

SFA Modernization Partner
United States Department of Education
Student Financial Assistance



**Data Warehouse Application and
Technical Architecture Standards**

Task Order #4
Deliverable # 4.1.5

February 16, 2000

Table of Contents

1	INTRODUCTION.....	4
1.1	SCOPE.....	4
1.2	ORGANIZATION OF THIS DOCUMENT	4
2	DATA MOVEMENT STANDARDS	6
3	DATA WAREHOUSING ARCHITECTURE OVERVIEW	8
3.1	DATA WAREHOUSE – A TECHNICAL INTRODUCTION.....	8
3.2	DATA WAREHOUSING ARCHITECTURE FRAMEWORK.....	9
3.2.1	<i>Data Sources</i>	11
3.2.2	<i>Architectures</i>	12
3.2.3	<i>Metadata Management</i>	20
4	DATA WAREHOUSE ARCHITECTURE.....	25
4.1	ARCHITECTURAL COMPONENTS.....	25
4.1.1	<i>Data Warehouse</i>	25
4.1.2	<i>Data Mart</i>	29
4.1.3	<i>Operational Data Store</i>	31
4.1.4	<i>Differences Between ODS, Data Warehouse (DW), and Data Mart (DM)</i>	33
4.1.5	<i>A Sample Data Warehouse Architecture</i>	35
4.2	DATA WAREHOUSE ARCHITECTURE DRIVERS.....	36
4.2.1	<i>Granularity of Data</i>	37
4.2.2	<i>Data Retention and Timeliness</i>	39
4.2.3	<i>Reporting Capability</i>	40
4.2.4	<i>Availability</i>	40
4.2.5	<i>Scalability</i>	40
4.3	DATA DISTRIBUTION AND PLACEMENT	41
4.3.1	<i>Distributed Data Warehouse</i>	41
4.3.2	<i>Data Segmentation via Distributed Data Marts</i>	42
4.4	DATA MODELING FOR DATA WAREHOUSING.....	43
4.4.1	<i>Data Model Overview</i>	44
4.4.2	<i>Normalized vs. Dimensional Modeling</i>	45
4.4.3	<i>Sample Model Diagrams</i>	46
4.4.4	<i>Dimensional Modeling Components</i>	47
4.4.5	<i>The Star Schema</i>	49
4.5	PHYSICAL DESIGN ISSUES.....	53
4.5.1	<i>Indexing Strategy</i>	53
4.5.2	<i>Partitioning Strategy</i>	54
4.6	DATA WAREHOUSE TECHNOLOGIES.....	56
4.6.1	<i>RDBMS vs. MDDB</i>	56
4.6.2	<i>Data Warehouse-Specific RDBMS Features</i>	57
5	POPULATION ARCHITECTURE.....	67
5.1	POPULATION PROCESS OVERVIEW.....	67
5.2	ARCHITECTURAL OPTIONS.....	70

5.2.1	<i>Process Distribution</i>	71
5.2.2	<i>Update Techniques</i>	73
5.2.3	<i>Control</i>	75
5.2.4	<i>Sequencing of ETL Steps</i>	76
5.3	ETL PROCESS	76
5.3.1	<i>Initial and Ongoing Population</i>	77
5.3.2	<i>Extraction Techniques</i>	78
5.3.3	<i>Transform</i>	83
5.3.4	<i>Cleansing</i>	85
5.3.5	<i>Summarization</i>	88
5.3.6	<i>Changes in History</i>	89
5.3.7	<i>Realignments</i>	90
5.3.8	<i>Sorting</i>	92
5.3.9	<i>Load</i>	92
5.3.10	<i>Staging</i>	94
5.4	ETL TOOLS	95
5.4.1	<i>Requirements for ETL Tools</i>	95
5.4.2	<i>Code Generators</i>	96
5.4.3	<i>Transformation Engines</i>	96
5.4.4	<i>ETL Tools Architecture Tradeoffs</i>	97
5.5	MIDDLEWARE	98
5.6	BATCH ARCHITECTURE ISSUES	99
5.6.1	<i>Bulk LoadBatch</i>	99
5.6.2	<i>Error Handling and Restart/Recovery</i>	99
5.6.3	<i>Data Movement</i>	100
5.6.4	<i>Exception Handling</i>	100
5.6.5	<i>Scheduling</i>	100
5.6.6	<i>Synchronization</i>	100
5.6.7	<i>Audits and Verifications</i>	101
6	END-USER ACCESS	104
6.1	END-USER ACCESS TOOL CATEGORIES	104
6.1.1	<i>Reporting Tools</i>	106
6.1.2	<i>Query Tools</i>	106
6.1.3	<i>Analysis Tools</i>	108
6.1.4	<i>Knowledge Discovery Tools</i>	116
6.1.5	<i>Data Mining</i>	117
6.2	END-USER ACCESS TOOL ARCHITECTURES	119
6.2.1	<i>Two-Tier vs. Three-Tier Architectures and Tools</i>	119
6.2.2	<i>Net-Centric Architectures</i>	121
6.2.3	<i>Web Architectures</i>	121
6.2.4	<i>End-User Access Security</i>	124

1 Introduction

This deliverable is part of Task Order 4, and defines both a standard framework, along with guidelines, and standards for building data warehousing infrastructures at SFA. This document is intended to serve as a guide for future data warehousing development teams.

1.1 Scope

The Data Warehousing Standards and Guidelines cover the following areas:

- A recommendation on data warehousing standards
- Descriptions of key data warehousing terms and concepts
- An overview of the data warehouse end-to-end architecture

The Data Warehousing Standards and Guidelines do not cover the following topics, since the focus here is on architectural standards:

- Detailed descriptions and comparison of specific data warehousing products
- Determining business case
- Requirements gathering strategy and processes
- Change management strategy, processes and techniques
- Application standards

1.2 Organization of this Document

This document is organized around two primary areas:

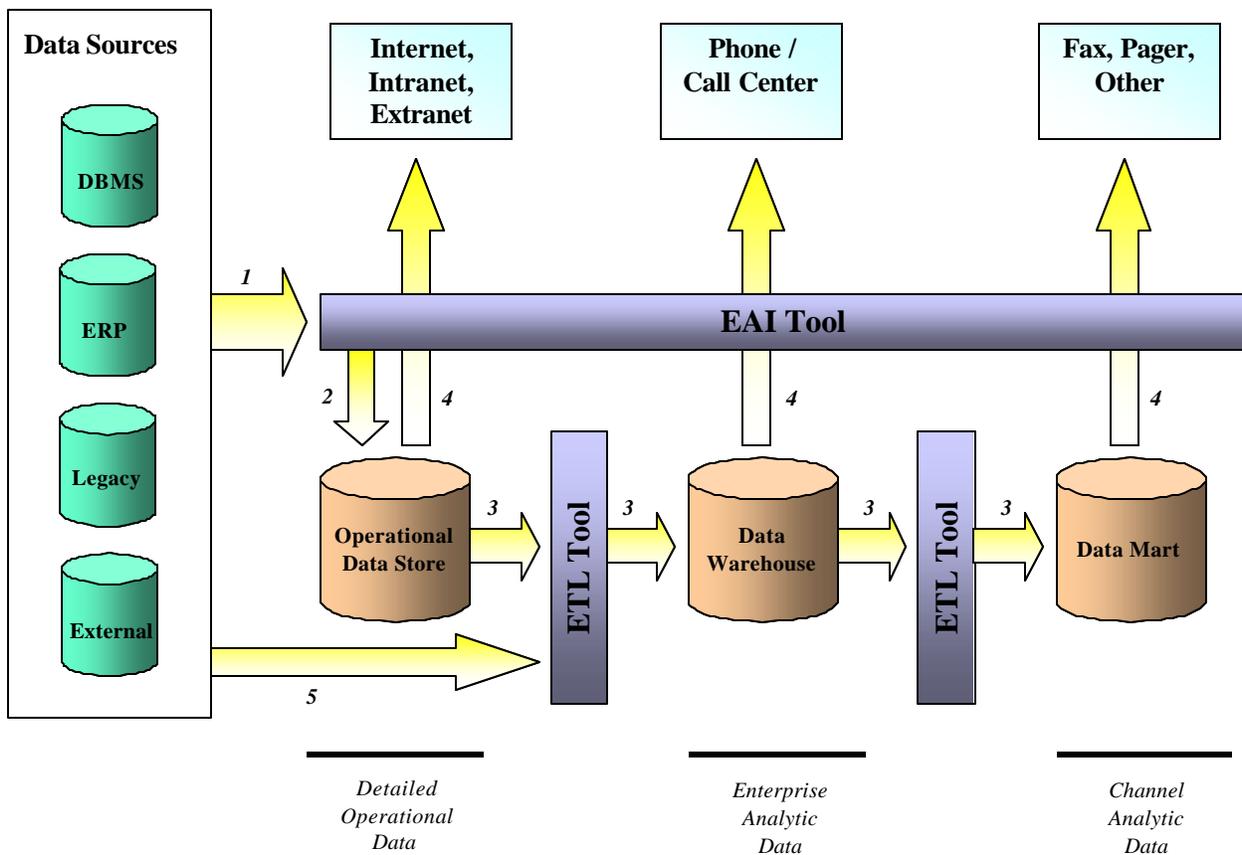
- Architectural standards recommendation, based on Andersen Consulting's Business Integration Methodology, our industry experience and best practices, and our knowledge of SFA's objectives, initiatives, and environment.
- Data warehousing terms, concepts and architectures, which have the following major sections:
 - *Data Warehousing Architecture Overview* – This section provides a high level architectural overview and conceptual framework for a data warehouse end-to-end technical architecture.
 - *Data Warehouse Architecture* – This section describes the various components of a data warehouse. It focuses on data placement and distribution issues, data modeling, and various technologies used to build the architecture.

- *Population Architecture* – This section describes the Extract, Transform, and Load steps to move from source systems into the data warehouse. It also focuses on tools and techniques to design and support these processes.
- *End-user Access* – This section describes common end-user access applications and architectures utilized for data warehousing.

2 Data Movement Standards

In this section, we have made recommendations on data movement standards and what tools to use in the architecture. Specifically, standards are for how data is written to repositories, and how data is distributed and disseminated. The rest of the document goes into much more detail on other data warehousing standards and guidelines.

The figure below provides a snapshot view of data movement from source to target systems. In the description that follows numbers on the diagram are referenced.



STANDARDS

High level data movement standards include:

- Standard 1 - (#1 and #2 on figure) - Use *only* the EAI architecture and tool to extract data from current transactional systems, and to load it to the Operational Data Store. EAI can provide near-real time extracts and loads, as contrasted with batch extracts and loads with the traditional ETL tools. The “currency” of data will be important for certain reports, web and call center activities.
- Standard 2 - (#3 on figure) – Use *only* the ETL (Extract-Transform-Load) tool to extract data from the ODS and load it into the data warehouse. Also *only* use ETL to “pump” data from the data warehouse to data marts. The ETL tool excels at transformations such as calculations and aggregations. ETL also provides robust metadata functionality, which gives users of the warehouse specific information about the data.
- (#4 on figure) – Most users, no matter what the access channel, will gain access to the data warehousing repositories through the EAI (middleware) architecture. EAI will handle security and routing such that right data gets to the right users from the right repository or systems. The exceptions are system, database and data warehouse administrators, and some power users.
- (#5 on figure) – While ODS will only be populated by the EAI tool, the data warehouse may need data that the EAI does not extract to the ODS. Therefore, we suggest using the ETL tool to get the additional data into the data warehouse as necessary.

The remainder of the document goes into detailed standards and guidelines.

3 Data Warehousing Architecture Overview

This section provides an overview of key terms and concepts within a data warehousing end-to-end architecture. It is structured within a conceptual model called the Data Warehouse Architecture Framework. This framework exists above the physical implementation level, although physical architecture choices and alternatives are discussed.

3.1 Data Warehouse – A Technical Introduction

A typical introduction to data warehousing often starts with a high-level physical schematic, as shown in Figure 3-1:

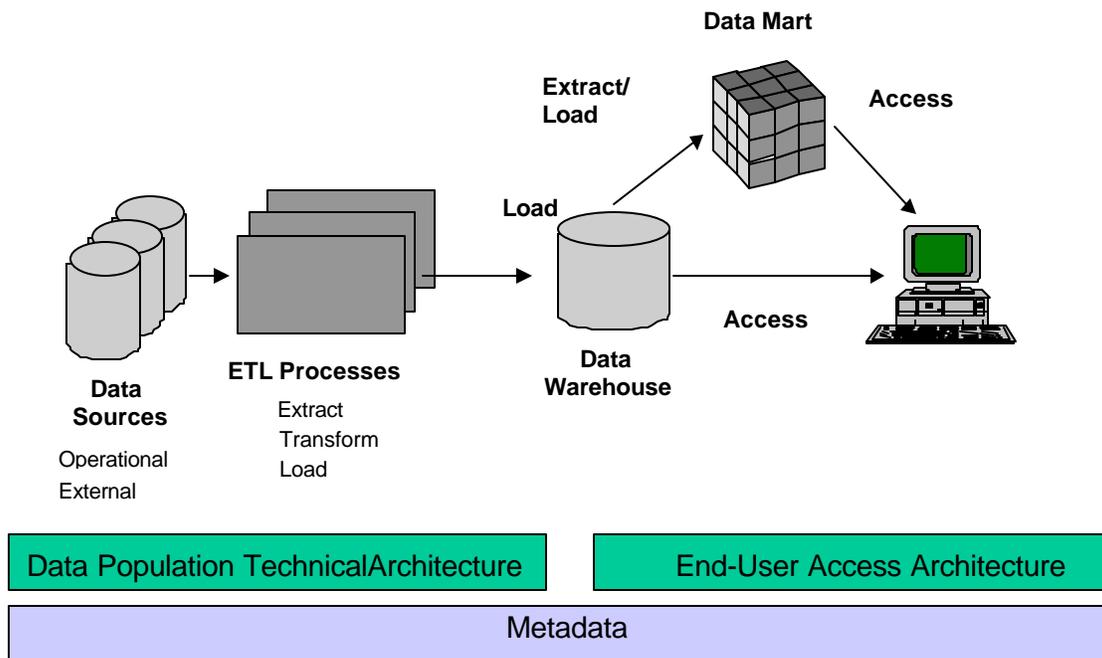


Figure 3-1: Typical Data Warehousing Architecture

Figure 3-1 highlights a traditional data warehouse end-to-end process. This is a fast and easy way to introduce a beginner to the concept of data warehousing, as it shows the following:

- The major components and processes of a data warehousing architecture in a physical context

- Data flow from source to user
- The dynamic of user interaction between a data warehouse and a "data mart" cube

However, for an experienced technical architect, this picture raises more questions than it answers. Many of the constructs within this picture are open to interpretation. For example:

- Is there just one physical data store for the data warehouse as the diagram might imply?
- Where does the Data Mart physically exist in relation to the data warehouse?
- Where exactly does the extract/transform/load occur?
- How does the user access the data?

To avoid the detailed and varying answers in which the above questions may lead, it is useful to conceptualize the architecture within a high-level framework for training and overview purposes. Physical issues such as those raised in the above questions will be discussed in detail throughout the remainder of these Guidelines.

3.2 Data Warehousing Architecture Framework

The Data Warehousing Architecture Framework (DWAF), displayed in Figure 3-2, shows the various components found in a typical end-to-end data warehousing architecture. A physical data flow can be implied within the left-to-right flow of the boxes from Data Sources to End-User Access. However, no relationship in terms of physical data stores and process locations should be assumed.

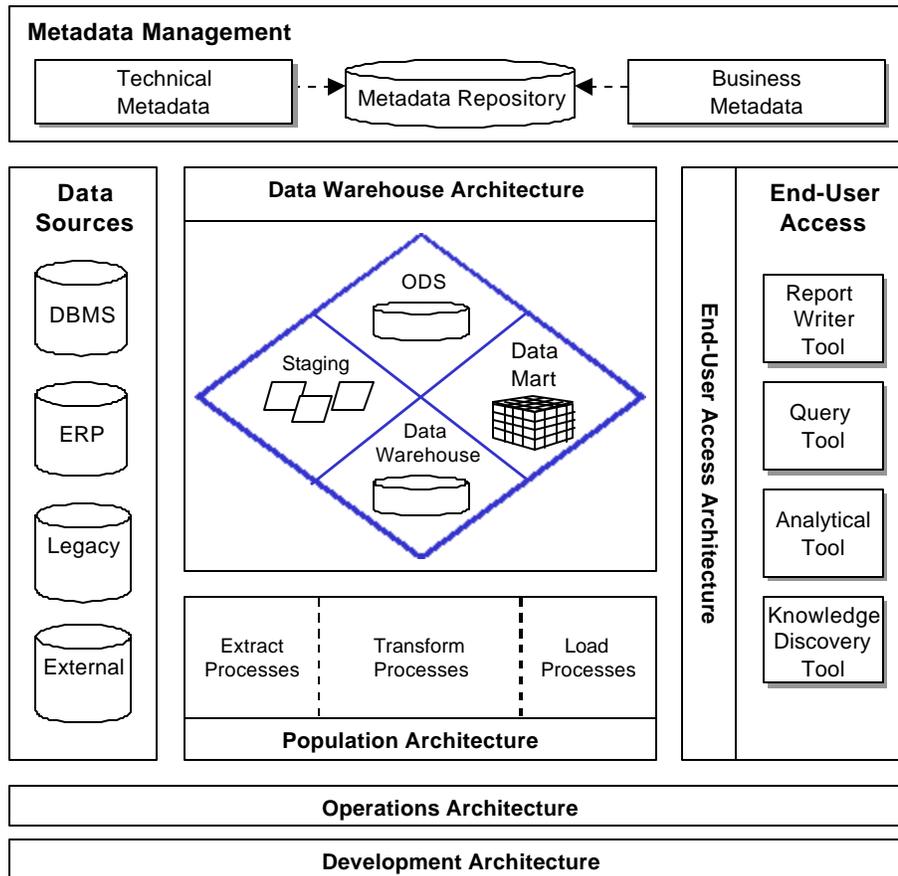


Figure 3-2: Data Warehouse Architecture Framework

The Data Warehousing Architecture Framework consists of the following components:

- **Data Sources** — The operational, legacy system and external databases in which the data that gets moved into the data warehouse resides.
- **Architectures** — The hardware and software that provide for and support the processing, storage and access of data as it flows from data source to end-user. Within the Data Warehousing Architecture Framework, specialized architectures exist for Data Population, End-User Access, the Data Warehouse, and supporting Operational and Development Environment services. Major components of these architectures include the following:

- Data Stores — The storage areas used to hold data as it is transformed from transaction records to information accessed by users. These storage areas include data sources, data warehouses, data marts and data storage on end-user PCs.
- Processes — The modules that perform the actual data extractions, transformations, and loads between data stores.
- Services — The supporting activities that ensure data is moved and accessed correctly and in a timely manner. These services exist within the Population Architecture (for example, transport, exception handling and audits) and Operations Architecture (for example, backup/restore and monitoring).
- Metadata Management — The tools and techniques that enable the definition, collection, and publication of information about the data itself (or ‘data about data’) – from the business meaning of the data for the users to the technical characteristics of the data (timing, format, etc).

The remainder of this section describes the major components of the Data Warehousing Architecture Framework, with a special emphasis on Metadata Management.

3.2.1 Data Sources

Data sources represent the stores of data collected and stored by operational and transaction processing (OLTP) business applications that are the origin of the information required for end-user reporting. Data warehouses generally start with a single source of data, or the smallest number of sources required to fulfill initial reporting and analytical needs, and expand in an iterative nature to include more source and external systems (as necessary) throughout the life of the data warehouse.

Source systems can include any of the following types:

- DBMS — A database management system which may be relational (DB2), hierarchical (IMS) or networked (ADABAS).
- Legacy — A catch-all term for data sources from mainframe-based systems. These are often older-technology or mature file systems (such as VSAM), and databases (such as DB2 or a networked or hierarchical database).
- ERP — Enterprise Resource Planning systems, such as Oracle applications, have been high-priority sources for data warehousing efforts at recent projects. These systems can have highly complex or unknown data models, so they often contain

specific mechanisms to extract and interpret data. This subject is discussed in more detail in the ERP Data Warehousing section of the Guidelines.

- **External** — Data warehouses are often supplemented by data from external sources to the organization, such as published marketing collateral, financial databases, and reports. These sources are sometimes included in later iterations of a warehouse release, as they often contain "nice-to-have" data that can supplement and enhance existing corporate data to improve the benefit of data already captured in a warehouse.

3.2.2 Architectures

The following architectures make up the overall end-to-end data warehousing environment. These architectures are defined in discrete sections which are often designed and built, to a non-trivial degree for large complex projects, by separate project teams. It is up to the overall technical architecture team to ensure that the separate sections described below come together within a single unifying architecture.

Data Warehouse Architecture

This architecture consists of all the data stores that exist within the data warehouse. It also consists of the physical distribution of data and processing, and the hardware and software to support these data stores.

Key to the notion of a data warehouse architecture is that even though a data warehouse, by its nature, is a centralized collection of data, there are still multiple ways to distribute data throughout the end-to-end technical architecture to meet user performance, availability and scalability requirements. The data warehouse architecture is formed based on decisions made to determine how data should be stored and physically distributed to meet both user and operational requirements.

The major data stores within the Data Warehouse Architecture are as follows:

STAGING

A temporary area in which data is staged for efficient transformation and loading into a data warehouse. No user access occurs within this area; files are generally deleted or overwritten when the next batch of data needs to be staged. Staging often occurs using temporary flat, unindexed files for use by the Transform and Load processes.

Architecture issues include determining if the staging area should reside on the same physical server or within the same physical database as the data warehouse, or if volumes and loads are heavy enough to warrant placing part or all of this area on a server that is separate from the rest of the warehouse. Many data sources and larger volumes require more sophisticated storage

mechanisms for the data (such as using a relational database instead of a flat file) to handle complex staging issues.

DATA WAREHOUSE (DW)

An integrated and centralized data store organized specifically for end-user reporting and analytical access. The data warehouse generally consists of enterprise-wide information over multiple subject areas, and contains low-level, granular data, kept over long periods of time for historical reporting purposes. The data warehouse must be physically optimized to handle high volumes of data and concurrent user access, and is generally lightly indexed and less normalized than data stores found within an operational system.

The term data warehouse is often used to represent the sum total of data stores and processes found within the data warehouse architecture. In complex and high volume environments, the data warehouse will likely consist of multiple centrally located data stores that are required to handle integration and summarization of data.

OPERATIONAL DATA STORE (ODS)

The storage of detailed transactional data in a normalized format for operational reporting purposes before being summarized into the data warehouse. An ODS is updated on a real-time or near real-time basis, sometimes from other operational systems or from actual user updates.

There is rarely a pre-defined need for an ODS – it is often created to handle architectural requirements for performance, scalability and near-real-time operational reporting.

DATA MART (DM)

A grouping of data specific to a single subject area, department or user class. This data is optimized for fast access and analytical reporting; therefore, the data structures will be highly summarized and indexed.

Using a data mart within a data warehouse architecture is not mandatory – it becomes necessary based mainly on the reporting needs of the end-user community. Data marts are also used for availability, scalability and performance reasons.

There are generally multiple data marts for a single data warehouse, developed in an iterative fashion. Data marts should be fed from a single point of collection – namely the data warehouse - for consistent data views throughout the enterprise. Feeding data marts directly from source systems run the risk of multiple inconsistent views of the same data throughout the enterprise, as well as multiple redundant processes requiring high levels of change and support when the source systems or reporting needs change.

The data mart can be implemented physically on the same server as the data warehouse, on a physical server separate from the data warehouse at the same central site, or distributed

regionally at the user sites. The end-user access tool associated with the data mart generally dictates the architecture and placement options available.

PATHS THROUGH THE DATA WAREHOUSE ARCHITECTURE

Given all the components within a data warehouse architecture, there are many ways to move data from the source to the end-user. These multiple paths of data flow exist depending on the types of end-users and their data and access requirements. Typical paths through this architecture are as follows:

- Staging – DW – User — The most straightforward data warehouse architecture; users access data directly from the warehouse.
- Staging – DW – DM – User — The number and location of data marts will vary based on performance, availability and scalability requirements of the users. Users may access data out of both the data mart and the data warehouse, depending on the necessary level of detailed data.
- Staging – ODS – DW – DM – User — The ODS is used for both operational reporting and staging into the data warehouse. The data warehouse is fed from the ODS, and potentially from other data sources as well. Any number of data marts may exist to support the DW. Users can access any of these constructs (besides staging), depending on reporting needs.

The processes that extract, transform and load data throughout the Data Warehouse Architecture are described in detail in the Population Architecture section of these Guidelines.

Population Architecture

The Population architecture provides the processes and services that move and control movement of data, resulting in the population of the data warehouse.

The architecture consists of the following services to ensure that data is moved correctly and in a timely manner:

- Scheduling — Ensures that jobs are timed and synchronized within batch windows to move data between data stores, physical servers, and the extract/transform/load processes that act on the data
- Batch file (run-to-run) control and restart — Ensures that files are handed off properly between job steps and reach their proper destination, even in case of server or network downtime

- **Error handling** — Ensures that processing modules can handle unusual or abnormal events in a recoverable fashion
- **Exception handling** — Catches records that should not be loaded into the warehouse
- **File transfer and data transport** — Utilizes the appropriate technologies to move data and files between databases and between servers
- **Audits and verification** — Ensures that data is properly moved between the source and target data stores without loss of information

Major decisions to be made for the population architecture include the definition, choice of technology and method of implementation for all of the above services.

The Population Architecture also supports the Extract, Transform and Load (ETL) processes. These processes occur when moving data between operational systems and the data warehouse, as well as between data stores within the data warehouse architecture. ETL also occurs between the warehouse and the end-user, though this is generally handled by the end-user access tools and architecture.

ETL processes can be custom-built or produced by an ETL tool. In complex environments, both tools and custom code can be used. An ETL tool can dictate much of the population architecture in how it structures and distributes its processes.

The logical and physical distribution of the ETL processes are a major component of the population architecture. These issues are expanded upon below:

EXTRACT

Extract consists of those modules that extract data out of the source data store. Processing occurs not only for the data sources feeding the data warehouse, but also between physical databases within the data warehouse architecture such as between a data warehouse and a data mart.

The Extract process often incorporates parts of the Transform process; for example, mappings and conversions – in the same steps as the actual extraction. The Extract process may also include logic that transports, or pushes, extracted files to another server for transformation and loading.

TRANSFORM

Transform consists of those processing modules that change the data from the source data store to the target data store format. Transforms encompass a wide variety of processing, including

field mappings, conversions, merging (of several source system tables or fields into a single table or field), summarization, cleansing (making fields correct and consistent between different sources) and verifications, to name a few.

Transform processing can occur in a variety of places – as part of the Extract, part of the Load, and/or in separate processing steps. The Transform step can be performed on the data warehouse server, the operational system server, or for complex processing and high volumes, on a dedicated server between the source and target data stores.

LOAD

Load consists of those modules which load the data into the target data store. The load is often optimized for performance via the use of bulk load utilities for high volumes and pre-sorts. Loads can be broken into multiple processing steps where merging and transformations occur to optimize load performance. For example, loads often contain portions of the Transform process, including summarization, data conversions and sorting.

End-User Access Architecture

This architecture supports the delivery of data from the data warehouse to the user via an end-user access tool. The tool or combinations of tools determine what the architecture can support; therefore, the end-user access architecture should only be defined after the following questions have been answered:

- What types of reporting and analysis is required by end-users?
- What degree of availability to the data is required by end-users?
- What degree of data detail and data timeliness is required?
- How should data be placed throughout the architecture to meet end-user performance and availability requirements? Should data be located centrally, regionally and/or on the end-user's machine?
- What mechanisms are required to deliver information to the end-users (web-based tools, hard-copy printouts, faxes, pager, etc)?
- What level of security is required to access the data?
- What types of end-user access tools meet these requirements? What architectures do they impose upon the data warehouse environment?

End-user access tools provide the various reporting, analytical, and discovery capabilities necessary for users to gain benefit from the data warehouse. These tools should be chosen with

a deep knowledge of user reporting and operational needs, as they influence, if not completely dictate, the end-user access architecture.

End-user access tools can be classified as follows:

- Reporting tools— Canned, pre-generated reports
- Query — Ad-hoc queries generated with little or no knowledge of SQL required, in a user-friendly and graphic environment
- Analytical — Fast and flexible views of data, including roll-ups, drill-downs, ranking, averaging, and trending over time
- Knowledge Discovery — Intelligent data mining and statistical techniques to find previously unknown patterns in large amounts of data

These tools provide the mechanisms and architecture to access and display data in an understandable and flexible manner to the end-user. Some tools provide additional storage capabilities, such as highly-indexed databases or other structures (as found in a data mart) for fast and efficient access. Data can be stored on the user client machine, or as a centralized or distributed component of the data warehouse, depending on the capabilities and architecture of the tool.

More information about these tool categories can be found in the End-User Access section of these Standards.

Operations Architecture

The Operations architecture provides the services to support the overall data warehousing architecture, to ensure that all data is moved properly and in a timely manner, while ensuring the system can recover from small and large interruptions in service. It overlaps with many of the components found within the Population Architecture, such as error handling, audits and scheduling. The architecture also includes the following:

BACKUP/RESTORE

Timely data warehouse backups and restores are arguably the most important operational activity for the data warehouse. Backup and restore procedures must meet user and business requirements for frequency of execution and speed of recovery. They must also be designed carefully to handle high volumes of data typically found in a data warehouse while meeting user availability requirements, scaling for future growth, and minimizing performance impact.

Backup procedures should be highly parallel for large data warehouses to minimize execution time and impact on users. These procedures need to pull data out of multiple database tables

and/or databases to multiple backup drives concurrently. Many data warehouses, especially those with global users, are backed up on-line so users are not removed from the system.

Also important is minimizing the length of the recovery window and ensuring that a restore can occur within a reasonable timeframe. If data becomes corrupted in a large data warehouse, there are many steps that need to occur to perform a restore. Data from the backup media must be reloaded into the database, updates since the last backup applied, and indexes rebuilt. Each of these steps may take days for hundreds of gigabytes worth of data without high levels of power and parallelism.

ARCHIVE/RESTORE

Archiving is an important but often neglected operational activity within a data warehouse. The data warehouse should not keep data indefinitely, given the cost of the additional disk storage and the potential complexity and performance impact of continually backing up, indexing, and maintaining excess detailed data. Backup procedures can be modified slightly to archive old and stale data from the warehouse to an off-line storage device.

The archiving process is driven by user requirements for how long data should be kept on-line, how long it takes for archived data to be restored from off-line storage, and how long off-line data should be kept before being erased. Often there is not one blanket requirement – there can be multiple archiving requirements depending on the level of detailed data, the subject area, or both.

SYSTEM AND DATABASE MONITORING

Monitoring may be more complex in a data warehousing effort compared to other applications. Monitoring needs to exist over multiple platforms, both at the operating system and the database levels. Generally, existing client tools and standards can be used to monitor the data warehouse environment, although a new data warehousing platform could require new toolsets and skills.

The monitoring of database usage is important but often neglected. This information is necessary to determine the popular information being requested from the warehouse and to identify performance bottlenecks. This information assists in both the ongoing tuning of the database to handle current and future volumes and the assessment of creating new summarization tables or data marts to handle highly-requested information.

HARDWARE SIZING AND CAPACITY PLANNING

Hardware resource sizing for CPU, memory, disks, and networks is a critical component to any data warehousing effort. Hardware is one of the largest expenses within a large data warehouse project, given the multiple hardware servers and hundreds of gigabytes or terabytes' worth of data that must be supported. This issue tends to receive high visibility at executive levels given the cost and impact on the bottom line.

Sizing must be performed for hardware for development, testing and production. Sizing must occur early enough to procure and install the hardware in time to meet development, testing and release timelines. New servers and disks and how they are configured generally command the most time and attention.

Sizing must also occur with the bottom line cost impact in mind. If the organization cannot afford or support the hardware necessary to meet requirements, scope or user expectations regarding performance and availability may need to change.

TRANSPORT

The Transport process moves data between physical data servers. This processing, an overlap with the Population Architecture, is usually part of the job stream that performs file transfer, control and scheduling. Some data movement and transformation software incorporates transport logic into their processing, to perform compression, conversion (EBCDIC to ASCII), messaging, reconciliation of record counts, or file transfer.

PERFORMANCE

Performance is an issue that underlies practically every component within the Data Warehousing Architecture Framework. Performance must be designed into every architecture described above – in areas such as loads, index builds, the transfer of large data files across the network, the response time of user queries, and the length of backup and restores, to name a few.

Performance is often the weak link that can undermine the success of the data warehouse in the eyes of the user. Like any large complex technical environment, proper gathering of user requirements, setting of expectations via service level agreements, performance testing and ongoing performance tuning all contribute to proper performance management within a data warehouse environment.

Development Architecture

The development architecture has a few special considerations worth noting in a data warehousing context:

- Access to test data from legacy systems may not be possible without interrupting production systems. Creating new test data may not be practical without creating potentially complex extracts to collect a test bed of data.
- Building a data warehouse concurrently with a new transaction system, as occurs frequently in ERP environments, is a challenge. Test data for a source system extract may not yet exist, or may not exist in the volumes necessary to perform a system or

performance test. Use the data results from the system performance test of the transaction system as input to the data warehouse performance test.

- Development and test hardware must be sized and procured in the same manner as the production hardware.
- Dedicated development and test environments may not always be available in resource-constrained environments. Data warehousing projects often share space with other concurrent projects.
- Data warehouse architectures cross multiple platforms, leading to especially complex development environments. Source code version control and migration of code between development, test, and release environments is challenging, especially in environments with heterogeneous platforms in which processing code resides.
- Because it is not always possible to mirror the production environment in the test environment, allow for at least two weeks of testing in the production environment before going live.

3.2.3 Metadata Management

Metadata Management incorporates the collection and publication of information about the data itself – both the business meaning and the technical characteristics of the data. It is not any one data store, process or architecture – it has components of each, and is dependent on the other components within the data warehouse architecture framework.

Because metadata exists within practically every component of the data warehouse architecture, it needs to be actively managed to be properly defined, collected, and utilized.

Metadata Defined

Metadata is often defined as "data about data." Metadata is not new to the application development process. Typical uses include file structure definitions, database field names, lengths and standards found in a data model, as well as calculations and formulas found in any field-to-field or field-to-report mappings.

A data warehousing architecture adds new metadata categories generally ignored or not tracked in the past, and adds stronger emphasis to metadata as being a key contributor to the success of the warehouse.

A description of major metadata types is as follows:

BUSINESS METADATA

Business metadata is information needed by end-users to be confident in the meaning, quality and timeliness of the data. Without this information, the most technically robust and high-performing data warehouse will not be used to its fullest potential. A brief list of important business metadata is as follows:

- Business rules describing what is and is not included within the data warehouse
- Definitions of business hierarchies and key performance indicators (KPIs)
- Common business definitions and calculations for data elements
- Transformation and conversion rules in business context
- Source system names/locations
- User security profiles

TECHNICAL METADATA

Technical metadata is used by the IT support organization to ensure that the data is valid, timely and accurately reflects what is being pulled from the source systems. This metadata is also used for change control and to ease the impact analysis and development efforts for future modifications and enhancements to the data warehousing architecture. A brief list is as follows:

- Data warehouse field lengths and definitions
- Field-to-field mappings between source and target
- Query response times
- Usage of queries and aggregation tables
- Timings of loads, updates, and archives into and out of the data warehouse
- Timings and verifications of success for batch file transfers

Metadata Management Processes

Metadata management consists of the processes that perform the definition, collection, control, and publication of appropriate metadata to the right people at the right time. Determining what kind of metadata should be captured, how it should be published and what degree of integration is necessary is all part of this process.

SOURCES OF METADATA

Figure 3-3 illustrates various entry points for metadata capture in a sample environment:

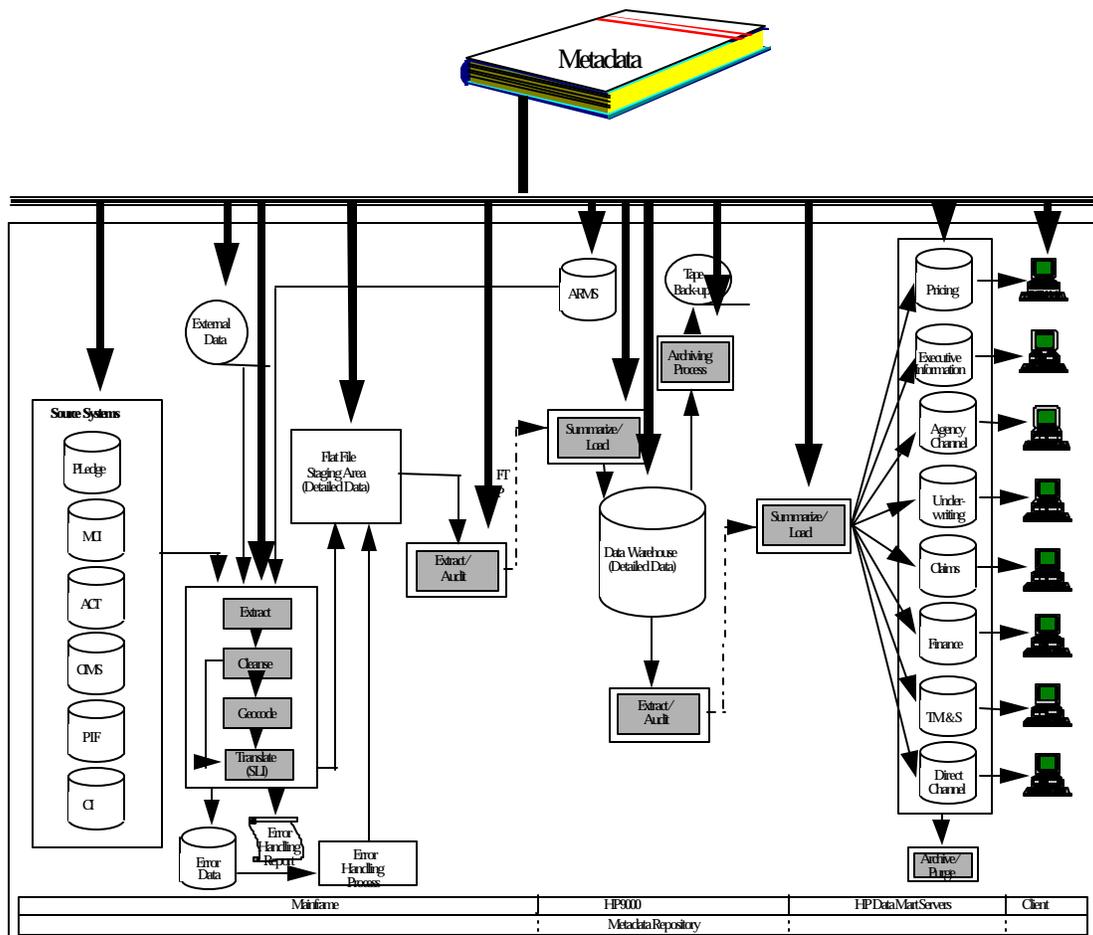


Figure 3-3: Points of Metadata Capture in a Sample Data Warehouse Environment

Metadata resides in multiple places throughout the architecture. It exists within the DBMS tool, the ETL processes and tools, the end-user access tool, as well as any tools or custom programs specifically utilized to collect and capture this information to a single location.

There is no way to tame this large amount of metadata without a manageable process. The lack of metadata management may lead to metadata that is ignored, over-engineered, over-simplified and/or published in a difficult to understand format. This, in turn, could detract from the quality and usefulness of the warehouse as a whole.

METADATA REQUIREMENTS GATHERING

Before determining the overall collection strategy for metadata, the development team must assess metadata requirements. This is not straight-forward, as typical business sponsors and end-users do not generally think in these terms.

It is the development team's responsibility to define and determine the scope and importance of metadata to be collected before a strategy is put into place. Phrasing these requirements in terms the client understands or has used in prior metadata collection efforts for legacy systems can be useful in creating and prioritizing this list. It is then the development team's responsibility to assign roles and responsibilities in the implementation, ownership and maintenance of the metadata.

If the requirements step is ignored, metadata collection will likely occur in a manner that is ad-hoc, minimal, non-integrated, low-tech (manual spreadsheets instead of automated procedures), and potentially, low-value.

METADATA COLLECTION STRATEGIES

Metadata needs to be defined, stored and accessed in a comprehensive manner, just like the data in the warehouse. The approach to storage and access constitutes the metadata collection strategy.

A metadata collection strategy has the following attributes:

- Integrated vs Non-Integrated
- Centralized vs Distributed
- Automated vs Manual

Project developers and end-users determine the breadth and degree of automation, centralization and integration required by the project to capture and publish this metadata. There is no one right way or best way as it will vary greatly on project needs and constraints.

Metadata collection strategies typically consist of one or a combination of the following:

Metadata Repository

Metadata repositories are utilized to integrated disparate metadata from multiple sources (tools, databases and processes) via automated feeds into a single point of collection.

A number of metadata management tools are available, but no one tool is sufficient for all requirements. This is primarily due to the sheer complexity of metadata management issues, but is also the result of a lack of common standards defining metadata needs, terms, and syntax.

Decentralized Metadata Capture

The use of an integrated metadata repository does not frequently occur for smaller-scale or tightly budgeted projects due to cost, complexity, and requirements not being strong enough to support this feature. In this case, metadata will be captured in a decentralized manner, within the tools, processes, and utilities that are utilized for the data warehouse architecture.

However, the metadata will exist in individual silos, out of context of other processes. These will likely not be integrated "out of the box" unless the tools are part of a framework specifically designed for metadata standards across multiple tools. Common metadata standards and frameworks, while still in their infancy, have been gaining popularity over recent years.

Automated or manual processes can be custom built to capture the key metadata out of DBMS, the ETL processes and tools, and the end-user access tools. Automated processes that gather and centralize metadata in this fashion can be time-consuming to build, and rarely have the business case supporting the cost involved. This is why the majority of projects today use manual metadata collection techniques.

Manual Metadata Collection Techniques

Even if centralized and automated methods of metadata collection cannot be utilized on a project, there are still high-benefit but manual-intensive methods of gathering and publishing metadata, including:

- On-line glossary of terms
- Independently maintained PC-based databases or spreadsheets for technical metadata, including field-to-field mappings and conversion rules
- Printed pocket guide for business users, with business definitions and key hints for successful use of the data warehouse
- Hard-copy printouts of the logical database design, with descriptions of entity relationships, data hierarchies and aggregations

All these methods require manual labor, not only in the collection, but in the ongoing maintenance of this metadata so it remains fresh and accurate. Keep in mind that unsophisticated and potentially manual-intensive metadata collection and publication is often of high-value, and is better than nothing at all.

4 Data Warehouse Architecture

This section describes the major components of a data warehouse and placement and distribution issues for a data warehouse architecture. This section also emphasizes data modeling and various technologies used to support the data warehouse.

A data warehousing project involves a range of activities from data extraction and transformation to the selection and integration of proper end-user access tools. However, the data warehouse itself is the central and most critical part of the overall system.

After reviewing this section, the reader will have a basic understanding of the following topics:

- Components of a data warehouse architecture
- Architectural designs and decision drivers
- Data modeling for data warehouses
- Metadata capture
- Database technology used in data warehouses
- The following issues are outside the scope of this section:
 - The use of “unstructured data” or binary large objects (BLOB) such as video, voice, and documents in the database
 - The characteristics, advantages, and disadvantages of individual DBMS products

4.1 Architectural Components

This section discusses data warehousing fundamentals and the key architectural constructs of a data warehouse architecture.

4.1.1 Data Warehouse

A data warehouse is a centralized database that collects, organizes and stores data from operational systems to provide a single source of integrated and historical data for the purposes of end-user reporting and analysis.

The ultimate goal of a data warehouse is to deliver reliable and flexible information that users need, when they need it. The data warehouse is merely a means to achieve this end, and is often the best means possible, given the many ways and places that information for any given subject area can be stored throughout the enterprise.

For example, there is generally more than one definition of a “customer” within a single organization, stored and accessed in a variety of technology platforms, tools and data formats. Disparate operational systems in many large, mature organizations have been created over many years independently of each other, utilizing separate and distinct technologies that were useful to that particular initiative at the time. A data warehouse is often the most appropriate

way to reconcile these differences and reach a common understanding of information within the enterprise.

The data warehouse is defined as a subject-oriented, integrated, time-variant, non-volatile collection of data in support of management's decision making process. The definition is further explained below:

- **Subject-oriented** — Data warehouse data are organized around major subject areas such as sales, claims, shipments, and enrollments. For example, a data warehouse for sales contains historical records of sales over specific time intervals.
- **Integrated** — A data warehouse provides the facility for integration in a heterogeneous, fragmented environment of independent application systems, where the data is stored in multiple, incompatible formats. For example, a department store may have information about the same customers stored in several databases using different format representations. The data warehouse brings the data together into a single representation.
- **Time-variant** — The data warehouse organizes and stores the data needed for informational and analytical processing over an extended historical time range. For example, a marketing analyst can analyze the sales history of five years from the information that was collected at the end of each year.
- **Non-volatile** — Changes to the data warehouse environment occur in a controlled and scheduled manner, unlike the more volatile OLTP environment in which updates continually occur. A similar query run in five minute intervals in an OLTP environment may yield different results, while the same query run within the data warehouse should remain stable and consistent. For example, an airline may capture frequent flyer information in its data warehouse. During check-in for a flight, the additional mileage for a specific passenger is immediately updated in the OLTP system, but is not yet reflected in the data warehouse until its next scheduled load.

While the previous points describe the defining characteristics of a data warehouse, the following points describe typical operational characteristics generally found in data warehouses:

- **Batch mode updates** — Current operational system and overall architectural limitations rarely provide for feasible and cost-effective real-time updates into a data warehouse

- Daily to monthly refreshes — Performance limitations and high complexity of extracting, moving and loading high volumes of data between databases generally prevent data warehouses from being updated any more frequently than daily
- Analytical and strategic reporting — Operational reporting dependent on up to the minute data is generally left with the transaction systems that contain this data
- Relational database technology — Relational databases handle large data volumes and provide high levels of maturity, scalability, interoperability and industry acceptance, making other types of storage technologies rarely considered for a data warehousing project
- Separate physical platforms from the source systems — Due to the differing performance and scalability demands found in OLTP and data warehousing systems (described in detail in the next section), implementing separate databases on separate platforms provide the ability to optimize performance for the unique demands of each system

Although it is possible to implement a data warehouse with more aggressive characteristics than the ones listed above, it is not recommended, due to the high levels of risk that may be incurred given the state of today's technologies and architectures.

Figure 4-1 illustrates a simplified typical data warehouse architecture:

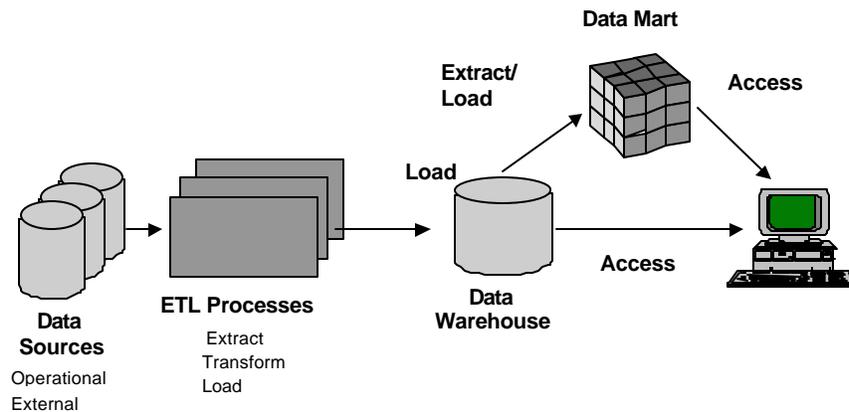


Figure 4-1: Simplified Data Warehouse Architecture

OLTP vs DW Environments

On-Line Transaction Processing (OLTP) systems are used to carry out the processing functions of a business and are characterized by high transaction processing (insert, update, delete) volumes and throughput. A data warehouse is optimized for end-user reporting and analysis of

large volumes of data at both detailed and summarized levels. The data is generally accessed in a relatively stable, read-only environment separate and distinct from the OLTP system(s). Table 4-1 summarizes the most common differences between OLTP systems and data warehouses.

Topic/Function	OLTP	Data Warehouse
Data Content	Current valued	Historical, Summarized data
Data Organization	Application by application	Subject areas across enterprise
Nature of Data	Dynamic, continual updates	Static until scheduled refresh
Data Structure	Structured for fast efficient read/write activity on a record by record basis	Structured for efficient read-only access to large volumes of data
Access Type	High volumes of single-record inserts, updates, queries	High volumes of records accessed in complex queries
Queries and Indexing	Optimized for fast direct access to individual records	Optimized for fast access to a large range of records with multi-table joins to associated detail and descriptions
Level of Data Detail	Records accessed and updated at most detailed, atomic level	Varying query and analysis needs may require both atomic detail and high-level summaries
Usage	Highly structured, repetitive processing	Highly unstructured, analytical processing
Response Time	Sub-second to 2-3 seconds	Several seconds to minutes

Table 4-1: OLTP and Data Warehouse Differences

Data Warehouse Terminology

As the data warehouse industry is still evolving, the term “data warehouse” may mean different things to different organizations. Some may see the data warehouse as a replication of OLTP data to be used as a reporting database. Others may define a data warehouse as any collection of historical data which may or may not span more than one subject area. Often, the

lines between data warehouse, data marts (defined later in this section) and other means of reporting become blurred in how the terms are used.

The most commonly used meanings within the industry for differing contexts of the term data warehouse are described below. Note that usages of these terms will vary from project to project.

- **Data warehouse architecture** — This term is used to describe the end-to-end processes and infrastructure required to move and support data between source systems through the data warehouse to the end-users.
- **Data warehouse** — This term should be used on its own only when describing the centralized database at the core of the data warehouse architecture. The data warehouse becomes a distinct term within environments that contain an Operational Data Store or a Data Mart (defined later in this section).
- **Departmental data warehouse** — This term is used to describe a warehouse which supports departmental analysis in one to five subject areas, drawing from one to a few data sources. This definition often overlaps on the low end with functionality of a data mart (defined later in this section). Typical data sizes can range between 10 – 200 GB, usually supporting between 5-50 users.
- **Enterprise-wide data warehouse** — This term is used to describe a warehouse which provides access to multiple subject areas and users in multiple departments and physical locations. The warehouse is typically fed from one to many data sources to provide a single collection point of information to access and share across the enterprise. There is a wide range of data sizes and number of users that fall within this definition; data sizes usually range between 50 GB – 2 TB, supporting between 20 – 2,000 users.

4.1.2 Data Mart

A data mart is a database of data gathered from operational data and other sources that is designed to serve a particular group of decision makers. The data may be derived from an enterprise-wide database or data warehouse. The emphasis of a data mart is on meeting the specific demands of a particular group of knowledge users in terms of analysis, content, presentation, and ease-of-use. Users of a data mart expect to have data presented in a format familiar to them, and with response times acceptable for detailed analytical reporting needs.

Typical analytical operations performed on a data mart are as follows:

- **Drilling down to detailed data from a higher summarization level.** For example, drilling down to school-level loan information from a regional view.

- Summarizing loans by quarters or months.
- Performing analytical functions such as time-based trending, moving averages and ranking; functions typically not found in simple query and reporting environments.

More information on data mart tools and analytical functions can be found in the End-User Access section of these Guidelines.

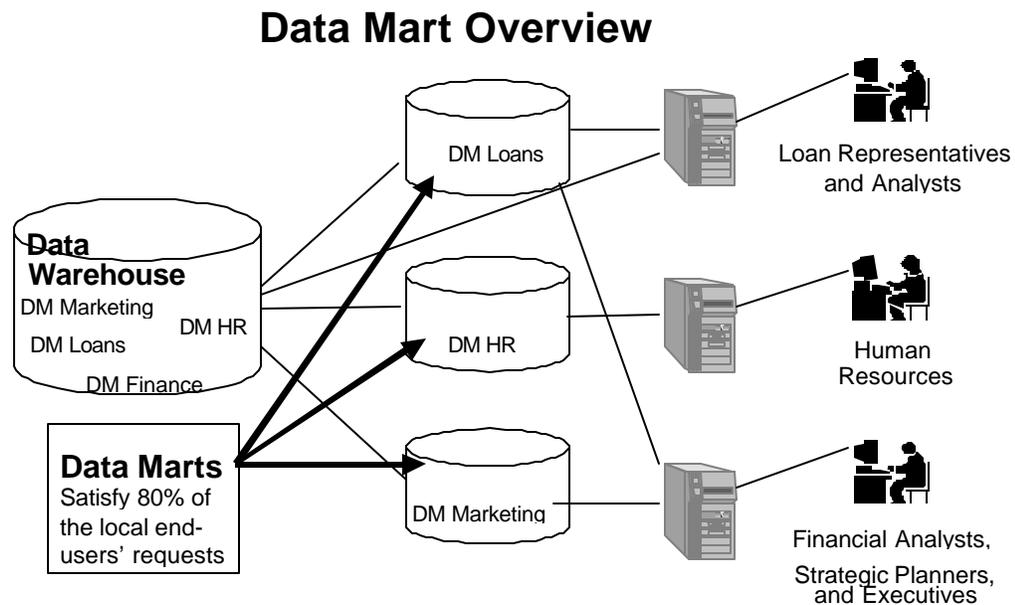


Figure 4-2: Data Mart Overview

Usually, a data mart represents a subset of the data contained in the data warehouse. The data mart depends on the data warehouse for data population. Typically, data marts are delineated along business units (such as customer service, loans, marketing, etc.), as shown in Figure 4-2. A data mart provides its departmental users with the ability to create summaries, aggregates, or data definitions specific to their business and decision processes, while at the same time providing corporate users with an integrated view of the organization through the data warehouse. By ensuring that the data from all data marts originate from the data warehouse, the organization's data integrity is assured while quality and definition issues can be centrally addressed.

The following summarizes some of the key facts about data marts:

-
- Data marts are common and natural extensions of the central data warehouse and can be co-located with the enterprise data warehouse or physically distributed in another location
 - A data mart is usually organized around a functional business unit (such as Financials, HR, or by Channels, etc.)

FAST TRACK DATA MARTS

A fast track data mart is required when the need to see real data and reporting results out of the operational system as quickly as possible outweighs the need to pull this data from an integrated data warehouse. Many vendors that promise a two month turn around for their data mart or data warehouse are actually describing scenarios in which data is pulled directly from an OLTP system into an analytical data mart product. This approach is tactical, yielding access to data for a limited number of users in a short period of time.

However, this approach sacrifices strategic, long term value for a tactical quick win and should not be mistaken for an appropriate long term data warehousing strategy. The risks of a fast track data mart approach over a long period, yielding multiple data marts without a central data warehouse as a core, are as follows:

- Performance — Performance issues, because the same data may be transported and processed several times
- Data consistency — Synchronization issues, where the results of the same query in different data marts yield different results
- Metadata consistency — Data definition may be inconsistent, since metadata among different data marts may not be same

4.1.3 Operational Data Store

An Operational Data Store (ODS) integrates data from multiple business operation sources to address operational problems that span one or more business functions. An ODS has the following features:

- Subject-oriented — Organized around major subjects of an organization (customer, product, etc.), not specific applications (order entry, accounts receivable, etc.).
- Integrated — Presents an integrated image of subject-oriented data which is pulled from fragmented operational source systems.
- Current — Contains a snapshot of the current content of legacy source systems. Historical is not kept, and might be moved to the data warehouse for analysis.

- **Volatile** — Since ODS content is kept current, it changes frequently. Identical queries run at different times may yield different results.
- **Detailed** — ODS data is generally more detailed than data warehouse data. Summary data is usually not stored in an ODS; the exact granularity depends on the subject that is being supported.

The ODS provides an integrated view of data in operational systems. As Figure 4-3 indicates, there is a clear separation between the ODS and the data warehouse.

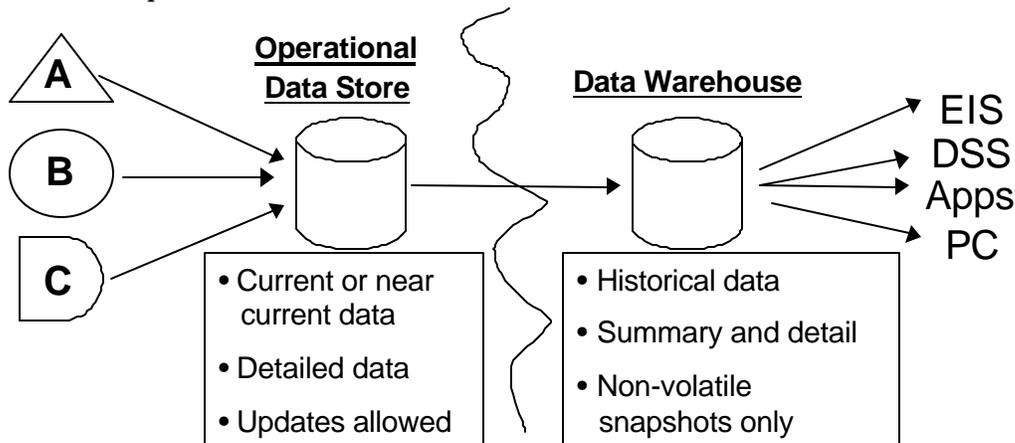


Figure 4-3: ODS and Data Warehouse Differences

The primary purpose of an ODS is to provide operational reporting on a near real-time basis. It is a truly operational system in which data is frequently pulled in from one or multiple systems. It can also be used as an intermediary site to a data warehouse and potentially other downstream systems. User transactions as well as reporting can occur concurrently within an ODS.

Most projects do not have a predefined requirement for an ODS. The ODS is used to fill an architectural requirement to handle detailed, volatile, integrated data for the purposes of operational reporting. Each project will determine if there is time and budget necessary to handle the complexities and reap the benefits an ODS presents.

Operational Data Store Benefits and Drawbacks

Benefits of an ODS include the following:

- Supports operational reporting needs of the organization

- Provides a complete view of customer relationships, the data for which might be stored in several operational databases -- this data can include data from an organization's internal systems, as well as external data from third-party vendors
- Operates as a store for detailed data, updated frequently and used for drill-downs from the data warehouse which contains summary data
- Reduces the burden placed on other operational or data warehouse platforms by providing an additional data store for reporting
- Provides more current data than in a data warehouse and more integrated than an OLTP system
- Feeds other operational systems in addition to the data warehouse

Drawbacks of an ODS include the following:

- Cost (development and maintenance) of another data store
- Continuous updates of ODS can degrade the performance of operational systems
- Possible misuse as a data warehouse might cause inconsistent information due to high data volatility

4.1.4 Differences Between ODS, Data Warehouse (DW), and Data Mart (DM)

DW vs. DM — A data warehouse is an enterprise-wide collection of data; a data mart is a database that emphasizes ease of access and usability of data for a particular purpose. In general, a data warehouse tends to be a strategic but somewhat idealist goal while a data mart tends to be tactical and aimed at meeting an immediate need.

DW vs. ODS — In environments without an ODS, a data warehouse can take on characteristics of an ODS, such as more detailed data or frequent levels of update. However, the data warehouse should not be allowed to degenerate into an ODS. Given the differences between ODS and data warehouse environments it becomes apparent why an ODS should not be used as a data warehouse:

- The ODS and the data warehouse contain different types of data; their content and structures are different
- There is a certain amount of operational data which will never be used in decision making and should not be in the data warehouse

- Although the ODS contains detailed records, the amount of data in an ODS is far smaller than that in the warehouse, which may contain years of corporate data
- An ODS does not usually contain historical data which is necessary for monitoring trends in data

Table 4-2 summarizes key differences between operational data stores, data warehouses, and data marts:

Criterion	Operational Data Store	Data Warehouse	Data Mart
Business Function	Operational support at the department level and below	Decision support at the enterprise, division, or business unit level	Decision support at the department level or below
Size	Range from 10s to 100s of gigabytes	Wide range from low Gigabytes to Terabytes	Usually below 100 gigabytes
Data Sources	Fragmented operational systems and external sources	Fragmented operational systems and external sources	Integrated data pushed or pulled from a data warehouse
Data Schemas	More normalized, flexible design	Dimensional or normalized depending on the use	Dimensional design to improve performance at the expense of flexibility
Subject Scope	Narrow	Wide	Narrow
Time Structure	Current; data synchronized with operational systems to enable operational reporting. Data are stored for the current accounting cycle only	Historical; data is historical to enable informational reporting and strategic planning. All data are associated in some way with time	Historical; data is historical to enable informational reporting and strategic planning. All data are associated in some way with time
Volatility / Update Frequency	High; data is constantly changing to reflect changes in operational systems	Low/Med; data is static and changes depending on frequency of update and level of historic data	Low; data is static and only change when updates to historic data are necessary
Data Aggregation	Low; mostly detail data. ODS summary data, unlike DW summary data, is volatile and represents a small portion of content	Med; some summary data required to improve performance and standardize analysis. Detailed levels of historic non-volatile transactional data.	High; large proportion of summary data required to improve performance and standardize analysis. Only selected detail data required for analytical, DSS-oriented reporting
Usage	Operational; read and write environment (read, update delete)	Informational; read-only environment	Informational; read-only environment
Audience	Clerical audience who make detailed, up-to-the-second decisions	Business analysts or management interested in long-term, strategic direction and decision making	Business analysts or management interested in long-term, strategic direction and decision making

Table 4-2: Operational Data Store, Data Warehouse, and Data Mart Comparison

4.1.5 A Sample Data Warehouse Architecture

Figure 4-4 shows a sample architecture of a data warehouse environment. Note that the following architecture is described only as an example and is not necessarily a recommended approach in all instances.

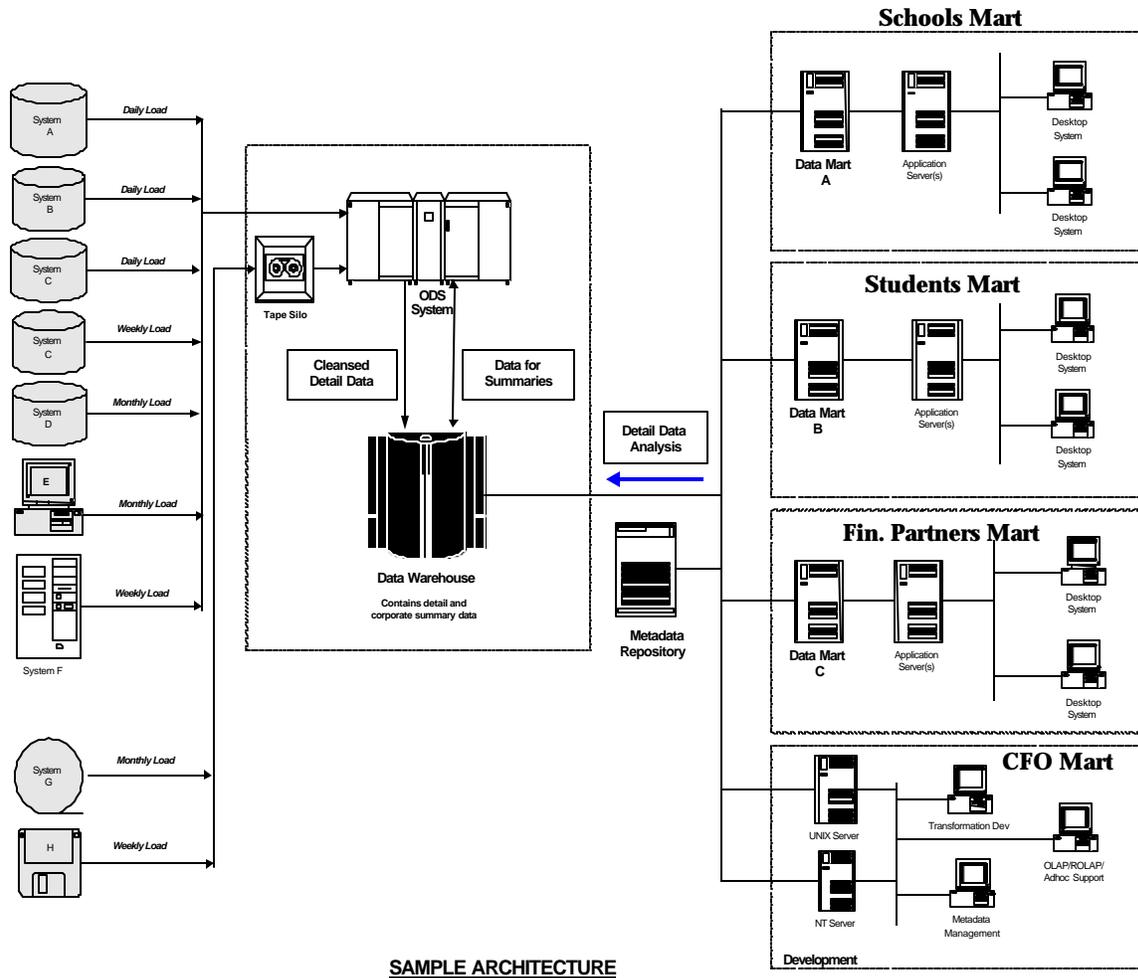


Figure 4-4: Data Warehouse Environment Sample Architecture

The sample architecture has the following features:

- Data from numerous internal operational sources with various formats are loaded at different time intervals
- Data from external sources may also be loaded at certain time intervals (not shown)
- An ODS provides a staging area for the integration of the data warehouse data
- The ODS system also provides the ETL processes, thus eliminating a load from the operational and data warehouse environments
- The Metadata repository is accessible by all users of the data warehouse, thereby defining the meaning of the data warehouse contents for all users
- The application servers can be augmented by web servers (not shown) to provide internet/intranet access to the warehouse for end-users
- The data warehouse contains the current detail data and summaries used at the corporate level
- The data marts contain the departmental level summaries; they may also contain detail data which is only of interest to a particular department. Note that this detail is not duplicated at the data warehouse level, but it passes through for integrity purposes

The centrally located data warehouse, as shown in Figure 4-4, is common to many warehouse implementations. The need for physically distinct ODS constructs and multiple distributed data marts vary considerably between projects. Generally, organizational concerns, budgetary constraints, and technological complexity of managing a distributed warehouse greatly favor this approach.

4.2 Data Warehouse Architecture Drivers

The data warehouse architecture design is determined by decisions on how to place and distribute data and the physical and logical design of the data. These decisions, in turn, should be based on the knowledge of user requirements. The requirements that drive a data warehouse architecture are as follows:

- Granularity of data
- Data retention and timeliness

- Reporting capability
- Availability
- Scalability

These requirements are discussed in detail in the following sections. Architectural decisions are discussed in subsequent sections.

4.2.1 Granularity of Data

Prior to the emergence of the data warehouse concept and its summaries, most analysts gathered data at the detailed level and created their own summaries. Such a process suffered from poor response time, duplicate effort, and data quality problems. A well designed data warehouse can improve this process by offering the right data at the right level of granularity. Data granularity refers to the degree of data aggregation or summarization. The lower the summarization (that is, the greater the detail or atomicity) of the data, the lower the level of granularity. Similarly, the higher the summarization, the higher the level of granularity.

As Figure 4-5 illustrates, data warehouses have a distinct data structure. The different levels of granularity help the users of the warehouse access data quickly and accurately. Three different levels of granularity are available:

- Detail (Atomic, raw) = lowest possible granularity
- Lightly summarized = medium granularity
- Highly summarized = highest granularity

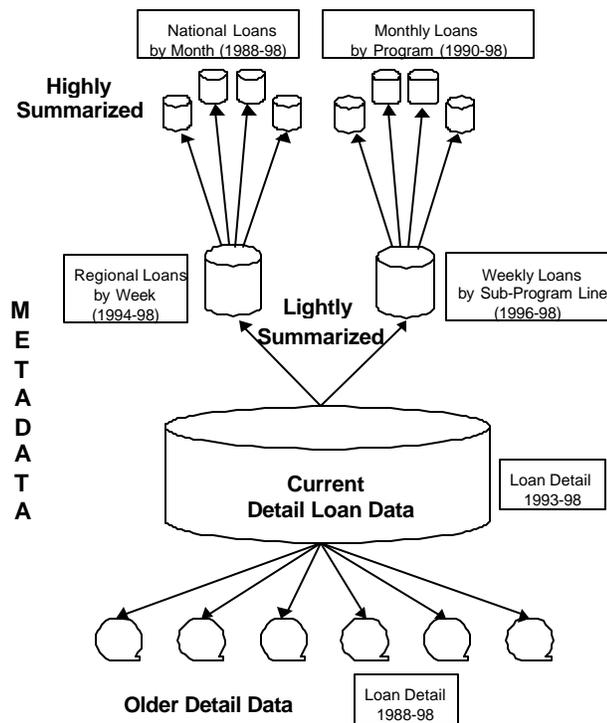


Figure 4-5: Data Structure Within the Data Warehouse

Detail Data — Two types of data exist at this lowest level of granularity; current detail data and older detail data. The current detail data is the most significant portion of the data warehouse because it:

- Reflects the most current transactions in the organization, often up to five years of data
- Is very large in volume and stored at the lowest level of granularity
- Is stored on disk, for faster access, which makes it the most expensive and complex to manage
- Is the basis for all the summary and aggregate data found in the data warehouse and data marts

Older detail data contains the same level of granularity as the current detail data. It is not accessed frequently and is usually stored in some off-line medium. Due to its large volume and low access frequency, it is usually not economical to use disk storage. As it ages, current detail data is normally rolled off and migrated to this off-line location based on a preset schedule (for

example, detail data older than five years). A key question is how much of this older data to maintain. One guideline is to maintain the same age of detail data as the oldest summaries, in case they need to be recreated.

Summarized data — Lightly summarized data is derived from the current detail level and is stored on disk. Highly summarized data is derived from the lightly summarized information. This level of summary is compact, easily accessible and may be located outside the data warehouse depending on business needs. Regardless of its physical location, highly summarized data is part of the data warehouse.

Summary tables are created to eliminate the duplicate processing that would be performed by many users in order to increase aggregation-specific performance. Most analyses start at the summarized level. Supporting evidence is found through drill-down.

The critical issues with all summaries are:

- The time interval used for summarization (such as year, quarter, hour)
- The contents (attributes) of these summaries

A critical success factor for the creation of the summaries is the involvement of the business owners. Analysis is performed to determine their needs; processes help determine the types of summaries required.

The different levels of data granularity in the warehouse (atomic, lightly summarized, and highly summarized) have different levels of usage. Generally, as the level of summary increases so does its usage. The more summarized the data, the quicker and more efficient it is to access. If an organization finds that its users are requiring a large amount of processing at the detailed levels, then its summaries may not be properly set and/or its users not properly trained.

4.2.2 Data Retention and Timeliness

This factor deals with issues, such as:

- How current must the data be according to user requirements?
- How long must this data be kept in the data warehouse before it is archived?

These factors can drive the replication and/or distribution of data warehouse data from the central site to reside closer to the user, or to the improvement in network bandwidth so that distributed users can perform reporting tasks from a central site. Also, the lower the granularity of the required data, the longer the ETL process takes and the less available or responsive the

data warehouse may be. This may also affect how data is stored and placed within the data warehouse architecture to give users the data and performance they need.

4.2.3 Reporting Capability

Reporting access needs can range from simple reports and queries to detailed analysis and knowledge discovery. The more processing power and knowledge discovery capability are demanded by the user, the more likely that data must reside near the user for acceptable processing response times.

4.2.4 Availability

Availability requirements are driven from the end-user perspective, such as whether they require 24x7 access to data and constant uptime. This can drive the need to either distribute data marts or the construction of a highly available data warehouse with high cost hardware and software (such as mirroring, logging). A distributed data mart, separate from the data warehouse server (see “data distribution and placement”), can provide access to users while the data warehouse server is down due to operational activity or a hardware failure. Extended availability requirements will escalate data warehouse costs dramatically. Every effort should be taken to avoid expectations that the warehouse will be availability 24x7.

4.2.5 Scalability

Scalability is the ability to increase the system load in terms of the number of users, data volume or processing without significantly decreasing the response time or incurring costly hardware/software expenses. The scalability of a data warehouse can be increased by careful attention to the following factors:

- DBMS software selection
- Technical architecture, and in particular, the selection and usage of staging, ODS, and distributed data mart components
- Hardware and operating system platform selection and the use of parallelization and clustering techniques

Any successful data warehouse experiences dramatic growth. The earliest architecture designs must reflect the expectation that scalability will be mandatory.

The above criteria are summarized in Table 4-3:

Criterion	Centralized Data Mart Architecture	Distributed Data Mart Architecture
Granularity	Lower	Higher
Reporting Access	Lower	Higher
Data Retention & Timeliness	Higher	Lower
Availability	Lower	Higher
Scalability	Lower	Higher

Table 4-3: Decision Criteria for a Centralized vs. Distributed Data Mart Architecture

4.3 Data Distribution and Placement

This section explains the issues involved in the physical placement and distribution of data. The sheer volume of data in the data warehouse constantly pushes the envelope of scalability of the underlying DBMS and hardware. Often, a data warehouse is already on the most powerful hardware and operating system platform that the organization supports, and therefore getting “a bigger box” is not a feasible solution. Technical limitations in today’s hardware and software, such as scalability and performance, dictate that data placement and distribution be considered a very high priority when making architecture decisions.

It may seem contradictory to consider “distributing” a data warehouse. After all, a data warehouse is meant to be a single, centralized database. However, there are two main methods to distribute data in a warehouse. One is rarely used, while the other is used frequently. The main architectural alternatives for data placement and distribution are as follows:

4.3.1 Distributed Data Warehouse

In a fully distributed architecture, the data is distributed over multiple databases and/or locations without maintaining a complete copy of the data in a central warehouse. This is sometimes known as a “federated database.” This configuration is intended to allow access to enterprise-wide data regardless of its location. With the current technology, the design and management of a fully distributed data warehouse is not easily implemented and should be used only in rare circumstances. Therefore, it is not discussed further in these guidelines.

4.3.2 Data Segmentation via Distributed Data Marts

A well designed data architecture includes the definition of departmental data marts. Departmental data marts, containing data derived from the central data warehouse, enable different departments within the organization to address their unique decision making needs by summarizing, aggregating, or defining data in terms which are relevant to that department.

Typically, these departmental data marts are designed along business functions, such as sales, accounting, manufacturing or shipping. In a strategic data warehouse architecture, these departmental data marts should be dependent on the central data warehouse for their content. Corporate summaries and definitions (quarterly revenue, gross margin, profit) remain in the central data warehouse where the entire corporation can access and view them in the same manner and with the same understanding.

To populate a data mart, data can be selected from a central data warehouse by any of the following criteria:

- Time intervals (year, quarter, month, day, etc.)
- Line of business
- Geographic region
- Product line
- Other characteristics
- Any combination of the above

The data marts can be distributed depending on performance, scalability, and availability requirements in the following ways:

- Centralized data marts — The data marts are in the same central location as the data warehouse the same server (scenario 1).
- Centralized data marts but on separate servers — The data marts are located in the same central location as the data warehouse but they are on separate servers. The data mart and data warehouse may share network drives through a high speed LAN (scenario 2).
- Distributed data marts — The distributed data marts are located close to the users (scenario 3).

Figure 4-6 illustrates these three scenarios.

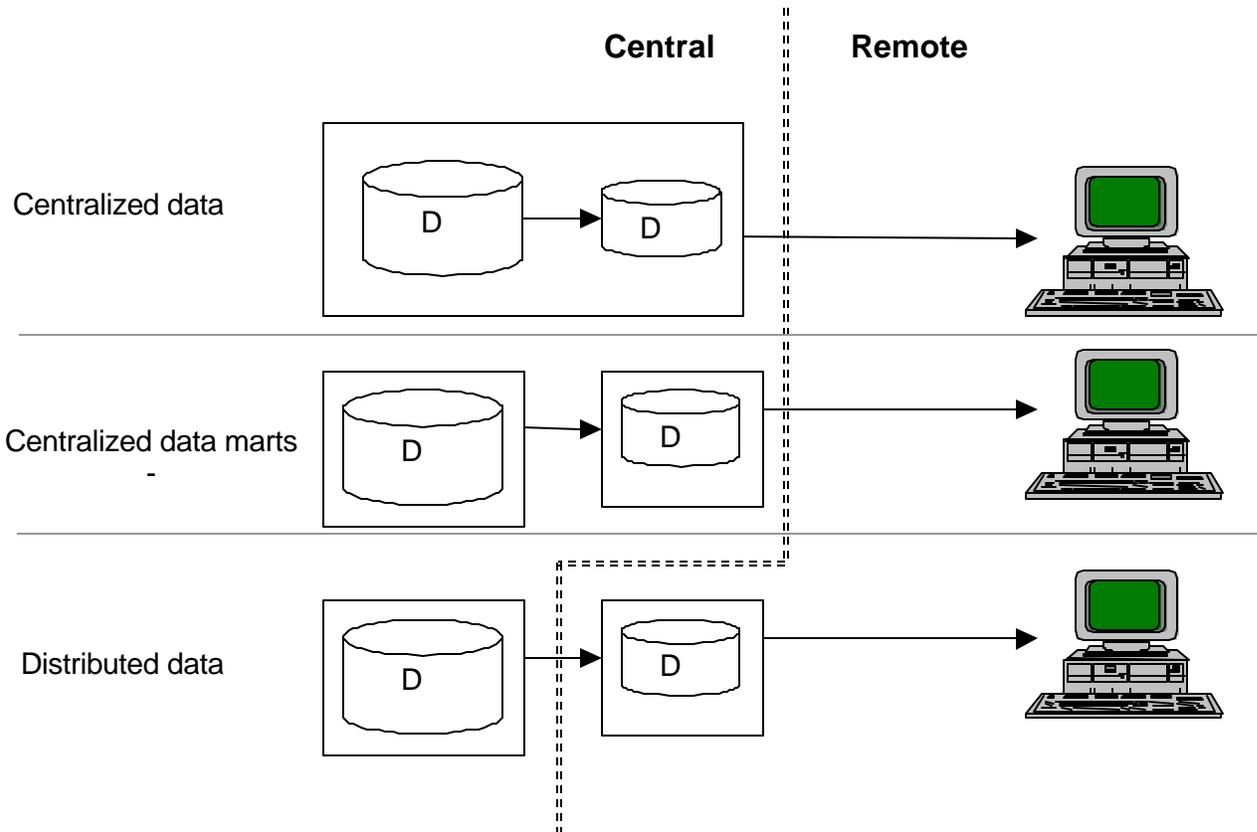


Figure 4-6: Data Mart Distribution Scenarios

4.4 Data Modeling for Data Warehousing

Data modeling is the most fundamental designing task in data warehousing. Data model detail can range from a basic one-page high-level diagram to a detailed model that covers the walls of the data administration office.

Data modeling is nothing new. It has been used in one form or another throughout the history of systems design. However, for data warehousing, data modeling takes on a whole new level of importance. New concepts, terms and techniques now exist to handle the specialized needs of a data warehouse.

4.4.1 Data Model Overview

The data model should function as the key building block to represent the information that supports the business. The information is broken into basic data elements to describe its name, type, relationship, and organization.

During a data warehouse project, several data models will be created depending on the requirements of the project. As the project complexity increases, a well designed data model becomes the blueprint of the data warehouse requirements.

Typical models created during data warehousing are the conceptual data model (CDM), the logical data model (LDM) and the physical data model (PDM). Each of the models listed are an evolution of the prior model and are refined with more analysis. Each model has specific features to serve a specific purpose.

Specialized modeling tools are used to generate the model diagrams, resulting schema, and sometimes SQL to generate and alter the actual database. However, basic drawing programs can also be used when the scope of the model does not demand a more robust tool.

Conceptual Data Model

During the Planning phase of the project, the conceptual data model is created to capture the high-level data requirements for the project. Since the model captures the highlights of the client's information needs, it is the only model that effectively reflects the enterprise level. Depending on the requirements, the enterprise-wide vision may need to be emphasized to help guide the client in the development of an overall data warehousing strategy. Detail models that reflect the project's scope will be created during logical and physical data modeling. The conceptual data model is the precursor to the logical data model; it is not tied to any particular solution or technology. Entities, relationships, major attributes, and metadata across functional areas are included. During successive releases, the conceptual data model should be validated and updated if necessary. An enterprise should have only one conceptual data model.

Logical Data Model

During the design phase of the project, the logical data model is created for the scope of the complete project. A portion of the conceptual data model will be fully attributed and completed as the logical data model. The logical data model reflects the technology to be used. In today's environment, this typically means either a relational DBMS or a multidimensional tool. But if the client should be using an older DBMS such as IMS or IDMS, the logical model will be quite different than if an RDBMS is to be used. The logical data model reflects a logical data design that can be used by the developers on the project. For an RDBMS, that means logical tables (views) and columns.

Physical Data Model

Like the logical data model, the physical data model is created during the design phase. This modeling activity should reflect the scope of the specific release of the project. The model's final design will be highly dependent on the technical solution for the data warehouse. The purpose of this model is to capture all the technical details required to produce the final tables, and physical constructs such as indexes and table partitions. The logical data model will serve as a blueprint to the project team while the physical data model is a blueprint for the DBAs. All the functionality reflected in the logical data model should be preserved while creating the physical data model. The generated table schemas will be identical to the physical data model.

4.4.2 Normalized vs. Dimensional Modeling

With the advent of data warehousing, database usage has expanded to decision support systems from mainly transaction-based systems. The main emphasis of read access to large amounts of data, both detail and summary, is unique to data warehousing. The transaction based systems' reliance on inserts, updates, and deletes of specific records results in a contrasting data and technical architecture. This difference has generated a great debate over data modeling. The two main schools are normalized and dimensional modeling.

The main objective of normalized modeling is to reduce data redundancy for ease of update, insert, and delete performance. Normalized modeling has been heavily applied in transaction based systems such as OLTP systems. Normalized modeling usually produces a very detailed entity-relationship diagram.

The objective of dimensional modeling is to represent a business framework that is intuitive and provides high access performance. Dimensional models are more intuitive; end users are better able to understand and use them. This discipline will produce a star schema. The dimensional approach benefits from shorter, less complicated join paths than the normalized method.

Reasons to use dimensional modeling for data warehousing include:

- Easily understood and communicated relationships of information needs
- Improved query performance
- Simpler access

Dimensional modeling is the discipline that is gaining increasing acceptance in data warehousing. Its structure and objectives are in line with the data warehouse functions. We generally recommend the dimensional modeling technique for data warehousing. Normalized modeling will continue to be valuable in the creation of transaction based systems.

Since dimensional modeling is most applicable to data warehousing, additional information on dimensional modeling will be provided in later sections.

4.4.3 Sample Model Diagrams

There are many data modeling disciplines and techniques, chosen generally based on project requirements, the modeler's particular experience and the client's predefined modeling standards. The technique will dictate the final diagram representation. The following data model diagrams are typically seen at data warehousing engagements:

Figure 4-7: Entity-Relationship – Most applicable to transaction systems.

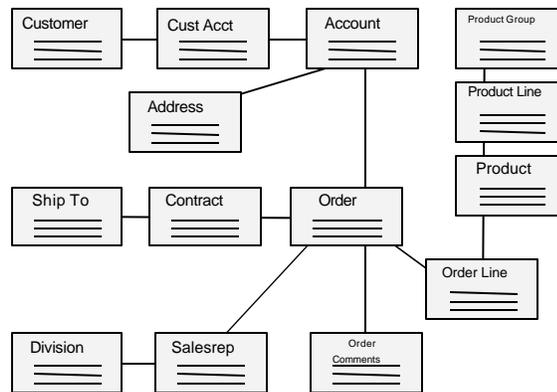


Figure 4-7: Entity-Relationship Diagram

Figure 4-8: Star Schema – Most applicable to data warehousing.

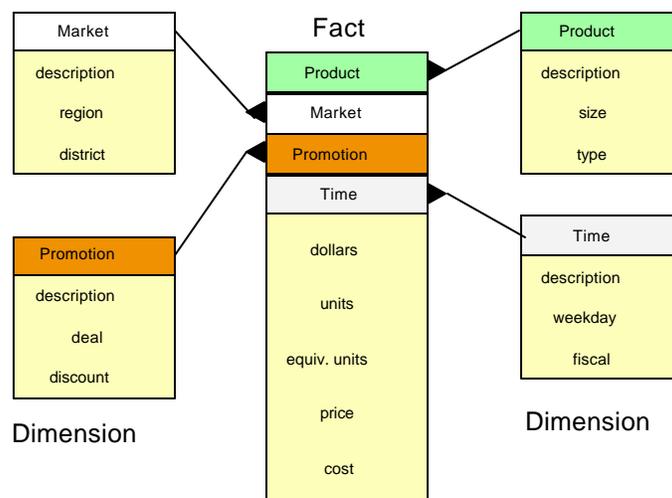


Figure 4-8: Star-Schema Diagram

Figure 4-9: Logical Diagram of a Single Dimension - Each attribute appears in a separate bubble. The entire collection of bubbles will comprise a single table. This technique is not recommended because it does not generate enough value in relation to diagramming effort.

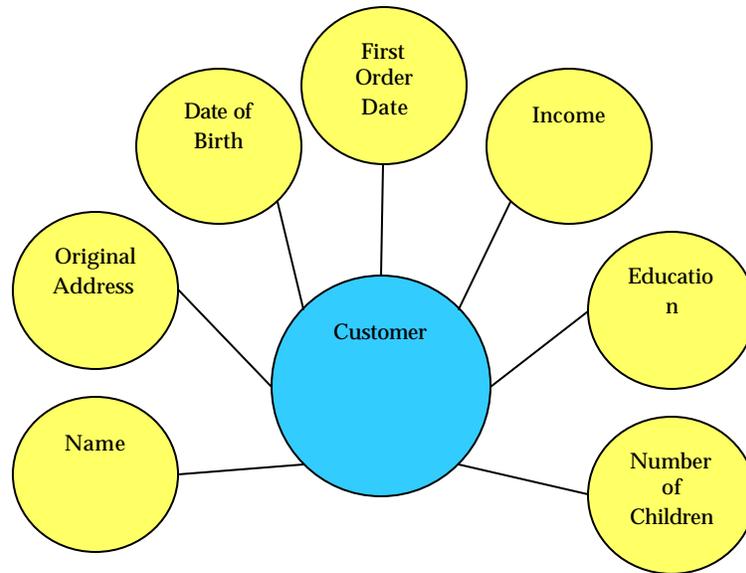


Figure 4-9: Logical Diagram of a Single Dimension

4.4.4 Dimensional Modeling Components

Regardless of the modeling discipline, data models basically result in entity, attribute, and relationship identification. An entity is some unit of data that can be classified and have stated relationships to other entities such as customer, account and product. An attribute is the specific component of an entity such as customer name, age, and birth date for the customer entity. Attributes are assigned to entities in accordance with their descriptions. Relationships are defined so that join results can be shown. Any number of the following relationships may be listed between entities: zero-or-one to one, one to one, one to many, many to many, etc.

The following terms are used in dimensional modeling:

- Entities are defined as dimensions
- Attributes that are textual and help define a dimension are still called attributes
- Attributes with unknown values that are usually numeric – often in unites of dollars or quantities- are called facts
- The dimensions are physically turned into dimension tables and the facts along with the foreign keys to the dimension tables are turned into fact tables.

There is no special term to represent relationships; all variations of relationships are supported.

Dimensions

Dimension tables are a composite of one or more columns as a primary key and other columns known as attributes. As an example, a time dimension is shown in Figure 4-10:

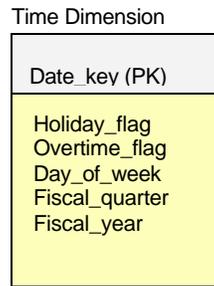


Figure 4-10: Dimension Table Diagram

Dimensions typically contain hierarchies. Hierarchies show the parent-child relationships between elements of the dimension. Hierarchies are used to logically group and analyze information within one dimension. In other words, dimensions contain hierarchies, which are natural structures within business dimensions, as shown in Table 4-4 below:

Description	Hierarchy
Geography Dimension - table is at the store level but can be rolled up through to region	Store within Zip Code Zip Code within City City within Country Country within State State within Region
Product Dimension Hierarchy	Universal Product Code within Product Line Product Line within Brand Brand within Category Category within Department
Time Dimension Hierarchy	Day within Week Week within Month Month within Quarter Quarter within Year

Table 4-4: Hierarchies

Facts

Fact tables are a composite of columns which are 1) foreign keys to the dimension tables and 2) columns simply called facts. Facts are the key measures (generally either quantities or dollars) that define the performance of the business. Samples of a fact include:

- A sale amount
- A net profit margin
- A purchase price
- A percentage of change in finished goods inventory
- A shipment amount

These facts are often summarized into key performance indicators (KPIs) that are based on critical success factors of the business. Typical KPIs include net income and inventory turns. Facts do not exist in a vacuum. They exist in the context of time, place, product, etc. For example, “units sold” means sales of a particular product, at a particular time, in a particular place. A fact stands at a cross-section of multiple dimensions. The combination of facts and the “who, what, where, and when” of these facts can be represented in a dimensional star schema.

4.4.5 The Star Schema

Dimensional modeling was popularized by Ralph Kimball to address the need for data designs that would give good performance for decision support systems despite being built on RDBMSs that were designed for OLTP applications. The model produced by dimensional modeling is sometimes referred to as a “star-schema” because of the star-like representation of the tables. Dimensional modeling stores data multi-dimensionally using fact and dimension tables. Foreign keys are used to join tables that describe the dimensions (dimension tables) to a central table containing the facts or KPIs (fact table). This produces a data model that contrasts sharply with the traditional normalized design.

Loan Program Sample Schema

The exhibit below depicts a sample loan schema. The schema contains all the important components of program information with dimensions.

Loan Schema

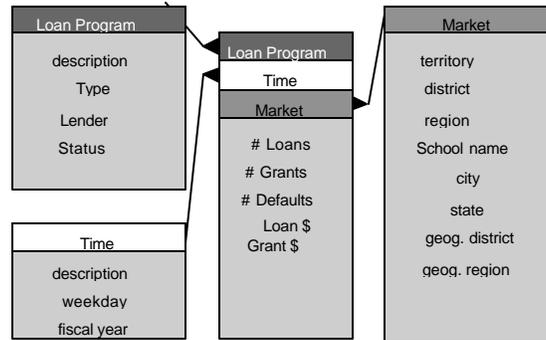


Figure 4-11: Example of a Product Profitability Schema

Outrigger Tables and the Snowflake Schema

An outrigger table can be considered a codes table for a dimension table. It serves to normalize the data by removing descriptions of items within the dimension that are only accessed infrequently. The descriptions in the outrigger table could be incorporated into the main dimension table, but are not included because the end users do not typically access the information using these items as constraints. This normalization occurs at the cost of the extra table join for the limited number of times the information is accessed. The extended star schema which results from the use of outrigger tables is also referred to as a snowflake schema. Figure 4-12 illustrates the use of an outrigger table:

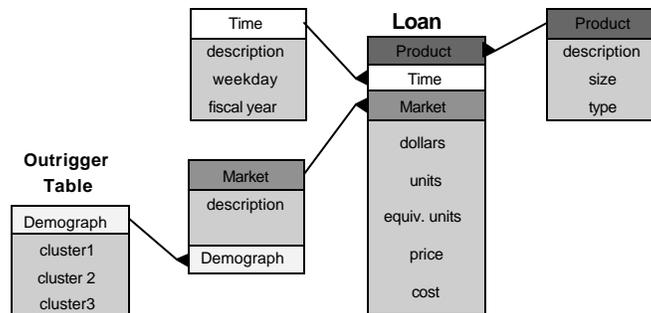


Figure 4-12: Outrigger Table Example

Star-Schema Families

Most data warehouses will not be able to use one single fact table that contains all the facts for every business system that is being modeled. While some of these systems will be very distinct, it is likely that they will share many identical dimensions. Dimension tables can be “shared” by

multiple fact tables when the dimensions are identical. When two or more fact tables “share” several dimension tables, they are called a “family” of star-schemas. This is illustrated by the Family of shipments and warehouse withdrawals shown in Figure 4-13. The product, market, and time dimensions are shared, while the carrier dimension is only associated with the shipments table.

It is important that data warehouses share dimensions whenever possible. If different departments build data warehouses with similar, but slightly different, dimensions, you can be confident that the dimensions will digress into disintegrity. Pulling dimensions into agreement so they can be shared produces *conformed dimensions*.

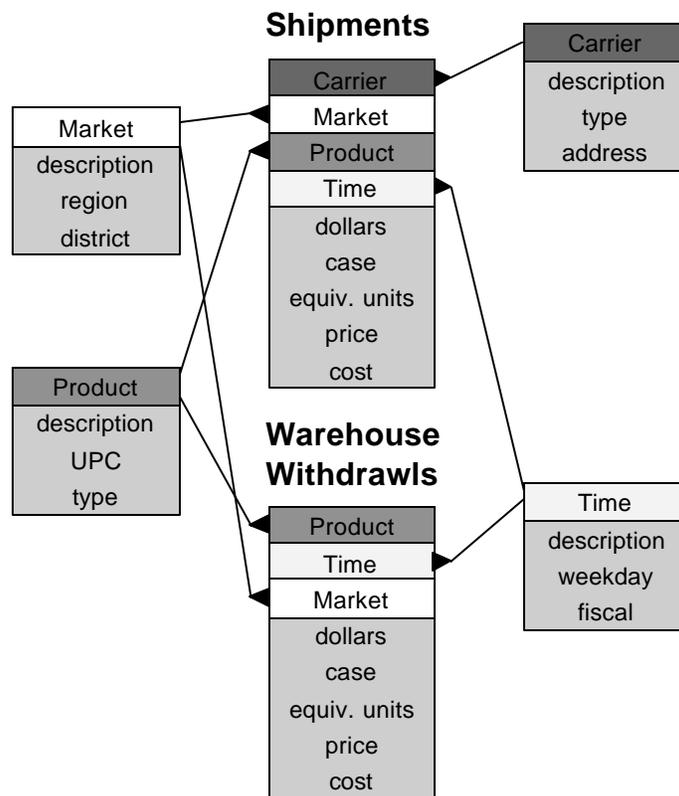


Figure 4-13: Family of Star-Schema Example

In a more complicated scenario, such as in a banking analysis, it may be possible to logically combine certain facts within the family, but not others. This creates a need for sub-fact tables which would contain the facts that are unique to each sub-group. The main fact table would contain facts such as account balance, withdrawals, and service fees. A sub-fact table for checking may contain number of checks written, overdraft charges, check fees, etc. A sub-fact

table for a savings account may have interest earned, minimum balance, and other facts not related to all the other account types. While it is possible to implement this type of schema using only one fact table, it would increase the amount of wasted storage space. Some unused storage space in the central fact table could be tolerated to avoid the complications of having multiple sub-fact tables while accessing the unique facts.

The Risk of Shared, but not Conformed, Dimensions

Occasionally two fact tables within a family share a common dimension, but have different meanings for the attributes of that dimension. For example, an insurance company with a marketing and a claims department share a customer dimension. While marketing department has defined the customer to include prospects, the claims department defined customer to mean only those with active policies. Marketing populates the customer dimension and doesn't realize there is a discrepancy in the definition. The star schema shown in Figure 4-14 reflects this sample scenario:

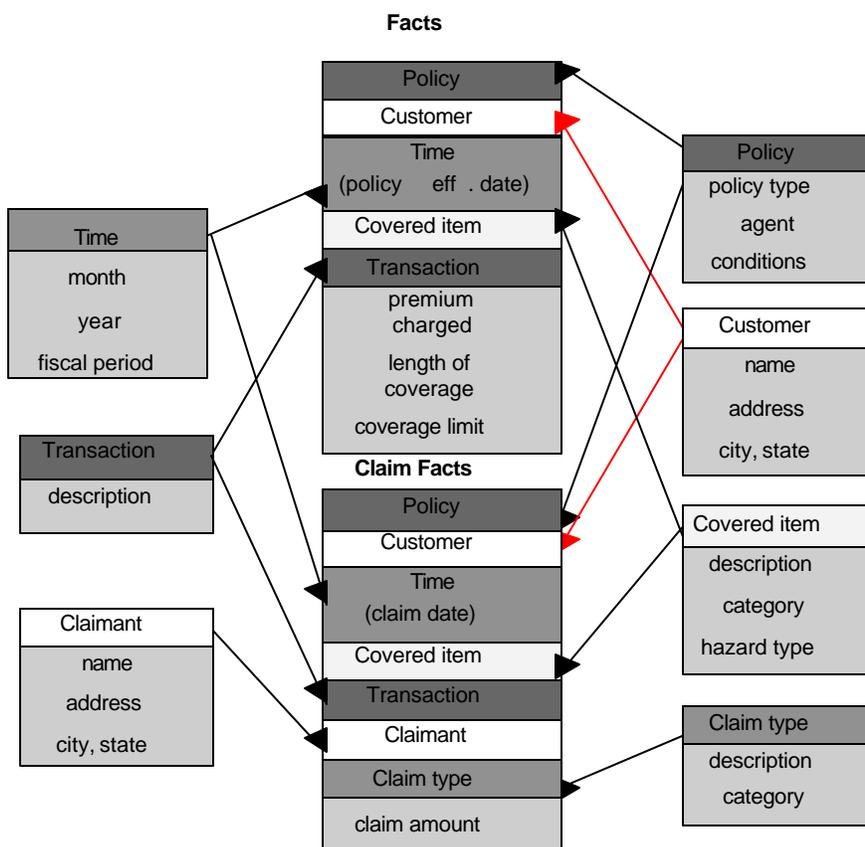


Figure 4-14: Example of Shared but not Conformed Dimension

The challenge with such schemata is to intelligently analyze the two fact tables to meaningfully analyze the company data.

4.5 Physical Design Issues

Once the data warehouse is modeled using a star schema diagram, the dimensions and facts must be translated into a physical model – it is generally close to a one-to-one relationship. However, this translation to the physical world does not take into account table indexing and partitioning. These major physical design issues are discussed below.

4.5.1 Indexing Strategy

The optimal indexing strategy will both minimize the number of indexes and optimize performance for all critical data accesses. It is tempting to add an index to satisfy every data access that hits a table. However, more indexes per table translate into additional time loading tables and restructuring indexes when inserting and deleting rows. More indexes also mean higher administrative overhead for example, each index must be considered when sizing and planning for growth). Methods to optimize indexing strategies are as follows:

- When the data warehouse is implemented in an RDBMS and the star schema defines the tables and columns of the database design, then indexes are generally defined on the primary key of the fact table and on multiple hierarchy elements within the dimension tables. Additional indexes on the fact and dimension tables may be necessary depending on the access patterns expected.
- Split tables by key — For example, a sales history table can be split into separate tables based on region. This leads to smaller tables, smaller indexes to support them, and faster access times to maintain (i.e., load or backup) each table. However, it makes it much more difficult to support queries that span regions, like questions about national sales by product.
- Use minimal indexes, or none at all, for tables with a high rate of inserts/deletes, tables that are accessed in large volumes in a specific order, or for smaller, highly accessed tables. In these cases, a sequential table scan may be acceptable. Investigate storing the records at load time in sorted order (so an index is not needed), or try using a clustered index, if provided by the RDBMS.
- Data warehouse usage patterns cannot be accurately predicted. Therefore, it is not feasible nor desirable to create indexes for every column in the warehouse in anticipation of its use. It is more prudent to create the primary and secondary indices needed and create more indices in the data marts, since this is where most of the users will be working.

- Choose the proper index type for critical data accesses. Using the 80/20 rule, determine the most important accesses that will require indexes. Then, determine the type of indexing required:
 - Sorted index
 - B-tree index
 - Hash index
 - Bitmap index
 - No index
- The index strategy of any organization should be regularly reviewed and evaluated against usage patterns. Indexes not used should be eliminated and new ones created.

The tradeoffs involved with these index choices are beyond the scope of this document. Many technical publications describe these options in detail.

4.5.2 Partitioning Strategy

Database partitioning is defined as the splitting of tables and indices within a single database; for example, the portions of the last past year's sales data can be partitioned into smaller monthly sales data for faster access. This is often supported by latest RDBMS products but can also be implemented manually by creating separate tables and allowing the application programmer to decide which to use.

Determining the most appropriate way to partition data may yield dramatic performance improvements for certain applications. An understanding of the data accesses by applications is key to this process. Most databases support partitioning - either horizontal (split by record - this is the most common) or vertical (split by column). Partitioning can also be built into applications.

It should be noted that partitioning is not without its share of pitfalls. Queries that need to access data from a single partition run much faster because the partition subdivides the database into smaller units. If a query needed to access data across multiple partitions, then it would run much slower than an equivalent query in a non-partitioned database. The other advantages of partitioning, namely, ease of database administration and faster loading, should be balanced against the end-user requirement of executing queries that require access to data across the partitions.

Data can be partitioned in several ways:

- Time intervals (year, quarter, month, day, etc.)

- Line of business
- Geographic region
- Product line
- Other characteristics

Usually, the easiest and the most appropriate partitioning strategy for the current data is along the time dimension. An example of partitioning is as follows:

An automobile company may choose to partition its data in the following physical units, using date and product line as criteria:

	Sedan Sales	Truck Sales	Van Sales
1995	1995 sedan sales	1995 truck sales	1995 van sales
1996	1996 sedan sales	1996 truck sales	1996 van sales
1997	1997 sedan sales	1997 truck sales	1997 van sales

Table 4-5: Example of a Partitioned Data Table

Choosing the right data partitioning strategy is one of the most important database design decisions. Benefits of partitioning include:

- Flexibility in migration — Partitioned data can be migrated from one processing complex to another with impunity
- Minimizing contention on database tables by distributing partitions over multiple distributed spindles
- Utilizing the benefits of SMP and MPP configurations
- Utilizing additional disk spindles to access data in parallel should improve query performance
- Flexibility in load and backups — Operations, such as backups and loads, can be managed in smaller, more discrete pieces (for example, one partition at a time). For loads, indices on new data can be created only on the changed partitions.

Data should be partitioned and broken down finely enough so that indices can be added freely. When the addition of another index causes the partition to exceed the allocated space, then the index should be created if absolutely necessary. This could also be indicative of a partition sized inadequately.

4.6 Data Warehouse Technologies

This section is composed of two sub-sections. The first sub-section addresses some of the confusion surrounding MDDB vs. RDMBS. The second section focuses on data warehouse-specific RDBMS features which are classified in the following categories: indexing methods, query processing, partitioning, and operating system/hardware support. Refer to the next section for examples of vendors, products, and selection criteria.

4.6.1 RDBMS vs. MDDB

There is often confusion in the data warehouse community in describing and deciding what technologies are appropriate for data warehouses and data marts. In general, the following rules apply:

- Data marts are generally implemented using a multidimensional database (MDDB) product. This term is a fancy way of referring to OLAP products, which are described in detail in the End-User Access section of these Guidelines. However, keep in mind that MDDBs can be implemented with the following technologies:
 - Relational databases — (ROLAP, used by vendors such as MicroStrategy and Information Advantage)
 - Sparse matrix technology — (MOLAP, used by vendors such as Arbor and Gentia); this technology is generally the one to be considered a true “MDDB”
- Relational databases are generally used to implement large data warehouses based on the size and scalability requirements. MDDB technology cannot yet support the size and scalability needs of a data warehouse unless it is relational technology.

Table 4-6 summarizes the differences between MDDB and RDMBS:

Criterion	MDDB	RDBMS
Decision support data analysis functionality (slice, dice, drill down, etc.)	Extensive	Minimal built-in, requires extensive application programming
Scalability with respect to data volume, # of users	Low	More than MDDBs by several orders of magnitude
Support for dynamic joining of data	Low	High
Maturity of technology	Low	Very robust
Speed and support for predictable data usage patterns and paths	High	Low
Speed and support for unpredictable data usage patterns and paths	May not be supported	High
Data Loading	Full loading required	Incremental load is also possible.
Concurrency control, transaction support	Immature	Proven

Table 4-6: Differences Between MDDB and RDMBS

4.6.2 Data Warehouse-Specific RDBMS Features

The major DBMS vendors (IBM, Oracle, Informix, Sybase, and Microsoft) are aggressively implementing data warehouse-specific features in their RDBMS products. These features have significantly reduced the need to use data warehouse-specific database products such as Teradata or Red Brick. Hence, it is important to understand the following concepts:

- Indexing methods
- Query processing
- Partitioning
- Operating system/hardware support

Indexing Methods

Most DBMS products feature basic indexing technology, such as sorted and clustered indexes. For very large decision support environments, these basic indexes may not be able to handle fast access over large volumes and multiple dimensions of data. Many vendors have implemented specialized technologies to support fast access for data warehousing databases, including:

- Bitmap indexing
- Join indexes for high-speed multi-table joins

As a general rule, always investigate the claims and limitations of these products and how they relate to your specific environment, as there is limited firm experience with these relatively new technologies. B-tree indexes are described below to set context.

B-TREE INDEXES

A B-tree index, or balanced tree index, is an indexing scheme which is common to most DBMSs. It is used to speed up searches on numeric or character fields by allowing a “phone book” type search (see Figure 4-15). The B-tree is a standard tree that provides:

- Improved search speed
- Balanced retrieval times

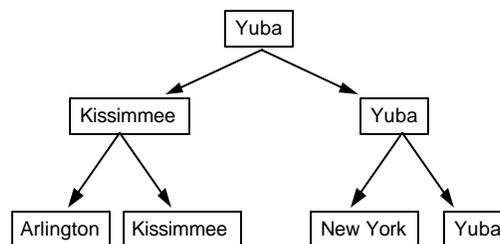


Figure 4-15: A Simple B-tree Index for City

BITMAP INDEXES

In its basic form, an inverted list or bitmap index creates an array for each unique value of a field and indicates for each record in the array whether the value is true (1) or false (0). Bitmap indexes address some of the limitations of B-tree indexes. Unlike B-tree indexes, bitmapped indexes efficiently represent low-cardinality data, are much smaller and easier to maintain, and can be processed simultaneously.

Bitmap Example — Suppose an analyst is investigating demographic trends of the company’s customers. A question that might arise in this analysis is “How many of our married customers live in the central or west regions?” A query tool could convert this request into a SQL query:

```
select count(*) from customer where marital_status =  
'married' and region in ('central', 'west');
```

Bitmap indexes can be used to process this query. One bitmap is used to evaluate each condition (or predicate), and the bitmaps are logically combined to evaluate ANDs and ORs. These AND and OR operations are extremely efficient.

Bitmap limitations— Some limitations of basic bitmap indexes are as follows:

- As the number of unique values increases, the amount of storage can grow rapidly. However, to make this indexing technology practical for data with many discrete values, some products offer compression techniques to take advantage of the binary nature of this storage approach.
- Bitmapmed indexes may be limited in their ability to aggregate data, implement relational joins, and retrieve the actual raw data values.

JOIN INDEXES

A star schema can be build in any relational database. This schema is simply a collection of tables that satisfies certain criteria, and the queries on a star schema are composed of standard SQL. However, while any RDBMS can store data in a star schema, it may not have a method for supporting the efficient execution of star queries.

- Basic method— In a basic OLTP-type approach, the system will choose to join the fact table to a single dimension table as the first step of query processing. The second step joins the intermediate result from the first step to the second dimension table. This process continues until all tables have been joined. The shortcoming of this approach is that the database is processing large amounts of data from the fact table multiple times (once for each join of a dimension table). Because the fact table is usually the largest table, this execution strategy can be very expensive.
- Join index — Although there are many variations of join indexes, these indexes all share two common characteristics: the index structure spans more than one table, and the index is used to improve the performance of relational joins.

The concept of a join index can be illustrated by an example involving the two tables EMP, containing employee records, and DEPT, containing descriptions about each department. Suppose, these two table share a foreign-key/primary-key relationship on the DEPTNO key. The following query might be used to find all of the employees in a given department:

```
select employee_name from emp, dept
where emp.deptno = dept.deptno
and dept.department_name = 'SALES';
```

This query can be evaluated using a join index. For example, there could be a join index on the dept.department_name column and the emp table. The index structure would list, for each possible value of dept.department_name, the rows of the corresponding employees in the emp table.

In a star schema, a join index can be created across a single fact table and all of its dimension tables. Thus, a join index provides immediate access to only the relevant rows of the fact table and delivers much better query performance compared to an RDBMS without join index support.

Weaknesses of Join Indexes

- A join index is a complex structure, and is necessarily more expensive to build and maintain than a single-table index. It may be difficult to update any of the indexed columns in a join index. Although a data warehouse is theoretically static, updates may be imposed by business changes such as the re-organization of a company's sales territory definitions. This may require rebuilding the entire index or even unloading and reloading the entire star schema.
- Join indexes are created on an ordered list of columns. The index structure is dependent on the ordering of the columns. Although the join index is very efficient when all the indexed columns are accessed or when the leading columns of the index are accessed, performance is sub-optimal when other columns are accessed.
- For example, one might build a join index on the dimension tables (TIME, PRODUCT, STORE, PROMOTION). A query that accesses the TIME and PRODUCT dimensions will efficiently use this index, because TIME and PRODUCT are leading keys. However, a query that only accesses the PRODUCT and STORE dimensions would perform sub-optimally, and a query on STORE and PROMOTION (the trailing keys) would perform even worse.
- The join index solution becomes much less attractive for star schemas with many dimensions. A join index spans all the dimension tables. If a given star schema has 20 dimensions, then the join index would span all 20 dimension tables. The difficulty is that a typical query on this 20-dimension star schema would only reference a few of the dimensions. A typical query may only reference five dimensions, yet the join index used to evaluate this query is still a twenty-column index. These star queries use an index that is approximately four times larger than necessary, in addition to any performance penalties associated with not using the leading keys of the index.
- A data warehouse could attempt to address the previous two issues by utilizing multiple, multi-key join indexes over different combinations of dimension tables. However, multiple indexes are often unacceptably costly both in terms of space utilization and machine resources for maintaining multiple indexes.

- A join index is not well-adapted for hybridized star schema. A join index may span a fact table, and all its immediate dimension tables. However, a join index does not directly address the performance issues of a 'snowflake' schema - a more complex variation of a star schema in which each dimension may be a collection of related tables rather than a single table.

Query Processing

When evaluating query processing techniques, a quick comparison against transaction processing applications is useful. Typical OLTP system transactions are predictable and each unit of work is small, accessing or changing just a few rows. The real challenge is in terms of performing hundreds, perhaps thousands, of these well-defined, relatively small units of work concurrently. In contrast, decision support queries usually involve large amounts of data and thus each unit of work is typically much larger. There may not be quite as many concurrent users, although that is starting to change.

Further, typical data warehousing queries involve complex operations such as multi-table joins, sorting, and aggregation. Often these operations are set-oriented, operating on groups of records meeting specified criteria, as opposed to the record-level operations of transaction processing.

Another main difference relates to the organization of the data being accessed. Unlike OLTP systems that employ transaction-centric modeling focused on efficiency, warehouses and data marts typically utilize query-centric dimensional models such as star schemas that facilitate easy visualization and analysis.

Finally, unlike transactional systems, where SQL is carefully hand-written and pre-tuned by developers, decision support queries are often not even written by humans. They are dynamically generated by end-user tools, posing some unique challenges for the query optimizer.

These differences translate into a need for specialized techniques in the database server, in every aspect of query processing-query optimization, access and join methods, and query execution. Here is a summary of the key techniques:

- Data awareness — The query optimizer needs to know the data. Specifically, knowledge of details such as the size of tables, range of column values, and data distribution characteristics is a fundamental requirement for effective query optimization with real-world data.
- Star schema awareness — Given their widespread use in data warehousing, an awareness of star schemas and efficient execution of star queries is a fundamental query processing requirement.

- **Intelligent choice of access and join methods** — The optimizer needs to make intelligent choices related to access and join methods based on the data, the nature of the query, and available resources.
- **Query rewrites** — Queries generated by end-user tools are often poorly written; that is, if they are executed as they are, they might be inefficient. The query optimizer needs to be able to transform or rewrite these queries for better execution.

ACCESS AND JOIN METHODS

- **Rich set of join methods** — The database server has to implement a comprehensive set of join methods including nested-loop, sort-merge, and hash joins to meet the needs of data warehousing. Further, the server needs to incorporate specialized techniques for dealing with star joins.
- **Specialized Indexing Techniques** — While traditional b-tree indexes perform well in typical transaction processing scenarios, warehousing calls for specialized methods such as bitmap indexes to deal with the wide variety of ad-hoc queries.

QUERY EXECUTION

- **Parallel Query Execution** — The database server needs to provide parallel query technology to enable efficient execution of queries against large volumes of data. The primary benefits of parallelism - scale-up and speed-up - are basic requirements in warehousing. The database server should offer a comprehensive set of parallel operators that covers every operation involved in query execution, including table and index scans, sorts, joins, aggregation, and grouping, as well as the creation of summary tables, and inserts, updates and deletes. The parallel query technology needs to support all the parallel hardware architectures: Symmetric Multi-Processors (SMP), Clusters, and Massively Parallel Processing (MPP) systems.
- **Simple optimization example** — An intelligent cost-based optimizer not only has information about the query and available indexes and join methods, but also includes information about the number of rows and blocks in each referenced table, the size of its indexes, and the distribution of data in each individual column.

When presented with a star query, such a cost-based optimizer recognizes that the fact table is very large and expensive to access relative to the cost of accessing the much smaller dimension tables, and it considers this cost when choosing the best execution strategy for the star query. For many star queries, the best execution strategy is to initially join all the smaller dimension tables together using 'Cartesian product' joins, and then join the fact table in the last phase of the query execution. A Cartesian product join is used for joining two unrelated tables. This

Cartesian-product-based strategy is often optimal because the expensive fact-table access occurs only once.

However, dimension tables may be large, so that building a Cartesian product of the dimension tables is an expensive operation. In these cases, the cost-based optimizer recognizes that the Cartesian product approach is not the best solution for this query, and chooses an alternative execution strategy. By accessing the fact table's data only once, the cost-based optimizer's star query strategy can be up to two orders of magnitude faster than the basic approach.

- Query acceleration — Parallel processing is one of the most effective ways to accelerate large, complex queries. The parallel query processing schemes of most database engines vary because of the complexity of the query processing event. Important questions to consider include the following:
 - Is there parallel SQL parsing? How efficient is the parallel distribution of query processing?
 - Are there parallel index scans?
 - How does the engine handle parallel joins, table scans, and parallel I/O?

It is important to benchmark query acceleration and to receive expert guidance in the best manner to utilize the database engine's increased potential.

Some database engines also offer indexing add-ons. The developer should be very careful to analyze specifically what the indexing scheme does and how it benefits the end-user tool's queries versus the cost of the add-on and the time required to build the indexes.

Finally, although not commonly implemented, certain SQL extensions can decrease the amount of processing necessary for complex SQL queries. Queries involving comparisons, percentages, and this year vs. last year analysis generally require the creation of temporary tables.

Elimination of these tables with new SQL commands can be beneficial. A key question with SQL extensions is whether the end-user tool is capable of using the extensions.

Partitioning

Traditional database implementations base parallel execution completely on data partitioning. In this case, data partitioning represents one-dimensional control for achieving two potentially conflicting objectives: manageability and parallel execution performance. What is desirable from a manageability and availability point of view may not yield acceptable performance and vice versa. Even if a partitioning scheme that optimizes both performance and manageability can be designed initially, it doesn't usually stay optimal over time. Significant changes in data distribution or access requirements may render the scheme ineffective and require repartitioning of the data at substantial administrative cost and application downtime. To address this potential tradeoff, some products provide parallel query execution at two levels:

-
- Inter-partition parallelism or parallel execution across partitions (the traditional approach).
 - Intra-partition parallelism or parallel execution within a partition.

Ideally, the added capability to perform query operations in parallel within a single partition enables a "best-of-both-worlds" approach to leverage data partitioning where appropriate, without being constrained. In situations involving data skew, where a small subset of partitions may contain much of the data, parallelism can be applied within these large partitions to avoid any degradation in performance. Administrators can make data partitioning decisions based primarily on manageability and availability needs, without worrying about query performance. Further, the need to repartition data following changes in data distribution or query patterns might be reduced.

Operating System / Hardware Support

This section discusses how a DBMS utilizes OS/hardware features such as parallel functionality, SMP/MPP support, and clustering. These OS/hardware features greatly extend the scalability and improve performance. However, managing an environment with these features is difficult and expensive.

PARALLEL FUNCTIONALITY

The introduction and maturation of parallel processing environments are key enablers of increasing database sizes, as well as providing acceptable response times for storing, retrieving, and administrating data. DBMS vendors are continually bringing products to market that take advantage of multi-processor hardware platforms. These products can perform table scans, backups, loads, and queries in parallel.

PARALLEL FEATURES

An overview of typical parallel functionality is given below. Refer to the upcoming section on SMP/MPP for more information about parallel processing platforms.

- Queries — Parallel queries can enhance scalability for many query operations. This, in turn, can improve response times for very large complex queries. For some products, scans, sorts, joins, (distinct and group) can all be performed in parallel. The work of processing SQL statements may be divided among multiple query server processes and distributed over several nodes.
- Data load — Performance is always a serious issue when loading large databases. Meeting response time requirements is the overriding factor for determining the best load method and should be a key part of a performance benchmark. Look at bulk data loads, insert/copy utilities, data partitioning strategies, how index building during the load is handled, and parallel loading/building. A parallel data

load can make it possible to load huge amounts of data into a database in a very fast and efficient manner. For example, data located on different tapes can be read in parallel and then distributed over a large number of disks.

- **Index creation** — Parallel index creation exploits the benefits of parallel hardware by distributing the workload generated by a large index created for a large number of processors. The time needed to create a large index can sometimes be diminished linearly with an increased number of processors. This makes it possible to generate indexes for single applications and remove them afterwards. By specifying the "unrecoverable" option, the logging overhead for building indexes can be removed.
- **Create table as select** — This feature makes it possible to create aggregated tables in parallel. This is especially useful in big data warehousing environments, where smaller data marts are created out of huge amounts of data coming from the data warehouse. This feature makes it possible to create aggregate tables for individual users in a timely manner. By specifying the "unrecoverable" option, the logging overhead during a create table as select can be removed.

SUPPORT FOR SMP/MPP PLATFORMS

Given the size of a data warehouse, it is likely that parallel processing technology will be appropriate for the volumes and scalability required of most data warehouse environments. Major DBMS vendors are continually releasing products that utilize or benefit from SMP or MPP environments. These products take advantage of multiple processors to perform in parallel tasks such as query optimization, loads, reorganization, and backups.

Which parallel processor configuration, SMP or MPP, is most appropriate for a data warehouse environment? SMP and clustered SMP environments, under the best of circumstances, have the flexibility and ability to scale in small increments. SMP environments are often useful for the large, but static data warehouse, where the data cannot be easily partitioned, due to the unpredictable nature of how the data is joined over multiple tables for complex searches and ad-hoc queries.

MPP environments ideally scale in large increments to a practically infinite degree. MPP works well in environments where growth is potentially unlimited, access patterns to the database are predictable, and the data can be easily partitioned across different MPP nodes with minimal data accesses crossing between them. This often occurs in large OLTP environments, where transactions are generally small and predictable, as opposed to decision support and data warehouse environments, where multiple tables can be joined in unpredictable ways. However, large DSS/data warehouse databases that can be partitioned and/or distributed with minimal access across nodes can be appropriate for MPP. In fact, data warehousing and decision support are the areas most vendors of parallel hardware platforms and DBMSs are targeting.

MPP does not scale well if heavy data warehouse database accesses must cross MPP nodes, causing I/O bottlenecks over the MPP interconnect, or if multiple MPP nodes are continually locked for concurrent record updates. Finding the optimal data partitioning strategy is key to utilizing MPP technologies.

CLUSTERING

A cluster is a group of independent nodes working together as a single system. Clustering offers a fairly mature architecture for high availability and scalability in business-critical computing applications.

In a client/server environment, clients interact with a cluster as though it were a single high-performance, high-reliability server. If one node in a cluster fails, its workload can be distributed automatically among the surviving nodes, thereby reducing downtime. By allowing the grouping of separate servers into a single computing facility, clustering provides a scalability option. Clusters utilizing inexpensive commodity components have the potential to provide excellent price/performance advantages over traditional mainframe systems in many areas. Note that clusters are not only limited to client/server environments, but can also play a role in other computing paradigms such as intranets and network computing environments. Data warehousing architectures rarely use clustered platforms given the high complexity and expense these environments usually require. However, as availability and failover requirements increase for data warehousing environments and hardware costs decrease, it is likely that clustered architectures will be utilized for large data warehouse environments with heavy concurrent use in the near future.

5 Population Architecture

This section of the Data Warehousing Standards and Guidelines discusses the process of collecting and moving data from various sources into the data warehouse. It describes the steps and techniques for this process and attempts to detail many of the issues and problems. It clarifies many of the frequently asked questions about the population process and increases the reader's understanding of population architecture requirements.

After reviewing this section, the reader will have a basic understanding of the following concepts:

- Steps involved with the population process
- Issues surrounding Extract, Transform, Load (ETL) architecture (process distribution, timing and control for the ETL, and ETL process step sequences)
- ETL application issues and concerns (initial population, various extraction techniques, source system replication, merging and mapping, cleansing, summarizing, changes in history, realignments, sorting, loading and staging)
- Transformation tools (code generators and transformation engines)
- Middleware
- Batch architecture (error handling and restart/recovery, data movement, exception handling, scheduling, synchronization, verification and audits)

5.1 Population Process Overview

This section gives an overview of the population process and summarizes all involved steps. It describes the different functions of these steps and lists examples.

According to industry experience, nearly 80% of the effort required in a data warehouse engagement revolves around the population process. This is due to several factors:

- Identification of relevant source data
- Assessment of the data quality and cleansing of data
- Data mapping from source to data warehouse
- Design and creation of the data model and database

- Technical effort required in physically extracting the operational data and deliver it into the data warehouse

Figure 5-1 depicts the conceptual architecture of a data warehouse system and illustrates the specific steps (Extract, Transform, Load) for the population process. It also demonstrates the detail functions, which are involved in these three steps.

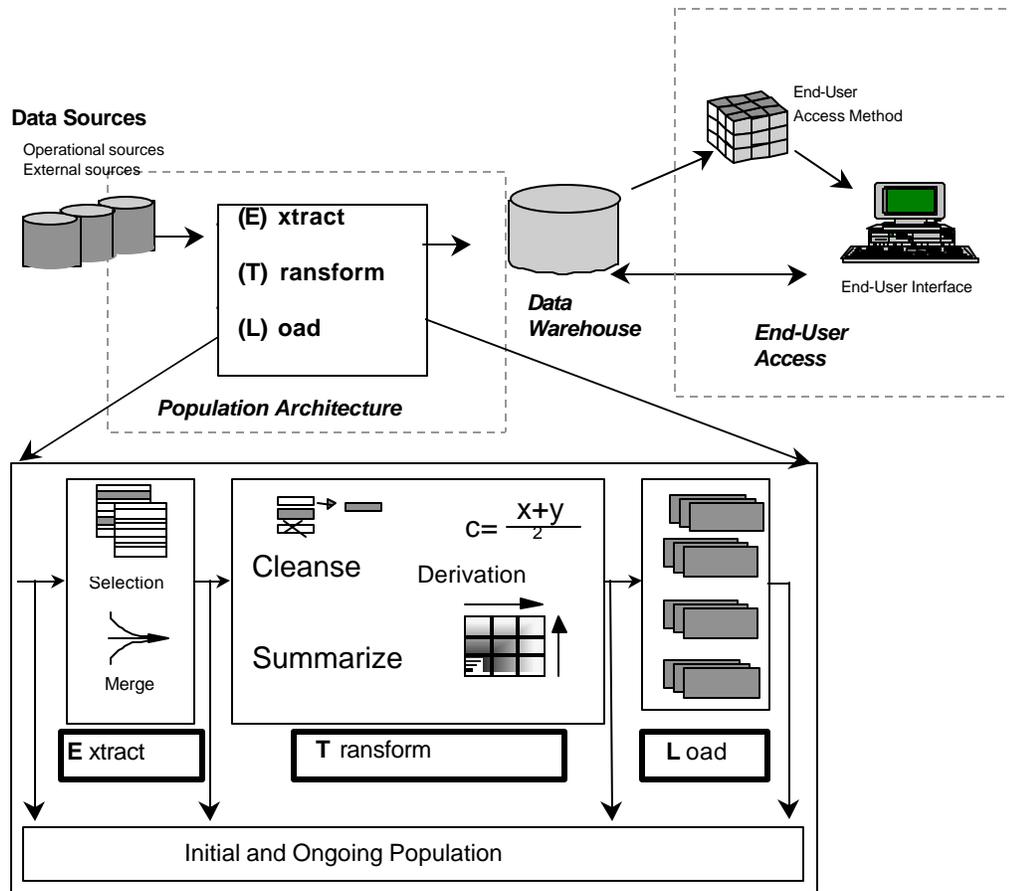


Figure 5-1: Data Warehouse Conceptual Architecture with the Population Architecture

The steps in the population process are described below:

- Determine the data sources — This operation is used to ascertain the origin point for information. The data sources contain the data relevant to the business processes supported by the data warehouse. Sources may include:

- Operational sources — The systems an organization uses to run its business. It may be any combination of the ERP and legacy operational systems
 - Strategic sources — The data may be sourced from existing strategic decision support systems; for example, like executive information systems
 - External sources — Any information source provided by an external entity, such as Nielsen marketing data or Dun & Bradstreet data to the organization
- Extract — This step is used to identify the best sources of data and to obtain the required data from these sources. It consists of the following functions:
 - Selection — This function is used to select relevant data records for the extract. The function criteria is made at two levels: column and row levels. Relevant attributes from the data model of the source system must be identified using the business requirements; rows are selected based on the population criteria.
Examples of selection processing include extracting all the customers from a particular region and selecting only the demographic attributes.
 - Merge — Data from multiple sources is integrated and merged to form a complete picture of a business/product relationship. It is used to create a single version of data to reduce duplicate record processing and decrease data model complexity.
An example of the merge function may include: a bank has customer information in multiple account systems which requires merging before it can be loaded or transformed for a data warehouse.
 - Transform — This step is used to ensure that the produced data is clean and integrated. The Transform step receives data from the Extract step and enhances the data prior to its being loaded to the data warehouse. It consists of the following functions:
 - Cleanse — Edits the extracted data for content, checks it for context and corrects it using business rules. This function is used to alert data warehouse staff and users of quality issues and unacceptable data within the data sources.
-

Examples of cleanse processing include checks for data value within a predefined range and overlaying the unacceptable data value with a default; for example, checking for gender codes of M or F and overlaying all other values with a blank.

- Summarize — Creates new higher-level roll-up records from a set of detailed source records. This function stores summarized data in the data warehouse rather than summarizes the rows on individual requests. Summarization can be performed as part of the transformations or completed after the data has been loaded in the warehouse.

Examples of summarize processing include summaries of sales by time period, such as daily, weekly, or monthly.

- Derivation — Generates fields that are commonly required in the data warehouse, but needs to be calculated from two or more fields at the data sources. This function is used to reduce internal calculations in the data warehouse. Note that this function can occur several times throughout the ETL steps in the data warehouse.

Examples of derivation processing include computing margin on each transaction, which may differ for each transaction due to differing volume or customer discounts.

- Load — This step is used to receive data from the Transform step and load it efficiently into the data warehouse. This operation is used to ensure that the data warehouse is loaded with the most current data. The Load step takes place on the data warehouse platform and typically utilizes platform-specific tools in order to optimize load performance.
- Data transport — This step is used to move data from its source to the target environment. This operation may be utilized between any of the above steps to not only move data between environments but also from one location to another in a single environment.

5.2 Architectural Options

This section describes basic architecture options to transport data from the source systems to the data warehouse. It describes techniques to be considered for the ETL process, timing and control for the ETL process, sequencing of the different ETL process steps, and illustrates that this process involves much more than a simple file transfer and load.

5.2.1 Process Distribution

This section describes how the ETL process is distributed to different environments. It illustrates the environments where the Extract, Transform, and Load steps occur and discusses each of the approaches. Note that these approaches mainly focus on the distribution of the Transform step, which can be performed on the data sources, separate environments, or on the data warehouse platform itself. Extract always takes place at the data sources; the Load step is performed on the data warehouse.

The different approaches are listed below:

Transform in the data warehouse environment — The entire Transform process occurs in the data warehouse. The extraction is accomplished in the data source environment and the load is performed in the central data warehouse environment as shown in Figure 5-2.

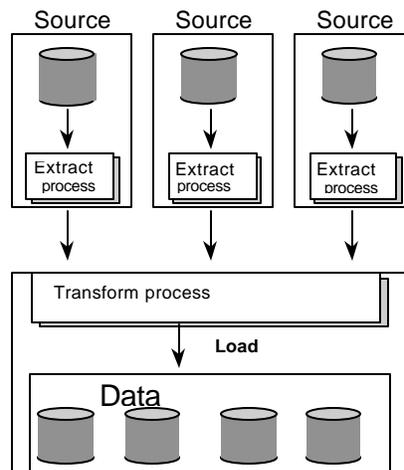


Figure 5-2: Transform Process Data Flow – Data Warehouse Environment

Transform in the data source environment — The Transform process occurs on the data sources. The extraction is accomplished in each source environment and the extracted data is sent to the Transform step within the same environment.

The central environment loads the transformed data into the data warehouse, as shown in Figure 5-3. This approach is used frequently where there are independent data sources that are not tightly coupled and can be loaded into the data warehouse independently.

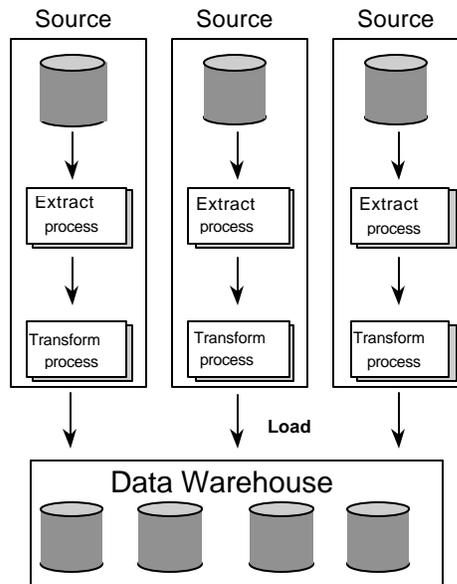


Figure 5-3: Transform Process Data Flow – Data Source Environment

Transform in a separate environment — The Transform process occurs in a separate environment. The extraction is accomplished in each source environment and the extracted data is sent to the Transform step within the transform server. This server transforms the extracted data. After finishing the Transform step, the transformed data is loaded into the data warehouse, as shown in Figure 5-4. This strategy helps avoid the consumption of legacy/data warehouse platform CPU cycles when extensive transformations are required.

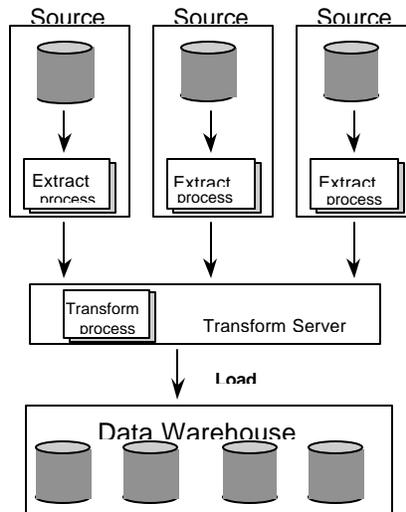


Figure 5-4: Transform Process Data Flow – Separate Environments

Distributed transformation — A combination of these three approaches, the Transform step can occur in multiple environments, depending on network, hardware, and software capabilities and constraints. The extraction is accomplished in each source environment and data is sent to the Transform processes residing in different locations and managed centrally.

Table 5-1 summarizes the decision criteria for each approach.

Decision Criteria	Transform Process on Data Warehouse Central Environment	Transform Process on Data Source Environment	Transform Process on a Separate Environment	Distributed Transformation
Batch Architecture Complexity	+ Moves data only between data sources and data warehouse	+ Moves data only between data sources and data warehouse	- Moves data between data sources, separate environments, and data warehouse	- Moves data among data sources, multiple transform envions, and data warehouse repository
ETL Work Separation	- Allows no workload splitting among systems; only reduces load if several sources are phased out	+ Allows some of the workload for the ETL process to be split	+ Allows workload for the ETL process to be split among different environments	+ Allows workload for the ETL process to be split among different environments
Maintenance Cost	+ Needs to support only one environment for the Transform step	- Needs to support transform processes on many source environments	- Needs to support another environment for the Transform step	- Needs to support additional environments for the Transform step
Resource Usage on Source System	+ Requires no resources for the Transform and Load steps on the source system	- Requires resources for the Extract and Transform steps on the source system	+ Requires no resources for the Transform and Load steps on the source system	- May require additional resources for the Transform step on the source system
Metadata	+ Single location of all transformations help with the management of metadata	+ Original data definition library and metadata are co-located		- Requires special attention paid to manage metadata across transformation platforms
Impact of Source System Change	+ Requires no additional installations and set-ups for the Transform step	- Requires additional installations and set-ups for the Transform step	+ Requires no additional installs and set-ups for the Transform step, unless it is handled as a remote process with tools from the central data warehouse	- Requires additional installations and set-ups for the Transform step
Resource Usage on Data Warehouse	- Requires resources for the Transform step	+ Requires no resources for the Transform step	+ Requires no resources for the Transform step	- May require more resources for the Transform step, even for centralized management
Transform Implementation Complexity	+ Has one central Transform process	- Has multiple Transform processes	+ Has one central Transform process	- Has multiple Transform processes

Table 5-1: Decision Criteria for Transform Approaches

5.2.2 Update Techniques

This section describes the data warehouse update techniques. It describes the different update approaches, decision criteria, and future trends.

There are several timing approaches to assist in transferring data between two environments, because some data in the data warehouse needs to be updated immediately after a change in the source systems, whereas other approaches update changes on a daily or weekly basis.

The important update approaches are described below:

- **Real-time** — Provides real-time updates to the data warehouse. An example of real-time data capture includes sending updates to the data warehouse as soon as the data is processed in the OLTP environment. Common real-time approaches include the use of database gateways. This update approach is often used when the data warehouse provides direct feedback to operational systems.
- **Batch-scheduled** — Provides data delivery at a predefined time period. Common batch-scheduled approaches use gateways or data loader routines. An example of batch-scheduled data capture includes sending sales balance data updates to the data warehouse at the end of the day, week, or month.
- **Batch-event** — Provides data delivery after a specific event. Common batch-event approaches use gateways. An example of batch-event data capture includes sending sales data immediately after a sales transaction has been registered.

Table 5-2 illustrates the decision criteria for the different approaches.

Decision Criteria	Real-Time	Batch-Scheduled	Batch-Event
Performance	- Requires additional update I/O to be performed at the peak use times of source and target environments	+ Able to tune and level-load	+ Requires additional I/O for updates; possibly during peak usage
Ability to Provide Up-To-Date Data	+Provides real-time data	- Timeliness of data dependent upon batch schedules	- Triggering event may not occur for a long time
Technical Complexity	- Requires mechanisms for real-time support (real-time OS)	+ Requires only a few additional mechanisms	- Requires special mechanisms (triggers etc.) to update all changes

Table 5-2: Decision Criteria for Several Update Approaches

Traditionally, data warehouse architectures have used a batch-scheduled approach. However, recent trends have encouraged real-time and batch-event timing. These approaches should only

be used to satisfy clearly defined and justified needs, as the necessary technology is complex and promotes a data warehouse that resembles the OLTP environment.

5.2.3 Control

This section describes approaches for controlling the ETL process. As the number of processes increases, the control and management of those processes becomes critical to the success of the data warehouse. This section explains a number of approaches and lists decision criteria.

There are three control approaches for controlling the ETL process:

- **Central control (master/slave control)** — This approach uses a central system to schedule and initiate batch jobs. This system requests that the extract be performed on the source system. It then asks the transform machine to get the extracted data and transform it. Finally, the data warehouse machine is asked to load the data.

Note that one ETL process step is always waiting for another. Central control from one of the environments is more generally known as master/slave control, where the master controls the job on one environment and the slave performs the master requests on another environment, and sends the result back to the master.

Table 5-3 illustrates master and slave environment possibilities for the master and slave. Possibilities include the data warehouse environment, transform, and data sources environment. Note that one ETL step is not always waiting for another.

Master	Slave
Data warehouse	Transform server; Data sources
Data warehouse	Data sources
Transform server	Data warehouse
Transform server	Data sources
Data sources	Data warehouse
Data sources	Transform server

Table 5-3: Master/Slave Environment Possibilities

- **Shared control** — This approach uses a shared mechanism to schedule and initiate batch jobs in multiple environments. It requests that the extract be performed in one environment. It then asks the transform machine to get the extracted data and begin transform processing. Finally, the data warehouse machine is asked to load the data in another environment.
- Note that this approach uses multiple schedulers, which can communicate with each other and that one ETL process is not always waiting for another.

Table 5-4 lists decision criteria for the different control mechanisms:

Decision Criteria	Central Control	Shared Control
Communication Infrastructure	+ Requires no complex communication mechanism	- Requires a complex communication mechanism
Implementation Complexity	+ Requires only one central control mechanism	- Requires a complex communication between the data sources, transform environment and data warehouse environment
Scalability	- Low	+ High; can be implemented throughout the enterprise

Table 5-4: Decision Criteria for Control Approaches

5.2.4 Sequencing of ETL Steps

Because ETL functions need not always occur sequentially, there is no single, correct, order. They are usually dependent upon the data warehouse architecture and sequencing requirements of the ETL, the environment, and other factors. Examples include the Transform step, which can occur logically or physically as part of the Extract or Load step, and the summarize function of the Transform step, which can occur after the Load step. It is also possible that steps must be performed multiple times, as exemplified by the merge function, which combines the records from the data sources but can also merge them for the summarize function.

However, most processing must occur sequentially, such as selection, merge, cleanse, summarize, and Load; or selection, merge, cleanse, derivation, and Load.

5.3 ETL Process

This section describes the ETL process and outlines techniques and issues surrounding the Extract, Transform, and Load steps. After review, the reader will be familiar with the following topics:

- Initial and ongoing population

- Extraction techniques and source system replication
- Transformation
- Cleansing
- Summarization
- Changes in history
- Realignments
- Sorting
- Loading
- Staging

5.3.1 Initial and Ongoing Population

Once the data warehouse architecture is designed, attention shifts to the population of the data warehouse. There are two principal strategies for accomplishing this:

- Full population — Extracting the entire set of data stored in the data sources and transferring it to the Transformation step, thus starting the data warehouse with the current set of OLTP data
- Incremental population — Incrementally updating the data warehouse while relevant transactions occur on the operational system

Data warehouses are often updated differently for the initial load and for ongoing regularly scheduled loads:

- Initial — There is often a full population from the source systems so that the data warehouse is current with operational data. If there is no data in these source systems or no historical data of relevance, transactional information is posted to the warehouse incrementally.
- Ongoing — :Due to the frequent high volumes of source-system information in data warehouse fact tables, it may not make sense to do a full population every time. The population architecture needs to accommodate posting changes to the operational system since the last load.

High volume transaction data is generally loaded into data warehouse fact tables via incremental population. This is advantageous for performance reasons (to prevent large bulks of records being moved at once) and if an audit trail of changes must be kept for historical purposes.

For low-volume data (such as master and reference data), incremental updates are not always necessary unless a history of changes must be tracked. Tables are often overlaid completely with data from the source system. Thus, ongoing population processes can be a mixture of full and incremental extracts and loads. These techniques are discussed in the following section. Despite the initial population method, going forward, incremental population is used for the data warehouse, ensuring that it will be periodically updated with new information from its sources.

5.3.2 Extraction Techniques

After initial population strategies are reviewed, several common techniques to extract data from the data sources can be considered. This section describes and lists decision criteria. It also discusses the replication of data sources.

To keep a data warehouse environment up-to-date, two extraction techniques exist:

- Full extract
- Delta extract

Full Extract

Data is entirely extracted from its sources; then, the Transform or Load step determines which data is needed. This is required for full population of the data warehouse.

There are two implementations of the full extract:

- Before/after images — A copy of the data at the last extraction (before image) is compared with a current copy of the data (after image) to determine net changes since the last extraction. Only the changed data is transformed and loaded into the data warehouse.
- Full data load — The full data from the data sources are extracted, transformed, and loaded, regardless of which records have changed.

Table 5-5 compares these implementations according to various decision criteria.

Decision Criteria	Before/After Images	Full Data Load
Processing Time	- Involves comparing before and after records	+ Requires no comparisons of records
Data Warehouse Transfer Time	+ Traffic can be reduced if compare occurs before transfer	- Full transfer of data over a network can be costly for high volumes
Complexity	- Requires mechanisms to implement efficient comparisons	+ Requires no comparisons
Load Performance	+ Only changed data is loaded	- Performance suffers from deleting /updating redundant records

Table 5-5: Decision Criteria for Different Implementation Approaches

Delta Extract

Only data which has been changed, inserted or deleted since the last extract is taken from the data sources. This typically isolates the needed data for the Transform and Load steps. The delta extract techniques can be classified as system level, table level and row or record level.

SYSTEM LEVEL

Archive logs — These standard database functions (typically used for backup purposes), which store before and after images of added, changed, and deleted rows, are used to identify net changes for the Extract step.

Figure 5-5 illustrates this method. Note the INSERT statement is coded as 0, the UPDATE statement is coded as 1, etc. All transactions are encoded.

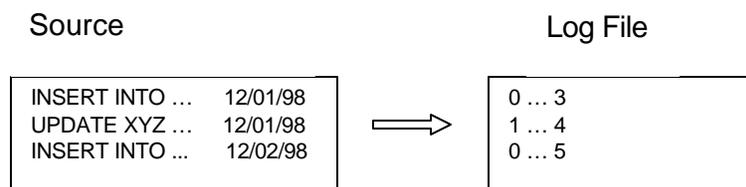


Figure 5-5: System Level Log File Tracking

TABLE LEVEL

Log tables — A separate log table, typically populated via database triggers associated with the table level updates, is utilized to store changed records. Figure 5-6 illustrates this method. When a change event occurs, the change is captured and updated in the log table. At a specific event, the update is executed against the designated table based on key value. In Figure 5-6, Table 2 is not affected since the log table does not contain a trigger for this change event.

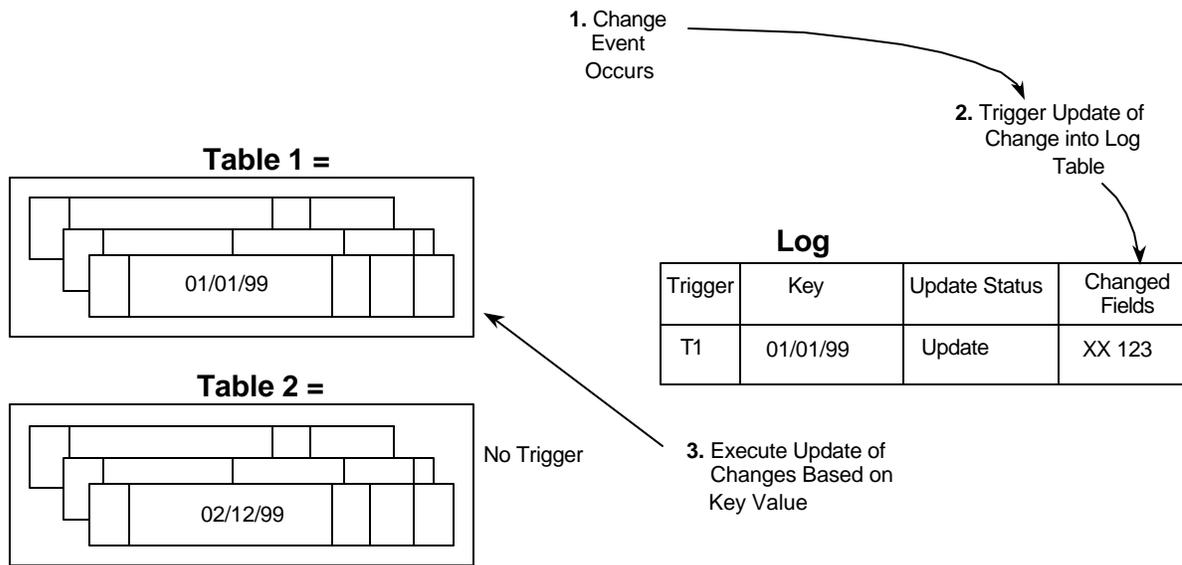


Figure 5-6: Log Tables Example

ROW LEVEL

Indicator flag tracking — A flag, separate for every record, is updated (typically through a trigger) whenever any field in that record is changed, updated or deleted. This allows extraction processing to easily identify any changed record.

Figure 5-7 illustrates this method, where 1 is used to indicate an update and 0 for no update.

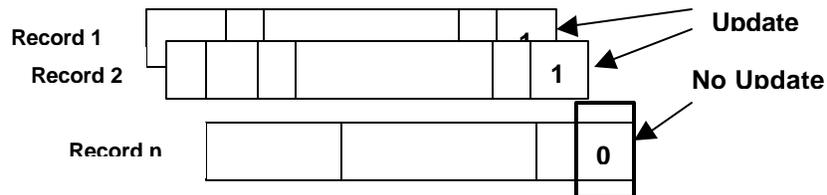


Figure 5-7: Indicator Flag Tracking Example

Timestamp tracking — Similar to flag tracking, timestamp uses a date/time value instead of a flag. Extraction processing must remember the last extraction date/time and compare each timestamp against this value.

Figure 5-8 illustrates this method, where the last extraction occurred on 11/10/98 at 5:00 and the current date/time is 11/12/98 at 5:00.

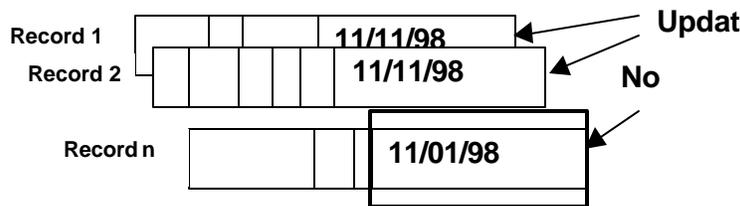


Figure 5-8: Timestamp Tracking

The indicator flag technique requires less storage and memory as compared to the timestamp technique. However, it is more difficult to maintain as each module requires encoded indicator flags.

DELTA EXTRACT IMPLEMENTATION COMPARISON

Table 5-6 compares these implementations according to various decision criteria.

	Row Level	Row Level	Table Level	System Level
Decision Criteria	Indicator Flag Tracking	Timestamp Tracking	Log Tables	Log Files Tracking
Complexity of Solution	+ Simple solution	+ Simple solution	- Update table must be analyzed	- A separate tool used to interpret log files; arcane formats with bad data, spanned records, & coded values
Efficiency	- Very inefficient	- Very inefficient	- Very inefficient	+ Good; log file formats are optimized for performance
Ability to Determine Exact Record Change Time	- Cannot be determined	+ Can be determined; stored in a separate field	- Cannot be determined	+ Can be determined but is encrypted
Performance	- Must handle the entire data table	- Must handle the entire data table and compare timestamps for each record	+ Changed records are already separated into a dedicated table	+ Changed records are separated into dedicated file, but needs separate tool to examine records
Redundancy	+ Needs only one additional field for the flag	+ Needs only one additional field for the flag	- Needs additional table structures	- Contains transactions not finished during invalid inputs

Table 5-6: Implementation Comparison

Full Vs Delta Extraction Technique Comparison

Table 5-7 compares full and delta extracts for a set of key decision criteria.

Decision Criteria	Full Extract	Delta Extract
Implementation Flexibility	+ Not dependent on how the data is stored and maintained	- Strongly dependent on how the data is stored and maintained
Performance Impact on Source System During Extraction	- Can tie up the source system for extended periods of time during extract	+ Extraction of only changed data generally can be accomplished quickly
Redundancy	- Information already in warehouse is transmitted; wasteful for high volume of data	+ Transmits only what has changed
Impact on Network Traffic	- Requires more bandwidth as larger data volumes are transported	+ Saves network traffic as only the changed or new records need to be transported
Performance Impact on Data Warehouse During Load	- Records already in the warehouse are reprocessed and reinserted	+ Only the new records must be loaded
Suitability for the Type of Data	- Applicable for small or reference tables	+ Applicable for even large tables

Table 5-7: Full and Delta Extract Comparison

To summarize, it is clear that each of these techniques have tradeoffs – there is no “one size fits all” solution. Decisions made depend on the source systems, data volumes, and skills available to implement the extraction. Project to project, every architecture will be different. Where there is a mix of different source systems, it may be impossible to limit the choice to a single extraction technique.

Source System Replication

To reduce impact on OLTP environments, legacy systems have historically been fully copied to non-production areas for reporting. This can also be applied to extractions to a data warehouse. Though this may seem simple, there are often many complicating issues including availability, volume, complexity, and cost. Mechanisms to pull data out of source systems, such as logging

and mirroring (RAID1, RAID5), must be examined to see if there is a true benefit to replicating the source. There may be opportunities to reduce source impact by pulling source data from the reporting systems.

5.3.3 Transform

This section describes the functions in this step and relate their importance. These functions are as follows:

- Merging and mapping
- Cleansing
- Summarizing
- Changes in history
- Realignments
- Sorting

Once information has been brought into a data warehouse environment, transformation becomes important. Encompassing a number of different techniques and steps, it refers broadly to the process of making warehouse information suitable for end use. The business rules used as part of the transformation process must be captured and maintained within the metadata in order to assure a proper and consistent understanding of the data.

Merging and Mapping

The two actions of merging and mapping ensures that data from different source systems are sent to the proper location in the data warehouse. They are briefly summarized below:

MERGING

Merging is the action of placing information from different sources together. This can include bringing together multiple tables to create a data warehouse, or feeding a single table within the warehouse with information from multiple source tables. Depending upon the number, locations, and formats of the information sources, merges can be quite complex.

MAPPING

Mapping is moving data according to a one-to-one relationship between source and target. Each piece of information brought into a data warehouse must be brought to a specific location.

Mapping is the process of defining where it should go and includes matching, elementization, and standardization.

Matching

Matching is the process of checking for similar data elements. As an illustration, consider two fictitious addresses:

Albert Einstein, 449 Page Mill Road, Palo Alto, California 94306

Maria Einstein, 449 Page Mill Road, Palo Alto, California 94306

Matching these addresses would recognize that these persons share the same household and prevents redundant storage, indexing, and searching.

Elementization

Elementization is parsing or separating data into a list of standard elements. As an illustration, consider a fictitious address:

Albert Einstein, 449, Page Mill Road, 94306 Palo Alto, California

Elementizing this address would produce the following form:

Addressee First Name: Albert
Addressee Last Name: Einstein
Street Address Number: 449
Street Name: Page Mill Road
Zip Code: 94306
City: Palo Alto
State: California

Standardization

Standardization is putting data in a single, consistent format. This action may also involve translation tables. As an illustration, consider a data element:

Page Mill Rd.

Standardizing this data element would produce:

Page Mill Road

5.3.4 Cleansing

Cleansing is the process of making sure that warehouse data is accurate and consistent. This subsection:

- Provides an overview of issues related to cleansing
- Describes various cleansing operations

There are three types of cleansing actions: syntactic, structural, and semantic which are described below.

SYNTACTIC

Syntactic cleanse issues are concerned with inconsistencies in the entry of data that may exist within a single source or between multiple data sources. They include:

- Spelling errors — Records may be mistyped or misspelled. An example of spelling errors include a customer name list with the record entry “Ted McKendall” in one data source while another data source has a customer “Ted McKendal.” If the addresses and phone numbers are identical, these probably refer to the same customer, but one is in error.
- Abbreviation inconsistencies — Records may use different abbreviations to describe the same item. For example, one business list contains the entry “Forty-Seventh St. Photo” while another, referring to the same business, has an entry “47th St. Photo.” Another example is “IBM” vs. “International Business Machines” or “International Business Machines, Corp.”

STRUCTURAL

Structural cleansing issues are concerned with the inconsistent representation of data within a single data source or among multiple data sources or between data sources and the data warehouse. They include:

- Encoding inconsistencies — Records meaning the same item may be encoded differently. For example, one customer name list contains the entries “m” and “f” to represent male and female, while another may use “1” and “0,” and a third may use “x” and “y” to represent gender.

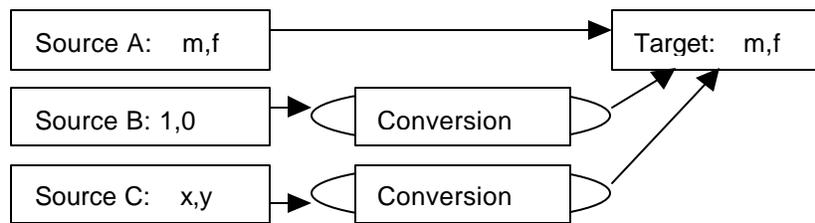


Figure 5-9: Encoding Inconsistencies

- **Character set inconsistencies** — Records may be represented by different character sets. For example, one set of data is represented in EBCDIC format while another is represented in ASCII format.
- **Unit set inconsistencies** — Numerical information may be recorded in different formats using different unit sets. For example, one set of revenue figures may use the Dollar as a currency basis, while others may use the Japanese Yen. Another example is sales represented in millions may use a value of “9” to indicate “9,000,000;” another sales figure represented in hundreds may use “9” to indicate “900.”
- **Data type inconsistencies** — Different data types may be used for different records. For example, one set of data uses a four-byte a floating point data type for a field while another uses an eight-byte floating point data type.

SEMANTIC

Semantic cleansing issues are concerned with different user interpretations of data. They may exist within a single data source, between multiple data sources, or between data sources and the data warehouse. They include:

- **Definition inconsistencies** — Records may have a different meaning or definition although they may share the same name. For example, one source system includes customer returns in its sales figures while another does not.
- **Integrity inconsistencies** — Records may use the same key for inconsistent information. For example, a customer list with the same customer ID is keyed to different addresses and phone numbers on different sources.
- **Text field inconsistencies** — Records may use different representations or data values in text fields. For example, one data field stores a customer’s full name while another stores only the customer’s last name.

CLEANSING OPERATIONS

Three cleansing operations are available to handle the syntactic, structural, and semantic issues listed above:

- Verification
- Conversion
- Calculation

They are introduced below; Table 5-8 illustrates which operation is appropriate for solving each cleansing issue.

Verification

Verification is the process of checking the consistency of standardized elements for errors and then correcting them. As an illustration consider the fictitious address:

Albert Einstein, 449, Page Mill Road, 94306 Palo Alto, Texas

Verifying this address would change the state from Texas to California based on zip code and city.

Conversion

Conversion produces standardized values and formats for the warehouse by performing operations - anything from arithmetic to logical translation - on incoming data.

As an illustration, consider international sales figures logged in several different currencies. To prevent confusion, conversion makes sure that all sales numbers are in terms of dollars, or another single monetary unit.

Calculation

Calculation alters the values of incoming numerical information to make it conform with standard definitions.

As an illustration, consider the example of definitional inconsistency used above, where sales defined in one system includes customer returns and another system does not. If it is decided that sales should exclude customer return information, then sales figures from the first system will be modified to reflect this change.

Once the cleansing functions are complete, the users should have higher confidence in the data, because these operations have reduced redundancy and improved the consistency of source data, as shown in Table 5-8.

Cleansing Issue	Cleansing Operation
Spelling Errors	Verification
Abbreviation Inconsistencies	Conversion
Encoding Inconsistencies	Conversion
Character Set Inconsistencies	Conversion
Unit Set Inconsistencies	Conversion
Data Type Inconsistencies	Conversion
Definition Inconsistencies	Calculation
Integrity Inconsistencies	Verification
Text Field Inconsistencies	Conversion

Table 5-8: Cleansing Issue and Operation Matrix

5.3.5 Summarization

Summarization is the process of concatenating information. It is used to match records at the same level of granularity in multiple source feeds to the data warehouse. In addition, summarization aids in avoiding redundant, time consuming, and resource-intensive calculations of commonly used or distributed information.

SUMMARIZATION OPERATIONS

Summarization consists of two operations, aggregation and balancing, as described below.

Aggregation

Aggregation is the process of rolling-up detailed data into entities that may be more useful to an organization. Figure 5-10 shows the distillation of transactional records into daily and weekly summaries.

Detailed transaction data..... summarized once... .. and summarized again!

SCANNER DATA			STORE ITEM DAILY SALES				STORE ITEM WEEKLY SALES			
CHECKOUT TRANSACTION NO.	ITEM NO.	QUANTITY SOLD	STORE NO.	ITEM NO.	DATE	QUANTITY SOLD	STORE NO.	ITEM NO.	WEEK ENDING	QUANTITY SOLD
P K	F K	NN	F K	P K	NN, DD	F K	P K	NN, DD	F K	NN, DD
1234001	1563	12
1234001	807	1	1	2	Jun 01	110	1	2	Jun 07	1363
1234001	2	1	1	2	Jun 02	126
1234001	149	4	1	2	Jun 03	123	2	2	Jun 07	456
...	1	2	Jun 04	144
...	1	2	Jun 05	102
...	1	2	Jun 06	344
1234005	402	3	1	2	Jun 07	410
1234005	2	2
...	2	2	Jun 01	50
...	2	2	Jun 02	47
1234027	2	3	2	2	Jun 03	92
1234027	807	3	2	2	Jun 04	20
...	2	2	Jun 05	37
...	2	2	Jun 06	144
1234046	177	6	2	2	Jun 07	126
1234046	807	1
1234046	2	3
...
1234039	2	1
1234039	177	1

Figure 5-10: Aggregation Example

Balancing

Balancing is the twofold process of computing summary data from detailed data and comparing it with an existing report. It is useful for verification purposes.

Figure 5-11 illustrates balancing. Daily transactional data on the left is summarized into weekly reports, which are then compared in the weekly summary shown on the right.

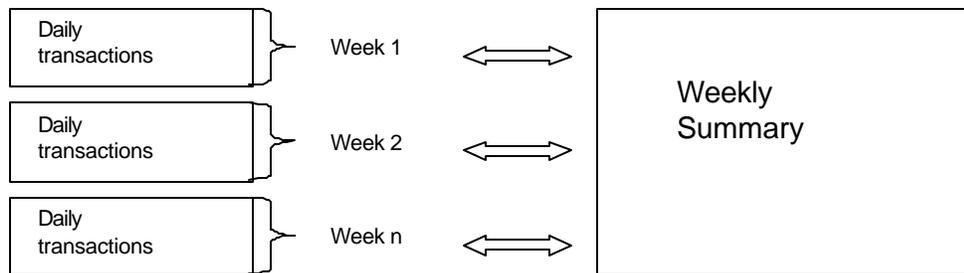


Figure 5-11: Balancing Example

5.3.6 Changes in History

Changes in history are alterations of warehouse data made in response to updates or the identification of mistakes.

As an illustration, consider the following example: a retail company’s daily and aggregated monthly sales figures for the first quarter of 1998 were extracted and loaded into the data warehouse. Due to mistyped information, the figures for 10 March 1998 were found to be in error. This necessitates not only an update for sales totals computed for that day, but also monthly totals computed for March.

Two methods of adjustment are possible:

- **Manual update** — The user manually makes all changes, first updating the lowest level of data and then refreshing the roll-ups.
- **Automatic update** — Through the monitoring of data or the reading of log files, changes are noted and all necessary updates are applied automatically.

Table 5-9 compares these methods.

Decision criteria	Manual Update	Automatic Update
Complexity	+ Very straightforward; requires only user action	- Setting up mechanisms to monitor data, read log files, and automatically make updates can be difficult
Change Efficiency	- Requires large expenditures of user time and effort	+ Changes can be made extremely quickly and transparently to users

Table 5-9: Automatic vs. Manual Update Comparison

5.3.7 *Realignments*

Necessary for the maintenance of data warehouse integrity, realignments are changes to a data model stemming from business changes. They present special problems for all processes relying on information hierarchies, such as drilling.

There are two types of realignments:

- **Transfers** — An entity is assigned to a new parent in a hierarchy.

For example, a retail store chain has is divided into two geographical divisions, regions A and B containing stores 1,2, and 3, and stores 4 and 5, respectively. A year after the data warehouse implementation, the divisions are realigned and store 3 is moved to region B.

- **Cancellations — An entity is removed from a hierarchy.**

For example, a salesperson leaves an organization and is no longer in the current hierarchy. Thus, applying the current hierarchy to historical data excludes that person's sales information.

There are two ways to handle realignments:

- **No historical data modification — Detailed data is not resummarized.**

In one year, each store in the retail example above had sales of \$10,000,000, giving region B a total annual sales of $\$10,000,000 + \$10,000,000 = \$20,000,000$. The next year, after realignment, each store had sales of \$12,000,000, giving region B a total annual sales of $\$12,000,000 + \$12,000,000 + \$12,000,000 = \$36,000,000$.

- **Historical data modification — Detailed data is resummarized.**

In the same example, using resummarization, region B would show total sales of $\$10,000,000 + \$10,000,000 + \$10,000,000 = \$30,000,000$ in the first year and $\$12,000,000 + \$12,000,000 + \$12,000,000 = \$36,000,000$ in the second.

Table 5-10 compares these methods.

Decision Criteria	No Historical Data Modification	Historical Data Modification
Focus	The past	The present and future
Historical Comparisons	Makes direct comparisons difficult for entities which change over time	Allows direct comparisons for entities which change over time
ETL Complexity	Simple; no resummarization	Complex; information resummarized
Retrieval of Historical Data	Intact historical data is available	Actual historical information is unavailable

Table 5-10: Historical Data Modification Comparison

5.3.8 Sorting

Sorting is the process of ordering rows of data by a specific attribute or property. Sorting may be accomplished using data warehouse management software or specialized high-performance sorting tools.

Sorting implementations usually have the following characteristics:

- Multiple data input support — Support for a variety of data and file types from heterogeneous data sources
- High performance — Proprietary algorithms to optimize the sorting process
- High scalability — The capability to use system memory resources such as memory and file systems to handle the gigabytes of data generated by large transaction processing applications
- Data editing and conversion capabilities — The capability to generate multiple, uniquely-formatted output files from a single pass of the input file
- Single job step for large data volume — The capability to dynamically segment, store, and merge input data if large sorts must be performed with a limited amount of disk space

5.3.9 Load

Loading is the process of taking transformed data and placing it in the data warehouse. There are three methods for accomplishing the load, though the appropriate one is determined largely by the output from the Transform step:

- Data warehouse management software — Transformed data is loaded directly into the warehouse without using flat files
- DBMS — Flat files are loaded into the data warehouse using the DBMS's load utility
- Third-party utility — Data is loaded into the warehouse using a specially-optimized loader

Table 5-11 compares these methods:

Criteria	Data Warehouse Management Software	DBMS	Third-party Tool
Load Performance	+ Saves time that would be otherwise spent writing the data to a flat file	+ Very fast; the DBMS's load utility is optimized for this purpose - Time must be spent writing data to a compatible file format	± Depends on the tool and whether it is optimized for the source and database formats
Resources	+ Requires less user effort and one fewer tool	- Loading may take place quickly, but source data must be written to a compatible file format	+ Depending on the tool, can be the quickest and easiest option
Accessibility	+Very fast; no intermediate step is necessary	+ Accessing data can be more complex	+ Can access data faster

Table 5-11: Data Warehouse Load Method Comparison

In order to facilitate the Transform step, records extracted from data sources are usually sorted early in the ETL process. If this is not the case, data should be sorted just before it is loaded into the warehouse because load performance will increase if data is ordered the way the RDBMS expects (such as by date first, then product, then market).

There are two options for performing load operations:

- Bulk load — Loads the entire set of records in the data warehouse; A DBMS utility is always used to perform this kind of load efficiently
- SQL or record-by-record load — Keeps track of changes to the information source and loads them into the data warehouse

The bulk load is significantly faster than the SQL load if the number of records to be inserted is large. The exact break-even point depends on the DBMS vendor and the table structure.

Consider the following issues when selecting an appropriate loading process:

- Data sorting
- Data volumes
- Physical database design

- Indexes
- Partitioned tables
- Operational issues in loading

Due to its size, complexity, and access requirements, loading a data warehouse can be a challenge. To help optimize this process, data loads should be an integral part of benchmarks and performance tests, since the logical/physical database design may require adjustments.

Consider the following operational points:

- The timeframe available for data loading must be determined. If an entire database must be loaded overnight (for example, in case of emergency recovery), or any other specific timeframe, a high degree of design and testing for the physical database will be necessary to meet this requirement.
- The frequency of data loading operations must be determined. This is dependent on DBMS availability requirements and the rate of growth or change to the database tables. For example, a decision support database with stable data may only need to be loaded monthly. However, tables that grow or change heavily may need daily loading and restructuring.
- The scope of data loading operations must be determined. A reload of the entire database may not be necessary when only critical and update-intensive tables need to be loaded or restructured on a regular basis.

Load performance is highly dependent on the availability of system resources. Low available system memory is one of the most common causes of performance degradation and loading failure; the greater the amount of available continuous space, the better the performance.

5.3.10 Staging

Staging refers to a temporary holding area for data being transferred from the data sources to the data warehouse. Staging often uses flat, unindexed files to facilitate the efficient transfer and load into the data warehouse. No user access occurs within this area; files are generally deleted or overwritten when the next batch of data is staged. Staging resource planning is more crucial when processing large amounts of data.

When data has been flagged in the source system as being appropriate for warehousing, it is extracted into staging until the Transform step is initiated. Once the Transform has completed, the staging resource is called in again to hold the Transform data until the Load step has been

started. The staging resource is freed up for another cycle once the data is no longer required by the ETL process.

5.4 ETL Tools

Traditionally, Extract and Transform solutions have been custom-coded for each project. These custom solutions can be inefficient to develop and result in a standalone extraction program for each data source, making them difficult to manage. In an effort to reduce the development time and improve manageability, software vendors developed packaged Extract/Transform/Load (ETL) applications to help control this major data warehouse expenditure.

This section provides an overview of transformation tools and explains how they transform the data from various sources to the data warehouse. After reviewing this section, the reader will have a basic understanding of transformation tools and their selection criteria.

5.4.1 Requirements for ETL Tools

Data transformation is a mainstay of many common IT projects, from application downsizing and package replacements to Year 2000 transformations to data warehousing. While data transformation is not inherently difficult, it can be expensive in a large enterprise due to several factors:

- Large numbers of data sources can result in numerous transformation modules
- Frequent addition of new data sources and modifications to existing sources can result in high maintenance costs
- Stringent performance, scalability, and robustness requirements are difficult to meet in an on-going fashion where complex workflows combine with complex configuration control requirements to challenge the management of the ETL process
- The first generation of packaged Extract, Transform, Load (ETL) applications that were developed to alleviate the above issues included code generating tools such as ETI Extract and Apertus PASSPORT. These tools offered several improvements over custom solutions, but still resulted in a large number of standalone programs that need to be managed in the runtime environment.

An improved, second generation of packaged ETL tools were developed in the mid-1990s: Transformation engines. Transformation engines offer a centralized approach to the ETL process. The “engine” is actually a server module that performs most of the processing, which is coupled with a relational database management system (RDBMS) that shares the transformation workload with the server module, acts as a staging area during the data

transformation process, and is used as a metadata repository. A graphical user interface (GUI) facilitates the transformation development and administration tasks.

5.4.2 Code Generators

Code generators are part of the first generation of ETL tools. The main improvement in these tools, as compared to the custom coding, is metadata management. In an enterprise with a large number of data sources, the metadata may be spread across a large number of custom extract programs. This unstructured, distributed metadata is difficult to manage and not easily accessible to new programs. Furthermore, if metadata of one source changes, the change must be propagated to several extraction programs. The code generators support metadata management by tracking changes to the source systems and target systems, and their respective transformation business rules.

5.4.3 Transformation Engines

Transformation engines are the second generation of ETL tools. They extract data from heterogeneous data sources, transform and cleanse the data using defined business rules, and load the data into target databases or files. An advantage that transformation engines offer is centralized control and processing. This reduces maintenance expenditures when changes are made to sources, targets, or transformations. The development environments of these tools are, in general, much more user-friendly than the environments of code generators.

Areas that have been identified as potential weaknesses of these tools are support for legacy sources such as ADABAS, IMS, VSAM, etc., and data throughput. However, since these tools are fairly new to the marketplace, improvements are being made at a rapid pace.

The specific components of a transformation engine depend on the vendor, yet the following components are standard across most transformation engine products:

- Metadata capture and management — Source, target, and transformation metadata are automatically captured during the transformation development process and are stored in a centralized repository. Business metadata can be written and stored with the technical metadata.
- Data extraction — The extract component supports extraction from all popular RDBMS products and many flat file formats, with increasing support for legacy sources such as ADABAS, IMS, and VSAM.
- Data transformation — Standard objects are provided for most common transformations, such as value-based filters, code table lookups, calculations, normalization, summarization, etc. Most products support custom transformation

development in a language such as C, C++, or Visual Basic. However, in many cases, the standard transformation modules are adequate.

- Data transfer — Transformation engines support multiple data transfer communication protocols such as TCP/IP, SPX
- Data loading — Bulk or row-by-row data loading is supported for most RDBMS products and many flat file formats.
- Transformation scheduling and management — Transformation engines provide simple built-in scheduling tools and basic exception processing.
- Versioning/development environment — Basic versioning and migration utilities are included with the products to provide a functional development platform.
- Data mart build — Transformation engines can generate the data definition language (DDL) to create target tables in many RDBMSs.

5.4.4 ETL Tools Architecture Tradeoffs

The selection of a transformation architecture is a major decision in the development of a data warehouse. Table 5-12 can assist the data warehouse architect in selecting a proper ETL technique:

Decision Criteria	Custom Code	Code Generators	Transformation Engines
Legacy Support	- Any source can be accessed provided it is documented	- Support a large number of legacy data sources e.g. VSAM, IDMS, ADABAS, or IMS	- Excellent RDBMS and flat file support; legacy support is not as advanced as code generators
Performance	- Inconsistent; depends on the developer's skills	- Consistent but not always optimal	- Bottleneck can occur, but many configuration options exist for optimization
Metadata Capture and Management Support	- Manually developed and maintained	- Manually maintained after initial program generation	- Source and target metadata are captured and verified at runtime
Licensing Cost	+ Inexpensive	- Moderate to expensive	- Moderate to expensive
Maintenance	- Difficult	- Moderate	+ Least difficult
Integration Capability	- Difficult	- Moderate	+ Least difficult
Scalability	- Limited by management effort	- Limited by management effort	- Facilitated by centralized management but limited by "bottlenecking"

Robustness in Functionality	- Inconsistent; depends on the developer's skills	+ Good; advanced code can be generated	+ Good; most transformations can be performed by standard "objects"
Programming Skills	- Requires advanced legacy programming skills	- Unfriendly development environments; COBOL skills often required to tune generated code	+ Most development can be performed through user-friendly GUIs
Stability	- Inconsistent	Moderate	Moderate
Exception Handling Support	- Custom	- Custom	- Basic processing provided; usually must be linked to external custom code
Scheduling Support	- No	+ Yes	+ Yes

Table 5-12: ETL Technique Selection Matrix

5.5 Middleware

Middleware deals with incompatibilities between the data sources, transform servers, and the data warehouse. It provides a set of software services, either custom developed or vendor provided, which enables elements during the population process to interoperate. These elements share function, content, and communication across computing environments. Important middleware types are discussed below:

- Database access middleware (DBAM) — DBAM enables clients to interact robustly with relational databases across disparate networks, protocols, and interfaces. There are three types of database access middleware: ODBC-like, proprietary, and gateway.
- Message-oriented middleware (MOM) — MOM refers to the process of distributing data and control through the exchange of records known as messages. MOM can be categorized as message passing, message queuing, and publish and subscribe.
- Object request brokers (ORBs) middleware — ORBs enable client objects to access server objects over the network and invoke operations (for example, functions, methods). ORBs typically provide interoperability between heterogeneous client and server environments: across languages and/or operating systems and/or network protocols.
- Remote procedure calls (RPCs) — RPCs enable clients to invoke remote services as if they were called and processed locally.

- Transaction processing (TP) monitors — TPMs provide queuing and other transaction services designed to support the efficient processing of high volumes of transactions. Services include load balancing, rollback, backup and recovery. It also acts as a database connection concentrator since the clients are now connected to the TP Monitor and not the DBMS directly.

5.6 Batch Architecture Issues

This section describes support mechanisms that effectively manages and distributes data from the data sources to the data warehouse. It discusses the following mechanisms: bulk load, error handling and restart/recovery, data movement, exception handling, scheduling, and verification and audits.

5.6.1 Bulk LoadBatch

Bulk load provides a high speed method of loading large volumes of data by disabling data integration checks, index creation, logging, and other operations that slow down the process. They support compression, blocking and buffering to optimize transfer times. Bulk load significantly increases the loading speed, so it should be used whenever possible. However, when using this technique, the technical architect should ensure that the requirements for load restart and recovery are met.

5.6.2 Error Handling and Restart/Recovery

Error handling and restart/recovery provide a strategy to solve an unexpected error condition during the load step. The following strategies are possible for such a problem:

- Checkpointing — Resume the load at the point where it stopped
- Roll back — Roll back to the state it was before the load was started; assumes that the database was backed-up before the load
- Internal database backup and recovery — Load the data from the database instead of a flat file

Unfortunately, many bulk load programs do not support these strategies. They leave the database in a half-updated state. If only new records are being inserted, this is not a problem, because the load can be restarted with the instruction to only insert new records. However, if many of the existing records are being modified, the load cannot be restarted from the beginning without updating some of the records a second time. Also, it is nearly impossible to reconstruct the state of the data warehouse before the load unless it was saved.

5.6.3 Data Movement

Data movement describes the functions necessary to transport data from the source systems through the ETL to the target databases. Data movement can be accomplished using middleware or in a batch mode by invoking FTP mechanisms.

5.6.4 Exception Handling

Exception handling provides the ability to handle expected error conditions. Consider Figure 5-12, where certain data fail to load into the data warehouse for various reasons (referential integrity, duplicate records, etc.). In addition, data may be discarded by the ETL process as being uncleanable. Usually, the ETL steps place these exceptions in reports or files for later examination. The user can inspect the data to determine the reason the data was not processed, repair it, and then send it back through the ETL process. The repaired exception file may be merged with new data just starting the ETL process. If the repairs were effective, the changes will go through cleanly. Otherwise, the changes will be written back to the exception file.

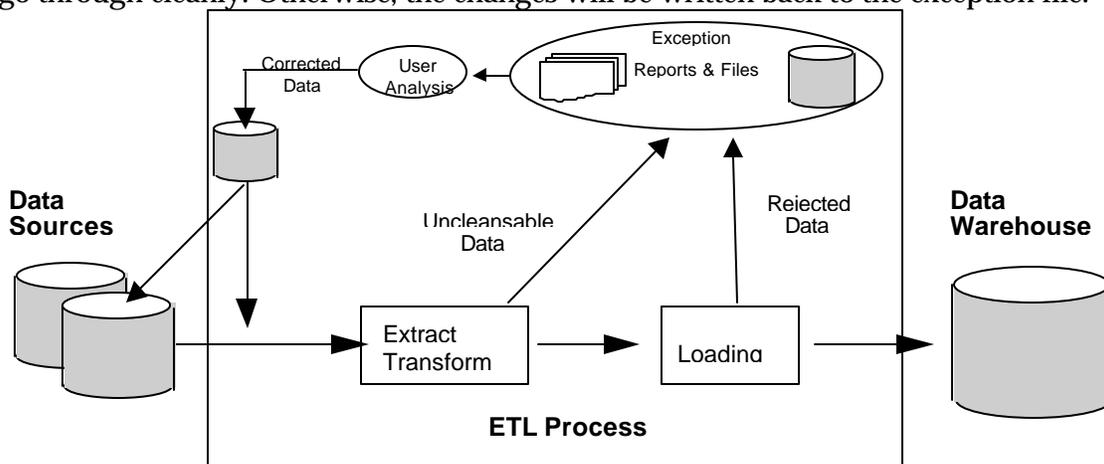


Figure 5-12: Exception Handling Process

5.6.5 Scheduling

Scheduling provides the ability to execute the ETL process in an automated manner. It is responsible for automated updates, because certain portions of the data may be updated at different intervals. For example, sales data may be transformed and loaded weekly, while marketing data is transformed monthly. Therefore, it is necessary to set up scripts, which define the update interval.

5.6.6 Synchronization

Synchronization provides the ability to track all changes made on each system and to update this changed information on each system.

5.6.7 Audits and Verifications

Audits provide the ability to perform an intensive data level analysis with statistical-sampling tools for large data sets. They look for synonyms and functional dependency rules that span different sources. They do not scrub or move data, but they provide the tools to audit the data and warehouse design effectively. Their goal is to discover the data rules by analysis of the data itself.

Most audit tools use metadata, which can make an auditor's job easier. It can reduce analysis time from the source systems since it contains information on transformation logic which was inserted to cleanse data, logic which was inserted to deal with technical issues, and logic which was inserted to deal with business issues.

Most DBMSs have repair or verification utilities to ensure that the metadata is complete, that pointers to the index and data pages are consistent and uncorrupted, and that space is properly allocated. However because legacy systems tend to change and the data warehouse team may not always be aware of the impact to data quality, strong client participation is required to ensure proper data quality control. Repairs are made to corrupted or inconsistent areas—usually automatically, unless there is permanent damage. For VLDB environments, these operations can add to an already limited timeframe to perform operations.

The following operational points should be considered when implementing audit or verification functions:

- The frequency of repair and consistency verifications performed on a database must be balanced with their effect on availability and response times.
- Verifications and repairs should be done at the table level. This may be far less costly than checks at the database-wide level.
- If possible, verifications should be completed on-line as they can be time-consuming for large data warehouses. Pulling them off-line for a lengthy check may be too costly.

ETL Tool Selection Criteria

This section highlights the major selection criteria to consider for ETL tools. Criteria which apply to all types of ETL tools are given, followed by those which pertain to each individual tool category.

GENERAL ETL TOOLS

- Scheduling support — The tool should be possible for the user to set integrated time and event-based schedules as well as transfer data “on demand.” He or she should also be able to schedule pre- or post-process events, simultaneously handle multiple transfers, and identify successors or dependencies.
- Error facilities — The tool should notify the user of errors with error codes rather than generic error messages. The user should also be able to define error handling, error log locations, and audit log locations. Rollback/commit should also be supported.
- Security — The tool should prevent the overwriting of metadata and data map definitions, the unauthorized generation of mapping engines, and the unauthorized access of mapping details. Also, it should not require System Administration security level to operate effectively.
- Platform support — The tool should be compatible with all applicable project platforms, legacy and operational sources, and data warehouse targets.
- Metadata repository support — The tool should support the bi-directional transfer of data, metadata reporting, full editing of definitions, shared repositories, and versioning of source and target metadata definitions. The tool should also support common DBMS and end-user access tool standards and definitions.
- Metadata loading - import and export — The tool should be able to capture definitions automatically and from both relational data sources and flat file (legacy) data sources.

EXTRACTION TOOLS

- Source data extraction support — The tool should be able to extract data from both relational and flat file (legacy) formats.

TRANSFORMATION TOOLS

- Data transformation support — The tool should support proprietary and non-proprietary transformation language, user-defined exits, data type and character set translation (such as EBCDIC vs. ASCII), table lookups, calculated fields, and point and click type source to target mapping.

- **Complex transformation types** — The tool should support summaries, user-defined fields, data type conversion, and conditional statements.
- **Self-documentation** — The tool should be able to provide a description of source and target systems and how they map now and in the future.

LOADING TOOLS

- **Target data insert/load support** — The tool should be able to load data into both relational and flat file target formats. The tool should also support bulk data loading.

6 End-User Access

This section of the Data Warehousing Standards and Guidelines describes how information is delivered to the end-user and utilized in the decision making process. Through presentation and analysis tools, end-user applications provide users access to information contained within a data warehouse or data mart to help them leverage people and information for increased business performance. These applications fall into many different categories, as distinguished by their various capabilities. However, their specific uses depend upon the requirements and knowledge levels of their users. Because they may vary greatly, it is uncommon for one end-user application to satisfy all requirements. As a result, an organization will commonly use more than one application to fulfill end- users' requirements.

After reviewing this section, the reader will have a basic understanding of the following topics:

- Categories of end-user access
- End-user access tool applications
- End-user access tool architectures
- End-user tool and vendor selection criteria

6.1 End-User Access Tool Categories

There are several categories of tools designed to access and analyze data which present advantages and disadvantages to users, depending upon their needs. These categories are:

- Reporting tools
- Query tools
- Analysis tools
- Knowledge discovery tools

Identifying a single category of end-user tool for a project is usually not possible because vendors, in order to serve many types of customers, are increasingly adding capabilities that place them in more than one category. It is important to focus not on category, but instead on the selection of an end-user access application that closely matches the user's functional requirements and abilities. Figure 6-1 shows the typical user and level of analysis supported by each type of tool.

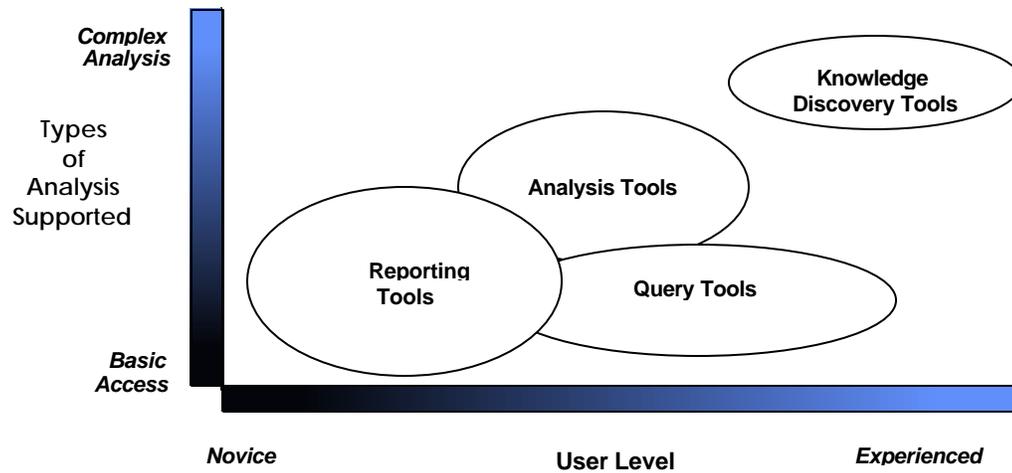


Figure 6-1: Matrix of End-User Access Categories

Table 6-1 summarizes how each end-user access type uses information and the typical users for each type.

Access Category	Access Tool	Information Usage	Users
Reporting	EIS	<ul style="list-style-type: none"> Status Reporting Summarizing Data Drilling Capabilities 	<ul style="list-style-type: none"> Executives High-level management
	Report Generation	<ul style="list-style-type: none"> Static Data Predefined Reports Limited Interaction 	<ul style="list-style-type: none"> Analysts <p>Experienced</p>
Query	Ad-hoc query	<ul style="list-style-type: none"> Fact Finding Querying 	<ul style="list-style-type: none"> Power Analysts Specialists
	Ad-hoc aliasing/ reporting	<ul style="list-style-type: none"> Fact Finding Reporting 	<ul style="list-style-type: none"> Analysts
Analysis	MOLAP, ROLAP, HOLAP	<ul style="list-style-type: none"> Exception Management Issue Resolution What-if Analysis Multidimensional Analysis 	<ul style="list-style-type: none"> Analysts Planners Specialists

Table 6-1: End-user /Access Type Comparison

Access Category	Access Tool	Information Usage	Users
Knowledge Discovery	Data Mining	<ul style="list-style-type: none"> • Rule Discovery • Pattern Identification 	<ul style="list-style-type: none"> • Specialists • Analysts
	Data Visualization	<ul style="list-style-type: none"> • Interactive Graphics • Pattern Recognition 	<ul style="list-style-type: none"> • Executives • Managers
	Impact Analysis	<ul style="list-style-type: none"> • Separation of Distinct Variables • Relative Importance of Variables • Projections 	<ul style="list-style-type: none"> • Specialists • Analysts
	Risk Analysis	<ul style="list-style-type: none"> • Pattern Identification • Projections 	<ul style="list-style-type: none"> • Specialists • Analysts
	Time Series Analysis	<ul style="list-style-type: none"> • Temporal Projections • Separation of Distinct Variables 	<ul style="list-style-type: none"> • Specialists • Analysts

Table 6-1: End-user /Access Type Comparison (Continued)

6.1.1 Reporting Tools

Aimed at less technical users usually at both ends of an organization, reporting tools are designed to facilitate the production and distribution of reports containing generally static warehouse information in the form of graphs, charts, and tables. These tools fall into two classifications:

- Report Generation/Publication — Report generation and publication tools are designed for the fast, easy, and (in some cases) automatic production of presentation-quality reports of key business information to be distributed through the web, e-mail, or corporate file servers in predefined formats
- Executive Information Systems (EIS) — More sophisticated than report generation and publications tools, EISs are easily programmable custom user interfaces allowing limited access to the data warehouse in the form of reports, newsletters, or newspapers

6.1.2 Query Tools

Querying capabilities are used to support the retrieval of basic information through Structured Query Language (SQL) or a dynamic data exchange (DDE). Users proficient with query languages and database structures may write their own retrieval code or use common tool functions which generate the appropriate code transparently using familiar business terms easily selected from menus or objects. These tools block out the complex technical structure and present information in simple and effective business formats.

A user who knows what information is required and understands how the data is structured can readily retrieve what he or she wants.

Typically, query tools are used by organizations looking for extensive access to their enterprise-wide data and adequate presentation capabilities, but no in-depth analysis of that data. This type of data access does not promote information discovery because the user must know what to look for and then issue a request each time new information is desired (see Figure 6-2). Query tools are most effective when they are used to deliver “canned” reports and are least effective for executive users who require an easy to use format to investigate enterprise-wide data.

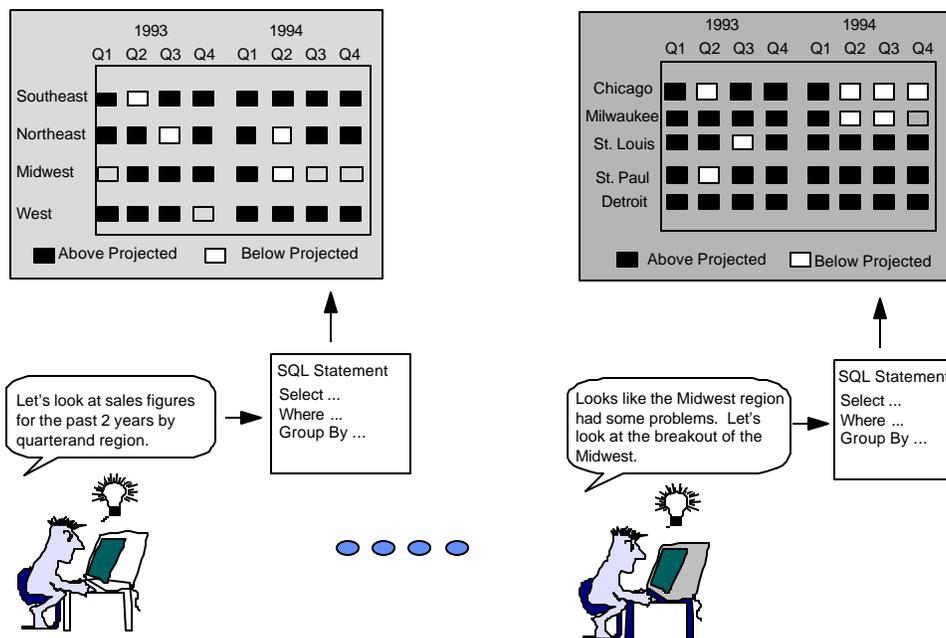


Figure 6-2: Query Scenario

Query tools usually do not include multidimensional storing or analysis capabilities, although some support built-in databases. These types of built-in databases act as filters that provide a representation or view of data specific to a group of users. For example, a marketing person may only need data relevant to marketing, such as “customer,” “sales revenue,” and “product,” where a human resources person may only need data such as “employee name,” “salary grade,” and “hire date.” This capability enables multiple users to access the same database, but only view the data important to their department.

SQL-based tools are limited by the functional capability of SQL itself. For example, simple business queries, such as, “How do sales this month compare to the average over the past three months?” are difficult to ask with SQL. They typically require a multi-step process; in this

example, first drawing the current month, then prior months into a spreadsheet before computing variances. This cumbersome approach limits its appeal and effectiveness. Functions that make SQL transparent to the user rely on menu-driven graphical user interfaces. They allow users to point and click to generate SQL statements instead of coding SQL syntax using common business terms rather than table and column names. For example, instead of “acc_prod_num1” the user might point to “account product number.” This enables decision makers and management level personnel (or non-programmers) to access needed information more easily.

6.1.3 Analysis Tools

Analysis tools allow users to view historical data at various levels of detail (or granularity) to:

- Examine trends
- Establish rankings
- Examine the effects of specific business decisions

These high level analyses are supported by On-line Analytical Processing, or OLAP, which is based on the extraction of data warehouse information through targeted querying.

OLAP Overview

Combining features of query and reporting writing tools, OLAP enables users to perform analyses in an intuitive presentation environment, and is the term that has come to represent the field of analysis tools.

Codd and Date have established 12 rules for OLAP systems. Although the term OLAP and the following rules have been widely accepted in the end-user access market, it should be noted that, presently, no one tool completely satisfies all these requirements. It should also be noted that these rules were established in conjunction with Arbor Software, designers of the multidimensional database product known as Essbase:

- Multidimensional conceptual view —Users are provided with multidimensional analysis capabilities
- Transparency — Complex physical schemas are hidden from the user
- Accessibility — Heterogeneous data sources are supported
- Consistent reporting performance — Performance is independent of number of dimensions

- Client/server architecture — Must be capable of operating in a client/server environment
- Generic dimensionality — Drill-everywhere capabilities
- Dynamic sparse-matrix handling — Physical data storage requirements are minimized
- Multi-user support — Concurrent access, integrity, and security are achieved
- Unrestricted cross-dimensional operations — Concurrent calculations across more than one dimension are possible
- Intuitive data manipulation — A customizable interface allows for flexibility
- Flexible reporting — Multiple reporting options are available
- Unlimited dimensions and aggregation levels

Most OLAP products enable a user to perform sophisticated multi-level analyses while avoiding complex access issues by giving him or her the ability to navigate or browse through a data warehouse intuitively and without interruption. For example (see Figure 6-3), a user can be viewing a graph of sales by year for all regions and decide it might be better understood if the view is rotated. He or she might then decide to drill-down to cities within the region and change the presentation to a table. All these manipulations can be easily performed by point-and-click or dragging and dropping using a graphical user interface. By promoting exploration, OLAPs contrast with ad-hoc querying, which forces the user to issue new queries for each new idea.

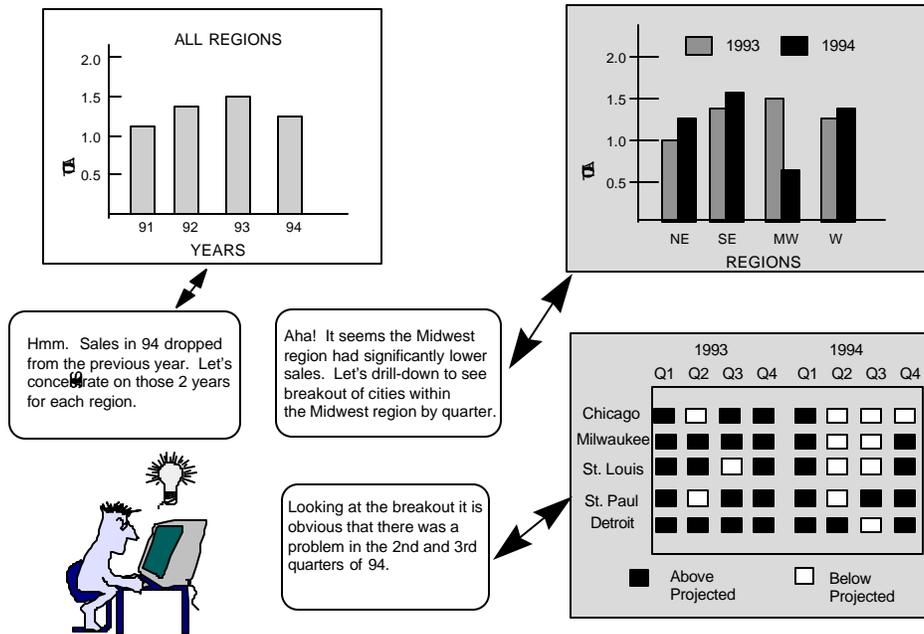


Figure 6-3: Data Exploration Scenario

Drill-down, drill-up, and drill-across, which enable users to investigate information at lower, higher, or the same information levels of an organization by point-and-click methods, are three important methods of effective knowledge exploration. See Figures 6-4 and 6-5 for illustrations of these techniques.

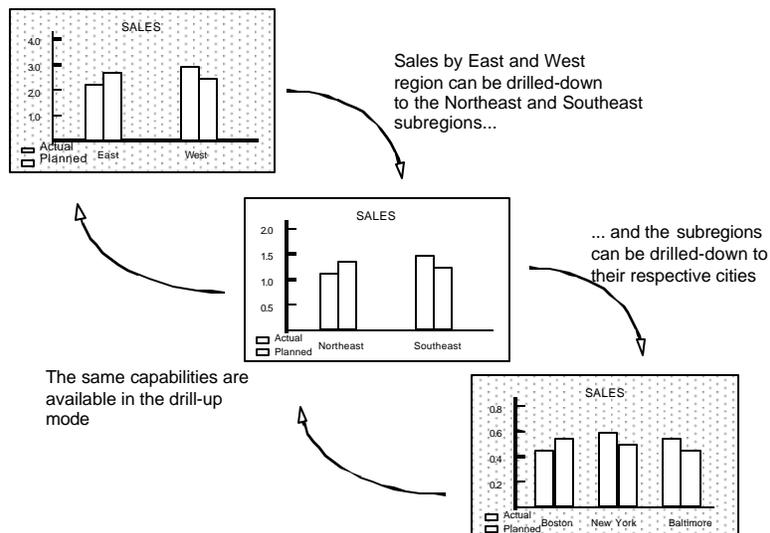


Figure 6-4: Drill-Down/Drill-Up Example

Drilling-across allows the user to view the same information for different entities at the same level.

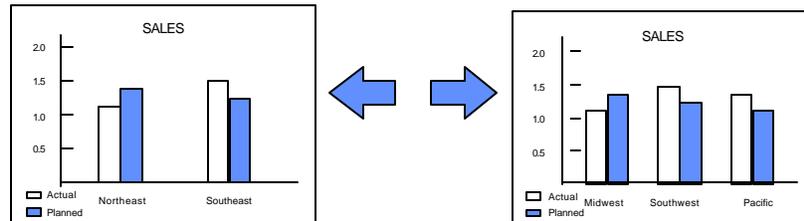


Figure 6-5: Drill-Across Example

The OLAP market has been marked in recent years by alliances among specialty software houses that develop and market programs designed for use in a specific industry (such as health care, retail, communications). Many vendors have also added OLAP functionality to their product as an “OLAP extension” or “OLAP engine,” which can be purchased separately or as an add-on. OLAP tools include:

- Multidimensional on-line analytical processing (MOLAP) tools
- Relational on-line analytical processing (ROLAP) tools
- Hybrid on-line analytical processing (HOLAP) tools

MOLAP

Multidimensional on-line analytical processing (MOLAP), requires the creation of a multidimensional information hypercube to handle all queries. The advantage of this approach is speed, as the retrieval of information from the cube can highly optimized for the types of queries desired by the end-user. However, depending upon the number of dimensions modeled, the size of the cube can be quite large (each cell inside the cube represents the intersection of a unique combination of dimensions). Most cannot scale beyond a certain threshold (usually between 20 and 50 GB), and the number of possible dimensions is limited (usually about 10). Queries are also restricted to information contained within the cube, and its setup and creation can also be quite time consuming.

Dimensions: Region, Product, Time
Items: midwest, computer, 2nd quarter
Numeric Value: 10,000

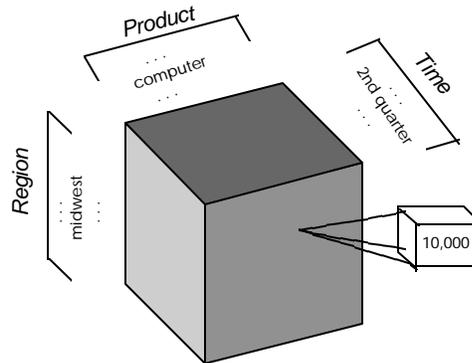


Figure 6-6: Data Location in a Multidimensional Database

The building blocks of a multidimensional database are dimensions, the items within the dimensions, and the numeric values associated with these items. Figure 6-6 illustrates the following example: 10,000 computers were sold in the Midwest region during the second quarter of 1994. Here, region, product, and time are the dimensions; “Midwest,” “computers,” and “second quarter of 1994” are items within the dimensions; and the numeric value 10,000 associates the dimensions to a business event.

A multidimensional tool enables the user to perform various forms of analysis through drilling, data rotations, data slicing and dicing, and multiple hierarchies. In performing different types of business analyses, these actions are very powerful and are also transparent to the user, which is unlike relational models in which users must be cognizant of table structures when attempting to perform tasks such as table joins. Multidimensional tools represent data in a business view by providing multiple perspectives and rotations, giving users the flexibility to navigate easily to more effectively understand business problems.

What were the *average margins and total sales* for *digital wireless calls over 10 minutes* from *Georgia* in *May 1997*?

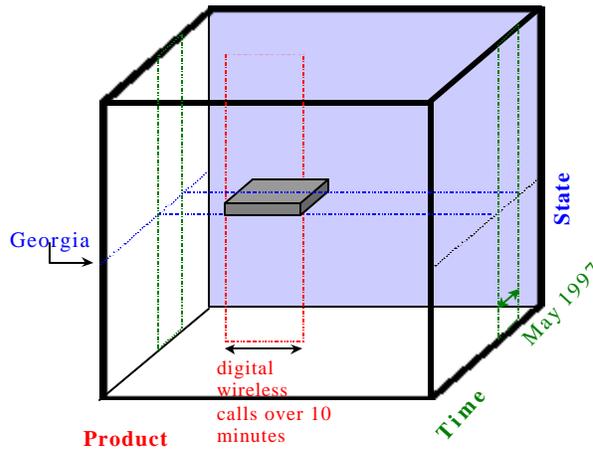


Figure 6-7: Slice, Dice, Drill, and Pivot

Multidimensional data structures enable users to have the ability to define multiple levels on either axis. For example, a multilevel cross tab may have row headings for country, state, city, and office, and column headings for year, quarter, month, and week. The multidimensional structure can then easily be rotated or pivoted for different views of the data. Within it, the grouping of similar items can be visualized by taking slices of the cube as shown in Figure 6-7. A product manager might focus on one product across many time periods and markets. A financial manager might concentrate on the current and previous time period for all markets and all products. A regional manager might study all time periods and products across some markets. And a strategic planner might focus on a subset of all corporate data and would therefore benefit from an ad-hoc view.

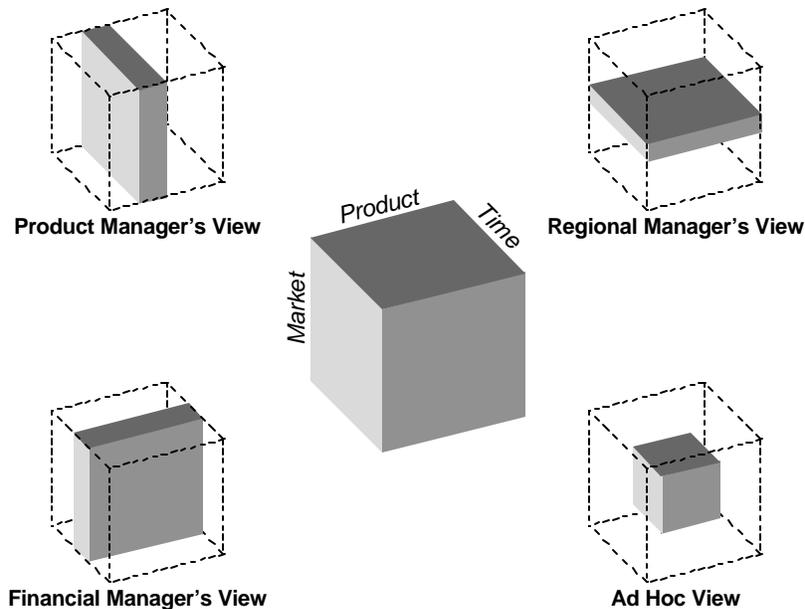


Figure 6-8: Views of the Multidimensional Database by Knowledge Worker

These data structures are very effective when used in systems (DSS/EIS) where quick and accurate access to business information is necessary. This is because they facilitate flexible, high performance access and analysis of large volumes of complex and interrelated data across various platforms.

ROLAP

Relational on-line analytical processing (ROLAP), is architecturally simpler than either MOLAP or HOLAP. In a ROLAP system, a user working through a GUI-based presentation layer creates SQL through a translator, the ROLAP engine. This SQL is then submitted to the warehouse, which returns the data to the end-user tool for cross-tabulation and final presentation. This allows almost unlimited flexibility in carrying out queries using information contained at all levels in the warehouse, or pre-calculated results, if available. It also allows fast set-up time, RDBMS schema independence, and, in addition to what is occupied by the warehouse, a minimal amount of hard disk space as the relational database handles data storage requirements and the ROLAP engine provides analytical functionality.

ROLAP supports a variety of different optimization techniques to increase response times:

- Database-specific SQL
- Application-level table partitioning

- Aggregate awareness and navigation
- Denormalization support
- Multiple fact table joins

Even with these techniques, ROLAP’s advantages come at a price, as its users can expect slow performance in retrieving information from large databases, necessitating performance enhancements such as caching.

HOLAP

Hybrid on-line analytical processing (HOLAP), is an attempt to combine MOLAP’s speed with ROLAP’s flexibility. Like MOLAP, HOLAP requires a cube. But, if the user requests information not contained within the cube, the HOLAP engine will generate SQL to retrieve new data from the RDBMS. HOLAP, however, is more difficult to implement and administer than either ROLAP or MOLAP, and its ROLAP component lacks the schema independence or multi-pass SQL generation abilities of their true ROLAP counterparts.

Analysis Tool Category Comparisons

Table 6-2 compares MOLAP, ROLAP, and HOLAP systems.

Criteria	MOLAP	ROLAP	HOLAP
Performance	<ul style="list-style-type: none"> • Fast for querying information within the cube • Optimization for particular queries possible 	<ul style="list-style-type: none"> • Moderate to poor, depending upon SQL complexity and optimization for particular databases 	<ul style="list-style-type: none"> • Fast for querying information within the cube • Slow for SQL-based retrieval of information from the warehouse
Scalability	<ul style="list-style-type: none"> • Limited by size constraints 	<ul style="list-style-type: none"> • Excellent; can support large amounts of data 	<ul style="list-style-type: none"> • Very good; data can be added to warehouse for SQL-based access
Flexibility	<ul style="list-style-type: none"> • Limited to queries and information provided for in the warehouse • Cubes can easily be “sliced” and “diced” 	<ul style="list-style-type: none"> • Very flexible; limited only by SQL itself 	<ul style="list-style-type: none"> • Very flexible; combines MOLAP and ROLAP access capabilities
Implementation	<ul style="list-style-type: none"> • Long build times for cubes 	<ul style="list-style-type: none"> • Very simple – only requires a connection to a data warehouse 	<ul style="list-style-type: none"> • Can be difficult and complex • User must decide what information is put in cube and what must remain in warehouse
Administration	<ul style="list-style-type: none"> • Cumbersome – updating the cube requires means rebuilding it 	<ul style="list-style-type: none"> • Very simple as long as warehouse is kept up to date 	<ul style="list-style-type: none"> • Tuning and builds difficult; requires rebuilding the cube

Table 6-2: MOLAP, ROLAP, and HOLAP System Comparison

6.1.4 Knowledge Discovery Tools

Knowledge discovery tools differ from query and analysis tools in that they, rather than users, ascertain important trends and patterns from information stores. These tools can be highly useful, as human beings are limited not only by their inability to differentiate between mere correlations and deeper cause and effect relationships, but also by their inability to determine the relevancy of patterns derived from historical data and extrapolate them into the future. Knowledge discovery tools include:

- Data mining tools
- Impact analysis tools
- Risk analysis tools
- Time series analysis tools

Knowledge discovery activities cannot be entered into lightly. They require technical expertise as well as a high degree of subject matter knowledge. The following issues are often encountered:

- **Modeling Accuracy**— Knowledge discovery tools may uncover statistical correlations between events or variables that have, in fact, no causal relationship. For example, it may be shown that rainfall in Madagascar might correlate with the S&P Index. Users must exercise good judgement in deciding which variables to include in models and which models make sense. Often key to the roll-out of a data mining system is a training period, in which the system and its users have time to work through and “learn” from their mistakes.
- **Informational Relevancy**— Informational relevancy has to do with deciding what patterns and rules found by the knowledge discovery system are important enough for alerts to be sent, and how and in what context this information should be presented to an end-user. This requires a sophisticated understanding of the business processes being modeled and explicit instructions from the user about what facts he/she finds important. A knowledge discovery system can potentially find millions of rules, only a handful of which has real value.
- **The Determination of Actionable Data** — Recommendations given to someone incapable of carrying them out or not tied to a verification system will have only limited utility. The determination of actionable data refers to the question of how to

best translate the recommendations of a knowledge discovery system into action by delivering the right information to the right people.

6.1.5 Data Mining

Data mining differs from traditional decision support systems in its attempt to interpret data with pattern recognition technology. Discovering relevant patterns in large data stores (at atomic levels) can be important for more effective decision making. More advanced data mining detects patterns in the data that are unknown to the end-user, generating hypotheses rather than reporting or confirming them. Primitive data mining searches for patterns requested by the end-user. Its Return on Investment (ROI) emerges from a reduced cost of analysis and, most importantly, through the discovery of important cost-saving or revenue-enhancing trends. Three simple scenarios, in sales, marketing, and field-service engineering, illustrate the potential utility of data mining.

Sales: A company wishes to sell product A to a customer in geographic region B and must compete with competitor C to do so. Given information about the product, sales history of the company, the region, and information about the competitor, the data mining system will inform the executive deciding whether to proceed that the likelihood of a sales opportunity is, say, greater than 63%. It then factors information about the actual outcome of the sale into future recommendations.

Field-service engineering: A company must provide service for product A in region B, and it has data about the product's repair history and knowledge of its usage patterns. The data mining system will predict the length of the service interval and will determine the optimal patterns for service calls, changing previous schedules to reduce call-out, thus saving resources.

Marketing: A company must decide how to allocate its available resources in order to maximize its return. Given information about the product being marketed, the region, and demographics, the data mining system will segment potential customers and make recommendations about how the marketing campaign should be targeted.

The following is a partial list of data mining techniques:

- Neural networks
- Case-based reasoning
- Rule induction
- Genetic algorithms
- Statistics (multivariate regression)

- Compression and regression trees

Impact Analysis

allows the user to specify a value to observe and a list of variables that may influence that value. The analysis then provides a level of impact that each of these factors would have on the variable of interest.

Example: A manager wants to investigate the impact on profits if the cost per unit resource increases, or the labor rate increases, or if both dimensions are simultaneously increased. When this what-if/impact analysis is performed in incremental steps, the effect on total profit can be viewed for each step. For example, if the unit resource cost increases from \$30 to \$31, \$32, or \$33/item and the labor rate increases from \$10.00 to \$10.33, \$10.67, or \$11.00/hour, the change in total profit can be viewed for each increment of change.

Risk Analysis

Risk analysis allows users to test various conditions, such as price changes or schedules, to determine the overall effect of the changes on production and/or sales. One form of risk analysis can be performed by Monte Carlo Simulation. This method allows a user-defined model of variables to be recalculated with randomly selected values and the probabilities of these generated models to be calculated using statistical distributions.

Example: A computer distributor is notified by its supplier of a decrease in production cost that will drive the price of the computer down by 5%; risk analysis capabilities allow the organization to estimate the (most likely) new stocked inventory levels that are required to handle the increase in sales.

Time Series Analysis

Time series analysis is concerned with the statistical analysis of data that arise in time, such as monthly sales revenues, daily stock prices, or hourly barometric pressure readings. It allows users to test the effect of changing variables over a specified time period and observe the results for different time intervals.

Example: A large retail store estimates a 15% increase in sales for all holiday weekends from the previous year, time series calculations allows the user to determine how this affects the overall profits for the year.

Other examples of time series analysis include forecasting schemes where the user can select data points on the graph to display the corresponding data values, including forecast values. The user may be able to select the most appropriate algorithm (or have suggestions of which is most appropriate) for the forecast computation.

Example: A company wants to forecast sales of product A over the next year. This may be achieved by using actual sales figures from the previous years (for more accurate forecast projections, more years of data need to be used) and a forecasting algorithm. By using statistical techniques, the new sales levels could be accurately projected from the same period in previous years.

Trend analysis is another example of time series analysis. It indicates the effect of static or stable model conditions over a user-defined time period, facilitating comparisons of various intervals. Trending functions can be represented by graphing simple regression analyses.

Example: A company has 100,000 product units in 175 stores in 35 states and wants to associate sales and inventory levels with in-store, print, and television advertising. By performing a trend analysis, a user can determine the effectiveness of various advertising media in different markets.

6.2 End-User Access Tool Architectures

System architecture plays an important part in the selection and evaluation of end-user tools, affecting performance, remote-use, and stability. This section presents the following:

- Client/Server architectures — Divides the application architecture into two or three tiers. With the data warehouse repository on one machine, designers can choose between end-user access applications residing entirely on client machines or splitting them into client and server portions.
- Net-centric architectures — Allows access to the data warehouse through the World Wide Web.
- Security issues — Makes sure repository information is kept private and secure.

6.2.1 Two-Tier vs. Three-Tier Architectures and Tools

Many of DSS/EIS applications allow two different configurations:

- Two-tier configuration — The data warehouse repository (RDBMS) is located on one machine and end-user access applications run on PCs. This corresponds to the

Remote Data Management client/server model, in which the presentation and application logic is entirely on the client side. Almost all report and query tools use this architecture.

- Three-tier configuration — The data warehouse repository is located on one machine and the end-user access application resides on separate client and server pieces. This corresponds to the Remote Presentation and Distributed Function client/server models, in which the presentation is on the client and all or part of the application logic is on the server.

In the two-tier approach, the DSS/EIS client application can access either the RDBMS or a multidimensional store directly. In the three-tier approach, the DSS/EIS client must go through the server piece to access the RDBMS or a multidimensional store. The server can reside on the same physical machine as the RDBMS or on its own dedicated machine. This provides additional scalability at the expense of additional software and network overhead.

Choosing the Appropriate Configuration

Table 6-5 summarizes decision criteria for selecting two- and three-tier configurations for end-user tools.

Criteria	Two-Tier	Three-Tier
Resource Allocation	<ul style="list-style-type: none"> • Heavy client processing; all application logic is on the client 	<ul style="list-style-type: none"> • Light client processing - the application is split, with the client only doing presentation plus perhaps limited processing
Hardware Requirements	<ul style="list-style-type: none"> • Requires more powerful and more costly client PCs 	<ul style="list-style-type: none"> • Client PCs need not be powerful • Additional hardware needed if application server and RDBMS are on separate machines
Remote Use	<ul style="list-style-type: none"> • Not well suited due to client hardware and network requirements 	<ul style="list-style-type: none"> • Remote use requires a connection to the server; if the laptop cannot communicate with the server, the end-user application is unusable
Standalone Use	<ul style="list-style-type: none"> • Well suited; most processing done on client independent of server • Requires large amounts of memory to store database information 	<ul style="list-style-type: none"> • Not well suited; most application logic is on the server
Network Requirements	<ul style="list-style-type: none"> • Causes heavier network traffic; all the data is brought back to the client for processing 	<ul style="list-style-type: none"> • Lighter network traffic (between the server and the clients) ; the more powerful server does most or all the processing, and only sends back the results
Performance	<ul style="list-style-type: none"> • Slow if the (less powerful) client is required to do a large amount of processing, such as heavy multidimensional analysis 	<ul style="list-style-type: none"> • Slow if many clients are competing for the server's resources • Slow if the server and RDBMS are on the same platform and competing for processing time
Scalability	<ul style="list-style-type: none"> • Limited by cost and the ability to upgrade client machines 	<ul style="list-style-type: none"> • Scaling is architecturally simple; may only require an upgrade of the application server • Physical separation of the RDBMS and the end-user application may be required
Availability	<ul style="list-style-type: none"> • Requires heavy use of processing power on the client PC • Does not facilitate batch processing 	<ul style="list-style-type: none"> • Background processing possible; reports and functions such as agents and alerts can run in the background on the server • Batch processing on the server possible
Flexibility	<ul style="list-style-type: none"> • Application distribution problems • Modifications to the end-user applications must be updated on every client 	<ul style="list-style-type: none"> • Updates to the application can usually be made just to the server piece

Table 6-5: Summary of Two- and Three-Tier Configurations

6.2.2 Net-Centric Architectures

Companies involved in data warehousing are increasingly turning to net-centric architectures, generally web-enablement, to magnify their competitive advantage. The tools used to accomplish this are new; therefore, the products and architectures supporting them are neither mature nor robust. Even as vendors continue to develop and expand this market, their clients are beginning to realize significant benefits.

Advantages and Disadvantages

Web-enabling a data warehouse presents the following advantages:

- Standard, simple interface — Hypertext, frames, and other characteristics of web pages are easy to understand and well-known
- Ubiquitous web browser software — Browser software is prevalent enough that everyone with access to the internet or an intranet can use it to accomplish data retrieval; also available free of charge
- Platform independent deployment — Web browsers run on all platforms
- Centralized and less burdensome administration — No software is installed on client machines and can be maintained centrally
- Components can be executed on either client or server hardware, allowing flexibility to leverage existing client/server infrastructure and computing structures

The disadvantages of web-enabling a data warehouse include:

- Security — The internet is an open network
- Reduced functionality and performance — Memory and bandwidth limitations restrict product capabilities

6.2.3 Web Architectures

There are three distinct architectures for web enabling a data warehouse, described below.

STATIC PUBLISHING

In static publishing, the most straightforward of architectures, users request pre-generated reports by clicking on hyperlinks. These reports are stored on the web server as HTML files, which the web server then retrieves and sends to the web browser. The execution architecture consists of only the web server and browser components. Reports are typically generated during batch runs with standard parameters. If a user wants a different report or a slight

modification, he or she must request a change which may involve programming. A disadvantage of this solution is that the data is only as up to date as the HTML reports. Security, however, is a major advantage since the database is never online and accessible to end-users.

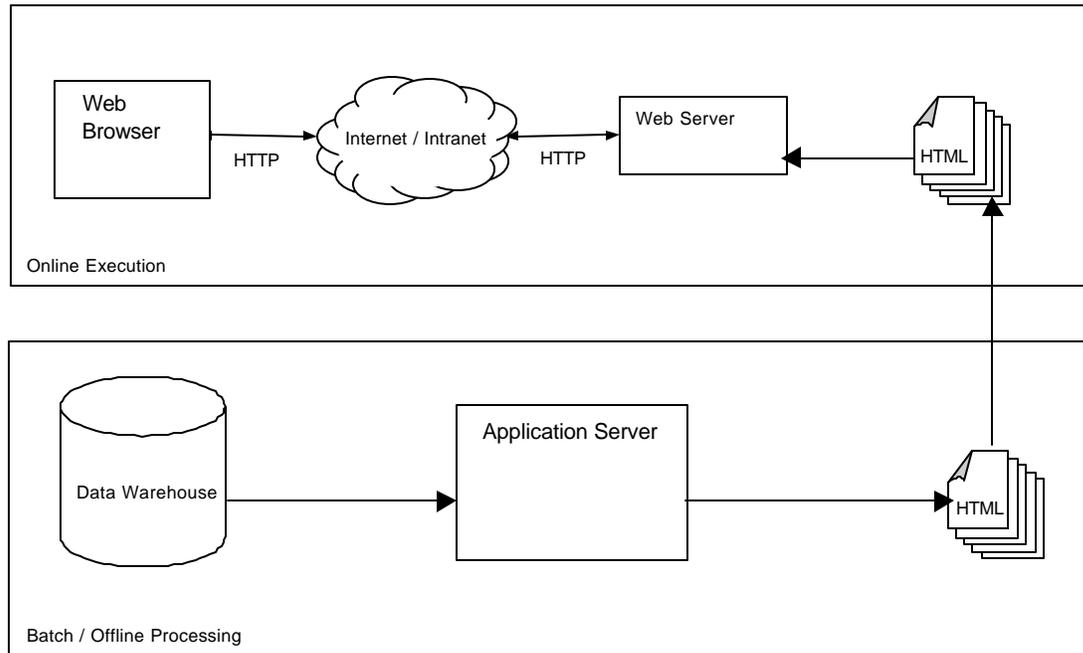


Figure 6-10: Static Publish

LIMITED INTERACTIVITY

This architecture differs from static publish by giving the user access to live data. When a user clicks on a hyperlink to retrieve a report, he or she is presented with a form allowing the specification of certain parameters such as time period, product family, or location. After pressing a submit button, the web server communicates (usually via Common Gateway Interface or CGI) with the application server, which then retrieves and formats the data into HTML. This HTML page, which may or may not be static, is then passed back to the browser for presentation to the user. If the page is static, once the report is delivered the user cannot alter it, and can only repeat the process with different parameters or a different report. If the report is interactive, then it may allow users the ability to alter the initial parameter settings or manipulate the formatting or level or detail of the returned data.

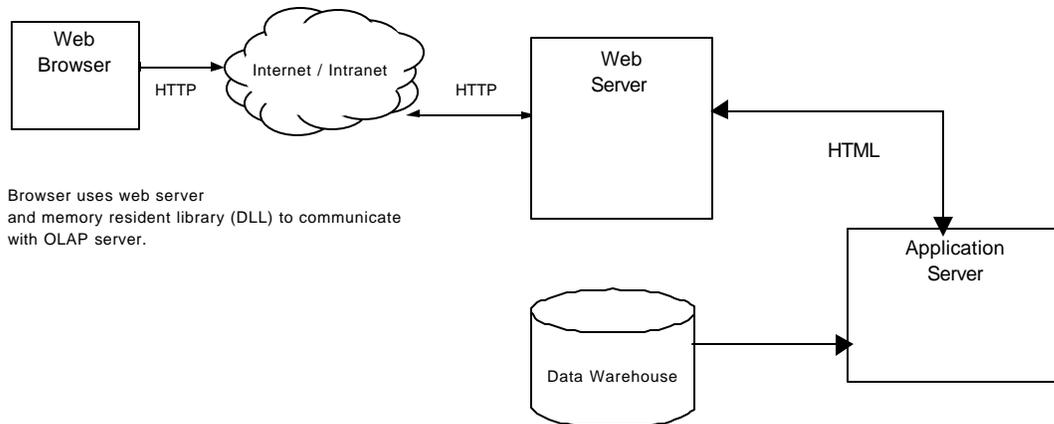


Figure 6-11: Limited Interactivity

HIGH INTERACTIVITY

This is the latest and most feature-rich of the architectures. Like its predecessor, it allows interactivity once a report has been delivered. However, in addition to HTML, this report may also contain embedded, interactive content such as drag-and-drop, pop-up menus and balloon help provided by scripting languages (JavaScript, VisualBasic Script, and LiveScript) as well as executable content (Java Applets and ActiveX controls). They allow the interface to more closely resemble the graphical interface of other client/server tools.

In addition to interface enhancements, this generation also includes two performance improvements:

- The use of Dynamic Link Libraries (DLLs) on the Windows platform and shared libraries on the Unix platform — DLLs and shared libraries facilitate memory-based interprocess communication between the web and application servers, eliminating the need for the web server to launch a communication executable for each request. This is slightly faster in terms of overall response time to the user, but more importantly can lighten the load and I/O requirements of the web server platform.
- Direct communication between the browser and the OLAP server — HTTP-based connections to web servers are stateless, meaning that information about a browser such as userid/password, previous reports viewed, current report being viewed etc., are not maintained. To avoid the overhead of setting up and tearing down TCP/IP connections proprietary protocols or the standard CORBA Internet, Inter-ORB Protocol (IIOP) maintains state information and allows the browser to bypass the web server and communicate directly with the application server.

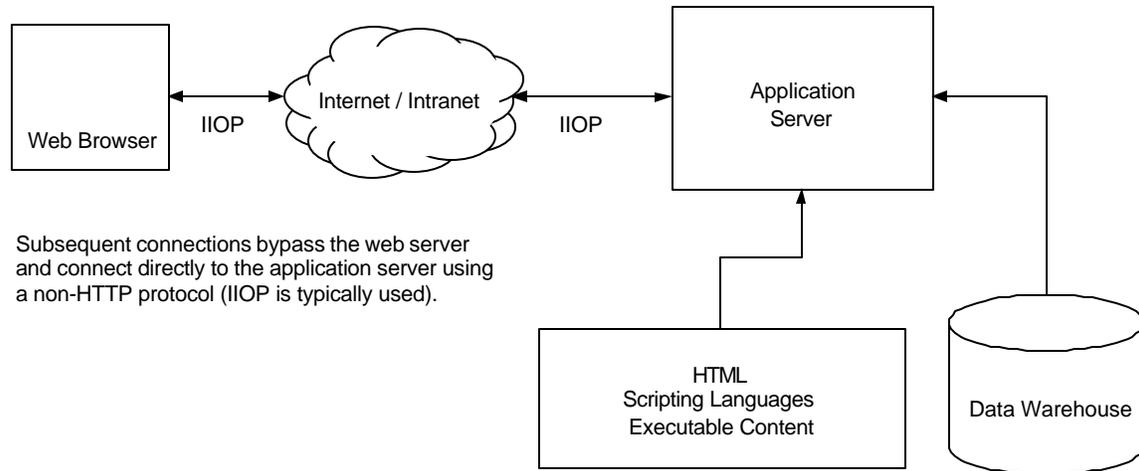


Figure 6-12: High Interactivity

6.2.4 End-User Access Security

The following topics comprise the issue of end-user access security:

- Data warehouse access security
- Network security
- Operating system security
- Internet/intranet security
- User access tool security
- Data security

These topics are outlined briefly below.

Data Warehouse Access Security

Warehouse administrators may want to restrict specific groups of end-users to certain information or procedures stored in a data warehouse, including tables and columns, models, views, and stored procedures. Typically resource-intensive, this type of security requires

constant monitoring of the warehouse and careful configuration of end-user applications to prevent unauthorized access.

Network Security

When end-user applications and the data warehouse are connected via a network, the network itself becomes an important part of the overall security framework. In an attempt to keep warehouse information secure as it is accessed by authorized end-users, system administrators must also utilize measures such as software authentication, network transmission encryption, and the physical restriction of access to the end-user applications themselves with security cards and special access terminals. Since network security is often out of the direct control of the warehouse administrator in increasingly open network environments, he or she must also regularly consult trade publications, security and user's groups, and industry experts to determine the proper strategy of securing the data warehouse while maintaining network access consistent with corporate access goals and protocols.

Operating System Security

The platforms on which the end-user access tools and the data warehouse sit are important security concerns. Consequently, the ability of administrators to implement all desired security protocols quickly and easily is a selection criteria for access tools and overall system architecture.

Maintaining operating system security is a time- and resource-intensive task. Appropriate measures include the following:

- Periodically changing passwords on all system and database default accounts
- Making sure that no company information is displayed before passwords have been authenticated
- Not allowing users to log directly on as "root"
- Insisting that all passwords contain both alpha and numeric characters
- Limiting the number of invalid logon attempts
- Making sure that all network transmissions use encryption
- Monitoring for and discouraging the use of one account by multiple individuals

Internet/Intranet Security

Companies, which for the most part, maintained information access on proprietary, limited access internal communication networks (intranets) are increasingly allowing outside internet-

based data access for various strategic and administrative reasons, often complicating warehouse security. This does not typically include access to the data warehouses, data marts, or metadata itself, which are of interest to administrators, but rather the use of end-user access tools processing that information.

Securing both intranet and internet architectures involves the implementation of similar network and operating system security procedures. Because the internet is open to the world and can expose an information network to a much larger set of potential users and interlopers, an internet-based architecture requires much more vigilant monitoring, a greater degree of participation in and awareness of discussions, users groups, and trade publications concerned with new products and methodologies having the capability to impact internet security, and a more in-depth understanding of the potential security weaknesses in end-user access tools themselves - including, potentially, various web browsers.

User Access Tool Security

There are two components to user access tool security:

- Using the tool's own security provisions to restrict the access of unauthorized people
- Limiting computer resource allocations

Using a tool's security measures involves a greater degree of administration to set up, evaluate, and monitor users, the level of effort for which depends largely on the tools themselves. Limiting computer resource allocations, on the other hand, is much more straightforward, and often requires little beyond the use of special rooms, access terminals, and security cards for intranet applications.

Data Security

Data security refers to the ability to present selective data to end- users based on their security profile. Data security involves the following facets:

- Read or select access — Only authorized users can access the data
- Write, delete, or update access — This is independent of the read access and control the users who can change the data
- Create or execute table access— Even the users who can change the data may not be able to create new data tables or formats

The data security can be implemented using the database access profiles such as:

- **System or table level passwords** —Database products which allow the set up of the system or table level passwords
- **Views** —Application programmers can create views that implicitly control data access
- **Triggers** — Triggers can be used to control data access at a very fine level of granularity