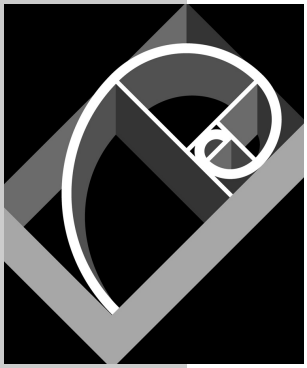


PowerDesigner® WarehouseArchitect™
The Model for Data Warehousing Solutions

A Technical Whitepaper from Sybase, Inc.

SYBASE®

Table of Contents

Section I: The Need for Data Warehouse Modeling

Introduction

Effective business decisions are a determining factor in the success of the enterprise. Key to effective decision making is the ability to readily access corporate information to assess company performance, market trends, and the resulting business activities.

Traditional, transaction-based information systems provide input (entering information into the database), processes (transforming that information), and output (delivering the results of processes). The focus is essentially on recording data and transforming that data through processes. The results of these processes are communicated to end users through the screens designed by the application developer.

New technologies have emerged to provide efficient decision-support tools working together in what is known today as a data warehouse environment. As a result, the emphasis has shifted from getting data into a database (transactional systems), to getting data out of the database (data warehouse) to serve business analysis purposes through user-defined screens.

Data warehousing integrates corporate operational data into a single informational system (the data warehouse) that business end users query using data-analysis tools. A data warehouse is a subject-oriented system specifically designed for decision support. A data warehouse environment organizes and provides information in a way that business users best understand.

The data warehouse environment manages the flow of information, from source to delivery. The source is operational information contained in various transactional database systems, programs stored in flat files, or external sources such as the Internet. Delivery is provided by end-user tools installed on a PC. These tools are used to define business queries that the data warehouse environment processes—queries that are answered in easy-to-read forms and graphical reports.

Metadata is an essential component in the data warehouse environment. Metadata is "data about data." That is, metadata determines how the data warehouse information is designed and structured. It defines mappings between the operational source and the data warehouse model, and specifies algorithms for summarization/aggregation.

This technical paper describes how PowerDesigner® WarehouseArchitect™, Sybase's data warehouse modeling tool, addresses data warehouse design and metadata creation in every phase of the data warehouse environment construction cycle.

The Data Warehouse Environment

Existing operational systems are not adequate for decision support because they use inappropriate data structures for business information analysis. Data is not integrated from the different systems used in the enterprise, and coverage of data is very detailed (atomic).

Operational systems are used for day-to-day create, read, update, and delete (CRUD) operations, and actually run the company. This means the information handled by operational systems is always up to date (of a changing nature).

Unlike operational systems, decision-support and informational systems use aggregated (summarized), historical information (of a static nature), where the concept of time is key to determining analytical patterns. Informational systems have become an essential planning tool for analyzing how a company should adapt to the market to become more competitive.

A data warehouse architecture can be divided into three layers:

- Source Layer—operational environment
- Transform Layer—extraction, cleansing (scrubbing), and transformation of source information
- Decision-Support Layer—data warehouse, data mart, and frontend business analysis tools

WarehouseArchitect supports all three data warehouse layers in terms of data modeling, metadata, and data import, and interfaces with the various third-party tools that perform important functions in the data warehouse environment (see Figure 1).

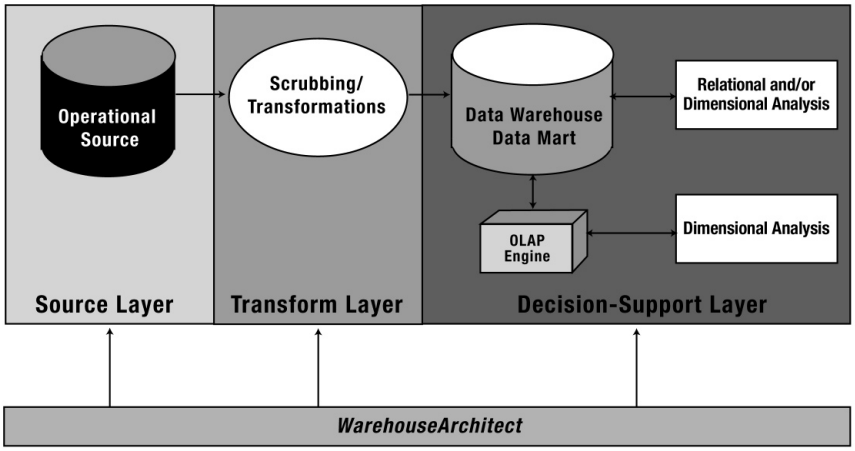


Figure 1: WarehouseArchitect supports all three data warehouse layers.

- The information contained in operational systems is the source for the data warehouse.
- Software tools are required to extract the needed data from databases, and/or files, and transform it before loading it into the data warehouse database system. These transformations are necessary because the same data elements, if used by different applications or managed by different database systems, may be inconsistent or redundant.
- A data warehouse database is generally stored in very large relational database management systems (RDBMSs). A data warehouse database can be subdivided into more manageable or domain-specific environments referred to as data marts.
- Data warehouse and data mart information may be managed by online analytical processing (OLAP) systems which transform complex data requests into dimensional SQL queries.
- Business information is accessed by decision-support systems sitting on PCs. Data warehouse and/or data mart information may also be directly accessed by managed-query tools such as Sybase's InfoMaker®.

The Importance of Data Design

Data design is the graphical representation of metadata. It provides a blueprint of the data warehouse database implementation. In addition, data design is the only automated way to optimize data warehouse performance and data access.

In the context of a data warehouse environment, data design becomes even more important because metadata must be traced from the source (operational systems) to delivery (business end-user tools). As a result, the overall data warehouse environment's design information must be driven from a single point of control—the design tool. Table 1 details how data design for informational systems is dissimilar from operational data design.

Table 1: Informational and operational design characteristics.

Informational	Operational
Value of data relative to the value of other data.	Absolute value of data (e.g, Product cost = $25).
Consistent, static data (comparison of data values between consistent sets, such as time frame, price range, etc.).	Current, up-to-date data.
Complex queries involving many records and complex join logic (optimized through highly denormalized data, ad hoc indexing schemes, aggregation, and partitioning).	Simple (predictable) queries at the record level.

This is why Sybase® created WarehouseArchitect, a new PowerDesigner module specifically designed to create data warehouse models and multidimensional hierarchies.

Section II: Introducing PowerDesigner WarehouseArchitect

WarehouseArchitect is based on the proven technology of PowerDesigner's DataArchitect™ physical data model (PDM). Specifically, WarehouseArchitect supports dimensional modeling, high-performance indexing schemes, and multidimensional hierarchies—all in a single, user-friendly design tool environment.

WarehouseArchitect provides the following data warehouse design functionality:

- Import metadata, including traceability links between operational source and the target data warehouse database with extraction commands to automate data transfer from operational databases to the data warehouse
- Multidimensional modeling (data warehouse schema) and multidimensional hierarchies (frontend analysis tools)
- Optimized code generation, both supporting multiple target data warehouse database systems (platform-specific indexing and in-place functions), as well as multidimensional frontend analysis tools
- Customizable report generation

Importing Metadata into WarehouseArchitect

WarehouseArchitect models (WAMs) include the metadata that defines the data warehouse database model. Typically, data warehouse metadata includes:

- Data structures that form the data administrator's view of the information
- Definitions of the source data from which the data warehouse database is built
- Specifications of the data transformations that occur as the source data is transferred to the data warehouse
- The data model of the data warehouse database (i.e., the data elements and their relationships)
- A record of when new data elements have been added to the data warehouse database and when old data elements have been removed
- Definitions of the views presented to data warehouse users; typically, multiple views are defined to suit the preferences of different groups of users

Data warehouse metadata is defined from external sources. In other words, the data structures created in the data warehouse model re-use the metadata defined in the operational sources. The actual data from the operational environment can be moved into the data warehouse database appropriately.

Because operational sources are heterogeneous, redundant, and inconsistent, they are imported into WarehouseArchitect via an extraction/transformation tool for reformatting. Alternatively, you can import external objects directly from operational sources when the source information is in a relational format (the only format natively supported by PowerDesigner), and when the number of external database systems is extremely limited.

Importing Source Information from an Extraction/Transformation Tool

In general, the heterogeneous information contained in operational databases must be extracted, scrubbed (or cleansed), and transformed by the extraction tool. Transformations include aggregation and summarization, as well as reformatting of the information to suit the target data warehouse backend.

The extraction tool output is imported to WarehouseArchitect and becomes directly accessible (and possibly aggregated) information (see Figure 2).

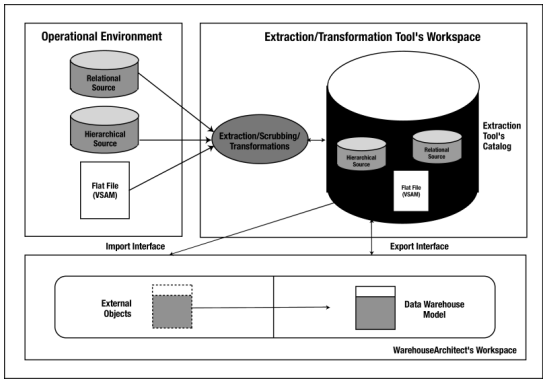


Figure 2: Extraction tool output is imported to WarehouseArchitect.

The WarehouseArchitect import feature allows you to select one or several operational (external) databases, which will be used as the source for your data warehouse model (see figure 3). Upon import, WarehouseArchitect allows you to select particular tables from each external database. The data type of each imported table column is converted into the appropriate corresponding data type of the target data warehouse database. Once imported, external tables are represented by dotted lines in your data warehouse model; the name of the source database system is in the external table title. Note that external tables are not generated; they are only used to marshal source metadata.

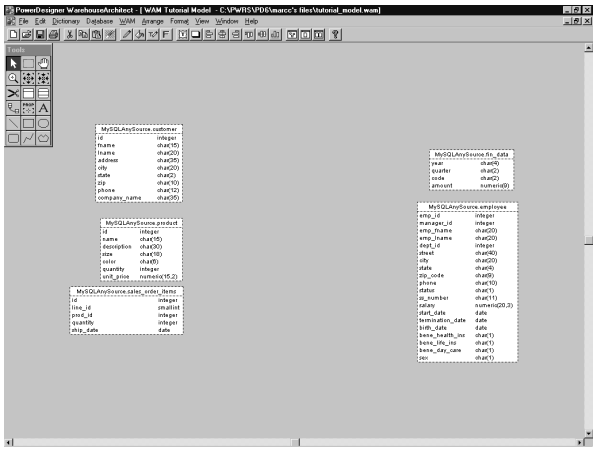


Figure 3: The WarehouseArchitect import feature allows you to select one or several operational (external) databases.

WarehouseArchitect allows you to automatically generate extraction expressions into an extraction file. The extraction file is sent to the extraction tool via the interface built between

WarehouseArchitect and that extraction tool. Each extraction tool interface is specific, in that they internally represent information in a unique format.

WarehouseArchitect includes traceability features that allow you to visualize the source tables and columns used to build the WAM. Traceability information is coded by extraction expressions for each table (SQL queries used to populate the data warehouse database). This information is key for extraction tools to transfer end-user data appropriately (more detail on traceability links later in this document).

Each time you modify the operational source structure, you need to reimport the new external table definitions. During import, external tables and columns are created or merged to define or update links between the operational system and the data warehouse. Once the external information is imported into WarehouseArchitect, you can begin designing the data warehouse multidimensional model.

Designing the Data Warehouse Model

The design of the data warehouse model largely depends on:

- Target Platform
  - *Mainstream RDBMS (e.g., Adaptive Server™ Enterprise)*
  - *Optimized RDBMS (e.g., Adaptive Server IQ)*
- Complexity of the data warehousing application (level of aggregation, number of variable values, etc.)
- Volume of the data to be handled by the data warehousing application
- Load frequency (i.e., how often the data warehouse data is refreshed)
- Number of users
- Hardware resources

These factors are a direct function of performance. As a result, appropriate data modeling and design strategies are key in optimizing the performance of the whole data warehouse environment.

The Dimensional Modeling Paradigm

The data warehouse backend database is designed to support the analysis and decision-support requirements for the whole data warehouse environment.

A multidimensional model is query-centric. It represents information in the same way the business end user utilizes that information. In contrast, a conventional (transactional) data model represents information the way it is to be implemented in the physical database, without considering how that information will be used by business end users. Table 2 represents these differences.

Table 2: Multidimensional data model compared to the conventional data model.

Multidimensional Data Model	Conventional (Transactional) Data Model
Entities are dimensions that define results over a given period of time.	Entities can be objects (e.g., customer, product, etc.) and transactions (e.g., order line item, sale line item, etc.).
Relationships are modeled implicitly. For example,the Product and Customer tables are not explicitly joined; they're related through a fact table.	Relationships are modeled explicitly. For example, the Product and Customer tables can be directly referenced (through their primary keys).
Analysis: For example, a typical informational query is "Why was there a downward sales trend in Europe over the first two quarters of last year?"	Audit trail: For example, a typical transactional query is "List every transaction over $5,000."

Thinking of the information in end-user terms allows the designer to optimize the data warehouse application according to both the target platform, and the complexity and volume of the information handled by the data warehouse or data mart.

In operational systems, transactions are small, involve few tables, and are often predefined and embedded in application programs or in database-resident code (stored procedures). As a result, the end user does not interact directly with the database definition.

In informational systems, queries can be very complex, involving many tables and joins. Unlike transactional queries, informational queries can be ad hoc (online) as well as predefined. When many tables are required to answer an informational query, the cube metaphor is used. Each side of the cube represents a dimension (or path) along which queries will be searched. The cube shows that several dimensions can be searched simultaneously and that each dimension can be detailed, depending on the level of query granularity.

Dimensions can include attribute hierarchies that facilitate and optimize query paths; this is known as "slicing and dicing" the dimensions, i.e., using the dimensions and dimension attributes relevant to a query. Appropriate (multidimensional) query tools allow you to "drill" into the informational system's backend database along those dimensions using drill paths defined by dimension attributes.

One of the advantages of informational systems is their ability to use selected attributes from several dimensions for a query. Sometimes, intermediate calculations (such as sub-totals) need to be stored (or aggregated) to be reused further in a query. The results to a query are returned from the information stored in fact tables.

WarehouseArchitect enables dimensional modeling using graphical objects that can be persistently stored and managed in PowerDesigner's MetaWorks™ dictionary including:

- Fact tables
- Dimension tables
- Tree hierarchies (facts, dimensions, and their respective attributes)
- Fact and dimension aggregation/summarization
- Fact partitioning
- Support for optimized indexing schemes
- In-place functions/macros

Defining Dimensional Objects in WarehouseArchitect

Fact Tables

Facts are variable values stored as numeric fields, which represent physical transactions occurring at a particular point in time. Examples of facts include Revenue and Expenses. Facts are recorded in normalized relational tables (fact tables) that contain multiple fact columns related by a compound key. A fact table is the central table in a star schema architecture.

Metrics

Metrics are analytical measures calculated from facts at runtime, and many metrics can be derived from just a few facts. Attributes represent conceptual metric qualifiers used for filtering and analyzing the information. For example, attributes of the metric "sales" may include city, state, store, quarter, year, category, and brand. Physical columns represent these attributes in the data warehouse database schema. In WarehouseArchitect, metrics are linked to a fact table column and can be the result of an operation involving several columns of the associated fact table.

Dimension Tables

Dimension tables are logical groupings of attributes with a common key relationship. Dimensional information is used to analyze the facts. In WarehouseArchitect, a dimension is linked to a denormalized relational table in a WAM (dimension table). A dimension table includes a single part primary key. A dimension can be broken down into a dimension hierarchy. Hierarchies are represented as (nested) subsets of dimensions and provide the navigation drill down, i.e., a hierarchy of dimensions shows the different levels of granularity available in the queries. In a snowflake schema, for example, the dimension hierarchy can be represented as links between dimension tables.

Attributes

An attribute is a field in a WAM dimension table. For example, Year is an attribute to the Date dimension. Attributes are also dimension qualifiers used in queries. In an alternative to the snowflake schema, WarehouseArchitect allows you to break down an attribute into an attribute hierarchy. Each attribute in the hierarchy corresponds with a column in the dimension table. The attribute hierarchy shows the different levels of granularity in the query for a dimension.

The Star Schema Design and its Derivatives

A star schema is a highly denormalized query-centric data model. In a star schema, facts are in a single place (the fact table) and the descriptions (or elements) that lead to those facts are in dimension tables. Dimensions store attributes for values in the fact table. Denormalization of a star schema has the direct effect of minimizing the number of joins and therefore increasing performance (one single fact table references several dimension tables).

The star schema design is based on direct paths (joins) between dimension tables and fact tables. Join processing is accelerated by indexes on fact table columns used as foreign key references to the dimension tables' primary keys.

The most important dimension in a star schema is Time. The Time dimension shows how values change over a given period of time, or how values between two periods of time can be compared. The Time dimension allows you to show time information in the data model itself, rather than in embedded database-resident code (time functions) or client applications. This is a more efficient way to design an informational database application because constraints (which are the crux of dimensions) help in defining the scope of time series.

An alternative to the star schema is the snowflake schema. Unlike a star schema, the dimensions in a snowflake schema are normalized—the result of splitting tables at the attribute level. Dimensions are "snowflaked" when denormalized star schema dimensions are too large (snowflake schema dimensions are low cardinality, i.e., there are fewer tuples in a table). WarehouseArchitect represents snowflaked dimensions in dimension hierarchies.

WarehouseArchitect lets you choose the best modeling solution for your data warehouse environment and lets you generate the appropriate data definition language (DDL) for the selected data warehouse database, using a wide range of dimensional modeling objects.

Representing Multidimensional Information in WarehouseArchitect

WarehouseArchitect allows you to break down facts into:

- Other facts (aggregation approach)
- Dimensions (star schema approach)
- Attributes (matrix approach)
- Metrics

Dimensions can be broken down into:

- Dimension subsets or hierarchies (snowflake schema approach)
- Attributes (mandatory in all dimensional models)

Modifications to a fact or a dimension are automatically propagated throughout the WarehouseArchitect model. Each object in a hierarchy can be created, deleted, renamed, moved, or copied using standard Windows commands.

Designing the WarehouseArchitect Model

Designing the data warehouse model begins with displaying the list of external columns and selecting those you need (see Figure 4). For example, you can start by creating a fact table in which you can add columns from imported external tables. You can then include metrics that become new members in the fact table. Metrics information may be obtained from operations carried out during extraction (either by extraction tools, or by scripts using SQL commands to select information and perform operations on that information—add, multiply, etc.). Dimension tables are defined in accordance with the source information you've imported into WarehouseArchitect.

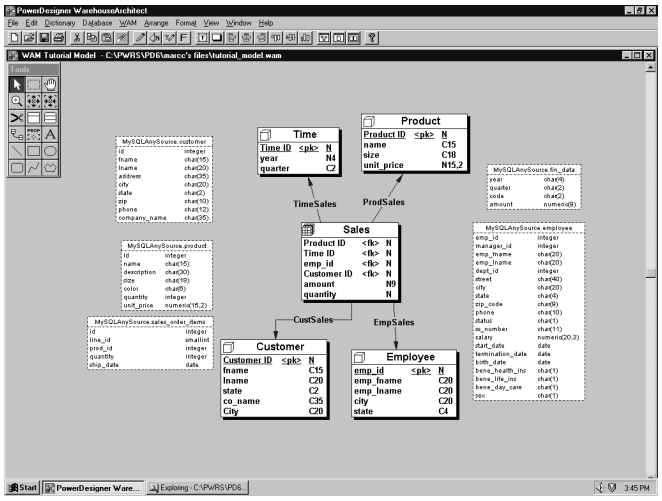


Figure 4: Users design the data warehouse model by displaying the list of external columns and selecting those they need.

Optimizing the Data Warehouse Model Design

Performance is key to using an informational system efficiently. WarehouseArchitect supports various performance improvement techniques:

- Denormalization—replacing an attribute column with multiple fact columns in the fact table, improving performance by reducing the number of joins required during query execution or reducing the number of rows to be retrieved.
- Summarization—creating partially redundant fact tables (aggregated facts). Frontend decision-support engines can then "re-path" queries to more aggregated fact tables at run-time. Aggregated (summarized) data is stored along frequently accessed hierarchies.
- Partitioning—segmenting the fact table into smaller parts. Multiple smaller fact tables are created into a fact hierarchy that stores atomic-level data.

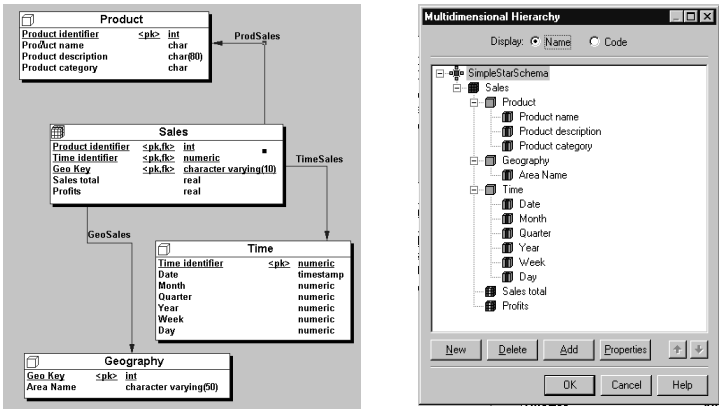


Figure 5: A star schema with its associated multidimensional hierarchy.

WarehouseArchiect's design artifacts are based on the use of a multidimensional hierarchy. With WarehouseArchitect you can map the data warehouse relational tables to a hierarchical structure that is used to implement partitioning, aggregation, and generation of data structures to frontend decision-support tools. WarehouseArchitect allows you to automatically build a multidimensional hierarchy from an existing dimensional data model (see Figure 5).

WarehouseArchitect's Aggregation Wizard

With WarehouseArchitect, both facts and dimensions can be aggregated. A wizard generates a new fact table referencing the selected dimension tables, letting you automatically create aggregation tables. Fact and dimension aggregation makes computation-intensive data warehouse applications more efficient. Aggregation optimizes queries by summarizing fact table rows along a specific dimension. Aggregation is very efficient for data models with complex dimensional structures that involve many joins between dimensions.

When you aggregate data, you summarize it along frequently accessed query paths, storing the data in a single aggregated fact table. The aggregated fact table and the original fact table form a fact hierarchy. For example, to create an aggregated fact table along the Time dimension, you select the Time dimension attributes that you want to keep (e.g., Year). The result of this aggregation is the Sales-Per-Year aggregated fact table and a new Time dimension, including the aggregated attributes.

Note: Aggregation in a WarehouseArchitect snowflake schema creates a new aggregated fact table attached to the parent fact table and to the relevant dimension in a dimensional hierarchy.

WarehouseArchitect's Partitioning Wizard

Partitioning optimizes queries by creating new fact tables in a fact hierarchy, according to one or more dimensions, for particular attribute values for dimension attributes. Partitioning is used when you want to focus a query on very detailed information (atomic data).

WarehouseArchitect's partitioning wizard first invites you to choose a fact table to be partitioned, and then to select the attributes of one or more dimensions to define the partitioning criteria. For example, you can select the Year attribute in the Time dimension and the Category attribute in the Product dimension and give labels (and values) to each (for example, 1996 and 1997 for the Year, and Deluxe and Standard for the Category). The net result is the creation of fact tables that store results along the selected dimension attributes. The number of partitioned tables you create depends on the number of values you specify for each dimension attribute. In this example, the following four fact tables would be created: Sales Deluxe 1996, Sales Deluxe 1997, Sales Standard 1996, and Sales Standard 1997.

The advantage of partitioning is that facts are very focused. This allows users to avoid traversing many rows to find the relevant information for the query. Fact partitioning creates a lot of graphical information in the data warehouse schema, and WarehouseArchitect allows you to select only the objects that you want to display.

Data Warehouse Backend Target Platforms

WarehouseArchitect invites you to select your backend target platform when you create a new data warehouse application. You can choose among the market-leading relational data-

base management systems, such as Sybase's Adaptive Server Enterprise, or you can choose database management systems optimized for data warehousing, such as Sybase's Adaptive Server IQ.

A server-based product, Adaptive Server IQ creates specialized indexes on either local or remote data sources, including flat files. Adaptive Server IQ then uses these indexes to resolve complex decision-support queries rapidly without having to refer back to the original data source. Adaptive Server IQ resides on the server between a frontend tool and an Adaptive Server database. You can also use Adaptive Server IQ to speed up query resolution without directly populating an Adaptive Server database.

Adaptive Server IQ enables optimized indexing against other data sources. The indexes for each database table are stored within Adaptive Server IQ indexsets. The choice of index depends on the cardinality and uniqueness of the data to be queried. Once you've built your indexes on the data, you can resolve standard database queries through the Adaptive Server IQ indexes.

WarehouseArchitect allows you to define the IQ indexes (FastProjection, LowFast, HighGroup, HighNonGroup) graphically through a dialog box, as shown in Figure 6. You define the type of index to generate for each column in an indexset, and you can define more than one index for a column when the column is used in different contexts.

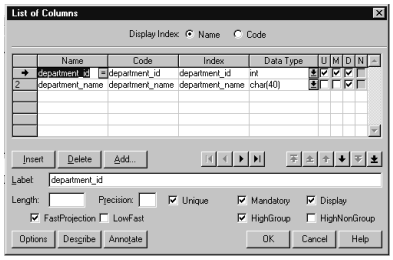


Figure 6 : WarehouseArchitect allows you to define indexes graphically through a dialog box.

Adaptive Server IQ Joined Indexsets

A joined indexset is an internal structure created when you predefine joins between tables with Adaptive Server IQ indexes on them. Joined indexsets let you prejoin tables, ahead of time—when you know which tables are necessary to carry out complex queries, improving query performance.

Join Relationships

When you create a joined indexset, you must specify the relationship between each pair of tables in the join as either:

- ONE>>ONE (e.g., a sales rep to a region)
- ONE>>MANY (e.g., a sales rep to several customers, each of which has only one sales rep)
- MANY>>ONE (e.g., several customers to a sales rep)

All join relationships supported in Adaptive Server IQ must have a hierarchy with one table at the top.

WarehouseArchitect handles join relationships automatically and seamlessly (see Figure 7). Simple joined indexsets are actually analogous to a star schema. A joined indexset that has two indexsets referenced to a top indexset is similar to two dimension tables and a fact table.

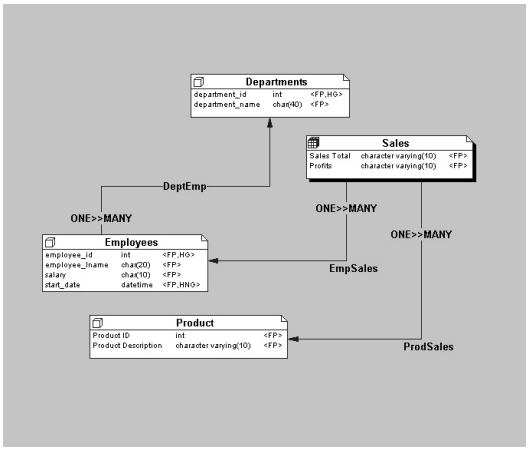


Figure 7: WarehouseArchitect handles join relationships automatically and seamlessly.

Populating the Data Warehouse

WarehouseArchitect stores information on how to move data from the operational environment into the data warehouse. Data may be moved from the operational database(s) into the data warehouse backend database by an extraction tool (as described earlier), or directly through an extraction script generated by WarehouseArchitect.

WarehouseArchitect provides interfaces with market-leading extraction tools. Supported extraction tools read a file (or table) that tells them where they can get the data, when they can get that data, and where they need to transfer that data. In other words, WarehouseArchitect provides a roadmap for extraction tools.

Using WarehouseArchitect, you can display the links between source information and the data warehouse model's objects. WarehouseArchitect displays that information in dual panes, with destination on one side and source on the other—as shown in Figure 8. This information is key for extraction tools to transfer end-user data appropriately.

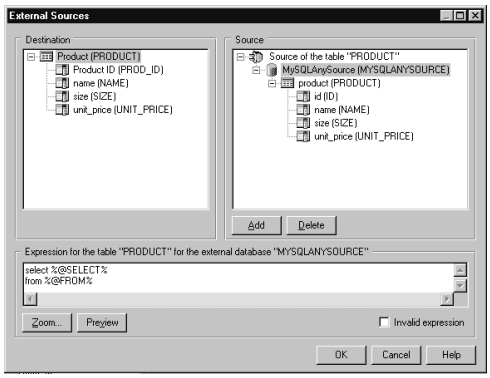


Figure 8: WarehouseArchitect displays information in dual panes, with destination on one side and source on the other.

Interfacing with Frontend Decision-Support Tools

So far, we've seen how WarehouseArchitect imports source information to design the data model (WAM) of the data warehouse backend database.

WarehouseArchitect allows you to transfer the data warehouse model information to front-end decision-support tools through data-generation interfaces (see Figure 9). This allows you to automatically build and generate structures(cubes, built and generated with wizards) and programs for market-leading frontend decision-support tools, such as desktop-and server-based OLAP environments, as well as more traditional managed-query tools.

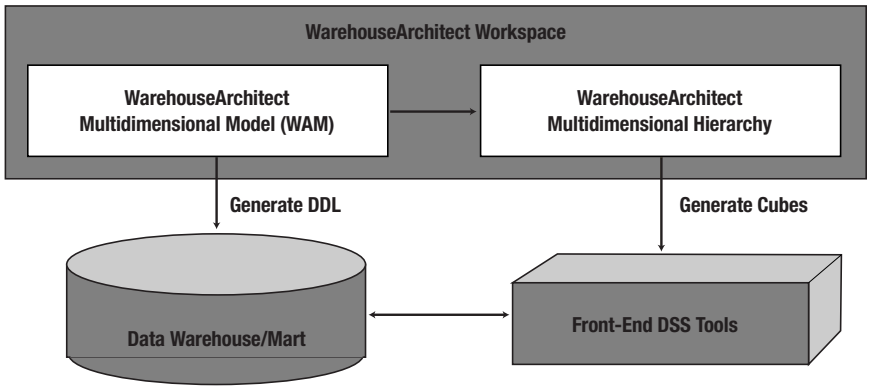


Figure 9: WarehouseArchitect allows you to transfer the data warehouse model information to frontend decision-support tools through data-generation interfaces.

Interfaces between WarehouseArchitect and frontend decision-support tools are invoked directly from WarehouseArchitect. Each interface is specific to the target decision-support environment. The WarehouseArchitect-generated information is directly reusable in the decision-support environment by simply invoking the appropriate file.

Generating WarehouseArchitect Reports

WarehouseArchitect generates customizable reports that include all the objects specific to multidimensional models:

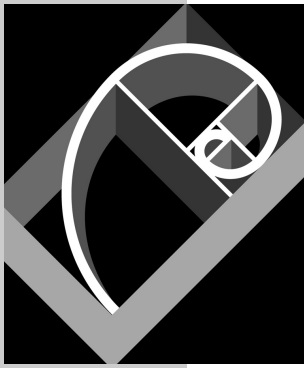
- List of external tables
- List of external columns
- List of facts (and fact hierarchies)
- List of dimensions (and dimension hierarchies)
- List of attributes (and attribute hierarchies)
- List of matrices
- Tree structures
- Links between source information and data warehouse destination tables

Objects such as tree structures are represented graphically in WarehouseArchitect reports.

Conclusion

Data design is key to the success of a data warehouse project. WarehouseArchitect supports every step of the data warehouse environment design cycle by allowing you to:

- Import source metadata into the WarehouseArchitect workspace
- Reuse the source metadata to design the data warehouse backend database model (support for market-leading RDBMS optimized for decision support)
- Interface with extraction/transformation tools for efficient data movement
- Generate data cubes from the data warehouse model for market-leading, frontend, decision-support tools
- Produce customized reports on every step of the design process



Sybase, Inc., Worldwide Headquarters, 6475 Christie Avenue, Emeryville, CA 94608 U.S.A.
Phone: 1-800-8-SYBASE (in U.S. and Canada); Fax: 1-510-922-3210.

World Wide Web http://www.sybase.com

L00451