SQL Server Technical Article

Implementing a Microsoft SQL Server Parallel Data Warehouse Using the Kimball Approach

Writer: Warren Thornthwaite

Technical Reviewers: Jesse Fountain, Barbara Kess, Stuart Ozer

Published: June 2011

Applies to: SQL Server 2008 R2

Summary:

This white paper explores how the Kimball approach to architecting and building a data warehouse/business intelligence (DW/BI) system works with Microsoft's Parallel Data Warehouse, and how you would incorporate this new product as the cornerstone of your DW/BI system. For readers who are not familiar with the Kimball approach, we begin with a brief overview of the approach and its key principles. We then explore the Parallel Data Warehouse (PDW) system architecture and discuss its alignment with the Kimball approach. In the last section, we identify key best practices and pitfalls to avoid when building or migrating a large data warehouse to a Microsoft SQL Server PDW system. This includes a look at how Parallel Data Warehouse can work with other Microsoft offerings such as SQL Server 2008 R2, the Microsoft Fast Track Reference Architecture for Data Warehouse, and the new Business Data Warehouse to provide a complete solution for enterprise-class data warehousing.

Copyright

This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 Microsoft. All rights reserved.

About the Author

Warren Thornthwaite started his data warehousing/business intelligence career in 1984 at Metaphor Computer Systems where he worked for eight years implementing major DW/BI systems and managing the consulting organization. After Metaphor, Warren became the Program Director of Stanford University's enterprise data warehouse (EDW) development project. He left Stanford to co-found InfoDynamics LLC, a data warehouse consulting firm in 1994. In 1997, Warren joined WebTV to help build a world class, multi-terabyte customer focused data warehouse. In 2003 he returned to consulting as a member of the Kimball Group with Ralph Kimball.

Warren holds a BA in Communication Studies from the University of Michigan and an MBA in Decision Sciences from the University of Pennsylvania's Wharton School. He is co-author for *The Data Warehouse Lifecycle Toolkit, 2nd Edition* (Wiley, 2008) and *The Microsoft Data Warehouse Toolkit, 2nd Edition* (Wiley, 2011). He teaches Kimball University courses around the world, and is a regular presenter at industry conferences.

The Kimball Group is the source for dimensional DW/BI consulting and education, consistent with our best selling Toolkit book series, Design Tips, and articles. Please visit www.kimballgroup.com for more information.

Contents

Introduction

Back in the 1980s, some of the biggest organizations in the world found themselves dealing with much larger analytic data sets than their mainframe database systems could handle. They turned to massively parallel processing (MPP) systems, which use a divide-and-conquer strategy by spreading the workload across multiple machines and having those machines work on the problem in parallel. These systems were very expensive, but were able to solve the problem.

In the last few years, many more organizations are finding themselves facing the same problem from a data perspective. At the same time, MPP technology that began as highly specialized and out of the price range of the broad market has advanced to enable more affordable and accessible solutions appropriate for companies of all sizes. Microsoft recently released a database appliance product called Microsoft SQL Server 2008 R2 Parallel Data Warehouse (SQL Server PDW) that is targeted at this new, broader market for large-scale data warehouse capabilities. SQL Server PDW uses Microsoft database software and pre-configured commodity hardware to provide MPP functionality at a lower price point than previously possible.

The main goal of this white paper is to help those who are familiar with the Kimball approach, and bumping up against the constraints of a single-server data warehouse database, to understand how their existing environment can be migrated to an MPP environment. At the same time, those without experience using the Kimball approach will be introduced to the methods and their applicability to the SQL Server PDW platform. To accomplish this, we will cover four main topic areas:

- Brief review of the Kimball approach
- SQL Server PDW massively parallel architecture
- Enterprise data warehouse architecture options
- SQL Server PDW implementation guide to build an MPP-based DW/BI system

Section 1: The Kimball Approach

There are a lot of misconceptions about dimensional modeling and the Kimball approach to building a DW/BI system. It's worth reading this section even if you are already familiar with the Kimball approach; you might be surprised at what you learn. The Kimball approach to creating an enterprise data warehouse has several core principles:

1. Follow a proven methodology; we recommend the Kimball Lifecycle.
2. Understand business requirements so you can engage the business, prioritize your efforts, and deliver business value.
3. Design the data warehouse data sets for flexibility, usability, and performance.
4. Build and deliver quick, business process-based increments within an enterprise data framework known as the data warehouse bus matrix.
5. Design and build a DW/BI system architecture based on your business requirements, data volumes, and IT systems environment.
6. Build out the extract, transformation, and loading (ETL) system with standard components to deal with common design patterns found in the analytic data environment.
7. Provide the complete solution, including reports, query tools, applications, portals, documentation, training, and support.

All of these principles are explored at length in *The Data Warehouse Lifecycle Toolkit book, Second Edition* (Wiley, 2008). We will examine a few of them here in detail.

Follow a Proven Methodology: Lifecycle Steps and Tracks

The Kimball Lifecycle is a detailed methodology for designing, developing, and deploying data warehouse/business intelligence systems, as described in *The Data Warehouse Lifecycle Toolkit, Second Edition*. The diagram in Figure 1 summarizes the key steps in the Lifecycle.

The Lifecycle is an iterative approach, with each pass delivering a coherent set of data and an initial set of associated reports and applications. Each pass can typically be completed in 6 to 9 months, depending on the data complexity. Building out the full DW/BI system takes multiple iterations, each one loading a new data subject area, which plugs into the overall enterprise data framework called the *bus matrix*.

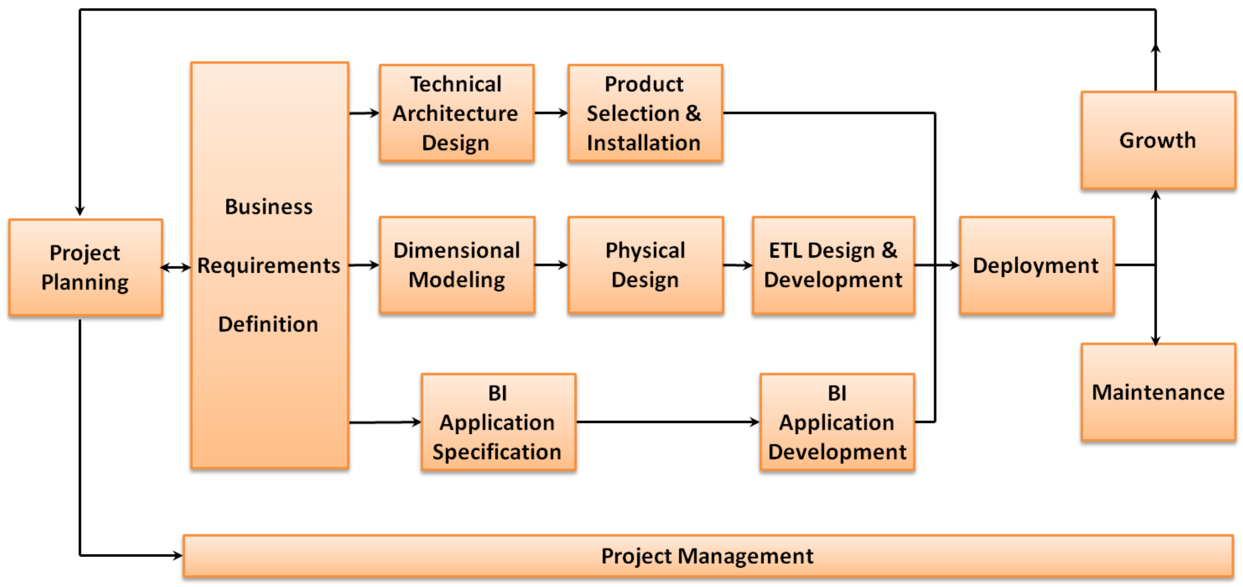

Project Planning
Business Requirements Definition
Technical Architecture Design
Product Selection & Installation
Dimensional Modeling
Physical Design
ETL Design & Development
BI Application Specification
BI Application Development
Deployment
Growth
Maintenance
Project Management

Figure 1: The Kimball DW/BI Lifecycle

The Kimball approach starts with understanding business requirements and determining how best to add value to the organization. The organization must agree on what the value of this data is before deciding to build a data warehouse to hold it. For example, capturing web browsing activity may allow you to gain deep insight into your customers' behaviors and preferences, opening up new ways to better meet their needs. If you clearly identify and deliver business value, the resulting impact on the business should easily justify your SQL Server PDW investment.

The ideal starting point for most organizations is to perform an initial set of interviews to gather and prioritize enterprise-wide high level business requirements for information. The result is a priority ordered list of business processes that generate data, along with high value analytic opportunities supported by that data.

Once the list of business processes and associated opportunities has been identified and prioritized, the next step is to take the highest priority business process and gather detailed business requirements related to it. This second pass at requirements is much more focused on understanding the specifics around the required data source, including attributes, definitions, business rules, data quality, and the range of analytics and applications that will be built on top of this data set.

Once these detailed requirements are in place, the Lifecycle moves into the implementation phase beginning with design steps across three different tracks. The top track in Figure 1 is the technology track. The main goal here is to identify the functionality and associated tools needed to meet the identified business requirements.

The middle track in Figure 1 is the data track. The initial step is to define the logical data model needed to support the analytic requirements. In the Kimball approach, this is a dimensional model. Once the logical model is in place, the team can build the target database in the database environment. The nature of the physical model depends on the target platform. Many database products work best with a physical dimensional model, although a more normalized model may make sense on a few platforms. The last data step is to create the ETL system that will populate the target database as required. The ETL system is a significant effort, often consuming a majority of the initial project resources.

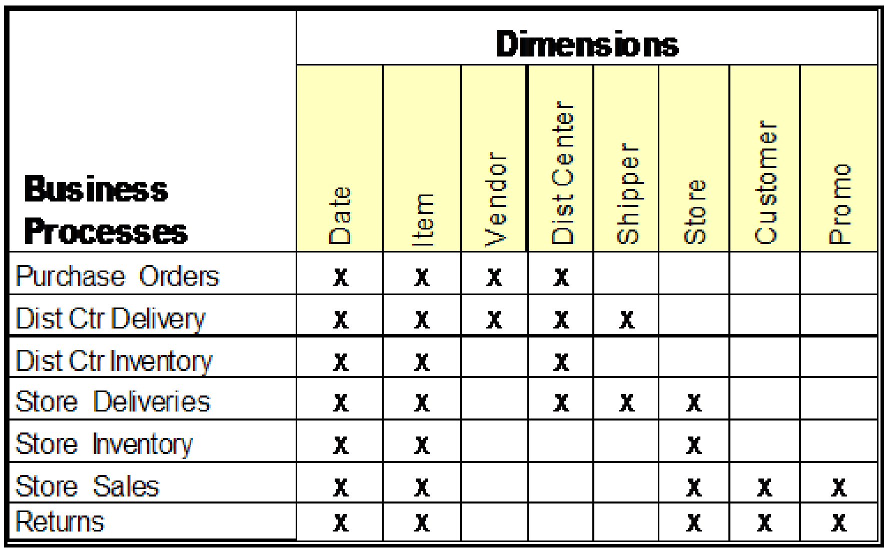
The bottom track in Figure 1 is concerned with the BI applications: the initial set of reports and analyses that will deliver business value to the organization. This track is split into two steps; the first is the design step where a small set of high value applications and reports are identified and specified in detail. The second step is the actual implementation where these applications and reports are built. This step often has to wait until near the end of the ETL development when data is actually available in the database. Note that these reports and analyses only serve as a starting point that helps solve a high-value problem. The dimensional model is not limited in any way to this subset of reports.

Once the three implementation tracks are complete, the Lifecycle comes back together to deploy the query tools, reports, and applications to the user community. This involves extensive communication, training, documentation, and support.

The next Lifecycle iteration usually begins during the deployment of the previous iteration, when the business analysts and designers can gather detailed requirements for the next highest priority business process, create the associated dimensional model, and start the process all over again. The Lifecycle's incremental approach is a fundamental element that delivers business value in a short timeframe, while building a long-term, enterprise information resource.

The Data Warehouse Bus Matrix

The Enterprise Data Warehouse Bus Matrix is the data framework for the enterprise data warehouse. Figure 2 shows a simplified version of a bus matrix for a retail organization with a customer affinity program.



	Dimensions							
Business Processes	Date	Item	Vendor	Dist Center	Shipper	Store	Customer	Promo
Purchase Orders	X	X	X	X				
Dist Ctr Delivery	X	X	X	X	X			
Dist Ctr Inventory	X	X		X				
Store Deliveries	X	X		X	X	X		
Store Inventory	X	X				X		
Store Sales	X	X				X	X	X
Returns	X	X				X	X	X

Figure 2: Example bus matrix

The row headers down the left side of the bus matrix define the organization’s primary business processes. A good way to think about these business processes is to think about the value chain of the organization. What are the activities in which your organization engages to provide your customers with the goods or services they need?

The column headers of the bus matrix represent the primary objects that participate in those business processes. Typical examples include customer, account, product, store, employee, patient, and date. These objects are called dimensions, and they must be pre-integrated to work with all the relevant business processes.

This pre-integration is called conforming, and it involves the hard organizational data governance work of deciding the standard names, descriptions, mappings, hierarchies, and business rules that will apply across the DW/BI system. This is essentially what master data management (MDM) is meant to do, and the DW/BI system benefits greatly from a separate MDM effort. In the absence of an MDM system, the DW/BI team must shoulder this dimension conforming effort. Once this definitional work is done, these dimensions become reusable components that can be applied to every associated business process. Most importantly, the conformed dimensions are the necessary framework for *integration*, where the results from two or more business process can be combined into a single BI deliverable.

Each row on the bus matrix is a business process data set that corresponds to a unit of work for the ETL system developers. Each business process data set needs a dedicated ETL module to extract the transaction facts, associate them with the conformed dimensions, and tie them together into a flexible dimensional model.

Data Model Design

The Kimball approach to data modeling takes a pragmatic look at the underlying database platform and chooses the appropriate physical model based on usability, flexibility, performance, and maintenance on that platform.

What Is a Dimensional Model?

Almost all dimensional models are classic star schemas, as shown in Figure 3. The numeric measurements (“facts”) of a business process are concentrated in the central fact table, and the context of the measurement is represented as a set of denormalized dimension tables, which surround the fact table. They keys that implement the joins between the dimension tables and the fact table should be anonymous integer keys. We call these *surrogate* keys.

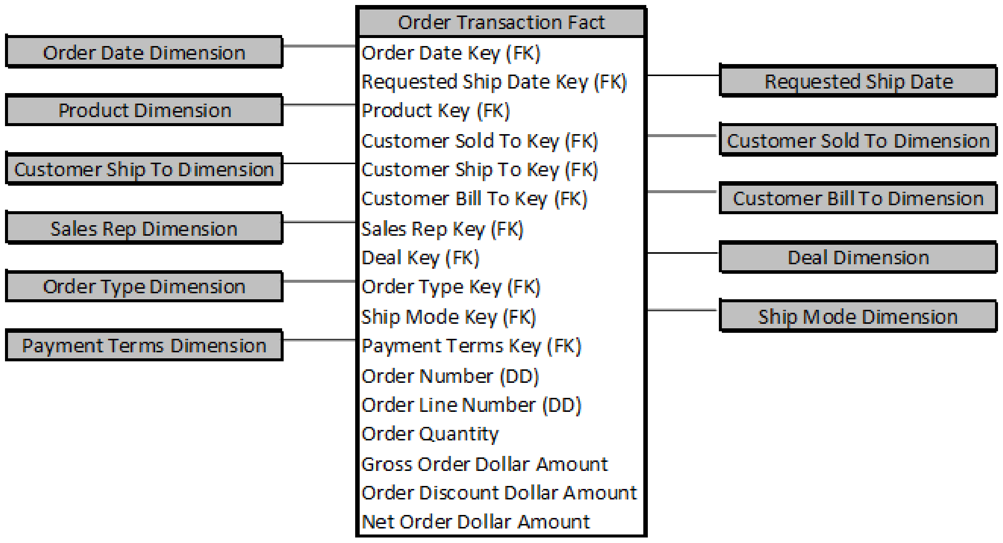


Figure 3: An orders business process star schema

Usability

All camps are in agreement that the most user-accessible data model in the data warehouse is the dimensional model. For example, a 2006 study in the journal *Decision Support Systems* found that dimensional models were significantly easier to understand and remember how to use than other more normalized models.

Flexibility

There is a school of thought that calls for a normalized, third normal form model at the atomic level of the data warehouse. Its proponents argue that this gives the most flexibility. While this may be true from a transaction processing perspective, it is important to remember we are building an analytic database. Most transaction systems are based on third normal form data models with the atomic level detail transactions captured in normalized fact tables. The third normal form school keeps this model as the data foundation of the enterprise data warehouse. This then requires additional transformation steps to get the data into its presentation form for user consumption, often involving another physical layer of departmental data marts.

There is a commonly held belief that dimensional models are based on a set of reports or analyses and are therefore less flexible. This is false and has never been part of the Kimball approach. The normalized model and a properly designed, atomic-level dimensional model are relationally equivalent. They can answer the exact same set of analytic queries.

Flexibility comes in part from the level of detail captured in the model. Another common misconception is that dimensional models are summary only. In fact, a strong design goal in the dimensional model is to always capture data at the lowest level of detail available, called the *atomic* level. The presence of atomic-level data allows users to roll the data up to any level of summarization required. Any aggregation prior to inclusion in the enterprise data warehouse means some detail will not be available, thus reducing flexibility.

Performance and Maintenance

The dimensional model keeps the atomic-level fact tables in their normalized form (by normalizing the dimension tables out of the fact table) for smaller size and better performance, but keeps each dimension in denormalized (flat) form. Note that such flat dimension tables contain exactly the same information as fully normalized (snowflaked) dimension tables but do not implement the separate tables and extra keys required to complete the normalization process. The dimensional model simplifies the physical design by dramatically reducing the number of tables and joins required for a given analytic query, which improves performance on most market leading database products running on single servers. In fact, all the major SMP database products, including SQL Server 2008, have built-in performance optimizations that leverage the dimensional model (search the web for “star join optimization” for more information on this). Using a dimensional model at the physical level is also easier to manage than a normalized model. Because it is already dimensional there is no need for a translation layer or separate data marts to make it user-presentable.

As we’ll explore in the architecture section, parallel processing platforms such as the Parallel Data Warehouse work a bit differently. Data is distributed across many independent query nodes across the server. Each of these nodes may hold a subset of the fact data, which may need to join to all of the dimensions. On SQL Server PDW, the standard approach is to replicate all of the dimensions out to each node so the node can perform local joins, thus preserving the physical dimensional model. However, in rare cases it may make sense to normalize and/or distribute very large dimensions on SQL Server PDW to save time in the replication process and to save space on each compute node.

Let’s be clear on what we are saying. In the best of all possible worlds, you would load the atomic-level detail into the data warehouse for flexibility. The user data model would be dimensional for usability, and the physical data model would also be dimensional for simplicity and performance. Our experience in the real world backs this up. We have found the dimensional model to be the most usable, flexible, best performing, and most maintainable data structure for analytic purposes on most platforms. We only compromise this design when the platform requires it for performance and the users can be transparently shielded from any increase in complexity.

Dimensions and Facts

As we described in the bus matrix discussion, dimensions are the objects that participate in an organization’s business processes. We generally model these as one table per object. Building the dimension in the ETL system involves joining the various normalized description and hierarchy tables that populate the dimension attributes and writing the results into a single table.

Figure 4 shows an example of typical product-related attributes in a normalized model.

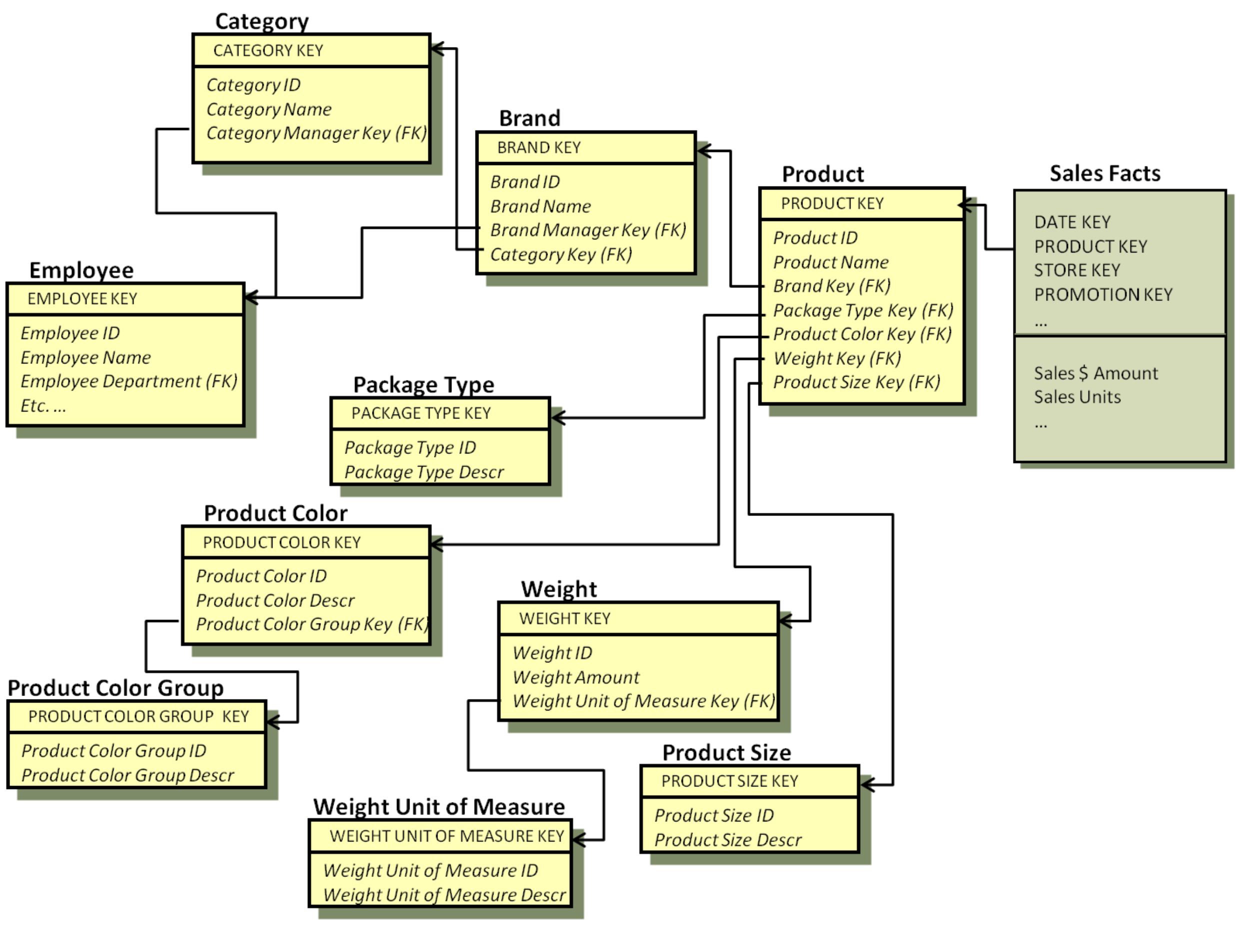


Figure 4: The normalized source tables for Product attributes

The base table is called Product and it connects to the Sales fact table at the individual product key level. From this set of tables, it's possible to create analytic calculation such as `SUM([Sales $ Amount])` by CategoryName, or by ProductColorGroupDescr, or any other attribute in any of the normalized tables that describe the product. It's possible, but it is not easy.

In the dimensional version of the Product table, we would join the product-related tables from Figure 4 once, during the ETL process, to produce a single Product dimension table. Figure 5 shows the resulting Product dimension based on the tables and attributes in Figure 4.

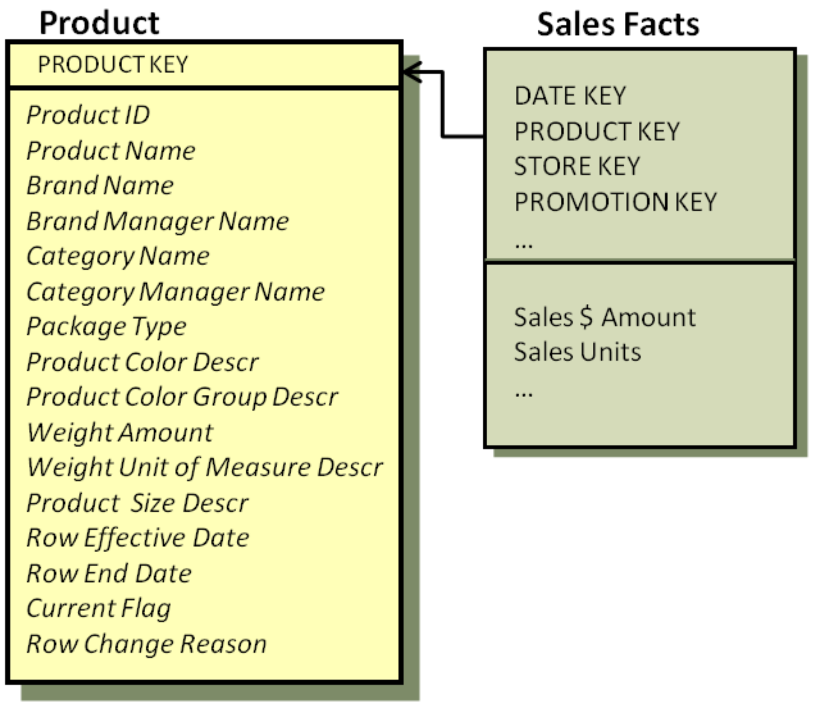


Figure 5: The denormalized Product dimension

Obviously, it is still possible to calculate `SUM([Sales $ Amount])` by CategoryName, or by ProductColorGroupDescr, or any other attribute in the Product dimension, which includes all the product related attributes from the normalized model. Note that the two models are equivalent from an analytic perspective.

Usability is significantly improved for BI application developers and ad-hoc users with the dimensional version. In this simple example, the ten tables that contain the 12 product attributes are combined into a single table. This 10 to 1 reduction in the number of tables the user (and optimizer) must deal with makes a big difference in usability and performance. When you apply this across the 15 or 20 dimensions you might typically find associated with a Sales business process, the benefits are enormous.

The main difference between the two approaches is that the normalized version is easier to build if the source system is already normalized; but the dimensional version is easier to use and will generally perform better for analytic queries.

Using BI Tools to Mask Complexity

Most BI tools have a metadata layer that can be used to display a normalized physical model as if it were a dimensional model. This does meet the usability requirement, but it usually has a price. First, it only works for those users working with the specific tool. Any other access, such as queries embedded in applications, will need to work with the normalized model. Second, this metadata layer puts these tools in the role of creating complex SQL, which is often poorly formed. In which case, the user is usually reduced to writing the SQL by hand. Third, the underlying physical model is still normalized, which may not perform well for analytic queries compared to the dimensional model. A fourth cost is in actual dollars; these tools are fairly expensive.

Tracking Attribute Variations over Time

Every analytic data store must provide a means to accurately track dimension attributes as they change over time. Tracking attribute changes allows the business to report on the state of the world as it was at any point in time, answering question like “What were sales by territory as of December 31st last year?” It also supports accurate causal analysis by associating the attribute values that were in effect when an event occurred with the event itself. For example, what postal code did a customer live in when they bought a certain product two years ago?

The most efficient way to capture these changes from both an ease of use and performance perspectives is to add a row to the dimension whenever an attribute changes by assigning a new surrogate key and capturing the effective date and end date for each row. These are commonly referred to as *slowly changing Type 2 dimensions*. You can see these control columns at the bottom of the Product dimension in Figure 5.

While tracking attribute changes over time places a burden on the ETL process, it improves performance for user queries because the joins between the facts and dimensions are simple equijoins on integer keys. This also improves ease of use because the BI semantic layer does not have to handle more complex, multi-column unequal joins in order to retrieve the correct dimension row for any given historical fact event.

Tracking changes over time is a mandatory business requirement, regardless of the underlying data model you use. It is possible to track changes in a normalized model, but the complexity of keeping multiple versions across dozens of tables associated with a single dimension is much greater than dealing with changes in a single, denormalized dimension table.

For a basic description of slowly changing dimensions, search www.kimballgroup.com for an article titled Many Alternate Realities (http://www.kimballgroup.com/html/articles_search/articles2000/0002IE.htmll).

For a discussion of more advanced change tracking techniques, see the article titled Slowly Changing Dimensions Are Not Always as Easy as 1, 2, 3 (http://www.kimballgroup.com/html/articles_search/articles%202005/0503IE.html).

For more information on tracking changes in a normalized model, please see Design Tip #90 Slowly Changing Entities at www.kimballgroup.com (http://www.kimballgroup.com/html/07dt/KU90SlowlyChangingEntities.pdf).

Performance Depends on the Platform

While the dimensional model provides the best performance and usability in a majority of standard scale DW/BI systems, it is not a one-size-fits-all answer. The underlying physical data structure needed to get the best performance on a given platform is the right choice, as long as it does not compromise the ease-of-use principle.

As we explore the Parallel Data Warehouse system, we will discuss the design and performance tradeoffs, and hopefully end up with the best of all possible worlds.

Section 2: Massively Parallel Processing and the Parallel Data Warehouse

Data growth is related to Moore's law. As computers get faster and more powerful, we are using them to process more data in more complex applications. Traditional sources such as ERP systems are processing more transactions as our organizations grow. Newer sources such as web browsing activity, mobile devices, and social media are creating data sets that are orders of magnitude larger than what we've dealt with in the past.

We call these large scale data sets "big data." Large organizations have always had to deal with big data, but the boundaries that define big data have expanded with the raw power of computers. The quantity of data you have to deal with only becomes a problem when it becomes difficult to work within your existing systems environment. For some organizations, this could be as small as 100s of gigabytes. For larger, more experienced organizations, the cross-over point is more in the tens of terabytes.

The term big data is a major buzzword in IT today precisely because big data holds the potential for big value. However, your ability to extract value from large data sets in your transaction systems and social media interactions is predicated on your ability to actually manage the data. Companies such as Google and Yahoo! have helped pioneer new technologies such as MapReduce and Hadoop to help deal with the massive quantities of unstructured data they collect. At the same time, mainstream technologies for dealing with more structured big data sets, such as massively parallel processing (MPP) systems and column store databases, are experiencing significant growth.

The common big data challenge in most organizations is to figure out how to make the large-scale volume of transactions in their corporate systems available and usable in an analytic environment. Dealing with newer sources of data, such as social networking, is important, but it is not the number-one problem in most companies today.

In this section we begin with a brief exploration of the strengths and limits of single-server systems when it comes to big data. We then dig into the MPP approach and explore the Parallel Data Warehouse system architecture.

The Strengths and Limits of Single-Server Systems

Most of the "servers" you work with are single servers with shared resources. Each CPU core can work with any section of memory or disk, and all memory and disk is available to each core. This all-in-one architecture is known as *symmetric multi processing* (SMP). As shown in Figure 6, the CPU cores connect to the memory and disk via a system bus. This shared connection supports high speed inter-process communication, memory, and data sharing, and is easier to manage because it is a single physical unit.

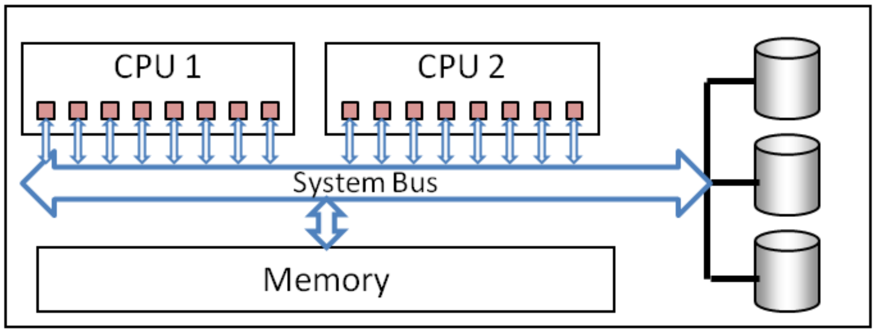


Figure 6: SMP architecture

However, SMP systems have a limit when it comes to big data; they can scale only to the point where their system bus gets overloaded. Too many CPUs making simultaneous requests for data on the system bus creates a traffic jam. As usage grows, the system bus becomes a bottleneck and limits the total amount of processing that can take place on an SMP system. There are ways to mitigate this contention by creating localized subsets of CPUs and memory, but this only extends the limit.

The Massively Parallel Processing Alternative

One time-tested strategy for handling large amounts of data is to avoid this bottleneck by distributing data and processing across many servers, or nodes, each of which has its own memory and disk so they can share the workload. This approach, known as massively parallel processing (MPP), has been around for several decades and is the basis for many of the largest super computers in existence today. Due to their high cost and complexity, MPP systems have historically been used by the largest companies and governmental organizations.

This massively parallel architecture lies at the heart of Microsoft's Parallel Data Warehouse system. Parallel Data Warehouse is a Microsoft SQL Server product designed to scale data warehouses from tens to hundreds of terabytes of data. It delivers the MPP architecture using an “appliance” model, providing preconfigured, optimized commodity hardware and software and a single point of support.

The SQL Server PDW Architecture

Figure 7 shows an abbreviated architecture for a SQL Server PDW MPP system. A user query request would come into the control node, which breaks the SQL into multiple parallel operations and distributes them out to the compute nodes where the actual data resides. A special module called Data Movement Services coordinates any needed data movement among nodes taking place between and handles any functions that need to be resolved centrally. When the compute nodes are finished, the control node handles post-processing and re-integration of results sets for delivery back to the users.

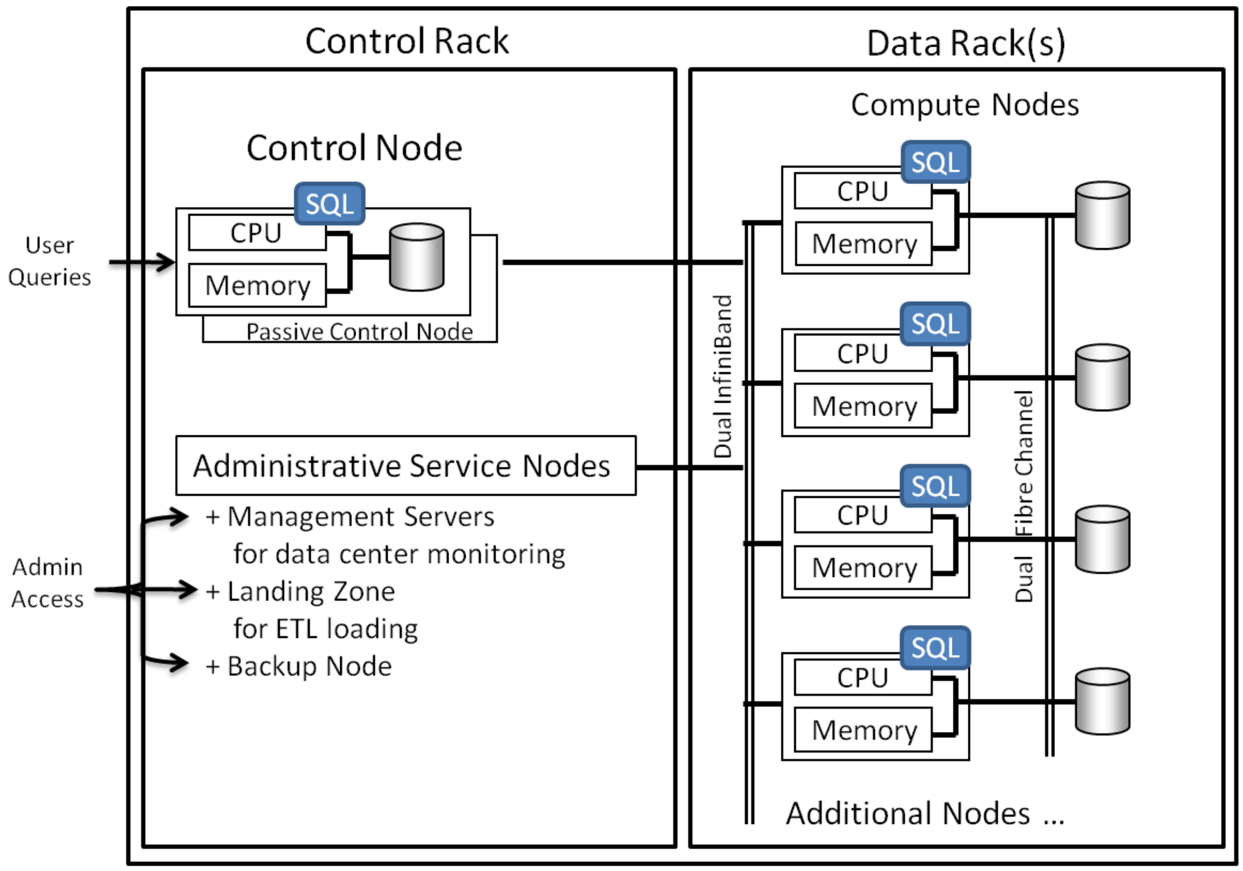


Figure 7: The SQL Server PDW massively parallel architecture

Each compute node is a separate SMP server running SQL Server. Compute nodes in current SQL Server PDW configurations ship with dual, hex-core CPUs, 96 GB of memory, and local **tempdb** workspace. They are connected together using dual InfiniBand network to support high-speed node-to-node data sharing for cross-node computations known as *data shuffling*. This network also connects the compute nodes to the control and administrative nodes to support high-speed data loading, extraction of query results, backup, and other administrative functions.

The disk subsystems for the compute nodes are managed by a storage area network (SAN) component with high-speed dual Fibre Channel connectivity. This data bus supports high-speed I/O, and failover redundancy. The compute nodes and disk drives are physically housed in the same rack, called a *data rack*.

There are three types of administrative service nodes that share the control rack with the control node. These include:

- Management Nodes, which provide the DBA or data center operations interface to access and manage the overall solution and support the system's internal network.
- A Landing Zone Node, where cleansed data is staged and prepped before loading into the data warehouse.

- A Backup Node and the appropriate associated storage. The Backup Node provides high-speed integrated backup at the database level. This is tied to the organization's overall backup strategy and systems.

The SQL Server PDW is a large-scale enterprise class system and has built-in redundancies:

- Primary data is stored as RAID1.
- Hardware redundancy includes redundant power supplies, spare disks, compute nodes, control nodes, and management servers, mostly designed to support automatic failover.

A Scalable Appliance

SQL Server Parallel Data Warehouse is sold as a data warehouse appliance: a set of commodity hardware and Microsoft software pre-configured to meet the needs of a range of data sizes and performance. This makes sense because configuring the individual components, network and connectivity throughput, and disk subsystem performance is a significant effort, more than most IT shops would care to take on. With the appliance, all components and network connectivity are carefully designed, configured, and balanced for optimal performance, and necessary software on all nodes is pre-installed and pre-configured.

The MPP architecture can be scaled up by adding racks of compute nodes. The base system starts with one rack. On an HP appliance, for example, a full rack holds 10 nodes, and additional 10-node racks can be added up to a total of 40 nodes. The 40-node limit is more due to the definition of the product and not an inherent limit of the system design. SQL Server PDW uses its backup and restore facility to make expanding a SQL Server PDW is fairly straight forward: back up the database, add the new rack, reconfigure, and restore. The database restoration automatically redistributes the data across all nodes.

Microsoft is working with several hardware vendors to offer SQL Server PDW systems. HP is the first to market with a publicly available product at this writing.

SQL Server PDW Data Management

The physical architecture of distributed nodes with local data means the large data sets have to be distributed across the nodes in a way that will support both data load and query processes. The goal is to get each node and CPU core working as hard as possible on every query. In the data warehouse, fact tables are distributed evenly across nodes so each node will have work to do.

Efficient processing on nodes results when local fact table subsets can join to local dimensions tables, which can be achieved if dimension tables are replicated to all nodes. SQL Server PDW allows you to specify distributed or replicated tables at time of creation, and then transparently manages placing the appropriate data on the appropriate compute nodes at load time.

The Kimball Approach on SQL Server PDW

How does SQL Server PDW fit with the Kimball approach? When you compare it with our principles, it fits quite well. It provides good usability and flexibility because in most cases, you can build a set of atomic-level dimensional models with conformed dimensions. It performs well because the workload is distributed across all the compute nodes rather than bottlenecked on a single server. SQL Server PDW gets an additional performance boost at the node level because SQL Server has functions to support dimensional models, including star-join optimization. And SQL Server PDW's support for replicated dimension tables allows many common query scenarios to be satisfied without more expensive data-shuffling operations.

Section 3: Enterprise Data Warehouse Architecture Options

Asking “What is the best architecture for an enterprise-wide data warehouse?” is like asking “Where is the best place to live?” The answer is it depends on what is important to you. We’ll start by describing the broad requirements for an enterprise data warehouse at a level which should be agreeable to everyone. We’ll then drill into each of these requirements to see how they can best be met.

Requirements for an Enterprise Data Warehouse Architecture

Of course, your architecture depends on your business requirements along with technical, historical, and political factors. While business requirements are usually business-specific, we can start with a list of broad requirements, or architecture goals, that most organizations would support:

1. Excellent query performance for users
2. High ease of use
3. Flexibility
4. Enterprise-wide usage and value
5. Good maintainability

From these broad goals, we can derive the primary components of an enterprise data warehouse architecture as follows:

Performance

Excellent query performance is a given requirement. What counts as excellent depends on your user expectations; obviously response times in the seconds are desirable when possible, though in some cases, response times in the minutes or even hours may be considered excellent, given the data volumes and query complexities. Just to be clear, excellent performance should be accomplished in a fashion that is entirely transparent to the user. There should be no need for the user to learn which aggregates to use, or which data mart has the required data, or how to add an optimizer hint.

Ease of Use

As we said earlier, the DW/BI industry generally agrees that the dimensional model is the easiest to work with. Ease of use from a business user perspective is ultimately determined by the BI tools that front-end the data warehouse database. Developers, or any users, who are writing reports and applications that directly access the data warehouse database generally end up writing their access code in SQL. Ease of use from a developer’s perspective is driven mostly by the physical database model.

It is much easier to present a dimensional model in the BI tool metadata layer if the underlying atomic data model is already dimensional.

Flexibility

The first determinant of flexibility stems from the grain of the fact tables you create. If your fact tables are captured at the lowest level of detail available, known as the atomic level, you can always aggregate the data up to any attribute of any dimension. Therefore, your DW/BI system must capture the atomic detail to provide maximum flexibility.

Conformed dimensions are also a contributor to flexibility. They allow users to query data from separate business processes such as sales and inventory, and correctly combine the results on shared dimension attributes, such as product or region. In effect, they allow users to compare apples to apples across the enterprise.

Note that a correctly defined dimensional model has the exact same flexibility from an analytic query perspective as a normalized model; they are relationally equivalent. A query to sum up Sales by Region will give the exact same answer in either model.

Enterprise Resource

A true enterprise information resource has three main components: all data is available to all users, data is aligned across disparate business processes, and there is one analytic system of record for each data element. Let's examine each of these statements in turn.

All data must be available to any analyst who might need it because all data is useful to everyone in the organization. The analyst in Logistics needs to know sales by geography and distribution center. The analyst is Sales needs to know sales by customer and region. The analyst in Marketing needs to know sales by product. These are all different summary queries on the same atomic sales fact data. Do not let these departmental differences lead you to think departmental data marts might be a good solution. As soon as you limit Marketing to product summaries, they will insist on customer detail to support a customer segmentation analysis. Every analyst ends up needing access to all the data at the atomic level at some point.

Data must be aligned across business processes because that's what allows users to combine data from multiple, disparate sources across the enterprise in a fast and correct manner. The data in the enterprise data warehouse needs to be integrated via the enterprise set of conformed dimensions identified in the bus matrix. Conformed dimensions and the work that goes into creating and maintaining them is a major component of this enterprise resource; conformed dimensions are the struts that hold the enterprise data framework together.

A single source reduces the confusion and wasted time that results from having multiple data marts with overlapping data content. Having a single analytic system of record may involve multiple physical copies for performance reasons, but this is a compromise. If multiple copies of the data are needed, these copies must be built from a single, central data warehouse database. While there may be transformations for analytic reasons, if the same tables and attributes exist in multiple places, they must be presented with the same names and definitions to avoid error and confusion.

The large-scale server power of a product like SQL Server PDW allows you to provide a true enterprise information resource: a single version of the truth without the extra time, resources, and maintenance required to copy data out to multiple data marts.

Maintainability
The simpler your architecture, the easier it will be to operate and maintain. A single, high-performance database with atomic-level detail and fast summarization based on a dimensional model is the simplest way to meet the broad enterprise requirements.

Architectural Compromises
You may need to adjust your ideal DW/BI system architecture if it cannot meet the enterprise requirements. Performance is the most common area requiring compromise; if the ideal architecture is not working, it's not so ideal. Before throwing in the towel on performance, it is important to make sure your ideal architecture is properly tuned. If it still doesn't work, the most common compromise is data distribution.

Performance Tuning
Performance is platform-dependent. Indexes and aggregates are the two standard performance tools in the DW/BI system, and these vary widely across database product and platform. For example, in an SMP environment, it can take a long time to run a query that asks for total sales for the last five years. In this case, it makes sense to create aggregated tables once during the ETL process that can be used over and over to answer summary level queries. (Note that these aggregate tables need to be transparent to the user to maintain ease of use.)

However, the MPP environment offers a third performance tool: parallel processing. Distributing query tasks across multiple nodes may allow summary level queries to be answered on the fly. This greatly simplifies the design, tuning, and maintenance of indexes and aggregates. You can generally rely on the brute force power of the underlying parallel processing architecture for excellent query performance in the MPP environment.

Distributed Processing
If performance tuning or parallel processing isn't enough, you may have to create separate subsets of the data warehouse and host them on downstream servers. These data marts may be departmental in focus; data is often limited to a few subject areas and summarized. (If the data marts contained atomic-level data from all business processes, you would be back where you started with the enterprise data warehouse.) From a performance point of view, the idea is to offload a subset of users and queries to a dedicated platform. This is a crude form of distributed processing, and is probably less effective than simply adding another rack to the SQL Server PDW machine where it could be used by the entire organization when needed.

There are times when this distributed strategy makes sense. Certain data may be useful or interesting only to a small analytic community. Other data may be sensitive and require strict physical access limitations. In some cases, the desire for a separate server is organizational; a certain department may insist on having its data on its own server. As we will describe in the implementation section, you may have existing data marts with extensive reports and applications built on them. In this case, it's much easier to initially populate these downstream marts from the SQL Server PDW rather than rewrite the reports and applications to work directly from the SQL Server PDW.

In these cases, the SQL Server PDW can act as the central source of the distributed data warehouse. SQL Server PDW has a Remote Table Copy feature that will propagate tables to these downstream SQL Server systems at high speeds. The target systems need to be physically located close enough to the SQL Server PDW so they can connect to the InfiniBand network, since this is part of the speed component. If the downstream systems are designed based on Microsoft's Fast Track architecture, the data transfer rates can be significant. The downstream systems can also be any data mart running SQL Server 2008 or above. This includes the new HP Business Data Warehouse, optimized for SQL Server 2008 R2, and the HP Business Data Warehouse, a BI appliance also optimized for SQL Server 2008 R2.

Extended Analytic Functionality

There are several ways to extend the functionality of the core data warehouse within the SQL Server platform. SQL Server Analysis Services online analytic processing (OLAP) brings more advanced analytic functionality and improved performance for complex queries. Analysis Services data mining brings predictive analytics that can leverage the ability of the parallel processing server to draw out valuable patterns and relationships from vast quantities of transactions.

Analytic Marts

Most organizations who have been working with DW/BI systems for several years have advanced beyond simple reporting. They are building complex analytic applications using predictive analytics and multi-faceted dashboards that draw key performance indicators from across the enterprise. It can be difficult to create the queries that populate these advanced BI tools because they need to make multiple passes against multiple fact tables at different levels of summary. In this environment, it is usually easier to create additional data sets that pre-integrate and calculate most of the analytics.

This can be done either as tables in the SQL Server PDW, or as a separate OLAP data mart. These types of calculated data sets in the relational data warehouse are often called *snapshot fact tables* or *accumulating fact tables*. Common snapshot tables include inventory balances at a point in time, or month end account balances in financial services.

The OLAP option is particularly compelling because OLAP databases, such as SQL Server Analysis Services, are designed to perform more advanced analytic calculations, and provide a performance boost through the creation and management of aggregates. The language used to access Analysis Services, called multidimensional expressions (MDX), was created to support analytics. It has a built in understanding of date relationships such as current month, year to date, and prior year. It also can navigate hierarchies, such as moving from district to region to country.

In either case, the SQL Server PDW core data warehouse would serve as the data foundation, and these analytic tables or marts would be built from its cleaned and conformed data store.

In an interesting combination of functionality, Analysis Services can also be used as a query management layer for SQL Server PDW in what is known as relational OLAP (ROLAP) mode. In this mode, Analysis Services retrieves data directly from SQL Server PDW at query time, rather than using a pre-loaded OLAP database within Analysis Services. It also provides full access to the advanced analytic capabilities offered by the MDX language. User queries are submitted to Analysis Services from the BI tool layer, translated into SQL, and submitted to the SQL Server PDW database.

Data Mining

Microsoft's data mining functionality can drive a range of interesting predictive analytics including forecasting, recommendation engines, and customer segmentation. The data mining component itself is an Analysis Services feature, and runs on an Analysis Services server.

Parallel Data Warehouse can serve as the data source that feeds the data mining engine, thus enabling models based on the vast amounts of transaction level detail stored in SQL Server PDW, in combination with the richly attributed dimensions. Generating the input data sets is often the hard part of data mining because these data sets typically involve multiple full-table scans to identify behaviors and changes in behavior over time.

BI Reporting and Applications

One of the original principles of the Kimball approach listed in the first section of this white paper is to provide a complete DW/BI solution. This includes providing user access for ad-hoc exploration and BI reports and applications that deliver value to the business that was identified in the requirements gathering process. Microsoft offers a set of reporting and analysis tools as part of its overall DW/BI product stack, and Parallel Data Warehouse is a fully participating member of this ecosystem. Reporting Services and Report Builder queries and reports, and third-party BI tools, can draw from SQL Server PDW like any other SQL Server database. The same is true for Microsoft Office tools such as Microsoft Excel and Microsoft PowerPivot. All of these user access methods can be hosted in Microsoft SharePoint and delivered in the context of a rich BI portal experience. .NET applications can access SQL Server PDW via ADO.NET drivers, and third-party tools can communicate with SQL Server PDW using OLE DB and ODBC. All of these drivers accompany the SQL Server PDW product.

Enterprise Data Warehouse Architecture Summary

In summary, the above goals lead us to the following components of the ideal enterprise data warehouse architecture:

Components	Goals Addressed
Atomic data	Flexibility, enterprise resource
Single data store	Enterprise resource, maintainability
Parallel processing and/or aggregates	Performance
Dimensional model	Ease of use for all user communities
Conformed dimensions	Enterprise resource, integration
Attribute change tracking	Enterprise resource, ease of use, accurate history

Table 1: Architecture components and goals

We include attribute change tracking even though it is more a function of the ETL process because it is mandatory from a business perspective, and its ease of implementation is a function of the underlying data model. Therefore, we list is as part of the core data warehouse architecture. A graphical model of this architecture can be depicted as follows:

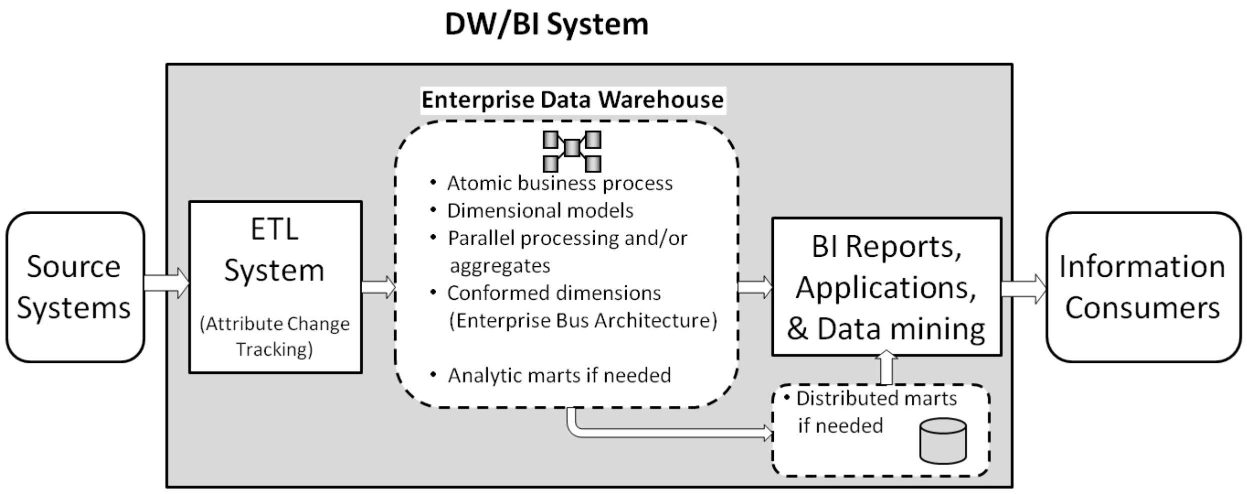

DW/BI System
Enterprise Data Warehouse
Source Systems
ETL System
(Attribute Change Tracking)
• Atomic business process
• Dimensional models
• Parallel processing and/or aggregates
• Conformed dimensions (Enterprise Bus Architecture)
• Analytic marts if needed
BI Reports, Applications, & Data mining
Information Consumers
• Distributed marts if needed

Figure 8 - High-level enterprise data warehouse in the DW/BI system architecture

Parallel Data Warehouse occupies center stage in this architecture. In many cases it can provide a no-compromise solution, with a single set of atomic-level data stored in dimensional models, using parallel processing to provide performance, and organized as an enterprise resource based on the bus matrix and conformed dimensions. SQL Server PDW can also take on many of the hard core ETL processes if need be, a function we will discuss in the implementation section coming up.

Section 4: Building the Enterprise DW/BI System with SQL Server PDW

Most readers considering a Parallel Data Warehouse already have a data warehouse in place and are looking for ways to help handle growing data and performance demands. Many of these next-generation, large-scale data warehouse/business intelligence systems are evolving from existing DW/BI systems that are designed based on the Kimball approach. In this case, the transition to SQL Server PDW will be straightforward.

In this section we run through the basic steps for converting an existing SMP-based Kimball data warehouse to a Parallel Data Warehouse server, including the impact of SQL Server PDW on the DBA. We'll also explore additional roles SQL Server PDW can play, including serving as the central source or hub, in a distributed data warehouse environment, as an ETL transformation engine, and as a platform for providing real-time analytic data.

Preparation and Installation

The SQL Server PDW system must live in a data center and involves at least two racks, so you should do some planning with your server management group before the truck shows up on installation day. Since it uses InfiniBand, any other servers you want to benefit from fast data transfer functions will need InfiniBand connections and to be located close enough to the SQL Server PDW server to meet any cable limitations.

Vendor installation is usually part of the purchase and takes a few days depending on what issues show up.

Part of planning should include some consideration of your overall conversion strategy. The options range from directly converting the existing data warehouse to completely re-architecting the system as part of the migration process. We will focus on the direct conversion approach in this section and discuss the re-architecting options later in this paper.

Data Migration

Once the machine is up and running, the next step is to create the new database, instantiate the target objects and their properties, and copy over the data. The Parallel Data Warehouse database is a SQL engine, but it is a bit different from the SMP-based SQL Server database. This is mostly because it is a parallel processing system, and some things don't work quite the same. Certain functions have an underlying assumption of serial processing that doesn't work in a parallel environment. Other functions, such as distributing data across nodes for parallel execution, don't exist in the SMP environment.

If you are converting an existing SMP SQL Server database to SQL Server PDW, you can use a tool the Microsoft PDW team has built to help. It creates tables, adjusts indexes and partitioning, suggests distribution strategies for the fact tables, identifies problems such as data types that do not have direct equivalents in SQL Server PDW, and generates the actual BCP out scripts to get data from SQL Server and load scripts to load data into SQL Server PDW.

If your existing data warehouse is not SQL Server, the initial data migration is still fairly straightforward as long as you have a solid set of dimensional models. It shouldn't take more than a few hours depending on the number of tables involved.

One big advantage of the SQL Server PDW system from the DBA's perspective is the simplification it brings to physical data management. The physical location of data, including filegroups, disk layout, LUNs, and **tempdb** location, is all handled automatically as part of the core SQL Server PDW system.

There is one high-level physical decision to be made when moving to a massively parallel environment: how the tables should be split up across the nodes. There are two primary ways to physically instantiate tables in SQL Server PDW: replicated or distributed. The CREATE TABLE DDL includes a distribution clause where this is specified.

Replicated Tables

A replicated table looks like a single table to anyone who accesses SQL Server PDW, but it is actually replicated out to all compute nodes on the server. That is, there is one copy of the table on each node.

The purpose of replicating tables is to improve performance by having local copies of data on each node to support local joins. Replicated tables are generally used for dimensions and lookup tables to support local joins to the fact tables.

The replicated tables are managed by the system transparently. From the DBA's perspective, the CREATE TABLE syntax is pretty simple:

```
CREATE TABLE Customer (
     CustomerKey int NOT NULL,
     Name varchar(50),
     ZipCode varchar(10))
WITH
     (DISTRIBUTION = REPLICATE);
```

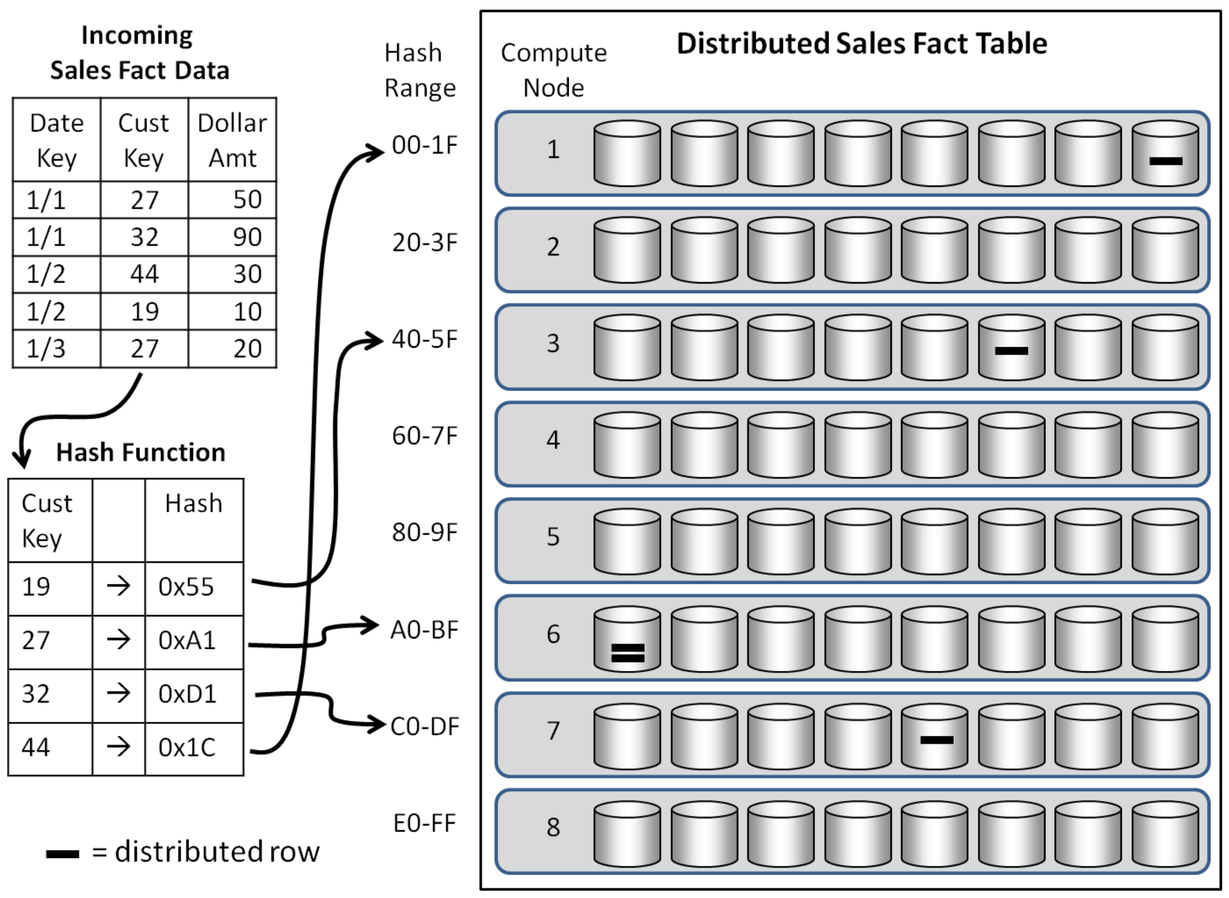
The default is REPLICATE if the distribution clause is omitted.

Distributed Tables

The rows of a distributed table are spread across all nodes as evenly as possible. Each row is written out to a distribution which is a storage location on a node. There are eight distributions on each compute node, each with its own disks. In other words, no copies are made; each row in the source table ends up in only one distribution on one compute node. The rows are mapped to the distributions using a hash function on a column from the table.

The goal of distribution is to improve performance by maximizing parallel processing. Fact tables are usually the largest tables in the data warehouse, and are usually distributed.

Figure 9 shows a simplified version of the distribution of a Sales Fact table across eight compute nodes based on the CustomerKey column.


Incoming Sales Fact Data

Date Key	Cust Key	Dollar Amt
1/1	27	50
1/1	32	90
1/2	44	30
1/2	19	10
1/3	27	20

Hash Function

Cust Key		Hash
19	→	0x55
27	→	0xA1
32	→	0xD1
44	→	0x1C

Figure 9: Fact table distribution

The Customer Key from each row from the Incoming Sales Fact Data in the upper left is put through a hash function. The hashed values map to a single distribution on a single node. For example, the row for customer key 44 hashes to 0x1C, which maps to the last distribution of the first compute node. Here is the DDL for the distributed table shown in Figure 9:

```
CREATE TABLE SalesFact (
      DateKey INT NOT NULL,
      CustomerKey INT,
      DollarAmount MONEY)
WITH
      (DISTRIBUTION = HASH(CustomerKey));
```

The choice of the distribution column is key, so to speak. If a few customers accounted for a large percentage of sales, using Customer Key would lead to an imbalance in the data distribution. One or two distributions would end up with a larger percentage of the data. This imbalance is called *data skew*. One or a few distributions with 10% more rows than average may cause problems, and a difference of greater than 30% will lead to poor performance. This makes sense because each query has to wait for all nodes to complete, and any node with significantly more data will take longer than the others when processing queries involving skewed data.

The primary criteria for selecting a good column for distribution are high cardinality and even row counts. There are other considerations for choosing the distribution column. For example, it's not a good idea to choose a column that is often constrained to a single value in user queries. If users typically constrain on a single day, then the DateKey column is not a good candidate because all the rows for that day will end up in a single distribution. Other factors come into play when selecting a distribution key, such as distributing multiple fact tables that may need to be joined together to support certain analytics.

The parallel processing power of the SQL Server PDW system allows you to test your distribution key choice. Pick a distribution key, load the table, and run some distribution queries and a representative set of user queries against it. If you find a problem, you can create another version of the distributed table from the first version by using the CREATE TABLE AS SELECT statement and changing the column in the DISTRIBUTION = HASH () clause. This is generally much faster than you would expect because of the parallel processing. Of course, you need enough space to make multiple copies of your large fact tables, even if they are only experimental.

Dealing with Very Large Dimensions

As we said, dimensions are almost always replicated in the SQL Server PDW data warehouse. As a rule of thumb, dimension tables that are 5 GB uncompressed or smaller should be replicated. You do have to allow for space on each node for the replicated table. A 5-GB dimension would compress to around 2 GB, which would take up a total of 20 GB once it is replicated across a 10-node rack. By the way, compression is automatic and mandatory in SQL Server PDW.

Just to get a sense of the dimension table size that qualifies for replication, a Product dimension like the one shown in Figure 5 with a 500-byte uncompressed row size could hold about 10 million rows before you might consider other options.

Dimensions larger than 5 GB uncompressed are not unheard of, especially when dealing with big data. If you have a dimension that exceeds the replication threshold, you have two main options in a parallel environment: distribution or normalization.

Distributing a large dimension leverages the same parallel processing power with the fact table. However, if the rows needed to resolve a query are not on the same node as the associated fact table, the required dimension keys must be "shuffled" between nodes. SQL Server PDW is designed to move data rapidly when necessary for a query processing step, but it's always faster to stay local.

In some cases, it may be possible to distribute the dimension using the same surrogate key as the fact table. This shared distribution key means the joins remain local because the required dimension rows are on the same nodes as corresponding fact rows.

The second option is to normalize very large dimensions to reduce their size and make their replication less burdensome. If the product dimension shown in Figure 5 had 10 million rows, it would require about 2 GB (depending on column widths and compression ratios), which is around the replication boundary. The normalized product table shown in Figure 4 would only require about 325 MB to hold 10 million rows. Obviously 325 MB is going to be easier to copy out to 10 or 20 nodes than 2 GB.

If most queries against a large dimension only return or constrain against a few columns, consider creating an outrigger dimension. That is, the core dimension will contain the commonly used columns. The rest of the columns are put into a separate table, called an outrigger, with the same surrogate key. The core dimension can then be replicated, and will join locally to the fact tables. Queries that require the less common attributes can bring them in with a single join to the replicated outrigger dimension. This is an easy way to get back into the range where replication works without having to completely normalize the dimension.

You can insulate users from the complexity of a normalized or outrigger design by providing views that re-combine the normalized columns back into a single dimension. Test these views to make sure they do not negatively impact performance.

You may have heard that normalization is a requirement for MPP systems. Some historical context might help explain this. Early MPP systems had tighter space constraints, lower bandwidth between nodes, and more costly storage. This led to a default practice of normalizing the dimensions in order to reduce the amount of data replicated onto each node. MPP vendors glossed over this need to normalize by arguing that you should use a normalized model because it is the “industry standard” for an enterprise data warehouse. Do not be fooled by this reverse logic. Normalizing dimensions is an MPP design choice made to improve performance by reducing the amount of data that must be replicated and stored across the nodes. Again, the need to normalize a dimension has been a rare occurrence in SQL Server PDW implementations to date.

Additional DDL

There are a few additional design decisions to make in defining the data warehouse tables. There are typically far fewer indexes on an MPP system because they are not needed. Do use clustered indexes where it makes sense. In most cases, this means creating a clustered index on the surrogate key of the dimension tables, and on the same column used for partitioning the fact tables. Use non-clustered indexes with care. In many cases, they are not needed because of the parallel processing speed, and they add maintenance, slow the load process, and take up space.

Fact tables may be partitioned for the same reasons you would partition on an SMP system, such as rolling window management or load isolation that uses a SWITCH operation. Partitioning is conceptually simpler in SQL Server PDW because it is fully specified as part of the table creation DDL rather than through a separate partition function and scheme.

Create an ETL System to Load the Target Model

Once you have tables defined in SQL Server PDW, the next step is to load data into them. The initial data transfers will most likely use scripts to bulk copy the existing data warehouse history into the SQL Server PDW. Moving forward, if you were using SQL Server Integration Services, your ETL system should function essentially the same with SQL Server PDW as it did with your prior data warehouse. For example, SQL Server PDW has its own source and destination connections you will use in your Integration Services packages. However, there are a few product differences that will impact your ETL system.

Surrogate Key Assignment

The IDENTITY property of an integer field is not supported in SQL Server PDW. This makes sense when you realize rows in a distributed table will be inserted across many separate nodes. The cost of keeping track of incremental identity assignments across multiple nodes in a parallel process would dramatically slow any insert process. If you were using the IDENTITY property to assign surrogate keys to your dimensions, you will need to manage this either in your ETL process by keeping surrogate key values in a table and assigning them incrementally, or in the INSERT statement by using the ROW NUMBER ACROSS function.

Cached Lookups Only

If you use Integration Services Lookup transformations in your existing ETL packages, make sure you select **Full cache** in the **Cache mode** section when querying SQL Server PDW, which pre-populates the lookup cache. Using the Lookup transformation to perform a non-cached SELECT operation against incoming Integration Services pipeline rows is inefficient with SQL Server PDW.

The Landing Zone

The SQL Server PDW system has a separate staging server as part of the control rack called the Landing Zone. Incoming data from the Integration Services connections or the SQL Server PDW bulk loader (DWLoader.exe) flow through the Landing Zone prior to being distributed to the compute nodes for permanent storage. The Landing Zone quickly reads incoming rows from files or Integration Services and sends them off to compute nodes in a round-robin fashion using a module called the Data Movement System (DMS) which, not surprisingly, handles data movement around the system. On each compute node, a DMS instance will hash the rows and send them back out to the DMS instance of the node to which they map. This receiver DMS inserts the row into a staging table where any sorting and indexing takes place. The final step uses SELECT INTO to copy the data from the staging table to the target table. All this happens behind the scenes and is managed by the system.

This whole flow keeps data loading in a highly parallel fashion and minimizes any processing work actually performed on the Landing Zone.

One benefit of parallel processing is the load process can run while users are querying the data. The loader processes get lower priority, so they have little impact on user queries. This means you can process yesterday’s load without having to limit user access. It also means you could do near-real time data loads to give access to current data where it’s needed.

Transact-SQL Compatibility

SQL Server PDW has its own variant of SQL with extensions to support parallel processing. Some functions in the SMP SQL Server product have not been implemented in SQL Server PDW. Some of these were omitted because they are functions that do not translate well into a parallel environment. For example, the IDENTITY property is not supported as described in the ETL section.

Transact-SQL compatibility with SQL Server SMP is not yet fully complete, and Microsoft continues to add functionality through frequent updates. You will want to test any existing scripts or stored procedures that are part of your current operations against the latest functionality provided by SQL Server PDW.

System Management and Tuning

SQL Server PDW has its own Central Administration console that provides easy management and monitoring of the system. It uses a product similar to SQL Server Management Studio that is aware of the multi-node nature of the system and monitors sessions, queries, loads, backups, node activity, and alerts and errors.

From a tuning perspective, it’s best to take a simple approach on SQL Server PDW, starting with minimal indexes as described in the physical design section and testing performance with a representative set of user queries once you get the data loaded. If it works, no problem. If not, you can use the Central Administration console to inspect individual query plans to see where the bottlenecks are. For example, your fact table distribution may be skewed, so most of the processing is on a single node. In this case, you can try a different distribution column using the CREATE TABLE AS SELECT statement as described earlier. You may need indexes for specific types of queries. For example, one customer needed to query individual phone numbers from the Customer dimension for some of their lookup reports. A non-clustered index on phone number did the trick. This is something you would typically consider for queries that are used often and have a large impact on the user community.

Additional Opportunities

There are several additional roles and requirements Parallel Data Warehouse can take on beyond hosting the enterprise data warehouse. From an enterprise information perspective, SQL Server PDW can integrate with existing systems by serving as the system of record for analytic data and providing that data to downstream bulk consumers. From an ETL processing perspective, SQL Server PDW can act as a large-scale ETL engine to manage the bulk transformation of big data sets. SQL Server PDW can also support near real-time data warehousing, which is critical for certain analytics.

Integration with Existing Systems

There are many situations where the enterprise data warehouse needs to feed large data sets to downstream systems. In many cases, these are extensions of the DW/BI system in the form of data marts which can be fed from SQL Server PDW in a hub-and-spoke fashion. The definition of data mart is quite fluid; it often describes a component that exists for historical and/or political reasons and adds significant work without adding much value. Data marts and other downstream data consumers can also include purpose-built architectural components. For example, it may make sense to create a subset of enterprise data on a separate server to allow integration with business unit or divisional data. We’ve also seen large chunks of data exported from the EDW to support research or data mining on a dedicated server. Operational systems such as a sales force automation system, or customer relationship management system, often import large subsets of the EDW to provide context to their processes. We've also seen subsets created for business-specific reasons. For example, one company wanted to provide sales data to their customers, but decided to create a separate data mart for each customer for security reasons.

If you need to integrate with existing systems, SQL Server PDW can help. Remote Table Copy is a high-speed table copying function that can transfer tables from the SQL Server PDW to SQL Server running on a locally connected SMP server. Data transfer rates can be as fast as 400 GB per hour. Once the data is in the target SQL Server machine, you would complete the ETL process to properly integrate it into the database with appropriate indexes, partitioning, and any other required constraints.

An Opportunity for Improvement

If you have downstream data marts that were created for historical performance and/or political reasons, and which no longer serve a true business need, we encourage you to examine them carefully. This multi-layered, multi-model approach adds significant work, time, redundancy, and cost to the enterprise DW/BI system implementation. Implementing a SQL Server PDW system offers a chance to re-architect these leftover appendages into a more efficient and effective enterprise information environment.

This platform improvement strategy seeks to replace the existing DW/BI system by unplugging the existing data marts and redirecting or rewriting BI queries and reports to pull directly from the SQL Server PDW. This approach is usually more disruptive and requires more effort, but ultimately it leads to a simpler, more robust, more responsive enterprise information resource. Simply integrating SQL Server PDW into the existing environment sounds appealing because it is low impact in the short term. However, in the long term, you may be perpetuating systems that are inefficient, confusing, and costly.

SQL Server PDW as the Transformation Engine

Organizations dealing with particularly large data sets and operating with narrow load windows may not have time to use a separate ETL system to process the data before loading it into the SQL Server PDW. In these cases, SQL Server PDW can serve as a large-scale transformation engine as part of an overall EDW architecture. This approach generally involves loading the data directly into tables in the SQL Server PDW database, and then performing ETL lookups as INSERT-SELECT operations joining staging tables to dimension tables to lookup surrogate keys in bulk. This approach applies the full power of the parallel environment to the core ETL processes.

Real Time Options

While most of the analytic data in the data warehouse does not need to be loaded on a less-than-24-hour basis, some business opportunities require more frequent data loads. SQL Server PDW's parallel load process supports "near real time loading" under the Read Uncommitted isolation level (Dirty Reads). Loads can be run while users query tables and these data loads have a low impact on the overall performance of concurrently-running queries.

Conclusion

SQL Server Parallel Data Warehouse offers a viable platform for supporting large-scale data warehouses into the hundreds of terabytes. The appliance nature of the system makes it relatively easy to configure, install, tune, manage, and expand. SQL Server PDW provides parallel processing of queries against dimensional models on atomic data to address the Kimball approach's goals of query performance, usability, and flexibility on an enterprise information resource.

For more information:

http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/pdw.aspx: Parallel Data Warehouse on SQL Server Web Site

http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx: Fast Track Data Warehouse on SQL Server Web site

http://www.microsoft.com/sqlserver/en/us/solutions-technologies/Appliances/HP-bdw.aspx: HP Business Data Warehouse Appliance on SQL Server Web site

http://www.microsoft.com/sqlserver/en/us/solutions-technologies/Appliances/HP-ssbi.aspx: HP Business Decision Appliance at SQL Server Web site

http://www.microsoft.com/sqlserver/: SQL Server Web site

http://technet.microsoft.com/en-us/sqlserver/: SQL Server TechCenter

http://msdn.microsoft.com/en-us/sqlserver/: SQL Server DevCenter

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback.