic data, crime data, airport data, vehicle collision data, and so on. Especially governments make mountains of valuable data available. At the time of writing 189,920 data sets with US government\(^6\) open data are available.

The challenge is how to integrate external data in the classic data warehouse architecture. If it’s treated as any other data source, these external data sources must be added to the left of the architecture. Data structures in the data warehouse and in multiple data marts must be added and many new ETL programs must be developed to process this external data. This requires a considerable amount of development time. Currently, users don’t have time to wait. They want to integrate these new data sources themselves, and thus bypass all the existing business rules. In addition, many users in such a situation are reinventing the wheel over and over again, because they all develop their own integration solutions.

**Duplication of Data** — A vast amount of duplicate data is stored in classic data warehouse architectures. For example, the content of most data marts is derived from that of the data warehouse, implying that 100% of the data in the data marts is redundant. Even a data staging area and a data warehouse contain masses of overlapping data. Additionally, inside each database a lot of duplicate data is stored in the form of indexes, materialized query tables, columns and tables with aggregated data, and so on. Obviously, there are several reasons for storing all this duplicate data, but the performance of queries, reports, and ETL scripts is a very dominant one.

Storage is not that expensive anymore, so what’s the issue? The issue is agility. The more duplicate data is stored, the less flexible the architecture is. Every change requires an additional change on the duplicate data. There will be costs involved in keeping duplicate data synchronized. Data warehouse architectures can be considerably simplified by removing most of the duplicate data.

**Diminished Data Quality** — When multiple copies of the same data exist, there is always a risk that data becomes inconsistent. In other words, storing duplicate data, which, as indicated, is done extensively in

\(^5\) P. Sondergaard, Bimodal Business Leads to Bimodal IT, December 2014; see http://blogs.gartner.com/peter‐sondergaard/bimodal‐business‐leads‐to‐bimodal‐it/

classic data warehouse architectures, involves a data quality risk. David Loshin\textsuperscript{7} formulates it as follows: “[...] Each time data is copied, it is also subjected to any number of data transformations, each of which provides an opportunity for the introduction of data errors. Each subsequent copy resembles the original data source less. Copying data can only lead to entropy and inconsistency.” In every data warehouse architecture one of the goals must be to minimize duplication of data to minimize data quality risks.

3 The Logical Data Warehouse Architecture

The logical data warehouse architecture is an alternative to the classic data warehouse architecture. It’s a flexible architecture for designing, developing, and operating a system that supports all forms of business intelligence. The essence of this architecture is that data consumers and data sources are decoupled from each other and in which meta data specifications are shared; see Figure 2.

Definition of Logical Data Warehouse Architecture – Definition of the logical data warehouse architecture:

The logical data warehouse architecture delivers data and meta data to data consumers in support of decision-making, reporting, and data retrieval; whereby data and meta data stores are decoupled from the data consumers through a meta data driven layer to increase flexibility; and whereby data and meta data are presented in a subject-oriented, integrated, time-variant, and reproducible style.

\textsuperscript{7} D. Loshin, Effecting Data Quality Improvement through Data Virtualization, Knowledge Integrity, Inc., June 2010.
Note that Bill Inmon’s definition (see Section 2) of data warehouse applies to the logical data warehouse. Nowhere in this definition does it state that the collection of data must be stored. The definition of the logical data warehouse architecture is clearly based on Inmon’s definition, except that in the tail of the definition the term non-volatile is replaced by the term reproducible. Inmon uses the term non-volatile to indicate that many users want to see consistent report results over time. Practically speaking, when users run a report a few times a day, they may want to see identical results. However, databases can be designed in such a way that they are volatile and still present consistent report results. Therefore, in the definition of the logical data warehouse architecture, the term reproducible is used instead.

**Data Virtualization Technology** — Several technologies exist for developing a logical data warehouse, such as database server views, enterprise service busses (ESB), data grids, in-memory database servers, and data virtualization servers. All these technologies have their merits, but data virtualization servers have the best fit. That’s why this whitepaper describes how to design and develop a logical data warehouse architecture with Red Hat’s data virtualization product called *Red Hat JBoss Data Virtualization Server* (JDV).

But note that the logical data warehouse is only one use case for data virtualization. Products such as JDV can also be used to ease the development of service interfaces to applications, to make access to data easier for users in general, to implement an overall data security layer on several data sources, to translate API’s for data access, and so on.

## 4 Four Layers of Views

The key building block of JDV is the view. Each view has a data structure and a definition that defines the virtual content. Practice has shown that when defining a logical data warehouse, multiple layers of views are required. In fact, the general recommendation is to work with at least four layers of views; see Figure 3. This is in line with the whitepaper *Designing a Data Virtualization Environment; A Step-By-Step Approach*.

In this architecture each view layer has its purpose (starting from the bottom):

- **Virtual Base Layer**: The virtual base layer contains views that show the data stored in source systems. For each physical table or file in a source system one or more views are defined. Each view definition may contain cleansing specifications to improve the data quality of the source systems. Besides correcting the data, the virtual contents of such a view is identical to the contents of the source system.

- **Enterprise Data Layer**: Views at the second layer present an integrated view of all the data in the source systems, hence the name enterprise data layer. The structure of each view is “neutral.” In other words, it’s not aimed at the needs of one data consumer, but at supporting as many forms of usage as possible. If possible, each view is structured according to the third normal form.

- **Shared Specifications Layer**: To avoid too many duplicate and possibly inconsistent specifications in the data consumption views, the third layer contains shared specifications. The purpose of this layer is to make the environment as agile as possible by avoiding duplication of specifications. Optionally, the shared specifications layer contains authorization specifications indicating e.g.
which user is allowed to use which view.

- **Data Consumption Layer**: The structure of each view at the data consumption layer is intended to simplify data access for data consumers. For example, for some data consumers it may be useful to see the data organized as a star schema, whereas others prefer to see all the data they require in one view consisting of a large set of columns. Filters, projections, transformations, and aggregations are specified at this data consumption layer to show data consumers only relevant data at the right aggregation level and in the right form.

![Logical Data Warehouse Architecture](image)

**Figure 3** A logical data warehouse architecture designed with data virtualization technology consists of four layers of views.

To compare these layers with the databases in the classic data warehouse architecture, the views in the data consumption together with those in the shared specifications layer relate very closely to the data marts of the classic data warehouse architecture, except that the data marts in the classic architecture are physical databases, whereas the views can be seen as *virtual data marts*. Likewise, the enterprise data layer corresponds loosely with the data warehouse of the classic data warehouse architecture.

## 5 Migrating to a Logical Data Warehouse Architecture

Many organizations already have a data warehouse architecture in place. Databases have been designed, ETL scripts have been developed, and reports are up and running. This section describes how an organization can migrate step by step from a classic to a logical data warehouse architecture without little
to no disturbance for the existing data consumers. It’s an evolutionary approach instead of a revolutionary approach. When, however, an organization wants to develop a new logical data warehouse architecture from scratch to support their business intelligence systems, a different set of steps is recommended. These steps are described in Section 6.

**Step 1: Import Source Systems** — It’s assumed that the JDV data virtualization server has been installed. Visit the Red Hat website that describes the installation procedures in detail for several operating systems.

Determine which tables in the data marts of the existing architecture are still in use. Define for each of these tables still in use a JDV source table; see Figure 4. Also, define a view on each of these source tables. The source tables and views defined in this step are part of the virtual base layer. These source tables and views must have a one-to-one correspondence with the tables in the data marts. So, no views are defined with complex transformations, joins or aggregations. Next, define on each view defined in the virtual base layer a view in the data consumption layer. Again, the definitions of these data consumption views are simple and must have a one-to-one correspondence with the virtual base views.

Existing reports are not changed in this step. They still access the same data marts of the classic data warehouse architecture as before Step 1.

**Step 2: Redirect Data Consumers** — In this step, one by one, data consumers are redirected to access the data marts via the views defined in the data consumption layer; see Figure 5. Instead of, for example, using a JDBC driver to access some SQL-based physical data mart, a data consumer now uses JDV’s JDBC driver to access JDV that accesses that same SQL-based data mart.
The effect of this step is that data consumers still access the same data in the same data marts and the query results remain unchanged, because the views defined in Step 1 have the same data structure and virtual content as the source tables originally accessed by the data consumers. So, JDV receives queries from the data consumers and passes them on to the underlying data marts unchanged. Results are returned to the data consumers. The only difference is that the queries are now processed via JDV.

**Figure 5** In Step 2 data consumers are redirected to access views defined in the data consumption layer.

### Step 3: Remove Derived Data

It’s common in classic data warehouse architectures to improve query performance by adding columns and tables with derived data to the data marts. For example, in the department table the total number of employees is stored and in another table the difference between products sold and products returned is stored.

In Step 3, a start is made with simplifying the overall architecture by removing as much of the redundant derived data stored in the data marts as possible; see Figure 6. The effect is that the definitions of the views in the data consumer layer change. For example, after Step 2, the number of employees per department is still retrieved from the corresponding column in a virtual base layer view, which retrieves it via a source table from a column in a data mart. In Step 3, the formula to calculate the number of employees is added to the definition of the view in the data consumption layer. The logic to calculate the number of employees is derived from the ETL script responsible for calculating the derived data. The ETL script must be adapted as well, because the logic to calculate the derived data must be removed. In addition, the derived column or table must be removed from the data mart (if no other data consumer uses the derived data anymore).
It’s important that the removal of derived data is transparent to the data consumers. Their reports must still show the same results.

**Step 4: Migrate ETL Logic to Views** — Most tables in data marts contain data derived from the data warehouse. In Step 4 the process to simplify the overall architecture is continued by minimizing the use of data marts; see Figure 7. The views in the data consumption layer are redefined to access the original data in the data warehouse. In other words, a view that accesses tables in a data mart is now redirected to access tables in the data warehouse containing the same data, although organized in a different set of tables and columns.

To make this work, first source tables and views are defined within the virtual base layer in the same way views were defined for the tables in the data marts in Step 1. They all have a one-to-one correspondence with the real tables in the data warehouse. Next, the logic used in the ETL scripts to load the data mart tables is implemented in the existing views in the data consumption layer. These ETL scripts contain the logic used to transform the data from the data warehouse tables to the data mart tables. The same logic is used in the definitions of the data consumption views. All this logic has to be translated to guarantee that the views still have the same data structure and the same virtual contents as the original data mart tables.

Like the previous step, these changes must be transparent to the data consumers. Their reports must still show the same results.
Step 5: Remove Obsolete Data Mart Tables — When data mart tables are no longer being accessed, because the queries have been redirected to tables in the data warehouse, they've become obsolete and can be removed. If all the tables of a data mart have been removed, the entire data mart can be removed; see Figure 8. In addition, the ETL scripts loading the data marts can be removed as well.

Until this step, nothing in the existing reports is changed. The reports still show the same results as before the logical data warehouse architecture was introduced. The performance of the reports, however, may have changed. It may have increased or decreased depending on the database and hardware technology used.

Step 6: Identify Common View Definitions — This step is like a clean-up activity. Multiple views (possibly coming from different data marts) may have comparable or maybe even identical definitions, because the structures of the views defined in Step 1 are identical to the structures of the original tables in the data marts and because they now extract data from the data warehouse. For example, two data marts may have contained the same product and customer dimension tables. The consequence is that after Step 1 two views with product data and two views with shop data are defined in the data consumer layer.

In Step 6, common view definitions in the data consumption layer are moved to views defined in the shared specifications layer; see Figure 10. It could even be that two views in the data consumption layer are 100% identical and can be merged. After Step 6, the entire set of view definitions is easier to maintain, because duplicate specifications are minimized.
Step 7: Define Enterprise Data Views — The data structures of all the views defined up until Step 6 are all directly or indirectly derived from the data structures of the existing data marts and data warehouse. These data structures may not be perfect. It is recommended to introduce well-defined views on the enterprise data layer that have neutral and normalized data structures. Next, these views are mapped to the views in the virtual base layer, and finally, the views in the shared specifications layer are redefined to work with the new views in the enterprise data layer.

Introducing an enterprise data layer can have the effect that definitions of business concepts are slightly adapted. If this happens, view definitions are changed as well as reports. Business users must be made aware of this.
Step 8: Make Room for Self-Service Usage — In the whitepaper Designing a Data Virtualization Environment; A Step-By-Step Approach the concepts of traditional usage and self-service usage are described. Traditional usage correlates to mode 1 development and self-service usage to mode 2; see Section 2. The views defined in the previous steps all belong to the area for traditional usage. For self-service usage a special area must be defined. This is an area within the data consumption layer and the shared specifications layer in which business users are allowed to develop their own views. In other words, the area principally exists to support self-service analytics and mode 2 development.

By only allowing business analysts and users to develop views on top of the enterprise data layer views, two problems are solved. First, because all the views on the data consumption layer retrieve data via the enterprise data layer, all share the same specifications. This increases the level of report consistency across mode 1 and 2 reports. Second, when a report developed by a particular business user must become available to other users (industrialization), the related views must simply be moved from the self-service usage area into the area for traditional usage.
Step 9: Optimize Performance — The performance of queries can be improved by, for example, defining caches, optimizing indexes, and updating statistical information. See the whitepaper *Designing a Data Virtualization Environment; A Step-By-Step Approach*¹ for some general guidelines.

Step 10: Access Source Systems — Until the previous step, all the reports retrieve their data from the data warehouse. Study whether it’s possible to remove some of the query workload on the data warehouse to the source systems. Especially some of the newer source systems may be able to handle the workload coming from the logical data warehouse. For users interested in working with operational data, this can be a great benefit.

If it’s possible to query the source systems, it’s important that the logical data warehouse architecture handles the cleansing of the operational data. This can be done by implementing cleansing rules in the views in the virtual base layer; see the whitepaper *Designing a Data Virtualization Environment; A Step-By-Step Approach*¹ for some guidelines and examples.

Final Remarks — The key benefit of this step-by-step approach of migrating to a logical data warehouse is that it is a seamless migration; it’s not a revolutionary, but an evolutionary approach. Existing data consumers won’t have to notice that changes are made to the system. In addition, the logical data warehouse architecture is significantly more agile. New reports and changes to existing reports can be implemented much faster.
In Step 10, some views are redirected to the operational systems. This means that it is no longer required to store some of the source system data in the data warehouse. But if parts of the data warehouse are dismantled, can we still call this database a data warehouse? Maybe another term, such as *data repository*, is more fitting. This topic is described in more detail in the next section.

### 6 Implementing a Logical Data Warehouse in a Greenfield Situation

When organizations don’t have a data warehouse and if they want to develop a logical data warehouse from scratch, a different step-by-step approach is recommended. This other set of steps is described in this section.

**Step 1: Develop Staging Area and Data Repository** — The first step for developing a logical data warehouse architecture begins with developing two databases, a staging area and a so-called data repository; see Figure 11.

![Figure 11](image.png)

The *staging area* in this architecture has the same use as that in a classic data warehouse architecture. It’s a database to store changes to the data in the source systems temporarily. Use ETL, ELT, or data replication for loading and updating the staging area. After the data has been copied onwards to the data repository, it can be removed from the staging area.

The *data repository* in this architecture is a database in which current and historical operational data is kept for longer periods of time. Many views in the virtual base layer are defined on the data repository’s tables. For several reasons this data repository is required:

- Not all source systems keep track of historical data. The data repository holds current and historical data to support historical reporting and analysis.
- Not all source systems can handle the required query workload. Performance may be too slow, or the queries may cause too much interference on the transaction workload.
- Not all source systems have the availability needed by the BI users. For example, some old source systems are still stopped in the evening and restarted in the morning. The data repository can offer a 24x7 availability.

Note that a data repository in this architecture is this not the same as a data warehouse. According to Bill Inmon’s definition, the data kept in a data warehouse is subject-oriented, integrated, time-variant, and nonvolatile. Data in the data repository isn’t always subject-oriented nor integrated. It’s the views in the enterprise data layer that are responsible for data integration. The time-variant and non-volatile requirements may apply. But the biggest difference between the two is that in a logical data warehouse
architecture the data virtualization layer offers all the data for reporting and analytics. However, not all that data is stored in the data repository. In the classic data warehouse architecture most (if not all) of the data used for reporting is stored in the data warehouse.

Two important decisions must be made when designing this part of the architecture. First of all, how much data cleansing is required, and second, how much data integration must take place? With respect to the first decision, implement most of the cleansing operations in the two loading steps. Each cleansing operation that isn’t implemented in these steps, must be implemented in views of the logical data warehouse architecture.

The guideline is to handle data cleansing upstream, as close to the source systems as possible. Cleansing data downstream (closer to the reports) is more complex and can be quite resource intensive. In an ideal world, data cleansing is fully handled by the source systems themselves. It should be impossible for users to enter incorrect data. And if incorrect data is entered, somehow the source systems must resolve that issue before the data is copied to the staging area. The effect would be that data stored in the data repository is cleansed, transformed, and normalized.

With respect to the design of tables in the data repository, normalize them: store each fact only once. If the table structures in the source systems are not really normalized, it’s recommended to let the ETL scripts transform the data into more relational structures. For example, if a table in a source system contains a repeating group, such as all the telephone numbers of an employee, a separate table must be created in the data repository for these telephone numbers. In this example, the effect is that data from that employee table in the source system is copied to two tables in the data repository.

The tables in the data repository must have a structure that can hold multiple versions of the same business object. For example, the customer table must be able to hold the current addresses of the customer as well as their previous addresses. In other words, the tables must be able to store historical data and the ETL scripts must know how to load new data and how to turn existing data into historical data. Several solutions exist to handle this correctly and efficiently.

**Step 2: Import Source Systems** — Install JDV and import all the tables from the data repository that may be needed for the first set of reports; see Figure 12. Define a source table and a view in the virtual base layer for each of the data repository tables.

If data from the data repository must be checked on correctness, then cleansing rules must be included in the views of the virtual base layer. These views are then be responsible for improving the data quality level and report consistency.

For some cleansing rules special tables must be created to help with cleansing incorrect values. Store these tables in the data repository. Data in these special tables is managed by the logical data warehouse architecture.
Step 3: Define Enterprise Data Layer Views – In the enterprise data layer, define views that represent business objects or properties of business objects; see Figure 13. Examples of business objects are customers, products, and invoices. This may require that data from multiple virtual base layer views is joined into one larger enterprise data layer view. For example, when customer data is distributed over multiple source systems, it’s integrated in the enterprise data layer to form one view showing all the customers with all their data.

Sometimes data from multiple data repository tables has to be integrated to develop an integrated view of a business object. For example, customer data may be distributed across several data repository tables. In the previous step, for each of these tables a view is defined in the virtual base layer, but in the enterprise data layer there should only be one view containing customer data. In other words, the enterprise data layer of views is responsible for integrating data and for presenting that data in a more business object-oriented style. This layer contains the data in a neutral or canonical structure. Therefore, this layer is sometimes referred to as the canonical data model.

Step 4: Define Data Consumption Layer Views – Develop the data consumption layer. The views of this layer are structurally aimed at the needs of a specific data consumer or group of data consumers; see Figure 14. For example, it might be that some data consumers work with a reporting tool that requires that tables form a star schema; other consumers may prefer to work with views that are derived from views defined in the second layer by defining filters or aggregations; or, some data consumers don’t want to see historical customer data, but only current data, which means that historical data must be filtered out.
Figure 13  In Step 3, views are defined in the enterprise data layer that represent business objects.

Figure 14  In Step 4, the views of the data consumption layer are defined to fulfill the reporting needs of data consumers.
The views in the data consumption layer can be regarded as virtual data marts. In physical data marts (as part of the classic data warehouse architecture) the structure of each table is also aimed at supporting the needs of the users and reports. As indicated, the same applies for the views making up the virtual data marts.

**Step 5: Develop Reports** — Develop reports on top of the views in the data consumption layer; see Figure 15. Most reporting tools also allow developers to enter specifications for filtering, joining, aggregating, and manipulating the data. Try to minimize this and implement these specifications in JDV so that they can be shared amongst reports and different reporting tools. It’s not uncommon that new business insights arise when new reports are developed. In this case, it may be required to apply changes to this layer of views.

![Figure 15](image)

*Figure 15 In Step 5 reports are developed on the views in the data consumption layer.*

**Step 6: Define Shared Specifications Layer Views** — As indicated, views in the data consumption layer are designed to support data consumer needs. These views are easy to change and if the right design techniques are applied, many specifications in the view definitions can be reused by implementing them in the shared specifications layer; see Figure 16. Common or shared specifications are removed from the data consumption views and implemented in the views belonging to the shared specifications layer. This step is very similar to Step 6 of Section 5.

**Step 7: Define Areas for Different Forms of Usage** — As with Step 8 of Section 5, to support Bi-modal development, reserve at least two areas for different forms of usage: traditional usage and self-service usage.
Figure 16 In Step 6 develop the reports and modify the views if required.

Step 8: Optimize Performance — The performance of queries can be improved by, for example, defining caches, optimizing indexes, and updating statistical information. See the whitepaper Designing a Data Virtualization Environment; A Step-By-Step Approach for some general guidelines.

7 Closing Remarks

Data Warehouse Augmentation — After completing all the steps, the logical data warehouse gives access to the data residing in the data warehouse or data repository. Nowadays, users also want to access external data sources and combine that external data with data from the data warehouse. The use of data virtualization technology makes it easy to make that external data source accessible through a JDV view to allow the users to analyze that data and combine it with the existing data.

The same applies for big data sources. Some are just too big to copy into the data warehouse. The copying process of all that data would take too much time and would be too expensive. It makes more sense to make the big data source accessible for the business users via JDV for reporting and analysis. As with external data sources this approach allows the business users to transparently combine big data with data warehouse data. They won’t even know they are accessing multiple systems.
This style of *data warehouse augmentation* (sometimes called *data warehouse extension*) is strongly recommended and shows the agility of the logical data warehouse architecture and the JDV technology.

**Vertical or Horizontal?** — Regardless whether a logical data warehouse is developed from scratch or migrated from a classic data warehouse architecture, a decision must be made on whether an entire view layer is designed and developed before work starts on the next layer, or that a limited number of views is defined in a layer before work on the next layer begins. The former approach is called the *horizontal approach* and the latter the *vertical approach*; see Figure 17. Using the terminology of *iterations*, with the vertical approach the iterations are short to very short, and with the horizontal approach the iterations can be long to very long.

![Diagram of horizontal and vertical approaches](image)

**Figure 17** *The horizontal approach is shown on the left and the vertical approach on the right.*

With the horizontal approach a layer must be completed before work starts on the next one. The horizontal approach corresponds with enterprise-wide approaches and waterfall design techniques. The benefit of the horizontal approach is that it’s easier to end up with a set of views that contains no or minimal redundant specifications.

With the vertical approach, just a limited number of views is defined within a layer before work starts on views in the next layer. The vertical approach fits very well with current *agile design techniques* where development must be highly iterative. Also, it works well for modern-day data warehouses where the requirement is to be able to develop more quickly and to change existing reports faster. These are the overall benefits of the vertical approach: high productivity and agile solution. If users need data from some source system, the required views can be defined in a minimal amount of time. In other words, the time to react will be short. In a green field situation, when the vertical approach is used, it won’t take several months before the first reports can be developed, but rather days or weeks. For most logical data warehouse projects, the vertical approach is preferred.

---

8 Overview of Red Hat JBoss Data Virtualization

Red Hat offers the only open source alternatives to enterprises looking to adopt and implement data virtualization. The open source subscription model lowers the cost as adoption barrier, while open community based innovation offers open, non-proprietary option. Following key features and capabilities make JBoss Data Virtualization a worthy choice.

Model Driven Development — JBoss Data Virtualization includes Teiid Designer (see Figure 18), an Eclipse-based graphical tool, which models, analyzes, integrates, and tests multiple data sources to produce relational, XML, and web service views that show business data without programming. You can map from data sources to target formats using a visual tool, as well as resolve semantic differences, create virtual data structures at a physical or logical level, and use declarative interfaces that are compatible with and optimized for your applications.

![Figure 18  JDV includes Teiid Designer, an Eclipse-based graphical tool.](image)

Centralized Data Security — JBoss Data Virtualization gives you the power to manage and monitor data services in a single unified environment and enforce and manage policies and roles across federated data for all data services. You can configure policies for data security, privacy, column-level data masking, and data sanitization of sensitive fields based on user roles. In addition to using the security capabilities of Red Hat JBoss Enterprise Application Platform, the software supports web services security and fine-grained access control for SQL data service and virtual table. In addition to column-level security, you get row-level security at the virtualization layer, which is independent of data sources. Transport and password encryption are available through SSL support.
Performance Optimization — JDV offers support for several advancing caching modes, including materialized views, result set caching, and code table caching that provide dramatic performance improvement. Configurable time-to-live, memory preferences, and updatability options are available for data caching.

Intelligent and automated techniques (e.g., cost- and rule-based query optimizer using information from source introspection, query capabilities, and constraints) include pushdown queries, dependent joins, projection minimization, partitioned aware unions, and copy criteria to optimize data sources join processing, and support for high-performance sub-queries. Comprehensive query trace is supported with manual plan override of automatic strategy selections for optimized query.
About the Author Rick F. van der Lans

Rick F. van der Lans is an independent analyst, consultant, author, and lecturer specializing in data warehousing, business intelligence, big data, database technology, and data virtualization. He works for R20/Consultancy (www.r20.nl), a consultancy company he founded in 1987.

Rick is chairman of the annual European Enterprise Data and Business Intelligence Conference (organized annually in London). He writes for Techtarget.com\(^\text{10}\), B-eye-Network.com\(^\text{11}\) and other websites. He introduced the business intelligence architecture called the Data Delivery Platform in 2009 in a number of articles\(^\text{12}\) all published at B-eye-Network.com. The Data Delivery Platform is an architecture based on data virtualization.

He has written several books. His latest book\(^\text{13}\) Data Virtualization for Business Intelligence Systems was published in 2012. Published in 1987, his popular Introduction to SQL\(^\text{14}\) was the first English book on the market devoted entirely to SQL. After more than twenty five years, this book is still being sold, and has been translated in several languages, including Chinese, German, Italian, and Dutch.

For more information please visit www.r20.nl, or email to rick@r20.nl. You can also get in touch with him via LinkedIn and via Twitter @Rick_vanderlans.

About Red Hat, Inc.

Red Hat is the world’s leading provider of open source solutions, using a community-powered approach to provide reliable and high-performing cloud, virtualization, storage, Linux, and middleware technologies. Red Hat also offers award-winning support, training, and consulting services. Red Hat is an S&P company with more than 70 offices spanning the globe, empowering its customers’ businesses.

\(^{10}\) See http://www.techtarget.com/contributor/Rick-Van-Der-Lans
\(^{11}\) See http://www.b-eye-network.com/channels/5087/articles/
\(^{12}\) See http://www.b-eye-network.com/channels/5087/view/12495
\(^{13}\) R.F. van der Lans, Data Virtualization for Business Intelligence Systems, Morgan Kaufmann Publishers, 2012.