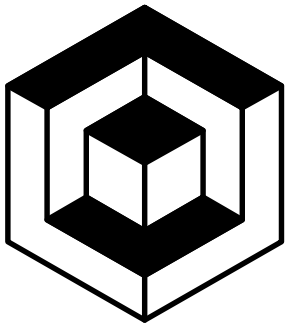
Getting Started with ADAPT™

OLAP Database Design



Symmetry Corporation

250 Tiburon Boulevard
San Rafael, CA 94901-5244
Phone: (415) 453-7966
Fax: (415) 453-8043
www.symcorp.com

Getting Started with ADAPT

In this white paper we discuss logical vs. physical data modeling, review currently available modeling techniques and why they are not appropriate for OLAP, and introduce the basic elements of ADAPT™ (Application Design for Analytical Processing Technologies). We explain each of the nine ADAPT database objects and their symbols and illustrate how to use the symbols with simple examples.

A) Logical vs. Physical Modeling

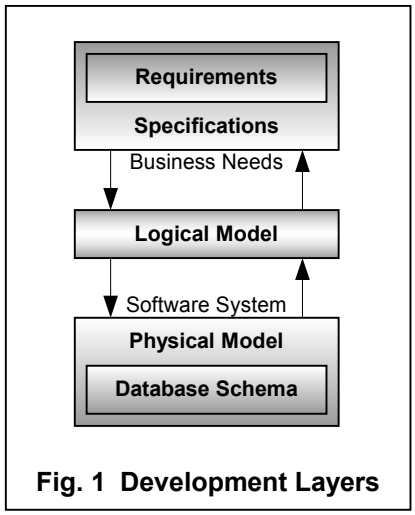


Fig. 1 Development Layers

The goal of a development project is to translate business needs into a working software system. Logical modeling acts as a translation layer (fig. 1). Application requirements become specifications. The specifications are used to construct a logical model, which is then validated against the requirements. Once the logical model is completed, work on the physical implementation begins. A physical model interprets the logical model based on the strengths and weaknesses of the chosen software and hardware platforms. The physical model is also used to create the database schema. Logical modeling can be thought of as a transformer that converts business needs into technical implementations.

Tight deadlines and easy to use software can tempt developers to forego logical modeling and go straight from the application description directly to the physical implementation. Bypassing logical modeling can result in an application that reflects the functionality of the chosen software, instead of the functioning of the business. In a perfect world, logical modeling would take place before the physical architecture is chosen. The reality is that hardware and software have usually been chosen before the project begins. Even so, OLAP systems are not so simple and OLAP software is not so good that rigor should be discarded in favor of expediency. Logical modeling helps ensure a focus on solving the business needs, regardless of the advantages or shortcomings of a particular implementation platform.

B) Current techniques

Before we get into details about the ADAPT modeling methodology, let's briefly examine three major application categories (transaction processing, data warehouses, and OLAP data marts) and their applicable modeling techniques.

The purpose of a transaction processing system is to support a business process. To model a transaction processing system, we need to examine the workflow, which tells us which data needs to be grouped together and processed as a unit. The data model for transaction processing is relational tables. Because transactions must be executed as quickly and efficiently as possible, data redundancy must be reduced as much as possible. The modeling technique of choice for transaction processing systems is entity-relationship (ER) modeling.

Unlike operational systems, data warehouses are not created to support a business process. Their purpose is the collection of data from the transaction processing systems into a single coherent database. In a data warehouse, analysis consists of examining the available data and determining the best way to organize it. The data warehouse data model is also relational tables, but with a slightly different slant from transaction processing. Manageability and accessibility of data are the criteria for success rather than transaction processing speed. The modeling technique of choice for data warehouses is dimensional modeling.

OLAP data marts provide an analytical environment. Their purpose is to solve business problems. Instead of starting with the available data to perform analysis, OLAP analysis begins with an examination of the business problem and works down to the data necessary to solve that business problem. The data model for OLAP is multidimensional cubes. An OLAP data mart needs to react quickly to changing business requirements.

Unfortunately, up to now there has been no modeling technique specifically designed to address the unique needs of OLAP database design. Without success, developers have tried to use the existing data modeling techniques of ER and dimensional modeling to design OLAP systems. With ER, designers model entities (tables), attributes (columns), and relationships (foreign keys) – all elements of relational databases. However, ER provides us with no good way of modeling hypercubes – the basic building blocks of OLAP databases. In ER, all entities are created equal and relationships define the associations between entities. Consequently, it can be difficult to distinguish the business data from the reference data in an ER diagram.

Unlike ER, dimensional modeling divides databases into two types of objects, fact and dimension tables. Dimension tables are de-normalized. This organization of data makes a database more understandable from a business perspective and allows a relational DBMS to execute large scanning queries (typical in a data warehouse) more efficiently. Dimensional modeling does a better job than ER of giving the model a context, but it does not go far enough for OLAP data modeling. For example, a dimensional model contains no information as to the precedence of level within a hierarchy. This lack of context has the undesirable impact of making it harder to validate the model. The result? Development is more difficult and the overall quality of the final system is reduced. Without context the meaning is unclear at best.

The most significant technical task of an OLAP system is to function as a calculation engine that creates derived data. Neither ER nor dimensional modeling has a way of representing the derivation of computed data. For the most part, computed data is not modeled because the derivation of data is generally considered a process function rather than a database function. At this point, it is important to segue briefly into a discussion clarifying the rationale behind this point of view. Database operations can be divided into the data definition language (DDL) and the data manipulation language (DML). Until now, data modeling has concerned itself only with the DDL operations. DML operations have been modeled using process modeling techniques, generally data flow diagrams (DFD). DFD is procedural in its approach and is used to define the programming modules required in an application.

The line between data and processing is blurring. Software is getting more sophisticated in how it treats analytical calculations. More and more of the calculation logic is embedded in the

database and the software itself handles applying that logic to the appropriate data. Particularly for designing OLAP data marts, the time is right for a new approach to data modeling that incorporates both data and process.

In addition to a lack of context and no support for derived data, both ER and dimensional modeling are too closely aligned with the physical model. Although physical modeling certainly has its place, modeling directly to a physical model skips a crucial step. There is a modeling step, logical modeling, that should take place between the application definition and implementation.

Modeling directly to a physical structure forces the designer to think about how to solve the problem before necessarily understanding the problem. For example, a designer has many different alternatives for physically implementing a hierarchy, among them are the star schema, snowflake schema, self-reference, indirect reference, and descendant designs. Which design should be used depends on the characteristics of the hierarchy and how the application software operates. The point is that a designer should have a logical understanding of how the hierarchy represents a business problem before modeling the physical implementation.

Lastly, because ER and dimensional modeling are so closely aligned with a physical implementation that reflects a relational database approach, they are no help if the implementation software is a multidimensional database (MDB). The data structures in an MDB are different than those of an RDBMS. With the growth in MDBs and hybrid OLAP systems that use a combination of relational and multidimensional software, a modeling technique that is wired to a specific physical implementation adds very little value.

The need to address the unique concerns of OLAP data modeling and the vacuum of modeling techniques for OLAP led to the creation of the ADAPT modeling technique for designing OLAP databases.

C) Core Objects

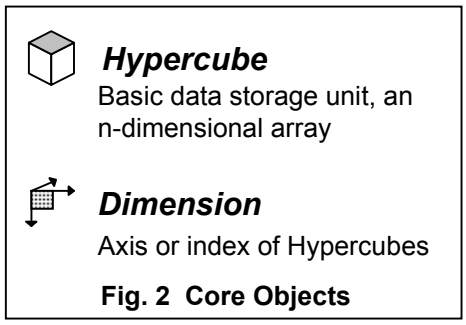


Fig. 2 Core Objects

The basic building blocks of ADAPT are hypercubes and dimensions, the core objects of the OLAP multidimensional data model. The symbols used for these objects are shown in fig. 2. In programming languages such as BASIC and "C", a data structure with no dimensions is called a scalar; with two dimensions, a matrix; and with three dimensions, a cube. In an OLAP database, a hypercube is the basic unit of storage for business data, an array with zero to many dimensions.

Because the term "hypercube" is considered too technical for marketing literature, some software vendors use either the term "measure" or "fact" instead of hypercube when referring to the business data in an OLAP database. To eliminate confusion in cases where a database contains a measure dimension, ADAPT uses the term hypercube, or "cube" for short. Likewise ADAPT avoids using the term "fact" because it is generally associated with fact tables, a term used when describing the physical implementation of a database.

It is important to clarify that the rules of array manipulation that govern programming languages such as BASIC and "C" do not apply to OLAP data structures. OLAP does not use the term “array” to reference its data structure because of significant differences in the rules governing the definitions of arrays and OLAP multidimensional data structures. In programming terms, two different three-dimensional arrays are structurally equivalent. In OLAP terms, two different three-dimensional data structures are only structurally equivalent if their dimensions are the same. For example, sales as defined by the dimensions time, store, and product are structurally equivalent to units by time, store, and product. Sales by time, store, and product are not structurally equivalent to financial actuals by time, unit, and account.

Arrays are indexed by subscripts, whereas OLAP hypercubes are indexed by dimensions. Subscripts are similar to but not the same as dimensions. A subscript is a positional index into an array: x[5] yields the fifth element of the vector x. A dimension is a named index into an OLAP hypercube, e.g., the sales data for the region west (where sales data is in the sales hypercube and the region west is a geography dimension member). In an OLAP system, you would never refer to the fifth element of sales.

Now that we’ve defined our core objects, let’s begin designing an application. The first step in designing an OLAP database is to determine the central hypercubes needed in the application and their dimensions. Consider a simple sales example for Acme Toy Company. There are three basic cubes: units, sales, and list price. Units and sales are dimensioned by time, product, and customer. List price is by time and product. The application has three cubes and three dimensions. The ADAPT diagram is shown in fig. 3.

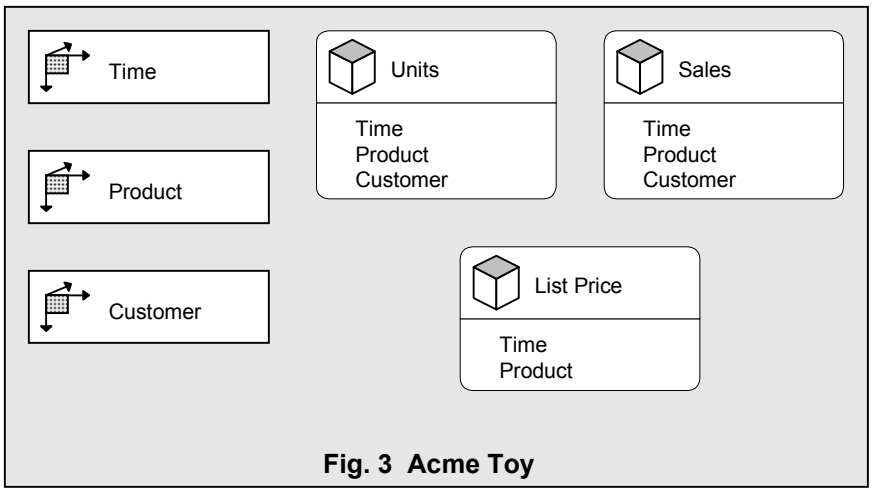


Fig. 3 Acme Toy

D) Hierarchies

At this point we know nothing about how the dimensions use the data in Acme Toy’s system. The most basic operation performed on data in an OLAP database is to aggregate it up a hierarchy. Hierarchies are generally made up of levels. In fig. 4 you can see the ADAPT symbols for hierarchy and level. Let's say we want to aggregate data along the time dimension using a standard calendar (fig. 5).

Hierarchy

Set of parent/child member combinations that define aggregation

Level

Collection of members used to define hierarchical precedence

Fig. 4 Dimensional Aggregation

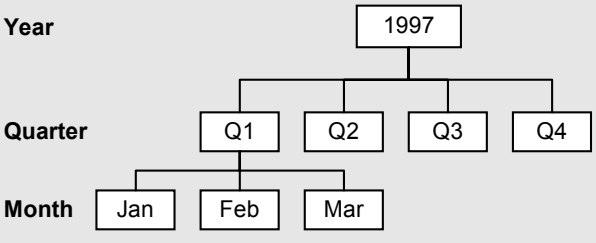


Fig. 5 Calendar Hierarchy

To draw this using ADAPT, we show a calendar hierarchy off the time dimension with year, quarter, and month levels within the hierarchy (fig. 6). We use a line with an arrow to show the calendar hierarchy is of the time dimension. The connectors between the levels show the precedence of the levels -- months roll up into quarters that roll up into years. Years are the highest level in the calendar hierarchy.

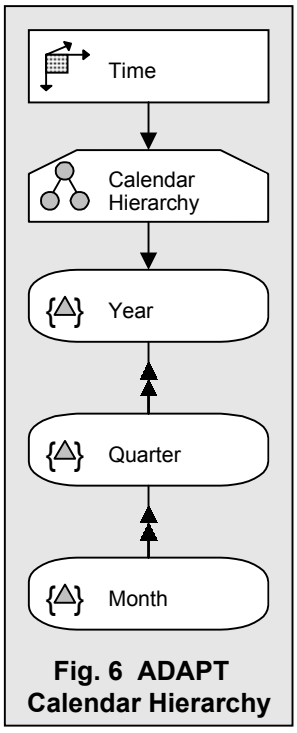


Fig. 6 ADAPT Calendar Hierarchy

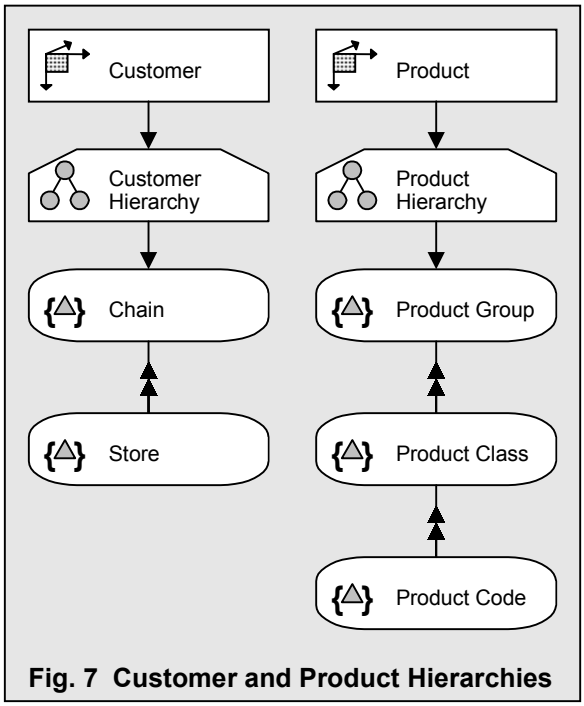


Fig. 7 Customer and Product Hierarchies

Acme Toy's customers are retail chain stores. Each product is assigned a product code and belongs to a product class (e.g., product code 111 belongs to a product class called baby dolls). Product classes belong to groups (e.g., baby dolls belong to the doll group). The ADAPT hierarchies for Acme Toy's customer and product dimensions are shown in fig. 7.

Now we need to show how these hierarchies are applied to the cubes. The units and sales cubes can be aggregated along all three dimensions. Fig. 8 shows how the units and sales cubes are aggregated. The price cube, on the other hand, cannot be aggregated along either the time or the product dimension. Assume the price of a product was $10.00 from January through August, and $11.00 from September through December. How do we determine the price of that product for the year? Different companies use different business rules to determine the prices they use in OLAP applications. The most straightforward approach is to say that the price for the year is equal to the price in December. It is equally reasonable to use an average, in this case calculating a price of $10.33.

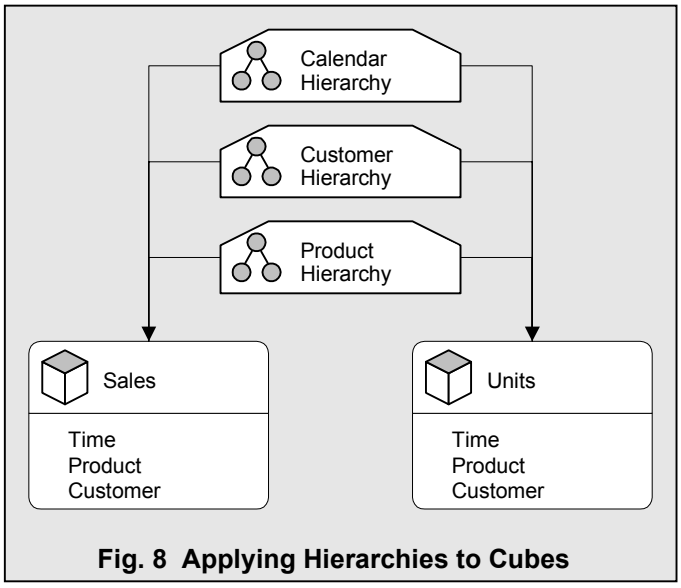


Fig. 8 Applying Hierarchies to Cubes

Reviewing a database design diagram that highlights an application's essentials often leads to the realization that information is missing. In our case, as we review the time dimension, we are reminded of the difference between a calendar and a fiscal hierarchy. Our initial design included only a calendar hierarchy, but the company's fiscal calendar begins in February (fig. 9).

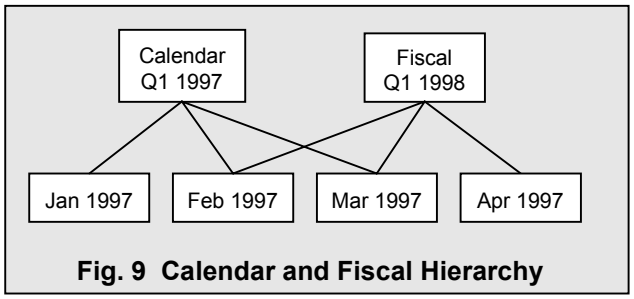


Fig. 9 Calendar and Fiscal Hierarchy

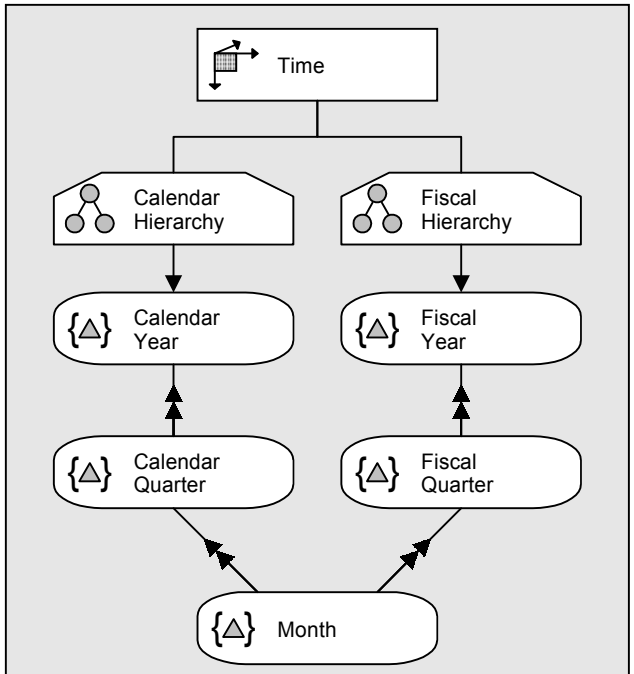


Fig. 10 ADAPT Calendar and Fiscal Hierarchy

We need to add a second hierarchy, a fiscal hierarchy, to the time dimension. The fiscal hierarchy also made up of months, quarters, and years. The quarters and years in the fiscal hierarchy are different from those in the calendar hierarchy. So that there is no confusion, the quarters and years in the calendar hierarchy are renamed to calendar quarter and calendar year, respectively. A fiscal hierarchy is added that includes levels of fiscal quarter and fiscal year. Notice that both hierarchies (fig. 10) share the month level, since a month is the same whether it is part of a calendar or a fiscal hierarchy.

Looking back at the diagrams we have created so far, we can see that we are starting to gain a better understanding of our sales application. We know about the cubes and dimensions in the application; the hierarchies of each of the dimensions (including the fact that the time dimension has two hierarchies, calendar and fiscal); the levels within the hierarchies; and the precedence of those levels. We see that sales and units are calculated as aggregates and equally important, we are now aware that we don't know which calculations to apply to list price. In just a few diagrams, we have captured the essence of the application in a logical description.

Whether the final system is implemented using an RDBMS, an MDB, or any other technology is immaterial to the logical design of the system. We have created a document that can be discussed and validated, thereby promoting communication. Discovering that the calendar hierarchy is not the only time hierarchy for the application has improved the correctness of the model. Adding a fiscal hierarchy to the time dimension has improved the completeness of the model. The investigation into calculations to apply to list price will create a model that is consistent. By creating ADAPT diagrams, we have realized the benefits of logical data modeling: completeness, correctness, consistency, and communication.

E) Dimension Objects

In order to flesh out the design of our application, we will need the following additional ADAPT symbols: member, attribute and scope (fig. 11). Members, attributes, and scopes are database objects that are associated with a dimension and are used to describe a dimension more fully.

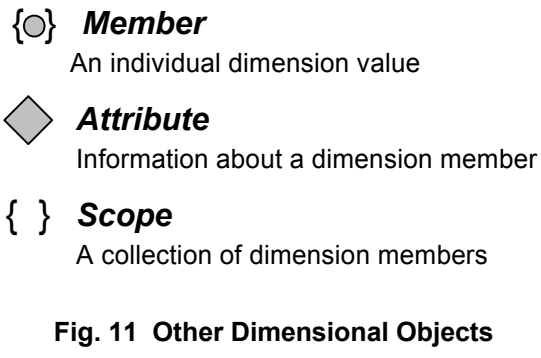


Fig. 11 Other Dimensional Objects

A member is an individual dimension value. January, 1998 and quarter one of 1998 are examples of members of a time dimension. California, Oregon, and northwest are members of a geography dimension. When modeling an OLAP database, you do not need to model all of the members of every dimension – only those important to understanding the design. A product or customer dimension can easily have hundreds of thousands of members. Modeling all of them is neither useful nor practical. However, there are some cases where there are particular calculations that apply only to certain members or where showing some members increases the understandability of the diagram. Assume that a U.S. company with global distribution uses different business metrics for its two territories, North America and international. In this example, for the geography dimension it will be important to show the territory members, so that the differences in calculations between territories can be highlighted.

A scope is a collection of dimension members, such as new products or the current year-to-date months. Scopes allow us to create a subset of database information, particularly useful in defining calculations, since a calculation does not necessarily apply to all the data in a cube. A product forecasting system might forecast new products differently than existing products. A financial system will need to know the year-to-date months and the forecast months to create a projected total for the current year. Scopes can be either enumerated or derived. The members of an enumerated scope are explicitly listed and are either manually managed or provided through an external source. The list of internal services departments in an organization dimension is an example of an enumerated scope. The members of a derived scope can be computed from other database objects. For example, the current year-to-date months can be derived from the current month. Just as with members of a dimension, it is not necessary or desirable to model all the members of a scope.

An attribute is descriptive information about the members of a dimension. An attribute can be of any data type. For example, the manager of each organizational entity is a textual attribute and the number of workdays in each time period is a numeric attribute. A constrained attribute is a special type of attribute whose values are constrained to a list of values stored in a dimension. For example, soft drink containers might be constrained to either 16-ounce cans or 32-ounce bottles.

Now let's apply these new concepts to the Acme Toy sales analysis application.

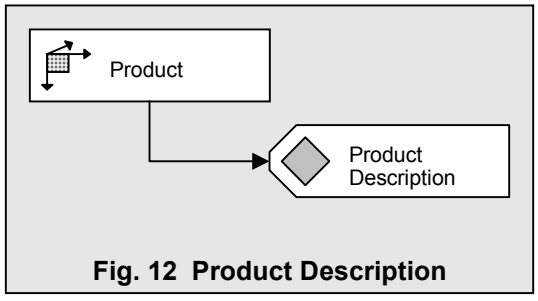


Fig. 12 Product Description

F) Dimension Attributes: Sample Case

Let's check Acme's dimensions to see what kinds of attributes are needed. Generally every dimension has a description attribute. We can immediately add a product description. Since every product will have a description, the attribute is hung off the dimension itself (fig. 12).

There is another product attribute, the Universal Product Code (UPC). The UPCs are not the same as the product codes assigned by Acme for internal use and must be included in the database for look up and selection purposes. Unlike the description attribute, the UPC attribute

is only valid for product codes and not for product groups or classes. To illustrate this business rule, the UPC attribute is hung off the product code level in the hierarchy (fig. 13) rather than the product dimension itself like the description.

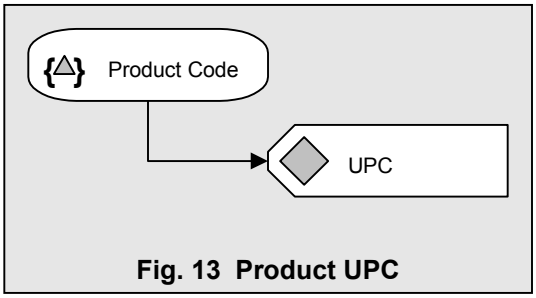


Fig. 13 Product UPC

Acme Toy is interested in other product attributes. Which gender the toy appeals to is an important product attribute. (Parents may try to be gender neutral, but toy companies do not…) As far as Acme is concerned, products appeal to either girls or boys or both. Since these are the only three values that gender can have, this is a constrained attribute. To represent this, we first create a new dimension called gender. "Girls", "Boys", and "Both" are the members of the gender dimension. We then create a product gender attribute (fig. 14) that is an attribute of both product and gender. This enables us to communicate that not only is gender associated with a product, but also there are products that appeal to a specific gender.

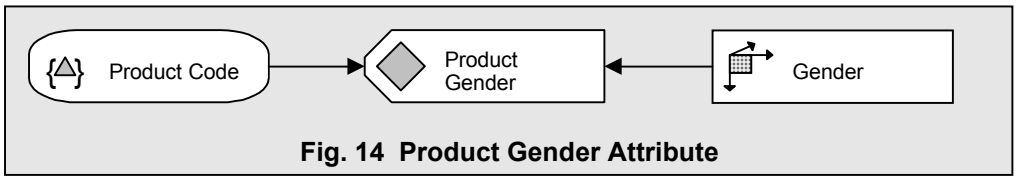


Fig. 14 Product Gender Attribute

Each product Acme Toy sells has a recommended age group: under a year old, 1 to 3 years, and so on. While the age group attribute is very similar to the gender attribute, there is an added twist. The age groups are part of a hierarchy that combine the age groups into age categories, such as

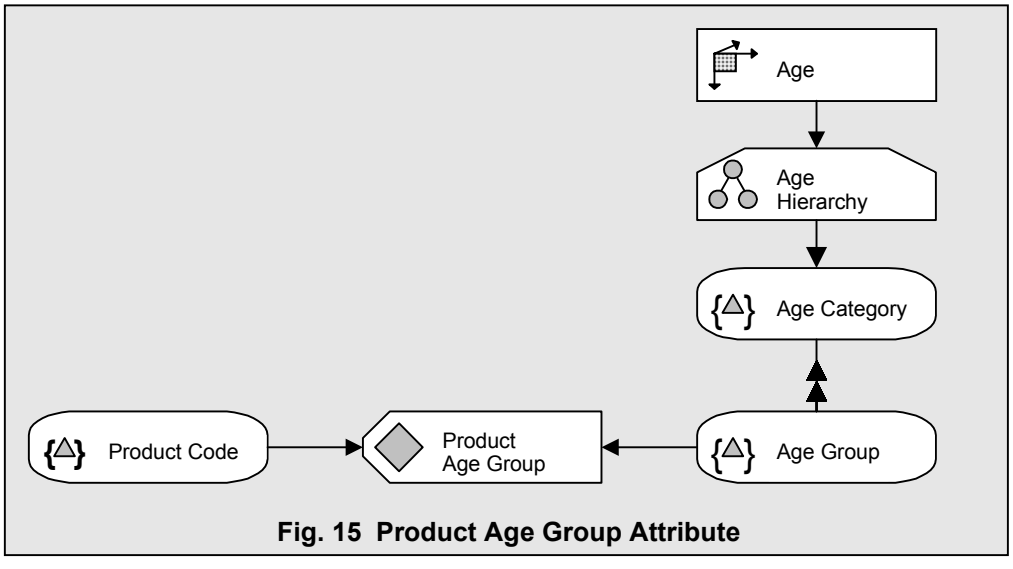


Fig. 15 Product Age Group Attribute

infant, child, teen, and adult. To model the age group attribute correctly, we need to create an age dimension with a hierarchy. There are two levels to the hierarchy: age category and age group. The product age group attribute hangs off of product code. Unlike product gender attribute that related product code to the gender dimension, the product age group attribute relates product code to the level age group (fig. 15). By creating a constrained attribute between product codes and age group, we can illustrate the relationship between the base dimension, product, and its property dimension, age.

False assumptions can ruin a database design and its subsequent implementation. Understanding system constraints is key to clarifying assumptions. The ability to show constraints clearly and graphically so they can be easily validated is a real strength of ADAPT. Throughout the examples in this white paper you will see how ADAPT presents the database objects in their proper context. Showing explicitly how specific objects relate to other objects gives the design the richest information content, minimizing misunderstandings about the system's function.

G) Dimension Scopes: Sample Case

Acme Toys wants to track some very interesting collections of dimension members. Like many other companies, Acme has a couple of flagship product classes. These are products that always generate the lion's share of sales. Barbie is a flagship product for Mattel. 501 Jeans are a flagship product for Levi Strauss. Acme's product marketing group evaluates sales in a completely different manner for flagship products. The flagship products do not have any special designation in the product hierarchy. To track them in the system, we will need to maintain a list of the so designated products. Dimension scopes are tailor-made to represent this type of database object. We create a scope off the product class level (fig. 16). Note that we do not model exactly which classes are flagship classes, only the fact that there is a special class of products called flagship.

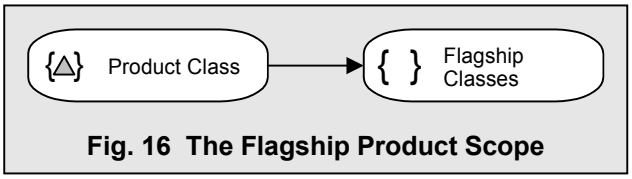


Fig. 16 The Flagship Product Scope

Acme Toys does not treat all of its retailers the same. Some retailers have joined a special program sponsored by Acme Toys and are treated as strategic partners. The product marketing group wants to track the retailers in this program to see if the program is paying off. To model this business requirement, we need to create a scope off the retailer level of the customer hierarchy (fig. 17).

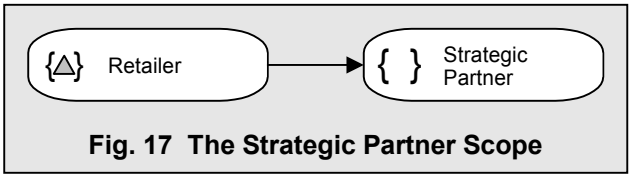


Fig. 17 The Strategic Partner Scope

Acme Toy wants to use its marketing resources to their best advantage. There is an internal debate as to whether specialty stores or super-stores are a better channel for its products. Some representative stores in key markets have been flagged as stores that need to be tracked so that this issue can be analyzed. Fig. 18 displays the ADAPT diagram. What is interesting to note in this diagram is that the tracked stores scope has its own scopes, specialty stores and super-stores. We are showing that the tracked stores are either specialty stores or super-stores. By modeling

this dependency, we have modeled a business rule. The ability to clearly show these types of business rules is a powerful aspect of ADAPT.

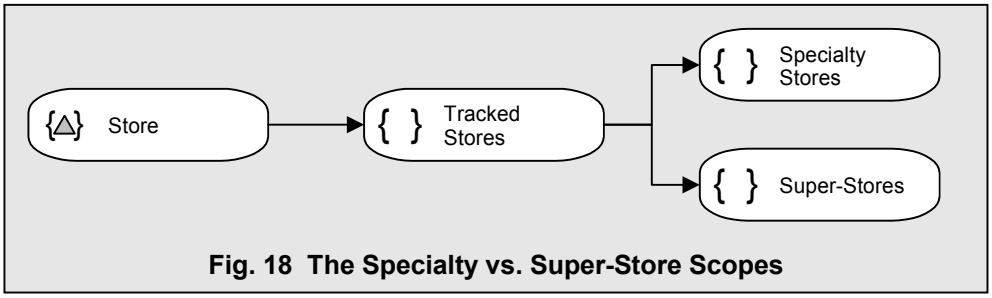


Fig. 18 The Specialty vs. Super-Store Scopes

Some of the products Acme sells are considered core products. The core products have other products associated with them that are accessory products. For example, a doll might be the core product and the dresses and shoes for the doll, accessory products. Or a racecar set might be the core product and other cars and scenery, the accessory products. Customers will not buy an accessory product without first purchasing the core product. It is very important that the brand managers at Acme Toy be able to analyze the relationship between core and accessory products. To model this relationship, we create two scopes, core and accessory, off the product code level. We then create an attribute that relates them (fig. 19). Note the relationship between core and accessory products. It is a self-relation of product to product, but we have limited the products that are included. In this case, by increasing the information content of the drawing we make the business relationship clearer.

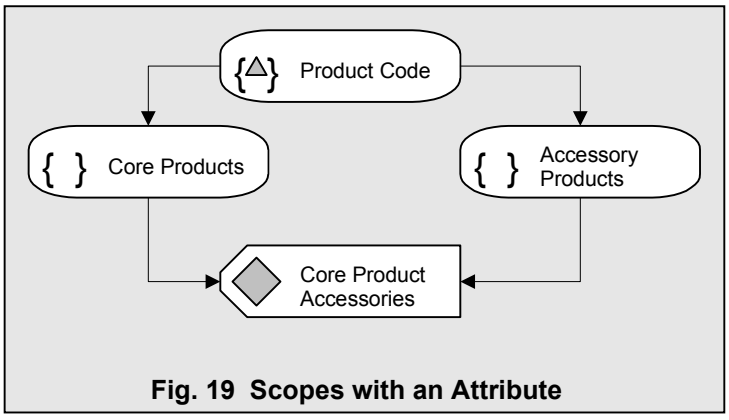


Fig. 19 Scopes with an Attribute

The time dimension always has a number of scopes that are important, and interestingly enough, the scopes tend to be common across applications. Virtually every OLAP application has a requirement to know current month, last month, and the same month last year (fig. 20). Current month scope would appear to contain only a single member, so why is current month a scope instead of a dimension member? Current month is not a distinct value in the time dimension because the value of current month changes every month. Current month is not a member of the time dimension -- it just references a member.

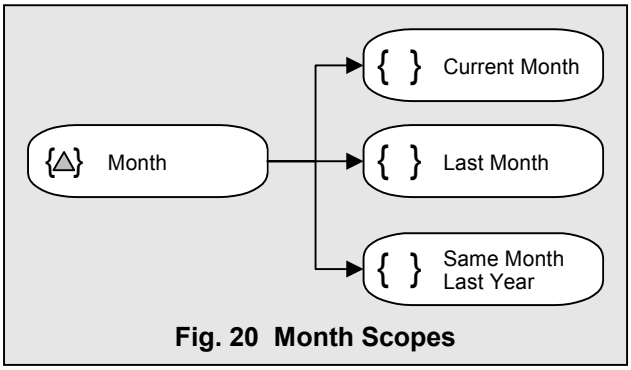


Fig. 20 Month Scopes

The last month and same month last year scopes are different from the other scopes we have reviewed. So far we have only looked at enumerated scopes, which are either manually maintained or are lists from external sources. Not only enumerated scopes but also derived scopes will be important to modeling the application. The last month and the same month last year are derived from the current month scope and the time hierarchy. Either the calendar or the fiscal hierarchy can be used -- both produce the same result. (The same month last year is identical regardless of which hierarchy is used.)

H) Dimension Members: Sample Case

Modeling dimension members is where discretion is the better part of valor in ADAPT. You want to model enough to adequately describe the system without getting mired in detail. As a rule of thumb, for most applications the number of members that should be modeled for any given dimension is under ten and more likely three or four. Let's look at an example in our application to see where modeling dimension members is useful.

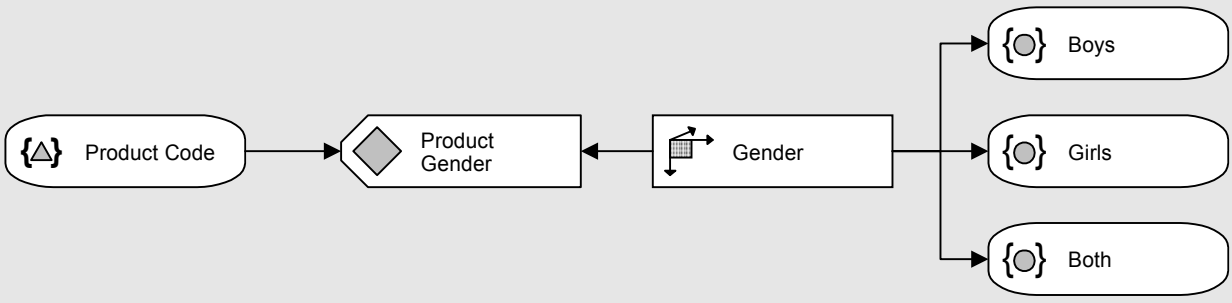


Fig. 21 Gender Dimension

Let's enhance our model of the product gender attribute (fig. 21). Notice in the drawing that we have chosen to model the members of the gender dimension. It is important in this case to show that there are three genders since we would normally assume there are only two. By modeling the dimension members we can ensure that we do not forget the gender that refers to both boys and girls.

We also need to model dimension members for the time dimension. A standard sales analysis compares this month with last month and this month with the same month last year. We need a place to reference these calculated results. To do this, we create two members of the time dimension, change from last month and change from last year. To further clarify their relationship with the time dimension, we create a scope of time variances and include the two new members within this scope (fig. 22). Creating a scope of time variances has the added advantage of grouping like objects together so that they are easier to spot.

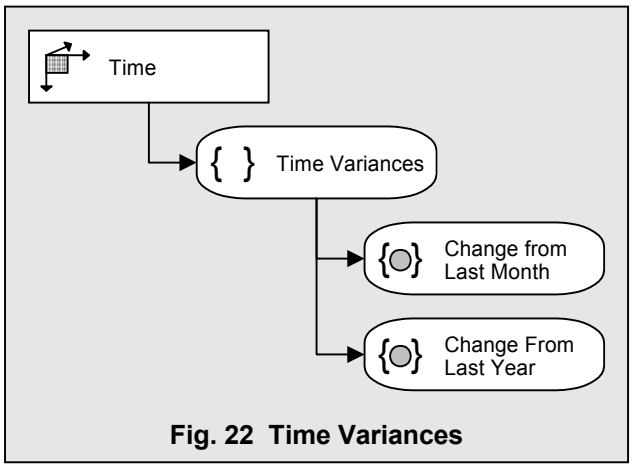


Fig. 22 Time Variances

I) Other ADAPT Objects

There are only two other ADAPT objects, model and context (fig. 23), completing the set of nine ADAPT database objects. Applying hierarchies to aggregate cubes is not the only type of calculation performed in an OLAP application. Very often we want to apply algebraic formulas to the data to perform calculations. We use the ADAPT model object to represent this. A model can represent a single formula, as in the case of computing variance between actuals and budget. A model can also represent a set of formulas executed together, as in a financial model.

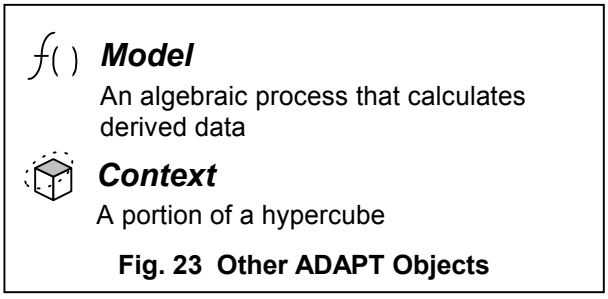


Fig. 23 Other ADAPT Objects

The purpose of using the model object is not to document the specific expressions used, but to document the source and target data for the calculation. While specifying derivation formulas in detail is critical to the application's final implementation process, it is not part of the database design. Detailing the specifics of the formulas as part of the database design can potentially lead to a quagmire of mathematical equations. Furthermore, overly specifying the formulas during the database design phase can result in physical implementation issues surfacing sooner than we would wish. During the database design phase it is important to focus on modeling the business metrics rather than how to implement the model.

A context is a portion of a cube that provides a context for analysis. This is a very powerful construct in ADAPT. We do not always use or compute an entire cube in a calculation. Many times we need to operate on only part of a cube. The use of the context object allows us to model an operation that applies only to a specific part of the cube. For example, the calculation of financial reserves for an organization's international units is likely to be different than a calculation for domestic units. The profitability of new stores is evaluated differently than mature stores.

Filtering a cube through one or many scopes creates contexts. A cube can be scoped along any of its dimensions. More than one scope can be used, even along a single dimension. Filtering a sales cube through the international scope creates a context of international sales. Further filtering the cube though the new products scope creates international new product sales. Filtering the cube still further though accessory products creates international new accessory product sales. The purpose of modeling contexts is not to model every possible query subset that users might request. The purpose is to model those subsets of data that have specialized processing. It is the exceptions that trip us up in applications development. Contexts help us recognize and model those exceptions.

J) Model Objects: Sample Case

All standard calculations should be shown in an ADAPT diagram, regardless of how trivial they might seem. Acme Toy wants to compute average selling price, a very simple calculation -- dollar sales divided by units sold. The reason for showing even such a simple calculation is that as application developers, we get into trouble by assuming we know what a business user meant. By showing the model (fig. 24), its inputs and calculation results, we avoid potential misunderstandings about the application's requirements and reduce the risk that the application will perform incorrectly.

The example in fig. 24 shows how to calculate the values in a cube from the values in other cubes. Sometimes we also want to calculate the values of one dimension member from other dimension members. Consider the time variances we have already created. These are computed values. To complete the diagram we should show the model that calculates these members (fig. 25).

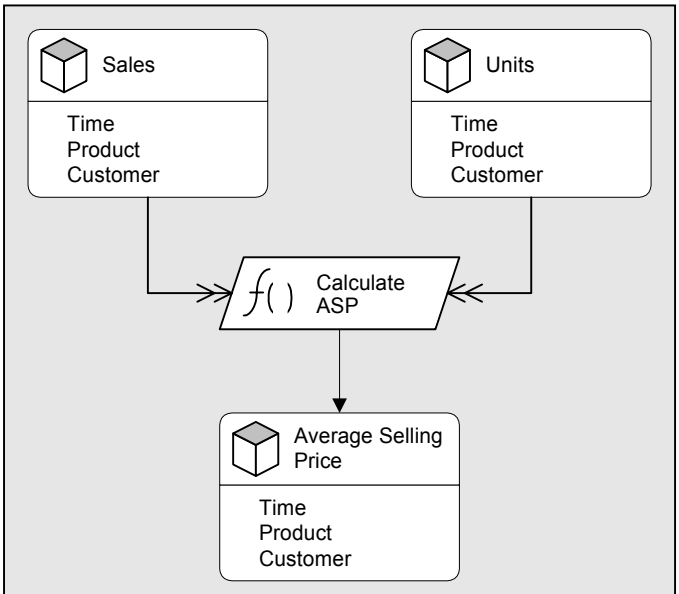


Fig. 24 Calculating Average Selling Price

By clearly showing the objects required and their dependencies, we further ensure that the application is complete and correct. Without the sales cube in fig. 24, average selling price could not be computed. Without the current month and last month scopes in fig. 25, variance could not be computed.

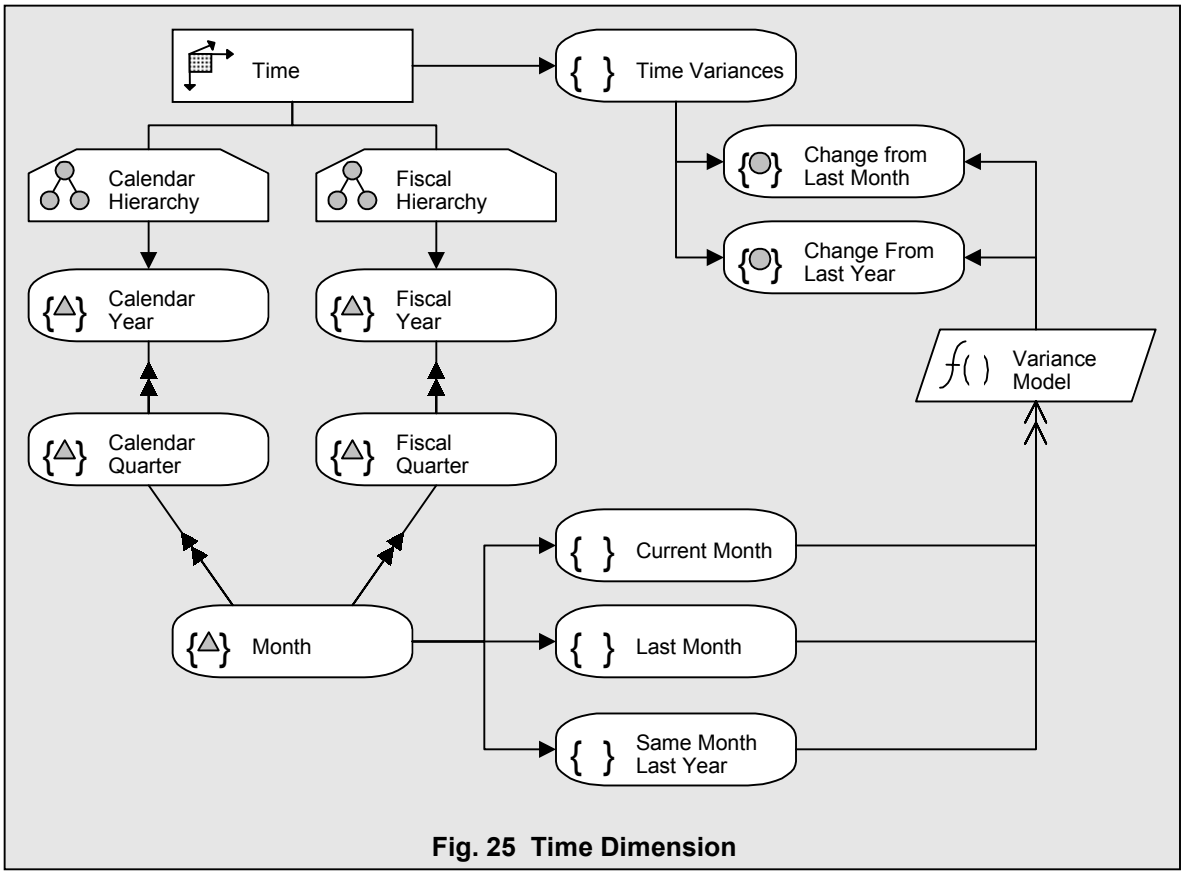


Fig. 25 Time Dimension

K) Context Objects: Sample Case

Acme Toy wants to extend their application to include a forecast. The forecast will use units sold to project forecasted units. The specifics of the forecast algorithm are not completely specified at this time but it will probably be based on the compound growth of the current business. First, we add to our database a forecast cube with time, product, and customer dimensions. We then create a model to represent the calculation of the forecast and show the units cube as input and the forecast cube as output. As usual, business analysis is never as simple as it seems at first glance. Upon further review, we discover that the forecast model is very different for the flagship products than the rest of the products.

The typical approach to this issue is to change the specifications document and insert many lines of if-then-else logic into the code. This is not the correct approach. The more the information about the application is designed as database objects, the more evident the business issues are and the higher the quality of the application.

To incorporate the new information into our diagram, we create two models, one for the general forecast and one for the flagship products forecast. To illustrate that the flagship model will only forecast some of the products, we create a context of the forecast cube. The forecast cube is filtered through the flagship classes scope to create the flagship forecast context. The destination of the flagship forecast model is the flagship forecast context, not the main forecast cube (fig. 26). Using ADAPT we can plainly document the exception processing that occurs.

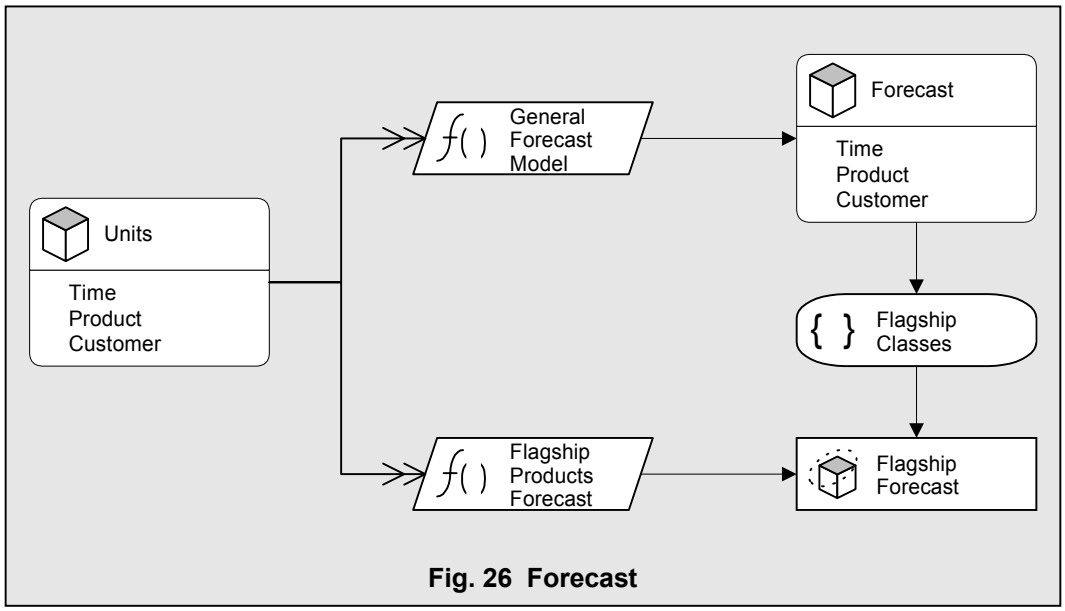


Fig. 26 Forecast

L) Putting it all together

The drawings in figs. 25 and 27 illustrate all of the information we have collected about the time and product dimensions for Acme Toy. It is important to note that there are no assumptions about physical implementation issues such as what type of software will be used or what the database schema will be. At this point it doesn't matter whether a relational or multidimensional database will be used. It also doesn't matter how hierarchies will be represented in a relational database or whether the MDB is a single or multiple cube architecture.

In the example we refer to aggregated sales, average selling price and time variances as "calculated" values. It is very important to realize the use of the word "calculated" refers only to *what* needs to be calculated, not *when* it needs to be calculated. Whether to calc-and-store or calc-on-the-fly is not a logical design issue and therefore not an issue that ADAPT addresses.

Once we are sure that the application is correctly designed, we are ready to translate the logical design into a physical design. It is at the physical design stage where the application designer considers the specific capabilities of the implementation software. At this point, the designer can take advantage of the features of the implementation software and overcome any software deficiencies. An upcoming book on ADAPT will cover how to represent more complex

relationships, will provide working examples that you can use as templates for your own OLAP applications, and will discuss translating the logical design into a physical design.

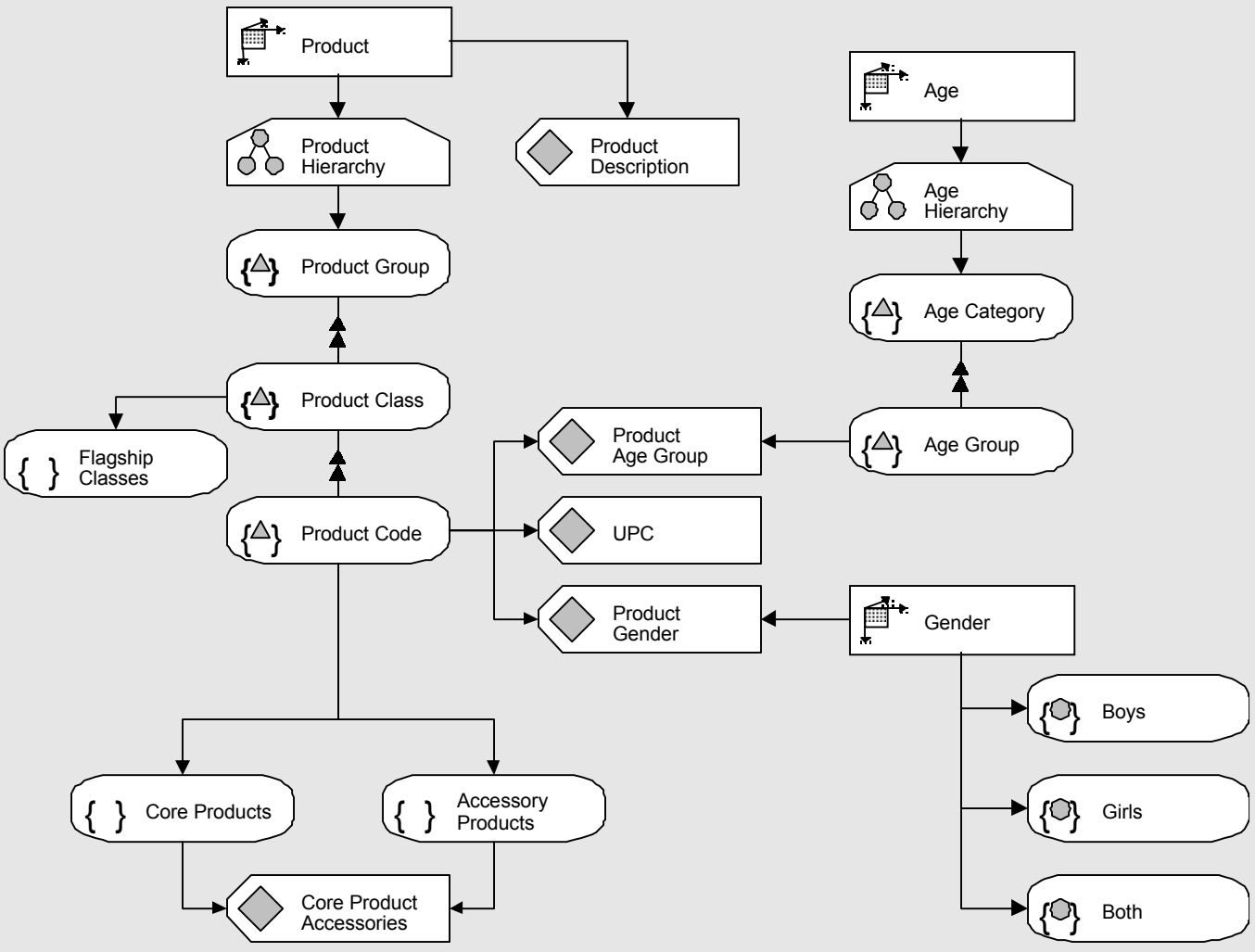


Fig. 27 Product Dimension

One of the problems with database diagramming is a general tendency to create extremely large diagrams. We actually witnessed a case where a client had a data warehouse database design document that was 5 by 14 feet! It is impossible to look at a multiple foot long diagram and do anything but have your eyes glaze over. The diagrams need to be reasonably sized, preferably on separate sheets of notebook-sized paper. We recommend you divide the layout of an ADAPT diagram into sections, with related objects in the same section. A section for dimensions and one for cubes is helpful. There should be a page for each of the dimensions. By grouping all the information about a dimension together, the dimension can easily be reviewed and validated. The same is true for cubes.

M) Conclusion

In this white paper we took a brief look at how to get started using ADAPT. We reviewed each of the nine objects ADAPT uses for a database design and we created diagrams for a simple sales analysis system. Even though our example was very simple and straightforward, something that never seems to happen when we do it for real, we saw the richness and value of ADAPT. We were able to represent an OLAP application in its entirety, without compromising our design as result of using a modeling technique designed for another purpose. The methodology enhanced, rather than limited, our ability to illustrate an OLAP database design clearly. With the information from our diagrams, we were able to locate problem areas and fix them, readily validating our design.

We saw that perhaps the biggest benefit of ADAPT is enhancing communication. Improved communication is a major factor contributing to higher quality software applications. It is impossible to provide a high quality application if you do not understand the business requirements. As a project team gets larger and the systems become more complex, communication becomes even more important. ADAPT provides project team members with a common basis for communication so that design review meetings can center around discussions on what the system does, rather than on explaining the design itself.