Chapter 3

OLAP Cubes

Chapter Outline

What You Will Learn in This Chapter

- Why is OLAP important or even necessary?
- Why are transactions and analysis difficult to combine?
- What is the process for designing and creating a data cube?
- How are OLAP cubes created using Microsoft SQL Server Analysis Services?
- How are dimensions created and modified to improve browsing?
- How can the cube provide more information?
- How can the cube be accessed outside of Analysis Services?
- How can the cube connect to external data such as Web sites and maps?
- How can simple data be provided to managers on a daily basis?

Winmetrics Corp.

In some ways, OLAP cubes are similar to reports—with the added features of interaction. Users can click to drill down and get more detail, or roll up totals to compare by region, product, or any defined category. With a few clicks, a user can quickly filter by any desired conditions. Setting up an OLAP cube often takes time and experience, so companies such as WinMetrics make money by helping companies configure their databases and OLAP cubes. WinMetrics describes the results at one company, which is the leading outsource collection agency for government debts in America. Initially, the company used fixed printed reports, including a 500-page "CARE" report that took more than 24 hours to generate and was difficult to use. It took WinMetrics about eight weeks to create a system to clean and transfer the data to SQL Server and build the OLAP cube. After training some in-house engineers to build additional cubes, the company eventually built cubes for most of the accounting functions as well. Instead of relying on fixed reports for standard items including general ledger, payroll, budget, and forecasting, OLAP cubes were created to enable dynamic access to the data. The underlying data was exported from the third-party accounting system and transferred into OLAP cubes in SQL Server. All of the accounting statements are now accessible through a few clicks of the OLAP cube, including the ability to drill down and see details such as the source of variances in the budget model. [WinMetrics]

Managers and analysts want dynamic access to data. The ability to explore and examine different aspects is useful to understanding the results, spotting problems, and making decisions.

WinMetrics Corp., How a Financial Services Company Developed a Performance Report for Clients, Saved $200K and Sold $167,000,000 of Equity in Just 9 Months. http://www.winmetrics.com/olap_casestudies.html

Introduction

Why is OLAP important or even necessary? Relational database systems were designed to collect and store transaction data efficiently. They have been effective at handling standard business data for dozens of years. Designing the database properly by splitting the data into tables is important. With separate tables, new data can be added without impacting the existing data—enabling the databases to become huge and still maintain performance when adding new data. SQL is a powerful tool that retrieves data from multiple tables, creates calculations and subtotals. So what is the reason for OLAP?

The problem with relational databases lies with retrieving the data. SQL and the visual designers are nice tools to use and they work well on small problems. But putting millions or billions (Jacobs 2009) of rows into multiple tables is going to cause problems. One of the biggest problems is created by joining tables. Joining tables by matching data keys is a slow process. Most relational database systems try to improve performance by building indexes—but the indexes take space and slow down updates, particularly with millions of rows of data.

Online analytical processing (OLAP) is designed specifically to improve performance for retrieving and analyzing data. The goal is to provide enough speed so that even with huge databases, analysts and managers can examine data interactively with minimal delays. To accomplish this task, OLAP systems typically store data in a new design format—which often entails pre-building all of the joins. Design issues are an important section of this chapter.

Even if the data is stored so that it can be retrieved efficiently, how are managers and analysts going to interact with the data? The presumption is that managers need to see various subtotals and to compare these values by different categories. For example, most managers need to see sales or production values at different points in time, and often want to compare values by different product or model types. As shown in Figure 3.1, a **cube browser** is a common OLAP tool that enables managers to select categories (such as Model Type, Time, and Location), and examine some critical fact for any desired subtotal. Generally, the cube is interactive and the manager can drag-and-drop items of interest and examine multiple levels of subtotals. Filters, such as Location in the example, can be used to display data for specific situations. All of these actions are accomplished by selecting or deselecting items on the screen—without writing any queries.

OLAP cubes are useful for exploring the data, but sometimes managers need even simpler tools. **Key performance indicators (KPI)** can be defined and used to create simple visual gauges that display the current values and trends of critical factors. For example, a sales manager or CEO might create a gauge that shows daily sales and comparisons to the prior year or month. Financial managers might track movements in stock prices or interest rates. Just as the gauges in an automobile provide information and feedback to drivers, **digital dashboards** are designed to provide critical information at a glance to managers.

Many tools exist to help build OLAP projects and analyze data. Some similarities exist across the tools, but all of them have their own quirks. This chapter focuses on the Microsoft business intelligence or SQL Server Analysis System (SSAS) tools.

RTCube C03.xlsx

Sale Price	Column Labels							
Row Labels	Hybrid	Mountain	Mountain full	Race	Road	Tour	Track	Grand Total
Calendar 1994	199,006	559,020		928,984	486,774	346,201	35,029	2,555,013
Calendar 1995	124,240	567,940		294,390	1,074,490	709,230	187,920	2,958,210
Calendar 1996	631,760	1,469,870		1,374,850	361,020	213,360		4,050,860
Calendar 1997	2,780	783,820	1,826,320	1,851,160	894,040			5,358,120
Calendar 1998	110,360	1,366,180	2,712,310	683,090	1,562,180	203,270		6,637,390
Calendar 1999		3,138,210	2,792,330	1,998,490	1,816,790	634,260		10,380,080
Calendar 2000	94,010	1,657,070	1,611,920	671,200	600,180			4,634,380
Calendar 2001	180,500	1,221,870	1,399,290	936,430	876,740	72,890		4,687,720
Calendar 2002		1,523,740	2,507,560	1,581,050	1,645,770	225,980		7,484,100
Calendar 2003		2,241,600	3,675,790	1,585,560	2,593,660	526,680		10,623,290
Calendar 2004		896,390	3,372,690	1,858,720	2,183,070	624,800		8,935,670
Calendar 2005		934,540	2,812,930	3,113,150	2,285,410	486,990		9,633,020
Calendar 2006		1,205,720	2,700,080	3,235,570	2,319,750	599,460		10,060,580
Calendar 2007		1,098,330	3,370,890	4,431,690	2,965,450	313,340		12,179,700
Calendar 2008		846,990	2,930,500	3,931,900	2,508,860	471,980		10,690,230
Calendar 2009		829,900	3,452,630	4,244,930	2,951,360	296,470		11,775,290
Calendar 2010		1,018,340	4,277,740	5,171,940	3,417,090	616,580		14,501,690
Calendar 2011		894,730	5,435,980	5,708,890	4,079,390	1,009,970		17,128,960
Calendar 2012	628,450	1,283,150	5,327,330	5,903,620	3,748,000	632,210		17,522,760
Quarter 1, 2012	111,940	164,510	1,063,010	1,154,660	716,840	161,460		3,372,420
Quarter 2, 2012	178,070	367,800	1,265,120	1,430,560	1,022,140	147,300		4,410,990
April 2012	67,780	158,450	478,280	576,620	348,270	64,520		1,693,920
May 2012	42,450	75,840	467,680	459,010	355,720	41,310		1,442,010
June 2012	67,840	133,510	319,160	394,930	318,150	41,470		1,275,060
Quarter 3, 2012	116,990	350,800	1,215,100	1,165,540	702,210	102,010		3,652,650
Quarter 4, 2012	221,450	400,040	1,784,100	2,152,860	1,306,810	221,440		6,086,700
Calendar 2013	736,360	1,286,090	5,437,450	6,093,490	3,687,020	695,840		17,936,250
Calendar 2014	691,900	970,200	5,792,490	6,101,470	4,560,560	588,610		18,705,230
Grand Total	3,399,366	25,793,700	61,436,230	61,700,574	46,617,604	9,268,121	222,949	208,438,543

Figure 3.1

OLAP cube browser. OLAP browsing consists of examining subtotals of a fact measure for any dimension. Typically, users can drag-and-drop to change dimensions. Hierarchies such as date provide the option to drill-down to see detail or rollup to see the summary level.

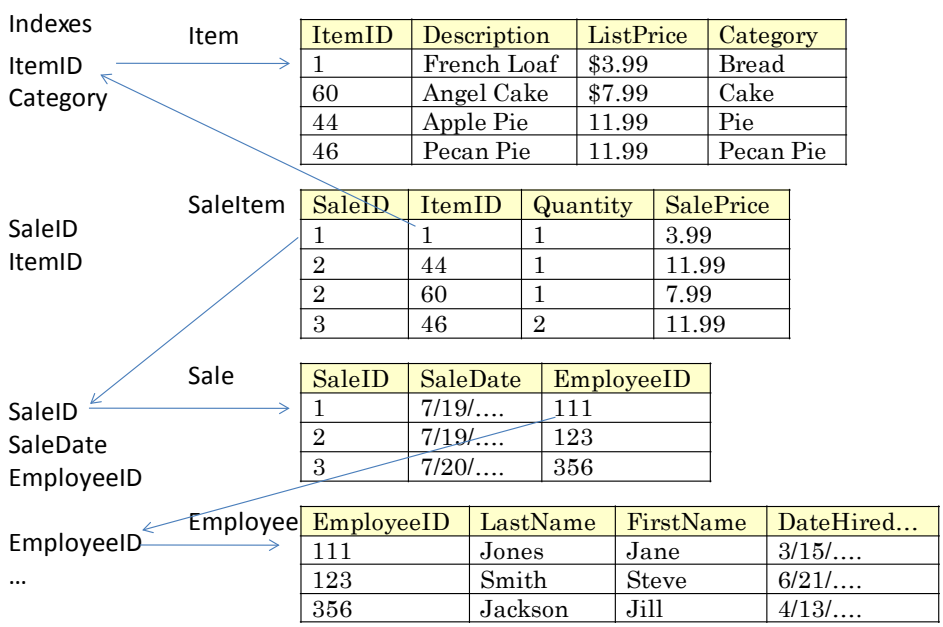
Challenges with the Relational Model

Why are transactions and analysis difficult to combine? Relational databases are designed for **online transaction processing (OLTP)**. The primary task in OLTP is to collect transaction data (such as sales) and store it efficiently. The systems also print basic reports, but these usually consist of simple receipts and totals. Each data object is stored in a separate table—isolating it from the rest of the data. Adding new data is fast and efficient. Data duplication is avoided (except for backups), and data is typically retrieved in small pieces.

However, data retrieval has always been an issue with relational systems (and with hierarchical and network systems before that). Splitting data into multiple tables means that the tables have to be joined to retrieve data. Most relational systems rely on indexes to improve performance for data retrieval. An **index** is a file that contains the key values sorted with a link to the rest of the data row. By sorting the data, items can be located with a binary search (or better). A binary search splits the data in half at each pass. Think about an alphabetical list of names. To find a name (say Smith), start in the middle. If the name there is less than Smith, discard the entire first half of the list. Split the remaining names in half and look at the middle again. Continue the process until the desired name is matched, then use the index pointer to retrieve the rest of the data row. Indexes are a fast way to improve retrieval performance. Searching through a million entries sequentially would take from 1 to 1 million data retrievals (averaging 500,000) without an in-

Figure 3.2

Relational tables and indexes. All key columns have indexes to improve performance. To improve queries, other columns often have indexes. More indexes mean faster queries. But inserting one row can require updating dozens of indexes, slowing down transaction performance.



dex. With a binary index, any entry could be found within 20 lookups because 2^{20} is greater than 1 million.

Indexes

As indicated in Figure 3.2, indexes are commonly used for key columns in relational databases. If managers want to create subtotals by other columns, such as Item Category, Employee name, or Customer City, these columns are typically indexed as well. The result is that each table could have many indexes. Adding a single row of data then requires updating every one of those indexes. The indexes are relatively efficient, but at various points, adding data requires restructuring large portions of the index. The result is that indexes are useful for improving the performance of retrieving data, but indexes slow down data storage and updates needed for transactions. This conflict is what leads to the need for a new storage method.

For large databases, the solution typically involves creating a new database—designed just for OLAP. This **data warehouse** often retrieves data from multiple sources, cleans it up, and places it into a completely new storage structure. Data within the warehouse does not change constantly. Instead, it is extracted from the relational sources on a periodic basis and the warehouse is updated in bulk. At that time, indexes can be rebuilt one time, making it ready for any type of data retrieval and analysis.

Data Warehouse

The main issues in OLAP design and construction are examined in the next section—particularly applied to Microsoft's tools. However, some basic principles apply to all of the tools and some questions have to be answered before any specific tool is selected. A data warehouse is a database specifically designed for data retrieval and analysis. Typically, it has a special design. It usually contains a copy of the data taken from a relational OLTP database. Figure 3.3 illustrates the basic concept of collecting the relational data into a central data warehouse.

Professional arguments still exist over how to handle the tradeoffs between storage and retrieval of data. Some experts and DBMS companies believe that the relational model with efficient indexes can handle most problems. Others believe data needs to be moved to a completely separate data warehouse to be efficient. One drawback to the warehouse approach is that the data being analyzed is not live—it is updated at certain intervals. Updates once a day are common, where the warehouse is reloaded and rebuilt overnight. Some companies update the data more often—perhaps a couple of times a day. Others delay as long as a week. The timing depends on the data and the needs of the managers. The question of whether data has to be live is an important one to answer when starting the process of designing an OLAP system. Microsoft's Analysis Services has the ability to store data using different methods so it can handle live and warehouse data at the same time—but performance is slower with relational data.

An intermediate solution to a full data warehouse is the use of materialized views. A **view** in a relational database is a saved query. When run, the query retrieves the data defined by the SQL statements from the desired tables and displays the results. A view is dynamic—it rebuilds the joins and queries the raw data each time it is executed. A complex view with many joins can take a long time to run—in an OLAP context, views are often too slow—taking many seconds or minutes (or worse) to retrieve data. And each time a user wants the same data, the

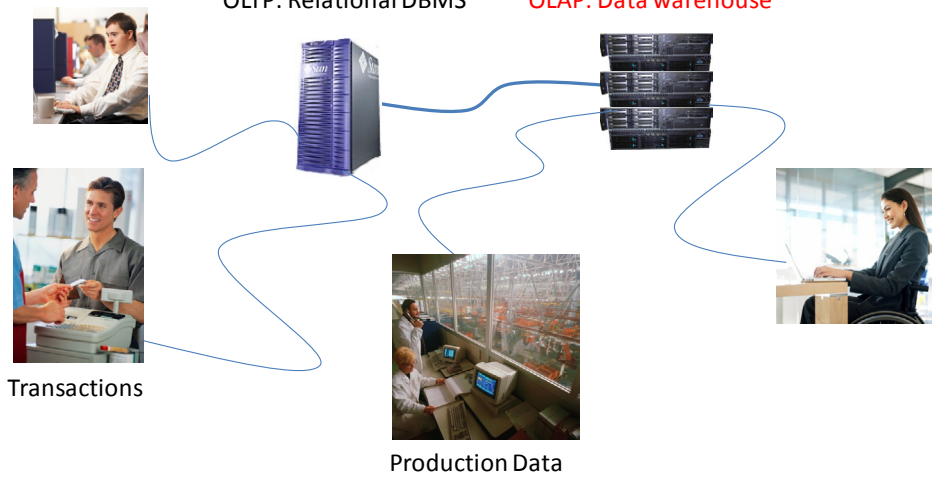


Figure 3.3

Data warehouse. Transactions data is processed in relational databases. The data warehouse extracts the data from multiple sources, cleans it up, stores it in special formats and processes it for analysis.

entire query has to be re-executed. Oracle, followed by most other DBMS vendors, introduced the materialized view. A materialized view starts as a query but saves the resulting rows as a separate table. Indexes can be built for the materialized view (but not for a regular view) because it contains the actual data. The data in the view has to be updated periodically to collect changes from the underlying tables. In a sense, a materialized view is a miniature data warehouse within the relational database. It is particularly useful for retrieving data that does not change often, but is heavily used. For example, customer and employee names are relatively stable. Product categories and descriptions might change for a few items, but the majority remains constant for long periods of time. All of these examples are items that might be used in data analysis—particularly for look up values and subtotals. If they are built into a materialized view, the resulting query can be significantly faster.

Extraction, Transformation, and Loading

One of the biggest challenges in creating a data warehouse is the need to build all of the connections and transfer functions that collect the needed data. These data transfers are executed on a regular basis so they need to be completely automatic. It is not a question of cut-and-paste data from one source to another. The system needs to be automated. Figure 3.4 shows that data can come from many different sources—even different types of databases. Business mergers are particularly challenging in terms of combining data because terminology, data format, and ID values might be radically different. Even older data often becomes a problem. Managers want to use the old data for comparisons, but it often has to be redefined to match the new data. It is easy to underestimate the amount of data an organization contains and the thousands of ways it can be messed up—until you sit down to define how to collect it all and transfer it into the data warehouse.

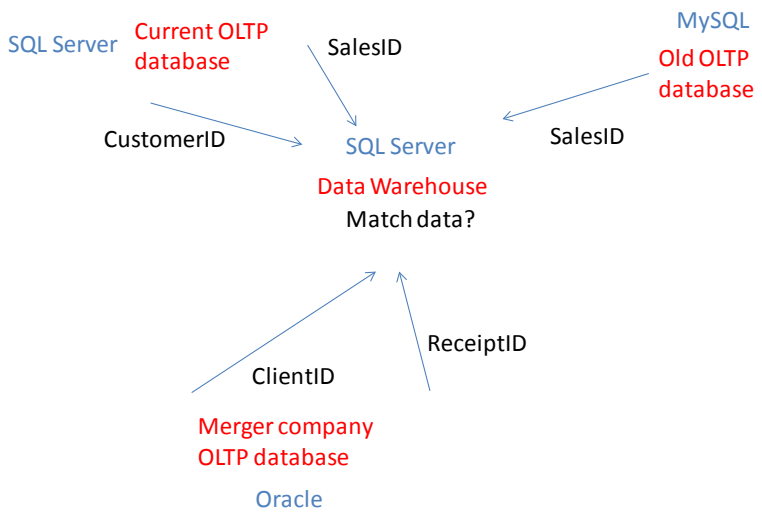


Figure 3.4

Extraction, Transformation, and Loading. Data comes from different sources in different formats. It often needs special procedures to extract. The formats might have to be changed, or ID values recoded. The entire process must be automated.

Extraction, transformation, and loading (ETL) is the term used to describe the process of collecting data from various sources and transferring it into the data warehouse. Because the updates have to be made every day or more the process must run automatically and unattended. Consequently, all of the data sources and transformations must be written down and procedures created to handle the various tasks. All of the steps and code have to be tested and debugged to ensure they work perfectly. The process of designing the ETL can take several months of effort. Even if all data is held in a single enterprise resource planning system with consistent definitions, it takes time to locate the exact data and ensure the proper formats and security permissions are established.

SQL queries are useful tools for transforming bulk data. For instance, SQL can be used to alter ID values with a single command. A useful trick if data from two sources has overlapping ID values and one of them needs to be shifted before merging the data. SQL is also good at identifying unmatched data—such as old product ID values in a sales item table that no longer have records for matching products.

The ETL process almost always requires database programmers to write the transformation code, establish the database connections, and validate the security controls. These programming and administrative tasks are beyond the scope of this book. If you are planning a data warehouse project, talk to the programmers early, particularly before any products are purchased.

MOLAP, ROLAP, and HOLAP

SQL Server Analysis Services can use one of three methods to store data in the data warehouse. **Multidimensional OLAP (MOLAP)** is the preferred storage method for most applications. MOLAP storage extracts the data from its source, pre-builds most joins, and writes the data into a new format with several indexes.

Retrieving data from MOLAP is faster than the other methods. The new physical format closely matches the logical structure of cubes described in the next section. **Relational OLAP (ROLAP)** leaves the data in the source database. The **metadata** or descriptions of the data are stored in the Analysis Services database. Because the data remains on the original source, the queries always produce up-to-date information from the live source. However, queries can be considerably slower than the MOLAP approach. On the other hand, MOLAP systems require processing time at the start to transfer the data and create the basic structures. Generally, the effect of this processing time can be minimized by performing it at night or when the data and systems are lightly used. The third method is **hybrid (HOLAP)** storage. With this approach, the original data remains in the relational database, but aggregations or subtotals are computed and stored in the warehouse. HOLAP might seem like a useful compromise, but its query performance has been reported to be similar to ROLAP, so it is probably not very useful.

One nice feature of Analysis Services 2008 is that data can be defined in partitions and each partition can have its own storage method. If the application needs live data, leave that portion in ROLAP structures. Put the rest of the data into MOLAP partitions. The logical data is treated the same and the data is easy to combine even from multiple sources on different partitions.

For the major elements of data needed in a project, someone has to decide if live, absolutely-current, data is needed. If so, this data is placed into ROLAP partitions. Other data should be placed into MOLAP partitions. Later, performance specialists can tweak the system to improve overall performance. Realistically, unless a project has at least 20 million rows, it is small enough that the choice of storage method is not going to have much effect on performance. All of the projects used in this book are small enough to easily use any of the storage methods—but MOLAP is the obvious choice because none of the data needs to be live.

OLAP Design

What is the process for designing and creating a data cube? Perhaps the first question is why OLAP browsing typically refers to a cube. The two questions are related. Figure 3.5 shows a sample three-dimensional cube. The data displayed on the cube are **measures** on some **fact**. The attributes on the sides are **dimensions**. The dimensions consist of factors that are important to decision makers. The cells in the cube show subtotals for the values of the two intersecting dimensions. A hypercube can have any number of dimensions, but it is hard to draw anything above three dimensions. In fact, most people can read only two dimensions at a time in a basic table. So, most OLAP tools provide a cube browser that is essentially a dynamic table. Analysts can choose the dimensions by dragging from a list. If the values in the dimension are small enough, they can be stacked in the table. Filters can be used to effectively slice the cube—changing the table values to show a single slice of the cube. For example, setting a filter to Month=Jan could be used to show a table with Customer location against Category for just that month. Dynamic changes and quick responses are key features of cube browsers. No one wants to wait minutes for a cube to update when a dimension is changed.

Figure 3.6 shows the Analysis Server cube browser with data from Rolling Thunder Bicycles. It currently displays totals for the dimensions of Model Type and Location of the customer. The date of the purchase is set as a filter dimension, but it currently displays data for all years. The dimensions can be changed simply by dragging them from the list in the left panel onto one of the three spots (row,

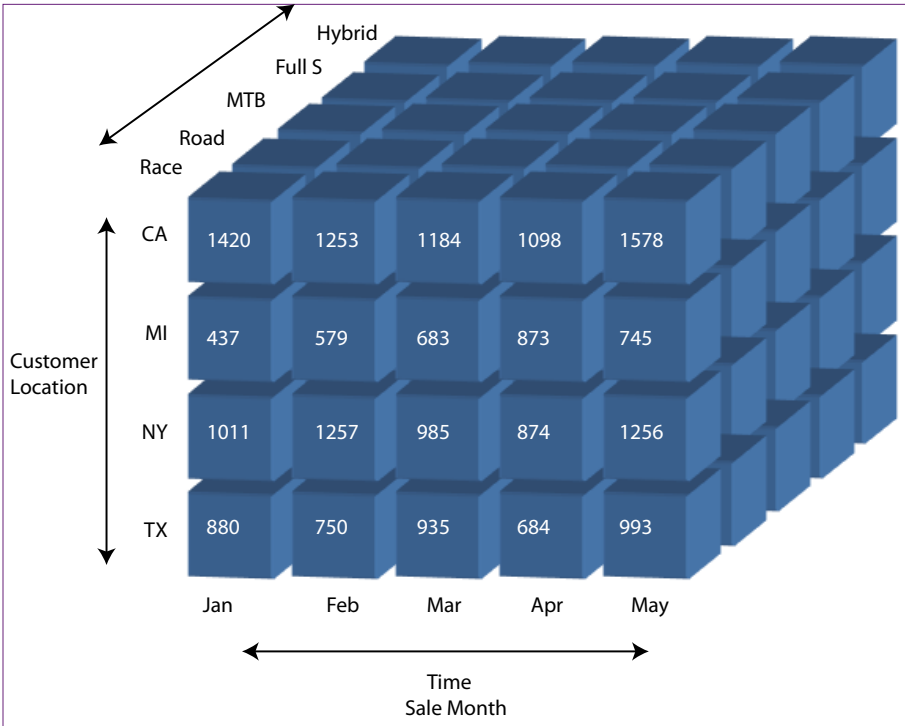


Figure 3.5

Hyper cube. The data are measures on facts. The attributes are dimensions on the cube. A cube browser shows the subtotals from different perspectives, and can examine slices of the cube from any perspective.

column, or filter). Dimensions are removed by dragging the off the table. It is possible to put multiple dimensions on the rows, but the table size quickly gets out of control and hard to read with too many values.

A dimension can be organized into a **hierarchy** or levels. In the example, location is a hierarchy from country to state, city, and ZIP code. Similarly, the date is a hierarchy from year, quarter, month, and date. These hierarchies have to be created within the dimensions, but once defined the cube browser handles the hierarchy automatically.

Notice that the left panel also contains a list of possible measures that can be used within the table. The cube currently displays subtotals for Sale Price. Other measures include List Price and the count of the number of bicycles. The measures can be dragged on or off the center of the cube. It is possible to display multiple measures at the same time side-by-side. However, the table can quickly get out of control when too much data is placed on it.

The default computation for values is to sum them. This computation can be changed to one of several options—but it cannot be changed from within the browser without some additional configuration. This restriction makes sense when you understand how the OLAP tool works. To provide near instantaneous response, the totals are computed only when the cube is processed. Browsing simply retrieves the values that match the displayed dimensions. So the next step is to learn how to setup and create the basic cube.

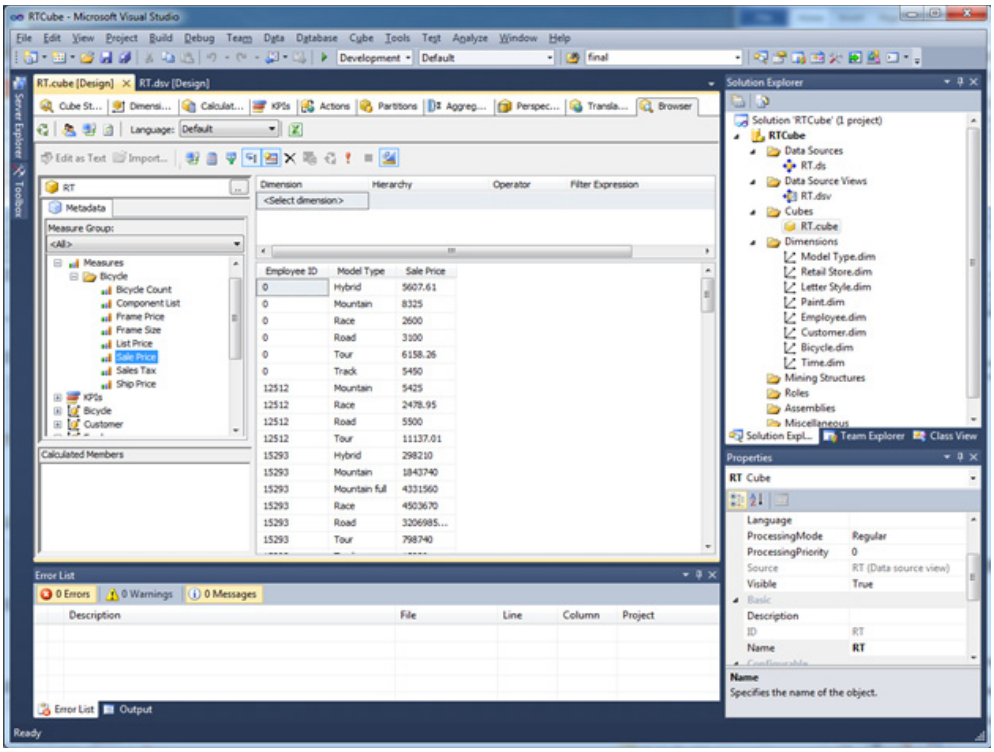


Figure 3.6

OLAP cube browser. Dimensions can be dragged from the left onto rows, columns, or the filter at the top. The data totals in the table update automatically as the dimensions are changed. Individual attribute values can be deselected with the drop-down arrow by the dimension name.

Facts and Dimensions

The cube image provides the foundation for the structure of how OLAP works. The key to understanding OLAP is to focus on the measures and dimensions. In fact, the multidimensional storage systems use this structure to organize the data for fast retrieval. The first step in designing an OLAP system is to identify the facts that need to be measured. In business applications, the facts typically involve sales, which are generally computed as price times quantity. In the bicycle case, all of the bicycles are unique, so the quantity is always one and Sale Price is the relevant measure. In other cases, it will be necessary to create a calculated column that handles the multiplication. Those details are explained in the next section. Some problems have multiple facts, but business cases typically focus on the value of sales and perhaps the number of items sold or number of sales. At the moment, it is not important where the fact measures are stored. Even if the data columns are stored across multiple tables, the OLAP system will combine everything into one set of measures.

Once the facts have been identified, the attributes that might affect those facts need to be identified. These attributes become the dimensions of the cube. At this stage, it is important to identify all of the potential dimensions that manag-

Product	Customer
Category Color Size Manufacturer	Location: City Gender Age Experience
Store	**Sale**
Country Region City Department Salesperson	Date Time of day Discount

Figure 3.7

Common business dimensions. Look at tables and categories for ideas about which dimensions to include. Ultimately, all of the dimensions are combined and treated equally.

ers and analysts might want to consider. Avoid looking at the data in relational terms. Simply look at the existing data and decide which attributes would be useful dimensions.

Figure 3.7 shows that common business dimensions include product attributes, customer demographics, salesperson, region or store, and sale attributes. Product attributes often include items such as category, color, size, and manufacturer. Customer demographics might include location (City), gender, age, and experience. The availability of data will depend on the type of business and the level of contact with the customers. Large organizations will want to track sales geographically by country, region, and city. Many will want to track sales by department and by salesperson. The Sale itself typically includes a date dimension which sometimes is tracked down to time of day. Some organizations provide different types of discounts and need to track those. The point is to go through all of the data tables and identify anything that might be used to define a grouping or subtotal.

Star Design

In the end, it is not important where facts and dimensions come from. The basic design in an OLAP system is the **star design**. Figure 3.8 shows an example of a star for the common set of business dimensions. The star name arises because the fact measures reside in the center and every dimension is connected directly to the fact table through single links—like rays emanating from a star. The star design is efficient for retrieving data because every dimension is directly tied to the fact measures. Pulling data from a relational database, the system often accomplishes this structure by duplicating key data. The data in a star structure does not have to be stored in relational normal form. Many systems improve performance even more by pre-computing all of the subtotals. The star configuration represents the MOALP or multidimensional storage. Data is stored according to the dimensions.

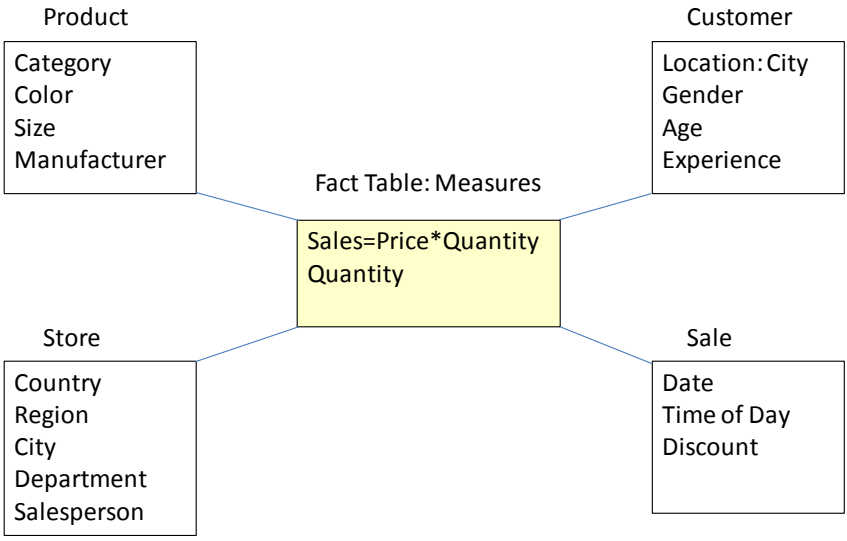
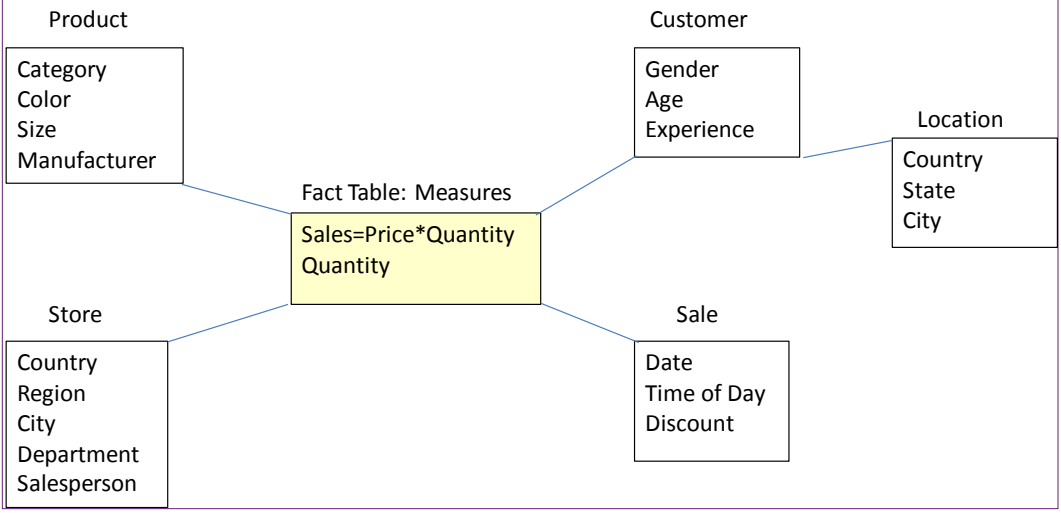


Figure 3.8

Star design. The fact table in the middle holds the measures. All other dimensions are one link away from the measures. The star design is often created by duplicating data. Keeping dimensions down to a single link improves retrieval performance.

Figure 3.9

Snowflake design. Dimension tables can have links to other tables, creating tables that are multiple links away from the fact measures. Complex snowflake designs require the OLAP engine to perform joins to obtain data and slow down the performance. Try to avoid them except for lookup tables.



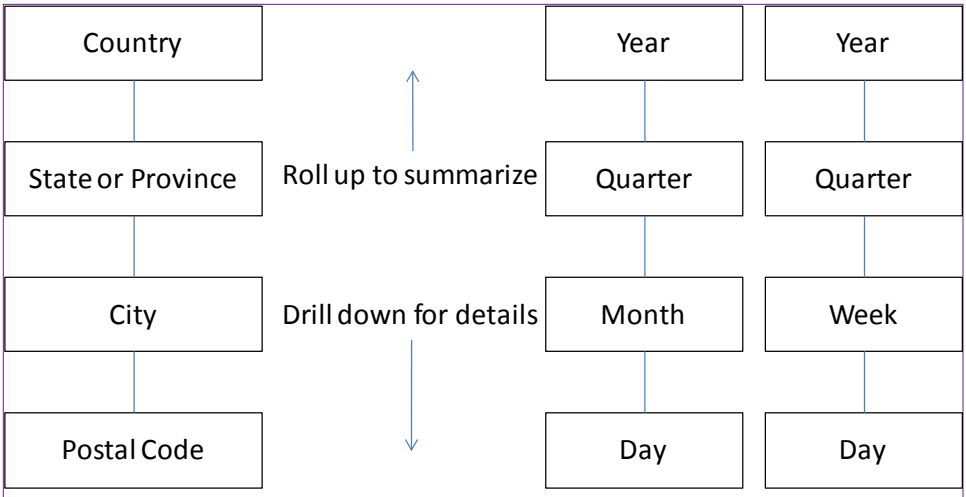


Figure 3.10

Hierarchies. Common natural hierarchies are based on geography and time. Sometimes, as with dates, multiple hierarchies can be defined from the same data.

Snowflake Design

Some OLAP systems support only the star design. Others provide more flexibility by allowing links to the tables holding the dimensions. Microsoft Analysis Services allows the creation of these links. Ultimately, these secondary links make the design appear more like a **snowflake** than a star. Figure 3.9 shows a simple example that links the City table to the Customer table. Throw enough secondary links into the picture and it begins to look like an idealized snowflake. The problem with these secondary links is that the OLAP engine has to build joins to retrieve the associated data. Although the query engine can build indexes, the links generally decrease performance—particularly with huge tables. The links often arise because of the way the data is stored in the relational OLTP database. In most cases, the OLAP tool supports methods to extract the data from the outlying tables and build it into a MOLAP structure. Even though the data might appear to be stored across multiple links, internally, it is denormalized and stored in a star structure. Still, it is best to avoid secondary links when possible. The example here uses it only as a lookup table, and that connection is handled later as a hierarchy.

Hierarchies

Hierarchies are dimensions that ultimately are perceived as a single dimension with multiple levels. People often use hierarchies to simplify problems. Looking at data by levels makes it easier to see the big picture and still provide the details needed to understand a system. Figure 3.10 shows an example of two common natural hierarchies: location and time. Notice that multiple hierarchies can be defined for the same set of data. Some retail companies use weeks to define sales instead of months, but analysts and managers might want both hierarchies for different purposes.

When hierarchies are applied to a cube, the browser displays buttons (often + signs) that enable the analyst to **drill down** to see the more detailed level. Similarly, detailed levels can be **rolled up** to see the totals for the higher level.

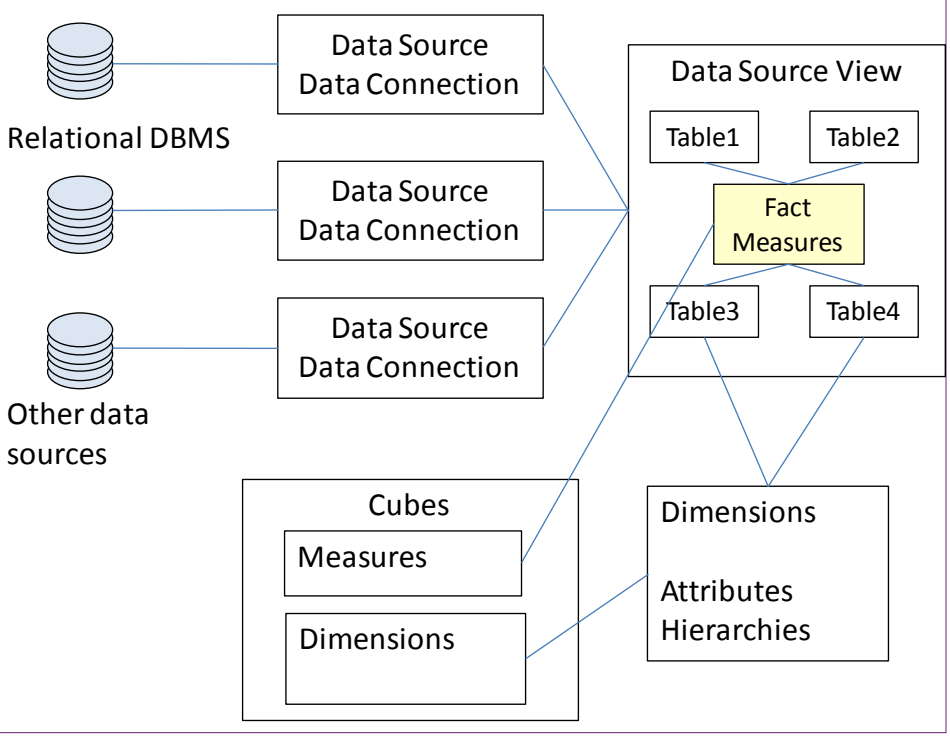
Microsoft Analysis Services has tools to help build common natural hierarchies, particularly for time and geography. It is also possible to build custom hierarchies, such as groupings of products, or employee teams. These hierarchies can only be created from existing data. If you know that certain hierarchies will be needed, be sure to include all of the related columns needed to define the hierarchy. For instance, an Employee table might include a ManagerID column to report a hierarchical reporting relationship, so ManagerID will have to be included in the dimension list. For workgroup teams, a separate table defining members of the teams will likely be needed.

Creating a Cube with Microsoft Analysis Services

How are OLAP cubes created using Microsoft SQL Server Analysis Services? The basic concepts of OLAP cubes are relatively common. The process of creating one is quite different depending on the specific product. The steps with Microsoft's Analysis Services are straightforward, largely because of the use of wizards. The basic process is to build connections to the data sources,

Figure 3.11

Structure of the Analysis Services design. 1. Define connections to the data sources. 2. Crate data source views to retrieve and configure the data. 3. Define cubes as a collection of measures and dimensions. 4. Refine dimensions and add hierarchies.



define views and calculations, create an initial cube, and refine the dimensions and create hierarchies. Figure 3.11 shows the basic structure of the objects within the Development Studio for building OLAP cubes.

Although other tools exist, the most powerful tool to define and control the OLAP structure is provided with the **Business Intelligence Development Studio**. The BI studio is based on Microsoft Visual Studio, which is a design environment for many types of programming. The BI projects are built using a specialized set of templates. The studio should be installed on your workstation using the Client disk from the SQL Server installation package. Even if a workstation already has Visual Studio for other uses, the BI templates need to be installed from the SQL Server setup.

The data defined for OLAP cubes is also used later for data mining. The statistical analysis tools are embedded in the Development Studio and the data can be retrieved through OLAP views or directly from OLAP cubes.

Data Sources

Almost any type of database or standard file can be used as a data source with SSAS. Microsoft has been building database connectivity systems for years, and most of them can be used to read data into the OLAP structure. One catch is that if multiple connections are going to be used, the first connection should be to a SQL Server database. Apparently, Analysis Services routes the connections to the other systems through the SQL Server database.

Start the BI tool and create a new project. Choose the Business Intelligence project type and select the Analysis Services Project template. All projects are created as a set of files in a folder on the local computer. Most of the files are XML text that describes the specific objects created in the project. A solution (.sln) file holds links to all of the files. When the cube is processed, the information is compiled into a specialized format and sent to the specified Analysis Server. For now, you simply need to choose a name and location for the project. If possible, stick with the default location. Create a name that describes the project that can be recognized later.

Initially projects are empty and Visual Studio displays a list of tasks in the Solution Explorer. The first task is to define a connection to a data source. A data source is typically a relational database. It might be SQL Server running on a central server or even on your local computer. The sample files for this book have scripts to build them on SQL Server, but the data also could be loaded into an Oracle or MySQL database. In a real-world project, the database should be running on a separate server.

Relational Database

Most OLAP projects retrieve data from a relational database. SQL Server is certainly easy to connect to with SSAS, but almost any common DBMS has drivers for establishing connections to Microsoft tools. To connect to an Oracle DBMS, you should obtain the OLE DB client from either Microsoft or Oracle. OLE DB is a Microsoft standard for establishing connections with database systems and the common DBMSs have providers—but you might have to install them separately. The providers will also have to be installed on the server running the Analysis Services engine.

To create a data source, right-click the Data Sources entry in the Solution Explorer and choose the New Data Source option. Click through the startup forms

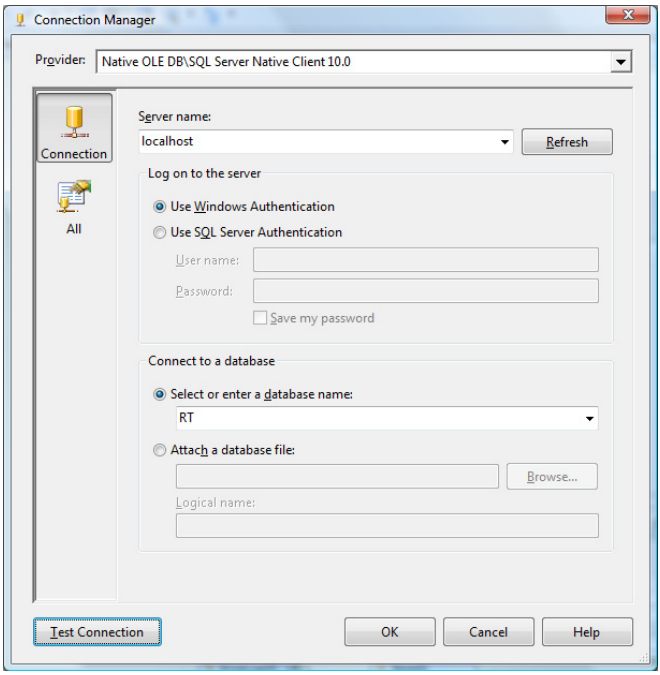


Figure 3.12

SQL data source configuration. Choose the correct server. If possible, use Windows authentication. Select the correct database and always test the connection.

and click the New button. Choose the provider from a drop down list—it defaults to the latest OLE DB driver for SQL Server. The configuration form shown in Figure 3.12 is standard for SQL Server connections. The form changes slightly depending on the provider selected. For SQL Server, choose the name of the server. If possible, use Windows authentication. Eventually, for production systems, it is better to create a separate SQL Server login. For now, select the appropriate database and always be sure to test the connection. Follow the steps to accept the choices and finish the wizard. When finished, an entry will appear under the Data Sources in the Solution Explorer.

At a minimum, the logon connection specified (either Windows or SQL Server) must have read permission on the tables or queries needed for the project. These permissions are set within SQL Server (or whatever DBMS is holding the data). Later, to create hierarchies within dimensions, the logon user will also need permissions to create new tables. Hierarchies are stored within separate tables. Time dimensions are the only exception—SSAS provides a mechanism to store time dimensions on the OLAP server for cases where the logon user cannot be given the permission to create a table.

Multiple sources

Multiple data sources can be added to a project. Simply follow the same steps to add new connections. Connections can be made to different databases, even those stored on different servers and different DBMSs. Always remember to test the connections as they are built—it will be difficult to identify the cause of problems later if the connection fails.

Think about what it means to use multiple data sources. The OLAP engine provides a level of abstraction. As long as Analysis Services can connect to a database and retrieve data, the data can be included in the project. In essence, this trick enables you to integrate data and even define relationships across data tables from different databases. Once the database connection is defined, any accessible data can be used in the project. From this point forward, you no longer care where the data is stored. The OLAP engine handles the details of retrieving the appropriate data. The process of connecting tables is handled in the Data Source Views, but all tables are equal at that point and the source does not matter.

Data Source Views

A data source view in OLAP has some similarities to queries or views in a database engine. The purpose of a view is to provide a perspective on a subset of the data. It is also used to join tables and define new computed columns. One big difference is that the data source view is required for OLAP. The data source view serves as a buffer and it is not possible to retrieve data directly from a table. To build a cube or to retrieve data for data mining, all of the data needed must be defined in a data source view.

A project can contain multiple views. Some projects are built by cramming all data tables into one view. With even a few dozen tables, this approach quickly gets messy. Sometimes there is no choice and all of the tables are needed together. However, in many cases, a better approach is to create separate data source views for each particular problem. Separate data views also make it easier to assign security. For example, a different view can be created for each specific group of users or analysts. Access can be controlled by assigning permissions to the views.

If it really is necessary to include dozens of tables in one data source view, the display can be cleaned up by creating separate diagrams. A diagram shows the relationships among tables and it is easy to create as many diagrams as necessary within one data view. The diagram does not affect the use of the data—which is good and bad. The good part is that the displays can be simplified and it is easier to work with a few tables at a time. All of the data is still visible later when building cubes and data mining models. The bad part is that the division only applies to the diagrams so every table and dimension shows up in huge lists when building cubes and selecting data. A diagram only applies to the display of the data source view on the diagramming window.

To reduce the overall number of tables and dimensions that groups work with, it is necessary to create separate data source views. Whenever possible, separate views should be used. Just be careful to give them descriptive names that everyone will recognize.

Creating a Data Source View

Creating a new data source view is straightforward. Right-click the Data Source Views entry in the Solution Explorer and choose the option to add a new one. The wizard will ask you to select the data source, although it has an option to create a new one if you skipped that first step. The second step is to select the tables and views (queries) from the relational database. Figure 3.13 shows the basic process. For complex problems, it will be helpful to study the relationship diagram from the underlying database. If the tables in the underlying database are linked indirectly through other tables, all of the linking tables must be included in the view to be able to reconstruct the relationships. If you are uncertain about which tables

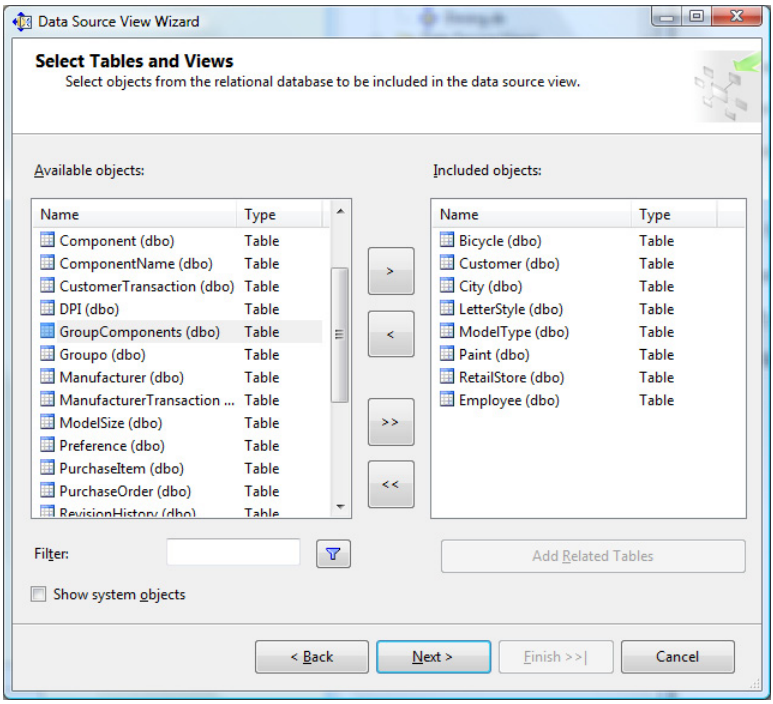


Figure 3.13

Selecting tables for a data source view. This view focuses on the bicycle and the sale. Most of the tables are lookup tables used for the dimensions.

are required, you could start with the table that holds the facts then click the button to "Add Related Table." The drawback to this approach is that it is likely to include many tables that you do not necessarily need in one view. But sometimes it is easier to add everything and remove the ones you do not need.

The last step is to name the data source view. Be explicit, use extra words. It is important to use a name that everyone will recognize later. A short name with abbreviations might seem cute now, but a year from now, it might be unrecognizable. This particular view might be called RT Bicycle Sales and Customers. But do not worry, views can be renamed later.

Creating a Named Calculation

Sometimes existing columns are not in the correct form or do not include the exact data needed for a cube or for data mining. Two of the most common situations are: (1) Common business items such as Value=Price*Quantity or Profit=Revenue – Cost, and (2) Creating concatenated columns to form a single primary key. The data mining routines all require a single column as key and it is often necessary to build one within the data source view. For now, it is useful to learn how to build a common business calculation.

In the Bicycle case, each bicycle is unique so there is no need to multiply price and quantity. However, the Bicycle table contains a ListPrice and SalePrice. The difference between the two is the discount amount. Some managers might want to explore the cube with respect to the discount values.

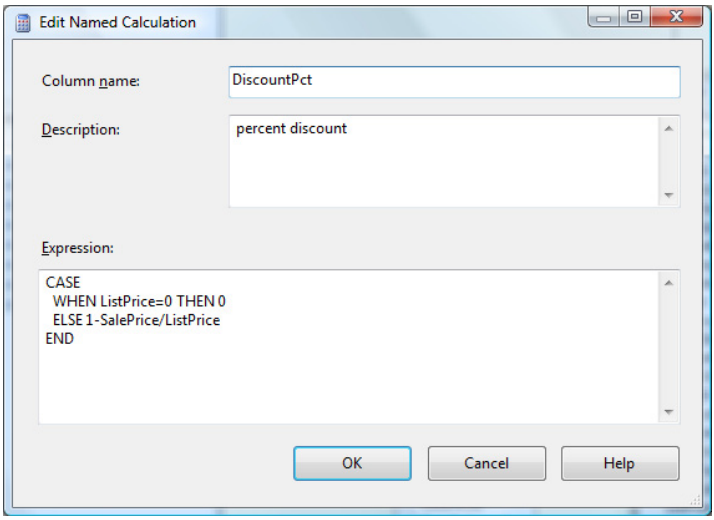


Figure 3.14

Named calculation. Right-click the title bar of the Bicycle table in the data source view and add a Named Calculation. Enter a name and the expression that defines the calculation. The expressions use SQL Server calculations and functions. Calculations are row-by-row.

A row-by-row calculation is created within a table in the data source view and is called a **named calculation**. It is equivalent to creating a calculated column within an SQL query. In fact, it uses the same functions and computations that are used in SQL queries. To create the calculation for the discount, right-click the top of the Bicycle table in the view and select New Named Calculation. It is important to select the title bar of the table; the option will not appear if you select the middle of the table.

As shown in Figure 3.14, enter a descriptive name. Enclose it in brackets if it contains a space or reserved character. A longer description can explain its purpose. The expression is the calculation. Discounts are typically evaluated as a percentage (ListPrice – SalePrice) / ListPrice. Expressions can be any SQL Server calculation or function. To obtain help building expressions, it is often easier to create the expression in a SQL Server query window, test it there, and copy it back to this OLAP expression box. Either way, once the expression is created, it is important to test it within the data source view. Right-click the main body of the table and choose the Explore Data option. Verify the computed column.

One problem with dividing is that it is important to test for zero values of the divisor. The SQL CASE statement is useful to create the two conditions needed (zero and not zero). If ListPrice is zero (most likely missing), set the discount to 0, otherwise define the percentage as 1-SalePrice/ListPrice, which is a slightly simpler expression of the percentage. As an advance warning—do not attempt to use this calculated column in a cube just yet. A section below on calculations explains an important issue.

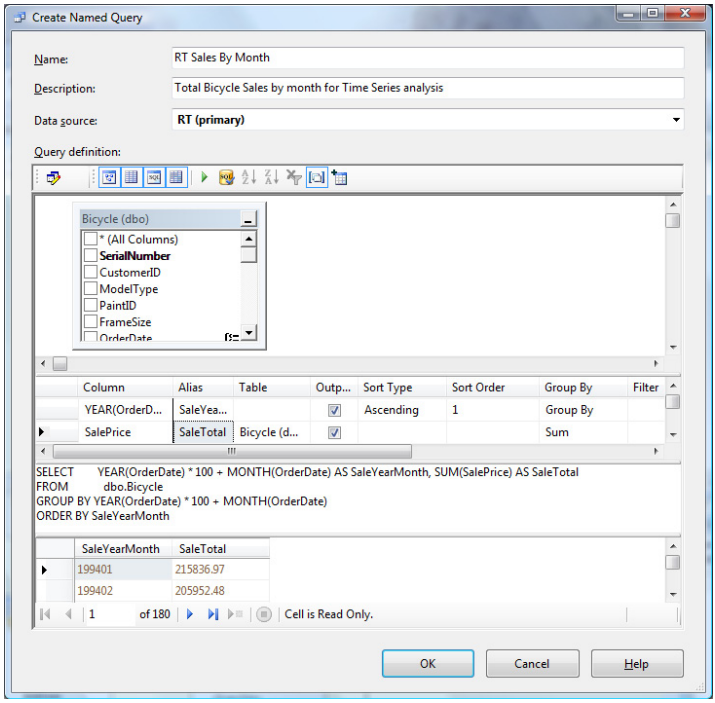


Figure 3.15

Create a named query. The edit window is similar to creating a query in SQL Server, but the named query is stored it the data source view. Use the icons or right-click to add tables and add Groups to create subtotals. The ORDER BY clause is used only to test the query. It must be removed to save it.

Creating a Named Query

It is also possible to create entire queries within a data source view. A **named query** is similar to queries created within SQL Server but they can use all of the tables within the data source view. If all of the tables come from a single SQL Server database, it does not matter if the query is created and saved in the underlying database or in the data source view. However, the strength of creating a named query is that it can us data from any table from any data source—so it can easily combine data from diverse locations. This feature will not be used for the example in this section, but remember that it is available.

Data mining problems often require data to be in specific formats. For instance, to perform a time series analysis on monthly data, the data totals by month need to be computed before starting the time series analysis. The time series tool does not have an option to compute subtotals—they have to be set up ahead of time in the data source view. The data mining tools also require a single column as the primary key, so it is often necessary to use a query to concatenate separate columns into a single key.

Creating a named query is similar to creating a query in SQL, but it is handled in the data source view. Right-click an open location in the data source view and choose the option to add a new named query to open the editor window. Figure 3.15 shows the edit window, which is similar to editing queries in SQL Server.

You can work with SQL directly or use the graphical interface to help build the query. Use the icons or right-click the table window to show the list of tables and to add the Group By options. Always test the query before saving it. In the example, the ORDER BY clause is used to help evaluate the results; but it must be removed before the query can be saved. The SQL Query is:

```
SELECT        YEAR(OrderDate) * 100 + MONTH(OrderDate) AS
SaleYearMonth, SUM(SalePrice) AS SaleTotal
FROM            dbo.Bicycle
GROUP BY YEAR(OrderDate) * 100 + MONTH(OrderDate)
```

Notice the way the YearMonth column is created in this query. Because a single column will be needed as the primary key for time series analysis, the Year and Month functions are used to generate dates of the form: YYYYMM. Multiplying the Year by 100 shifts it to the left by two places to create space for the month number. Also note that views cannot contain the ORDER BY clause. It is sometimes useful for testing, but needs to be removed before saving the named query.

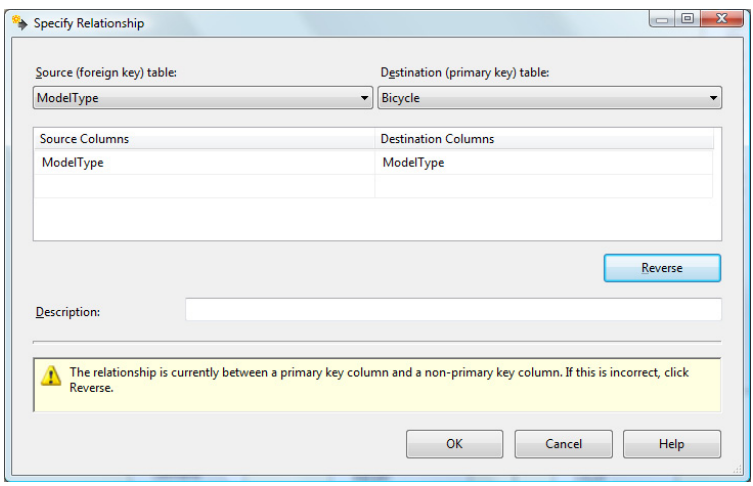
Once the named query is created, it is important to set the logical primary key for the new query. On the data source view, right-click the column name (SaleYearMonth) and choose the option to Set LogicalPrimaryKey. Defining the primary key within the named query sets the default values for any tool that uses that query.

Creating a Relationship

Most of the time, when tables are imported from data sources the relationships between the tables are also imported. However, when creating named queries or importing tables from multiple data sources the relationships will not be built automatically. In those cases, it is straightforward to create the relationships manually.

Figure 3.16

Creating relationships. When using drag-and-drop to create a relationship, drag from the foreign key (many) table and drop onto the primary key in the related table. Relationships created the other way need to be corrected by pushing the Reverse button.



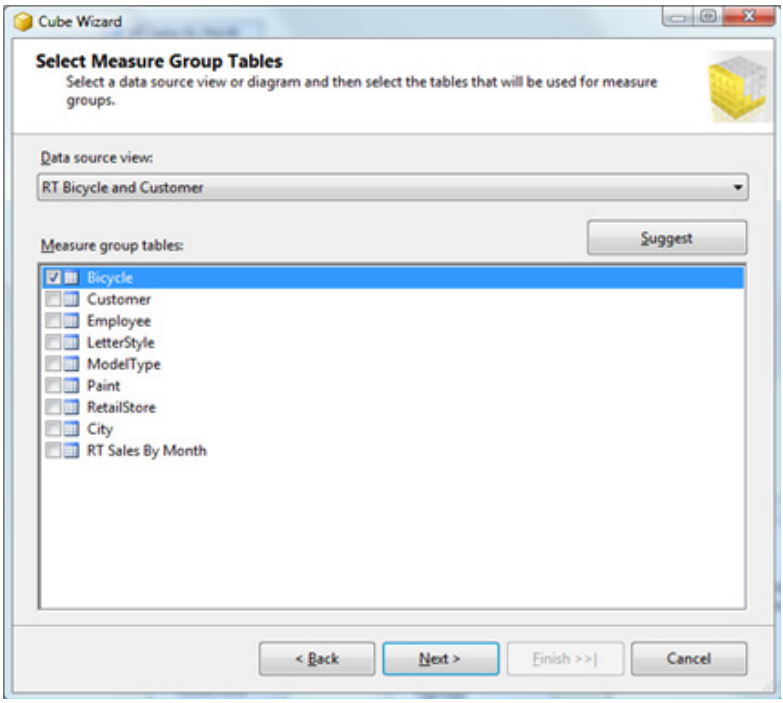


Figure 3.17

Select tables that hold measures. First select the appropriate data source view in the drop-down list. Then choose the Bicycle table which holds the sales information for the RT case.

Relationships can be created by right-clicking the data source view window and choosing the option to create a new relationship. However, it is easier to drag a column from one table and drop it on the matching column in the related table. As shown in Figure 3.16, the big catch is that you need to drag from the foreign key (many) side of the relationship and drop onto the primary key of the related table. The relationship editor is not smart enough to identify the relationship automatically. If you try to create the relationship the other way, the warning message explains the problem. Click the Reverse button to switch the direction. If that does not solve the problem, either the keys on the tables are wrong, or the wrong columns have been selected for the relationship.

Cubes

When the data source and data source view have been created, a hypercube can be defined with the selected data. The main steps in creating a cube are to choose the columns for the fact measures and select the columns that can be used as the display dimensions. If the data source view is built correctly, these steps are straightforward with the help of the cube wizard. When the cube has been built, it can be explored with the browser. Next, more complex dimensions can be created by building hierarchies.

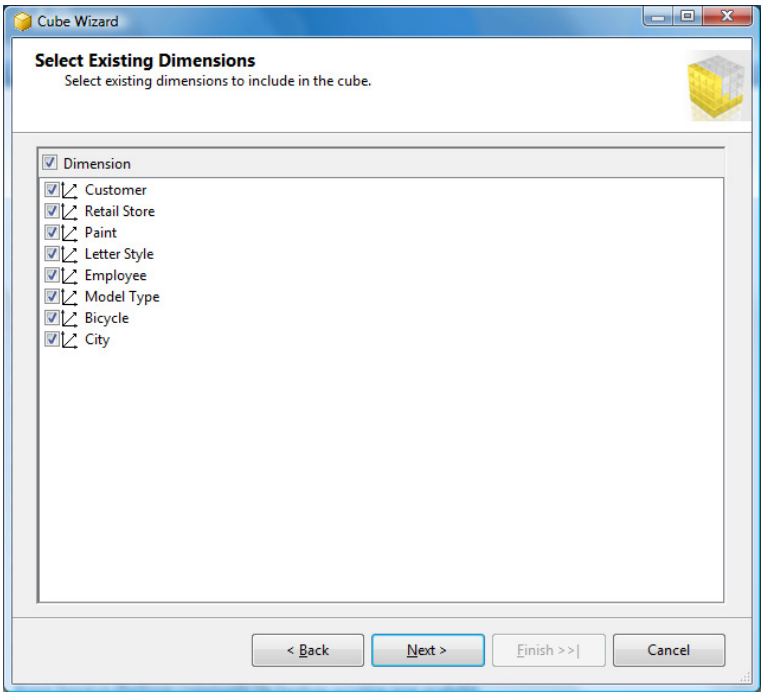


Figure 3.18

Dimension tables. Tables in the data source view that are linked to the measure tables are automatically included as dimensions. The City table can be removed later when the location hierarchy is created.

Wizards

Start the wizard by right-clicking the Cube entry in the Solution Explorer and choosing the New Cube option. The first screen has options for selecting the data. The default choice of using existing tables is generally the best way to start. As shown in Figure 3.17, the next step is to choose the data source view from the drop down list. The view should have been created to hold both the measures and dimensions. The main purpose of the form is to select the tables that hold the facts or measures that will be used in the cube. The RT case uses the Bicycle table to hold standard sales data, so select that table.

On the next screen, the wizard automatically selects all of the columns in the chosen table to use as measures. Because only a couple of the columns are actually measures, click the checkbox at the top to deselect all of the columns. Then select the three useful measures: List Price, Sale Price, and Bicycle Count. The last entry (Count) is not an actual column, it is a measure added automatically by the wizard. It essentially counts the number of rows. It is useful in cases such as the Bicycle table where no quantity exists. Similar situations exist for table on people, such as customers or employees. For more traditional problems containing a Quantity column, including the row count could lead to confusion, so it should be left out.

The next step is to choose the dimensions that can be used in the cube. The wizard automatically selects the tables linked to the fact table. In the RT custom-

er data source view, the dimension tables include: Customer, Model Type, Retail Store, Paint, Letter Style, Employee, City, and Bicycle. Essentially, those tables are lookup tables that provide a description for the ID value stored in the Bicycle table. Recall that the City table was linked through the Customer table. Later, it can be removed when a location hierarchy is created to handle the lookups. Also, notice in Figure 3.18 that the Bicycle table is included by default because it is the measure table. This table needs to be included with the dimensions because it contains the sale date information needed to explore sales over time. If the list lacks some tables that you feel should be included, you should cancel the wizard and return to the data source view to ensure that all of the needed dimensional tables are linked to the measures table. At this point, do not worry about hierarchies because they will be configured after the basic cube is built. The final screen summarizes the cube and enables you to enter a descriptive name.

That is the entire process needed to create a cube. The cube now needs to be transferred to the analysis server and processed. Then it can be browsed. It might not be perfect yet, but it is a good practice to test it to ensure it processes correctly and that it has the fundamental data needed.

Deployment and Processing

Most of the work to this point has taken place on the client workstation. The data connections, data source views, and even the cube definition are currently stored as definitions in XML files. The cube with data does not yet exist. To use the cube, the definition has to be transferred to the Analysis Services server and processed. Processing consists of building the dimensions and pre-computing most of the subtotals for each of the dimensions. Executing this step is straightforward: right-click the new cube name in the Solution Explorer and choose the Process option. The cube will be deployed to the server and a job schedule created to process the data. In most cases, for testing purposes, just click the Run button on the processing form to start the computations. In a production environment, the cube could be huge and take large amounts of resources and time to process. The processing form has options to estimate the impact on resources and limit the number of processors used. But these options are more useful when a cube is being reprocessed to load new data.

After the processing runs, the results will be presented on a progress form. It is critical that the processing complete successfully. Any errors will be displayed on the form and they need to be corrected. Some error messages are confusing, but two common errors are (1) The server is not running or not accepting your connection, and (2) The cube definition has errors—often due to problems with dimensions and hierarchies. Server connection errors

By default, all projects attempt to deploy to the Analysis Services running on the local computer (localhost). In many cases, a central server will be used to run Analysis Services and the deployment location needs to be changed. For instance, students might be asked to write data to a central server, and projects set for final release need to be deployed to a central server. The process for changing the deployment server is not obvious, so Figure 3.19 shows the configuration form. Right-click the project name in the Solution Explorer and choose the Properties option to open the configuration form. Click the Deployment link. Change the name of the Server from localhost to the network name of the server running the Microsoft SQL Server Analysis Services. Your Windows account will need appropriate developer permissions on the Analysis Server. In most cases, it is simpler

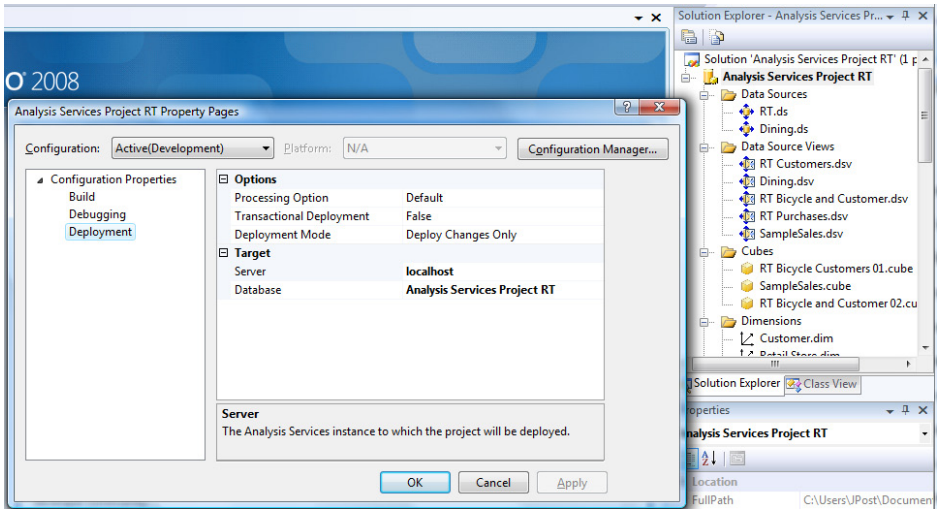


Figure 3.19

Change the deployment server. Right-click the project name in the Solution Explorer and choose Properties. Click the Deployment link. Change the name of the server from localhost to the network name of the central server.

to build and test all projects on your local computer, because security controls are easier to set.

Browsing the Cube

Finally, you are ready to browse the cube and explore the data. The Visual Studio OLAP project has a cube browser, but the 2012 version is quite limited compared to the earlier 2008 version. In particular, the dimensions can be displayed only as rows, not columns. For browsing, analysts will probably want to use an Excel PivotTable instead. The PivotTable can connect to the SQL Server cube, so SQL Server performs most of the processing work. The PivotTable browser allows placing attributes on columns as well as rows so some types of data are easier to read. The Cube Browser in Visual Studio 2012 has an Excel button that will open Excel and automatically add the current cube. (Details for creating a PivotTable independently are covered in another section.) The PivotTable is initially empty with markers for where to drop fields: Row dimensions on the left, Column dimensions near the top, Totals in the middle, and Filter dimensions at the top of the cube. It is easy to drag dimensions to different locations at any time, so the initial placement is not critical. Typically, the main factor in deciding whether to put a dimension on a column or row is the size of the dimension. More rows than columns can be displayed on the table, so larger dimensions are easier to read as columns.

Figure 3.20 shows one variation of the initial cube, formed by placing the Sale Price in the middle, Model Type as columns, and Employee ID as rows. The figure shows the tree structure of the measures and dimensions in the right panels. Any of these items can be expanded and the specific dimensions dragged onto the cube. The cube computes subtotals for each dimension. In the example, a single cell shows the total sales value of a specific Model Type made by the Employee

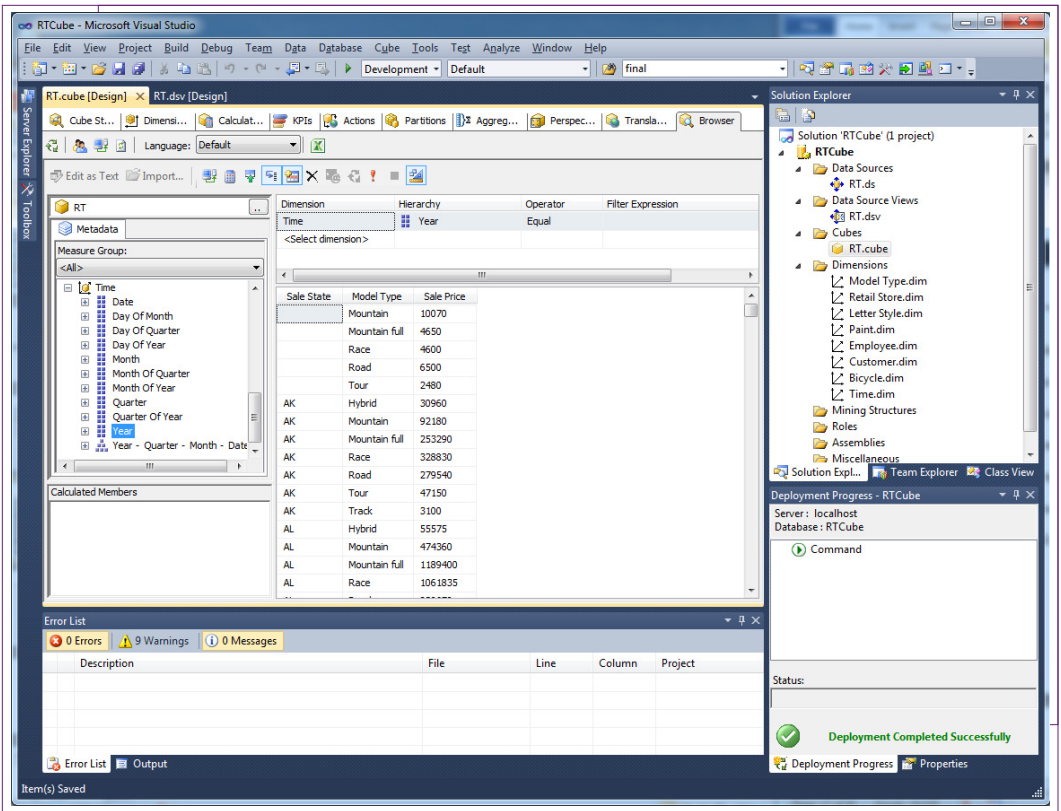


Figure 3.20

Browse the initial cube. Drag the Sale Price measure to the middle/values. Drag the Model Type dimension to the top/columns. Drag the Employee ID dimension to the left/rows.

shown on the row. Notice the Order Date is not included yet, so the values represent totals for the entire time the company has been in business. Note the difference with the SSAS cube browser, where the PivotTable can display dimensions as columns.

Dimensions

How are dimensions created and modified to improve browsing? The initial cube is somewhat disappointing. Check out the dimension choices: Employee ID, City ID, Letter Style (ID), Paint (ID), Retail Store ID, and so on. Where are the actual descriptive names? What about hierarchies? Sale Date needs to be configured into at least Year-Quarter-Month-Day, and it would be nice to track customer location by state and city name. And it would be nice to format the totals to remove the decimals and add commas to make the values easier to read. All of these improvements need to be made to the cube's dimensions.

The last one, formatting is probably the easiest to fix, so consider that one first. Switch to the Cube Structure tab (the first one on the left). In the Measures panel, expand the Bicycle entry to see the three items that were selected for this cube. Select the Sale Price entry and open the Properties panel. Under the Basic section,

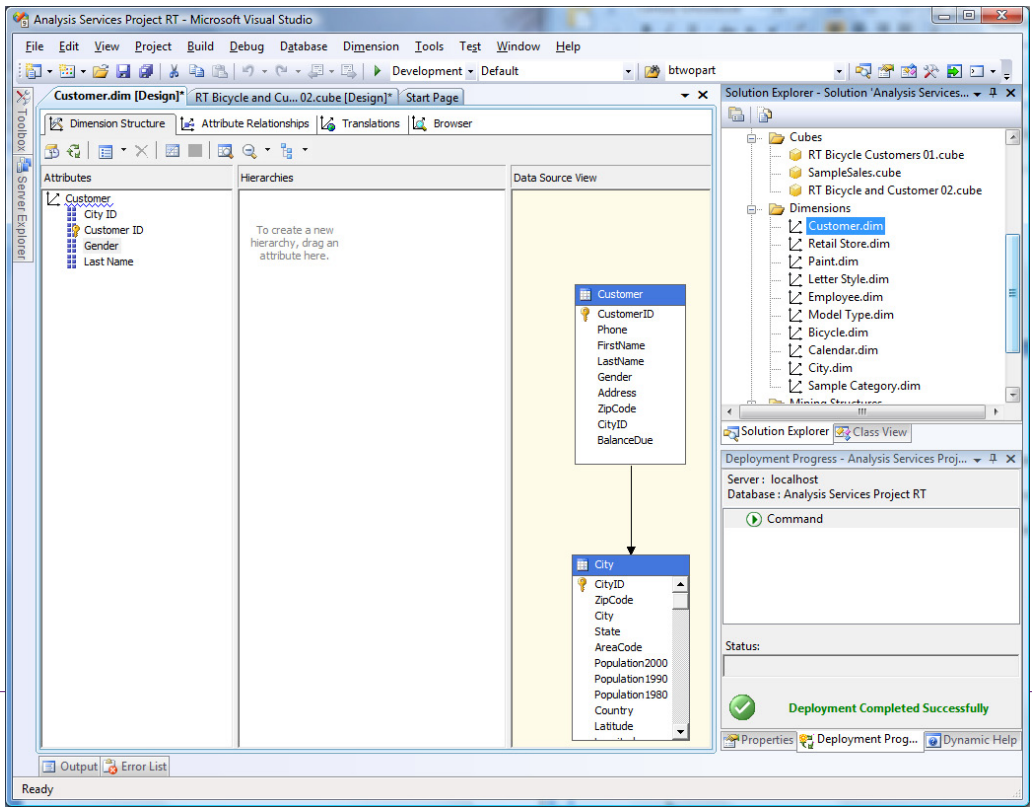


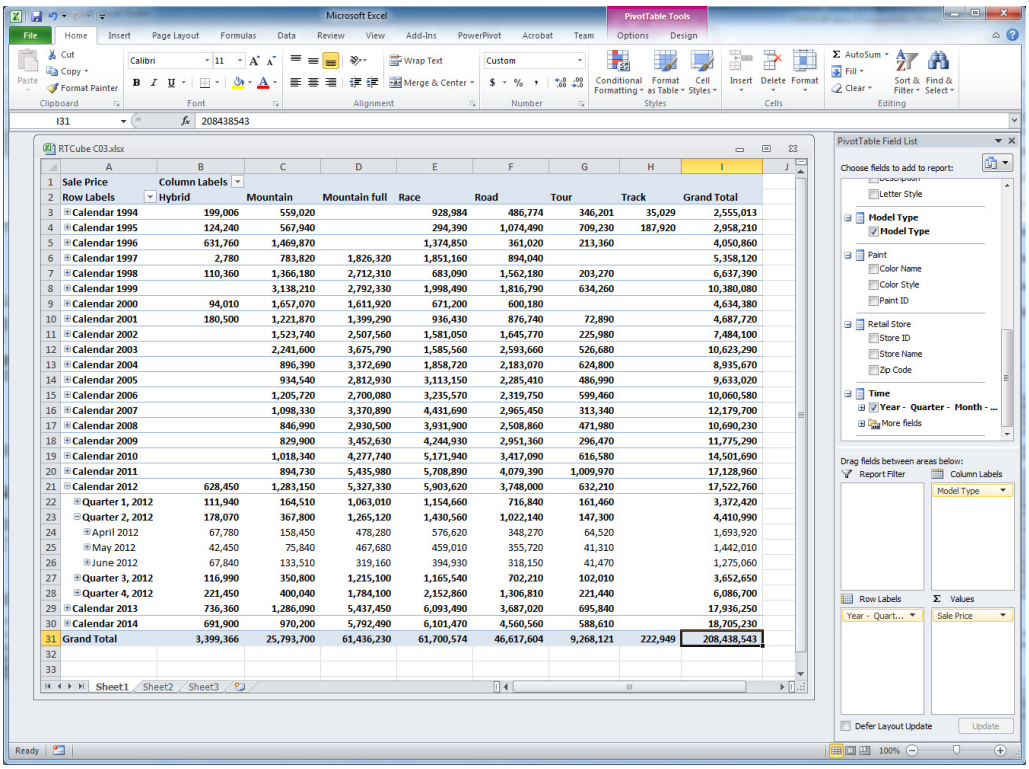
Figure 3.21

Edit the Customer dimension. Double-click the Customer dimension in the main list to open it. Drag the Gender and LastName columns from the Customer table into the Attribute panel on the left.

change the FormatString to: #,##0;-#,##0 which adds commas and removes the decimal places. Repeat the process for List Price. Bicycle Count is probably fine without formatting because the numbers are not too large. To test your changes, Process the cube and switch back to the Browser tab. Click the Reconnect icon to refresh the data.

It is more important to fix the dimensions so that managers can see the names of the dimensional values instead of just the ID numbers. No one wants to memorize lists of ID numbers. All of these require editing the underlying dimensions. First, understand that all of the dimensions are defined independently from the cube. Dimensions exist outside of the cube and then are used within any cube. This process might seem strange at first, but it makes the dimensions reusable, so they can be defined one time and applied to many cubes.

Figure 3.21 shows the basic process for editing the Customer dimension. Open the list of dimensions in the Solution Explorer and double-click the Customer dimension to edit it. Drag the Gender and Last Name columns from the Customer table into the Attribute panel on the left. After the cube is reprocessed, these col-



Sale Price	Column Labels							
Row Labels	Hybrid	Mountain	Mountain full	Race	Road	Tour	Track	Grand Total
Calendar 1994	199,006	559,020		928,984	486,774	346,201	35,029	2,555,013
Calendar 1995	124,240	567,940		294,390	1,074,490	709,230	187,920	2,958,210
Calendar 1996	631,760	1,469,870		1,374,850	361,020	213,360		4,050,860
Calendar 1997	2,780	783,820	1,826,320	1,851,160	894,040			5,358,120
Calendar 1998	110,360	1,366,180	2,712,310	683,090	1,562,180	203,270		6,637,390
Calendar 1999		3,138,210	2,792,330	1,998,490	1,816,790	634,260		10,380,080
Calendar 2000	94,010	1,657,070	1,611,920	671,200	600,180			4,634,380
Calendar 2001	180,500	1,221,870	1,399,290	936,430	876,740	72,890		4,687,720
Calendar 2002		1,523,740	2,507,560	1,581,050	1,645,770	225,980		7,484,100
Calendar 2003		2,241,600	3,675,790	1,585,560	2,593,660	526,680		10,623,290
Calendar 2004		896,390	3,372,690	1,858,720	2,183,070	624,800		8,935,670
Calendar 2005		934,540	2,812,930	3,113,150	2,285,410	486,990		9,633,020
Calendar 2006		1,205,720	2,700,080	3,235,570	2,319,750	599,460		10,060,580
Calendar 2007		1,098,330	3,370,890	4,431,690	2,965,450	313,340		12,179,700
Calendar 2008		846,990	2,930,500	3,931,900	2,508,860	471,980		10,690,230
Calendar 2009		829,900	3,452,630	4,244,930	2,951,360	296,470		11,775,290
Calendar 2010		1,018,340	4,277,740	5,171,940	3,417,090	616,580		14,501,690
Calendar 2011		894,730	5,435,980	5,708,890	4,079,390	1,009,970		17,128,960
Calendar 2012	628,450	1,283,150	5,327,330	5,903,620	3,748,000	632,210		17,522,760
Quarter 1, 2012	111,940	164,510	1,063,010	1,154,660	716,840	161,460		3,372,420
Quarter 2, 2012	178,070	367,800	1,265,120	1,430,560	1,022,140	147,300		4,410,990
April 2012	67,780	158,450	478,280	576,620	348,270	64,520		1,693,920
May 2012	42,450	75,840	467,680	459,010	355,720	41,310		1,442,010
June 2012	67,840	133,510	319,160	394,930	318,150	41,470		1,275,060
Quarter 3, 2012	116,990	350,800	1,215,100	1,165,540	702,210	102,010		3,652,650
Quarter 4, 2012	221,450	400,040	1,784,100	2,152,860	1,306,810	221,440		6,086,700
Calendar 2013	736,360	1,286,090	5,437,450	6,093,490	3,687,020	695,840		17,936,250
Calendar 2014	691,900	970,200	5,792,490	6,101,470	4,560,560	588,610		18,705,230
Grand Total	3,399,366	25,793,700	61,436,230	61,700,574	46,617,604	9,268,121	222,949	208,438,543

Figure 3.22

A better cube using Excel PivotTable. The formatted values are easier to read and the dimensions use names instead of ID numbers.

umns will be available to be displayed on the cube. Test it by reprocessing and reconnecting the cube. Then replace the CustomerID dimension with Gender.

Use a similar process to edit the other dimensions and add descriptive columns for: Employee, Letter Style, Paint, and Retail Store. Save the changes and close the dimensions after making the changes to reduce the clutter in the editor. Reprocess the cube after making all of the changes. One quick note about names. The attributes for names are usually listed separately (Last Name and First Name). The dimension editor will treat these columns separately. If they need to be combined into a full name (such as Smith, John), it has to be done with a Named Calculation in the data source view. Then the new Full Name column can be used as a single attribute in the dimension. Rolling Thunder has only a few employees, so they can be identified by last name. Most organizations will want to use a longer identifier, and will probably need to include the ID, phone number, or date hired to distinguish employees who have the same first name and last name.

Figure 3.22 shows the results of the changes. The cube is better because the formatted values are easier to read and the dimensions have recognizable names instead of numbers. The sample cube shows how dimensions (Paint Color and Customer Gender) can be combined to examine multiple levels of detail. This version is better, but the cube still needs hierarchies for time and location.

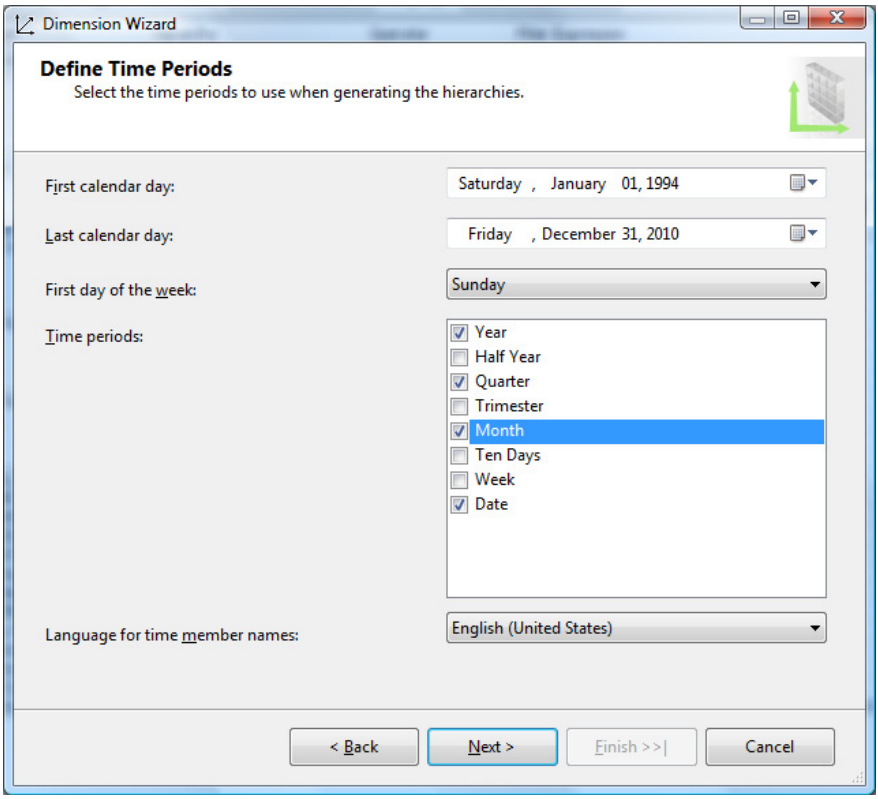


Figure 3.23

Creating a time hierarchy. Hierarchies on the analysis server require a starting and ending date, but it is okay to add extra dates. Choose the levels that will be used in the hierarchy.

Hierarchies

Many dimension attributes need to be created as hierarchies to show different levels. Three common hierarchies in business are: (1) Dates, (2) Location, and (3) Managers or divisions. Time and Geography are so popular that Analysis Services has wizards to help create them. Internal hierarchies by managers (or perhaps products) can be created but usually require custom definitions.

Hierarchies are built within dimensions and an important aspect of dimensions is that they are created independently from the cubes. They can be applied to multiple cubes. The same concept is important to hierarchies. A hierarchy is create separately and can be applied to multiple cubes. For example, the common Year-Quarter-Month-Day hierarchy applies to many dimensions. This process is handled by defining the hierarchy and all of its data levels as a separate dimension and then building a relationship between the lowest level (Day) and the underlying dimension. With this process the date hierarchy can even be applied to multiple dates within the same table, such as Order Date and Ship Date.

Think of hierarchies as lookup tables with a specific structure. The structure and the data have to be created and stored someplace, which is independent of the cube. Then the levels of the structure can be linked to a specific data cube.

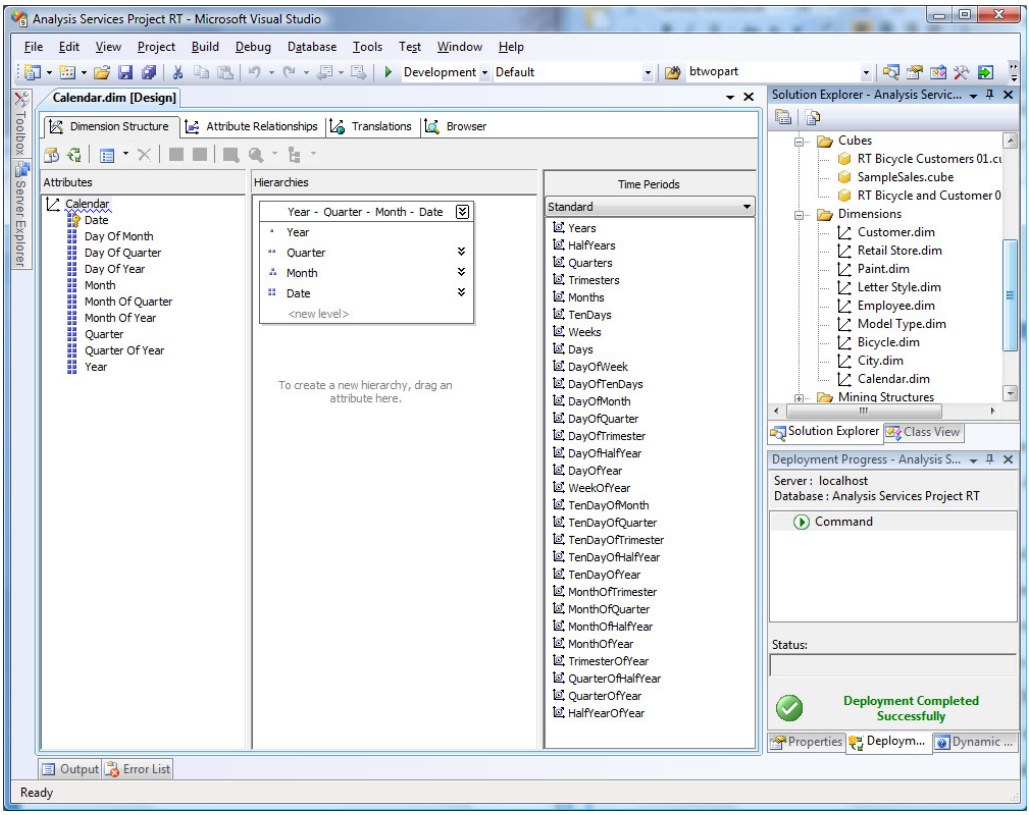


Figure 3.24

Calendar hierarchy. This dimension design form is the standard way to create and edit dimensions and hierarchies. Notice the hierarchy in the middle pane. All of the attributes in the left pane will be visible to the cube browser. Additional attributes in the table in the third pane can be dragged to the attributes pane if they are needed.

Time Dimensions

Time dimensions are so important to most organizations that Analysis Services has several methods to create time hierarchies. Remember that hierarchies are dimensions and all data displayed in the dimensions must be stored someplace. For time dimensions that means every year, quarter, month, and even day have to be predefined. However, because definitions of time levels are relatively standard, the dimension wizard can automatically create and populate tables with the data. The main decision you have to make is where you want to store the table: (1) Create a new table in the data source, or (2) Create a time table on the analysis server.

Because time hierarchies are the only ones that can be created on the server, that is the approach used in this chapter, simply to illustrate the process. Also, obtaining CREATE TABLE permission on the data source can be harder than it appears. To use the other option of creating a time table on the server, it is probably necessary to use the Windows login connection.

The one drawback to storing time hierarchies on the server is that the starting and ending dates have to be specified when the hierarchy is built. At a minimum,

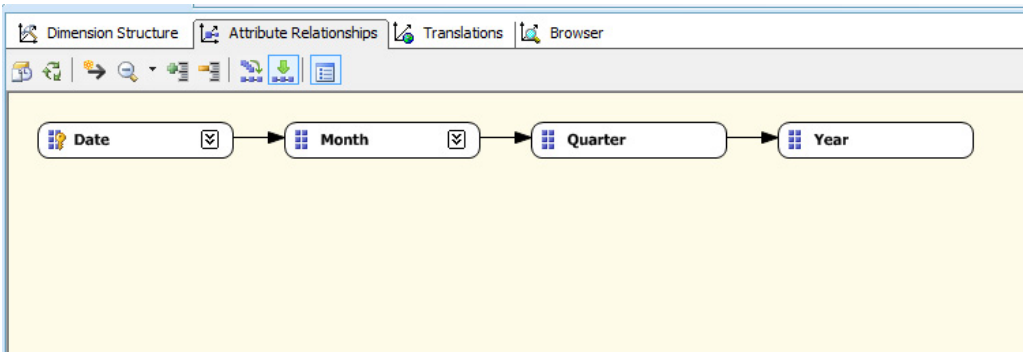


Figure 3.25

Attribute relationships. To improve performance, all levels in a hierarchy should be related in a chain from lowers to highest level. The relationship property should be fixed—signaled with a solid arrowhead.

the values need to include all of the dates in the database, so you need to know those values ahead of time. Fortunately, additional dates can be stored in the hierarchy. If dates in the hierarchy do not match the data, they are not displayed in the cube browser.

To create a time hierarchy, right-click the Dimensions heading in the Solution Explorer and choose the New Dimension option. The type of dimension is the first question that needs to be answered. Choose the option to generate the time table on the server. Figure 3.23 shows the form used to specify the details of the hierarchy. Because the table will be stored on the server, the dates are independent of any existing data, so you have to set the values for the starting and ending date. The Rolling Thunder database runs from 1994-01-01 through 2008-12-31 but like a real company, new data gets added over time. So set the ending date to the end of 2010. Later, when the hierarchy is used in the cube browser, dates that do not exist in the database will be hidden.

The time wizard supports several types of calendars. Some of the basic options can be configured on the main hierarchy screen—such as the starting day of the week and the language. Others are specified on the next screen. The two main choices are a Regular calendar and a Fiscal calendar, although several others exist, including the ISO 8601 calendar that displays dates in a standardized format. Note that it is possible to generate multiple calendars for the same hierarchy, so different users in the organization can choose a calendar to meet their individual needs. For now, stick with the Regular calendar.

Figure 3.24 shows the dimension hierarchy created by the wizard. This screen and the hierarchy are worth examining because the same form is used to create other hierarchies and it is nice to have a correct example to work with. The left pane shows the attributes that will be displayed in the cube browser. The terminology needs some explanation, although it is clear once the data is displayed. Specifically, what is the difference between Quarter and Quarter of Year? In Microsoft's terminology, Quarter is a specific quarter in the time period, such as 1994 Q1 or 2007 Q3. Quarter of Year is a generic quarter without the year, such as Q1—which represents the first quarter for all years. Similar definitions apply to the Month and Day variations.

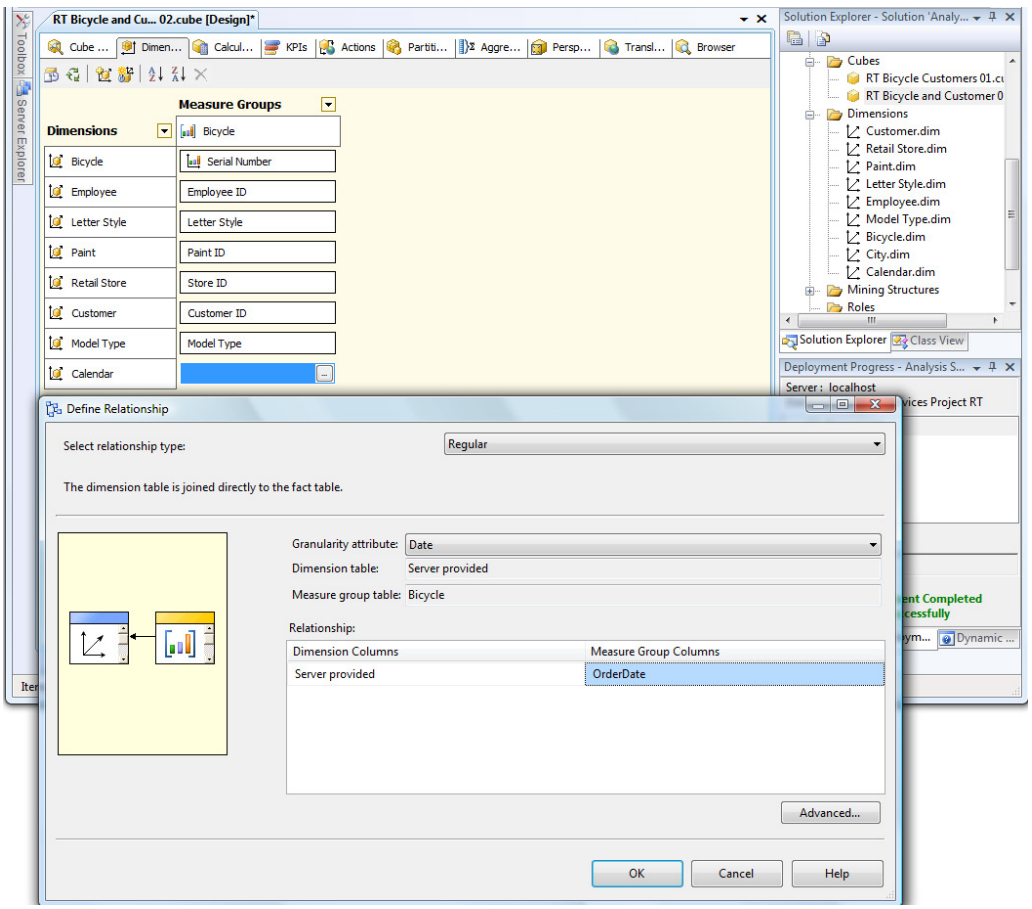
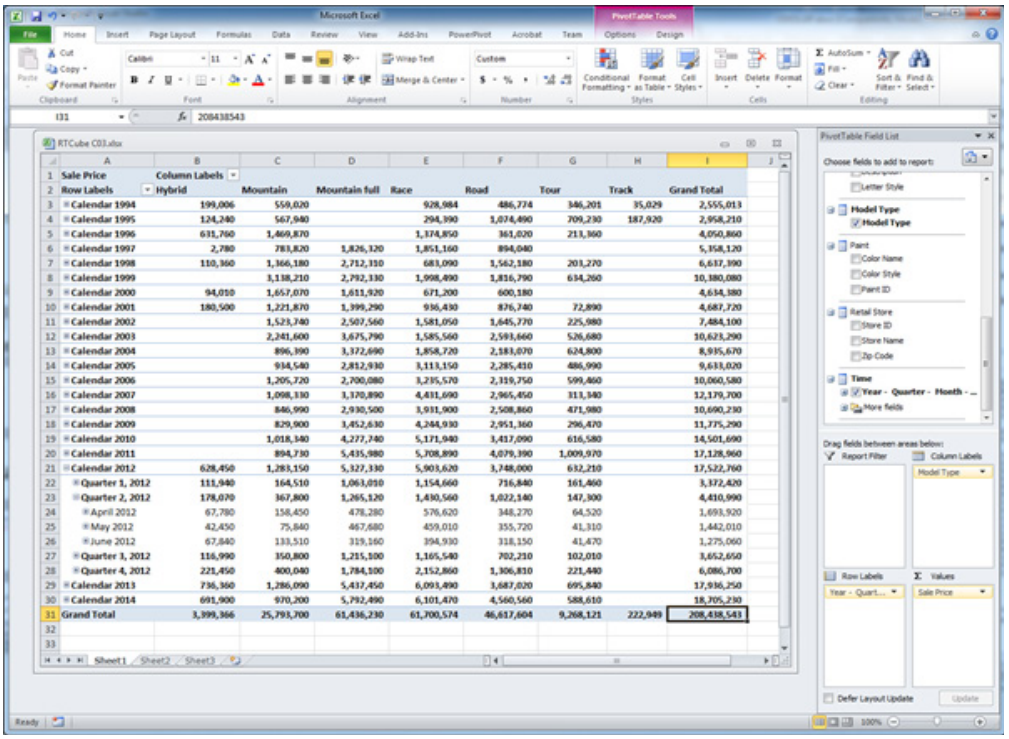


Figure 3.26

Create a relationship from the dimension to the cube. The ellipses button starts the wizard. Choose a Regular relationship in the drop-down list. Select the lowest level (Date) in the hierarchy list. Choose the OrderDate from the cube attributes to form the relationship between the cube and the dimension hierarchy.

The middle pane shows the actual hierarchy in top-down order. This hierarchy can be created or modified by dragging attributes from the left pane and dropping them at the appropriate level. Generally, it is best to work from the top level down to the lowest. The third pane shows all of the columns available in the data source table. Any of these can be added to the attribute list by dragging them over. However, because the wizard has finished, you will have to manually create levels with the new columns.

Technically, the hierarchy will be usable as long as it is defined correctly in the middle pane. However, to improve performance, Microsoft strongly recommends defining **attribute relationships** among the items in the hierarchy. Figure 3.25 shows the most efficient relationships are a simple chain from the lowest level to the highest. The wizard automatically created the attribute relationships, but for other problems you will have to build your own relationships. Relationships are displayed as the arrows and are created by selecting the lower-level node and



Sale Price	Column Labels							
Row Labels	Hybrid	Mountain	Mountain full	Race	Road	Tour	Track	Grand Total
Calendar 1994	199,006	559,020		928,984	486,774	346,201	35,029	2,555,013
Calendar 1995	124,240	567,940		294,390	1,074,490	709,230	187,920	2,958,210
Calendar 1996	631,760	1,469,870		1,374,850	361,020	213,360		4,050,860
Calendar 1997	2,780	783,820	1,826,320	1,851,160	894,040			5,358,120
Calendar 1998	110,360	1,366,180	2,712,310	683,090	1,562,180	203,270		6,637,390
Calendar 1999		3,138,210	2,792,330	1,998,490	1,816,790	634,260		10,380,080
Calendar 2000	94,010	1,657,070	1,611,920	671,200	600,180			4,634,380
Calendar 2001	180,500	1,221,870	1,399,290	936,430	876,740	72,890		4,687,720
Calendar 2002		1,523,740	2,507,560	1,581,050	1,645,770	225,980		7,484,100
Calendar 2003		2,241,600	3,675,790	1,585,560	2,593,660	526,680		10,623,290
Calendar 2004		896,390	3,372,690	1,858,720	2,183,070	624,800		8,935,670
Calendar 2005		934,540	2,812,930	3,113,150	2,285,410	486,990		9,633,020
Calendar 2006		1,205,720	2,700,080	3,235,570	2,319,750	599,460		10,060,580
Calendar 2007		1,098,330	3,170,890	4,431,690	2,965,450	111,340		12,170,700
Calendar 2008		846,990	2,930,500	3,931,900	2,508,860	471,980		10,690,230
Calendar 2009		829,900	3,452,630	4,244,930	2,951,360	296,470		11,775,290
Calendar 2010		1,018,340	4,277,740	5,171,940	3,417,090	616,580		14,501,690
Calendar 2011		894,730	5,435,980	5,708,890	4,079,390	1,009,970		17,128,960
Calendar 2012	628,450	1,283,150	5,327,330	5,903,620	3,748,000	632,210		17,522,760
Quarter 1, 2012	111,940	164,510	1,063,010	1,154,660	716,840	161,460		3,372,420
Quarter 2, 2012	178,070	367,800	1,265,120	1,430,560	1,022,140	147,300		4,410,990
April 2012	67,780	158,450	478,280	576,620	348,270	64,520		1,693,920
May 2012	42,450	75,840	467,680	459,010	355,720	41,310		1,442,010
June 2012	67,840	133,510	319,160	394,930	318,150	41,470		1,275,060
Quarter 3, 2012	116,990	350,800	1,215,100	1,165,540	702,210	102,010		3,652,650
Quarter 4, 2012	221,450	400,040	1,784,100	2,152,860	1,306,810	221,440		6,086,700
Calendar 2013	736,360	1,286,090	5,437,450	6,093,490	3,687,020	695,840		17,936,250
Calendar 2014	691,900	970,200	5,792,490	6,101,470	4,560,560	588,610		18,705,230
Grand Total	3,399,366	25,793,700	61,436,230	61,700,574	46,617,604	9,268,121	222,949	208,438,543

Figure 3.27

Sample cube with Calendar hierarchy. Click a + button to drill down to a detailed level. The cube is now easy to navigate for different time periods.

dragging it onto the next level. To create the first relationship in the example, you would drag the Date node and drop it onto the Month node. Notice that the arrow is solid—that signals that the relationship is fixed—the days in a specific month will never change. If necessary, this property is assigned by right-clicking the relationship arrow and setting its properties. No changes are necessary here.

Save the new Calendar dimension. Remember that it is currently a standalone dimension and is not linked to the cube. In fact, because it is a server-based dimension, it cannot be used until it is created. Save it, then right-click its name in the Dimension list of the Solution Explorer and choose the option to Process it. This method processes only the new dimension and generates the values for all of the dates. If any errors appear in processing, delete the new dimension and start over.

The generic Calendar dimension can now be assigned to the RT cube. Open the cube and switch to the Dimension (second) tab. Right-click the main window and choose Add Cube Dimension. Select the new Calendar dimension from the list of available dimensions. The calendar dates will now be available to the cube, but it is critical to link the calendar to a specific date in the data source view. As shown in Figure 3.26, select the gray box next to the new Calendar dimension and click the ellipses button. Choose Regular relationship from the drop-down list. A relationship links the hierarchy values to the data in the cube, so it is necessary to specify the attribute in the hierarchy that matches an attribute in the cube. For

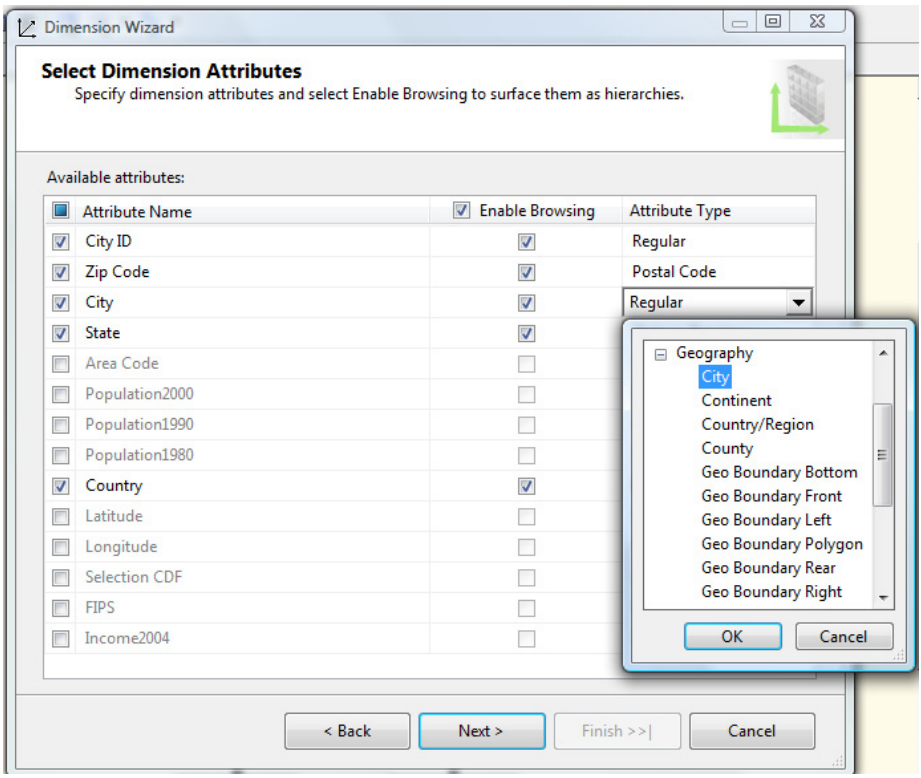


Figure 3.28

Assign geographic levels. The wizard understands common geographic levels and columns in the City table need to be assigned to the proper level by setting the Attribute Type. The primary key (CityID) must be included and it must be left as a "Regular" type.

dates, choose the lowest level in the hierarchy: Date. Then choose the Order Date in the cube to build the link. It is possible to repeat the process and add a calendar for any other dates in the cube (such as Start Date and Ship Date), but keep the cube simple for now and ignore those two dates. Save everything and process the cube. Browse the cube, reconnect if necessary, remove the existing dimensions and rebuild it with Model Type in the columns and the Calendar in the rows. Figure 3.27 shows a version of the cube using the PivotTable. It is now possible to drill down and examine sales at different time periods.

Custom Geographic Hierarchy

Geographic or location hierarchies are also common in many business problems. However, the geographic divisions are less standardized than the time divisions. The basics (Country, State, City, Postal Code) are relatively common, but the data that falls within those categories is more variable. Plus, companies often define regions or sales territories to the list and these are always defined differently. Even something as commonly used in the U.S. as "the Midwest" has many variations.

Analysis Services has a wizard to help define a geographic hierarchy, but Rolling Thunder Bicycles already has a City table that includes basic geographic data.

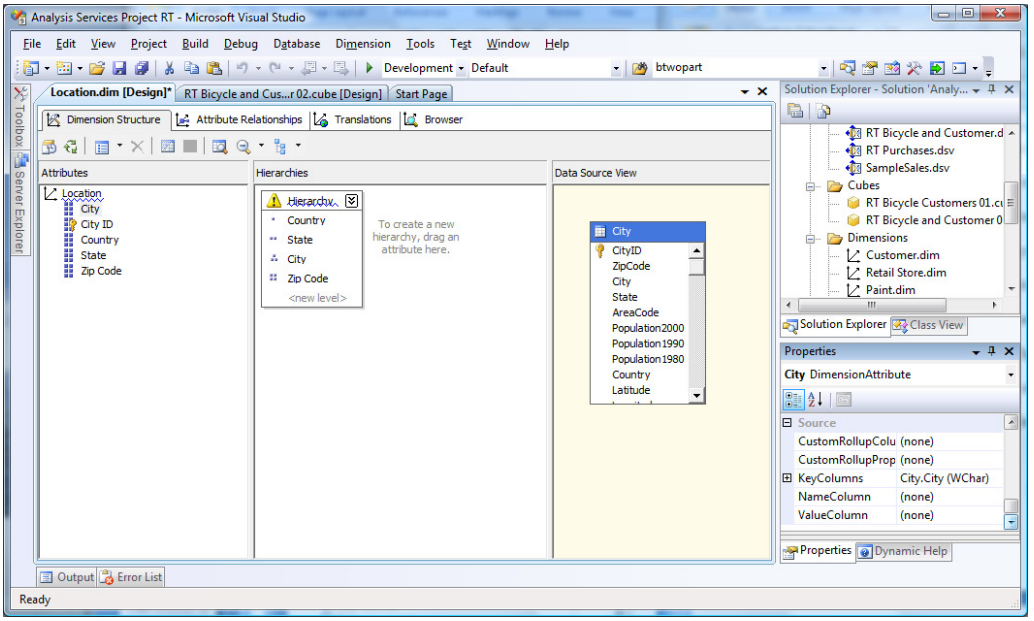


Figure 3.29

Building a hierarchy of attributes. The hierarchy begins as an empty panel. Drag the top level (Country) onto the middle panel. Drag State and drop it just below Country. Repeat the process for City and Zip Code.

Fortunately, tables of cities, states, and countries can usually be created from existing data. If a company has been in operation for years and collected customer data, the internal databases should already contain lists of addresses that can be extracted to form a City table complete with ZIP (Postal) codes. City databases can also be found online, some government agencies provide standardized databases. The Rolling Thunder Bicycle case was built with a City database that includes relatively detailed information. But, if you look closely at the data, you might spot one issue. The ZIP code aspects of the RT data are okay, but not perfectly realistic. Cities can have multiple ZIP codes, and in a few cases, a ZIP code can apply to multiple cities. This many-to-many relationship is difficult to handle in a database that relies on generating data.

Creating a geographic hierarchy is straightforward. Begin by right-clicking the Dimension entry in the Solution Explorer and choosing New Dimension. Choose the option to "Use an existing table." On the second screen, select the data source view and choose City as the main table. The wizard automatically picks up the CityID column as the primary key.

Figure 3.28 shows the screen where columns from the City table are assigned to specific geographic levels. The common values of City, State, Country, and ZIP code are assigned to City, State or Province, Country/Region, and Postal Code. Any hierarchy must also include the primary key (CityID) and its attribute type remains as Regular. This key value will be used to link the hierarchy dimension to data stored in other tables, such as Customer, Employee, and Manufacturer. The final screen shows the attributes and has a box for changing the name. Keep the name simple, such as Location, because the hierarchy is generic and can be used for different tables.

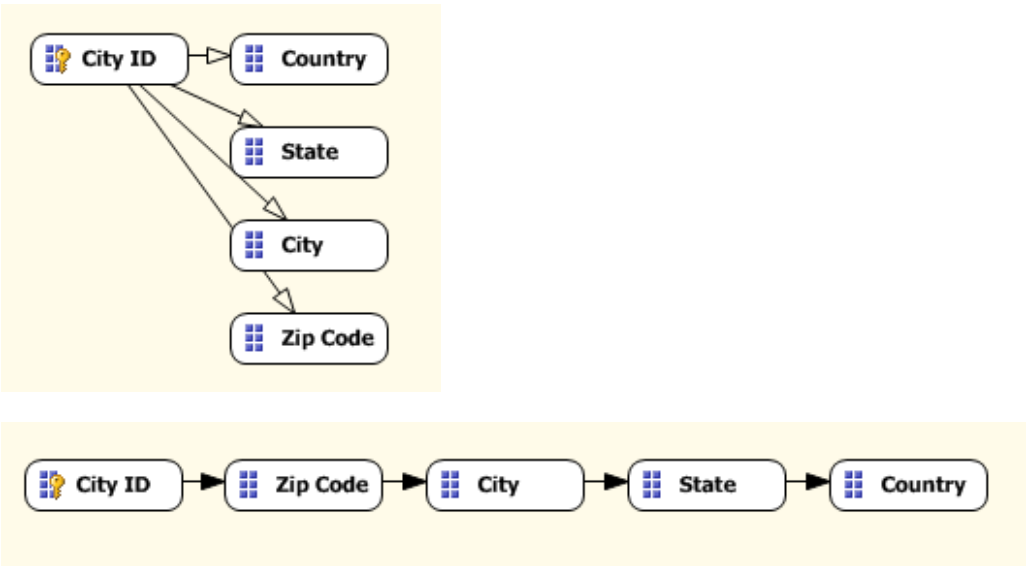


Figure 3.30

Build attribute relationships to improve performance. In the initial diagram at the top, work from the bottom up. Drag Zip Code and drop it onto City, then City onto State, and State onto Country. Double-click each arrow and select the Fixed relationship until the diagram matches the one at the bottom.

Building a dimension from an existing table does not automatically create the hierarchy. The attributes still need to be organized. Figure 3.29 shows the basic edit screen. Drag attributes from the left panel onto the middle pane (hierarchies). Begin with the top level (Country), and drag State onto a new level below Country. Repeat the process down to the ZIP Code attribute.

Attribute Relationships

Building hierarchies from scratch requires another critical step. As shown in Figure 3.30, the attributes need to be linked together with relationships to improve performance. Switch to the Attribute Relationships tab. Initially, all of the attributes are tied to only the CityID. The hierarchy needs to be specified in terms of relationships. In terms of the arrows, it is built from the bottom up: CityID à Zip Code à City à State à Country. The relationships are created by dragging the lower-level and dropping it onto the next level (Zip Code onto City and so on).

Also, to improve performance, the relationships should be defined as rigid instead of flexible (the default). Double-click a relationship arrow and change the setting in the drop-down list. The arrowhead will be filled in to indicate the strong relationship. If you make a mistake when building relationships, a relationship can be deleted by selecting an arrow and pressing the Delete key.

Now comes the tricky part. Try to process the new dimension. An error message will be generated and the processing halted. Attribute relationships follow a very precise rule: There must be a one-to-many relationship between each level. For example, a State can have many Cities, but a City can exist in exactly one State. This relationship holds if "City" is defined as CityID, but City in the relationships

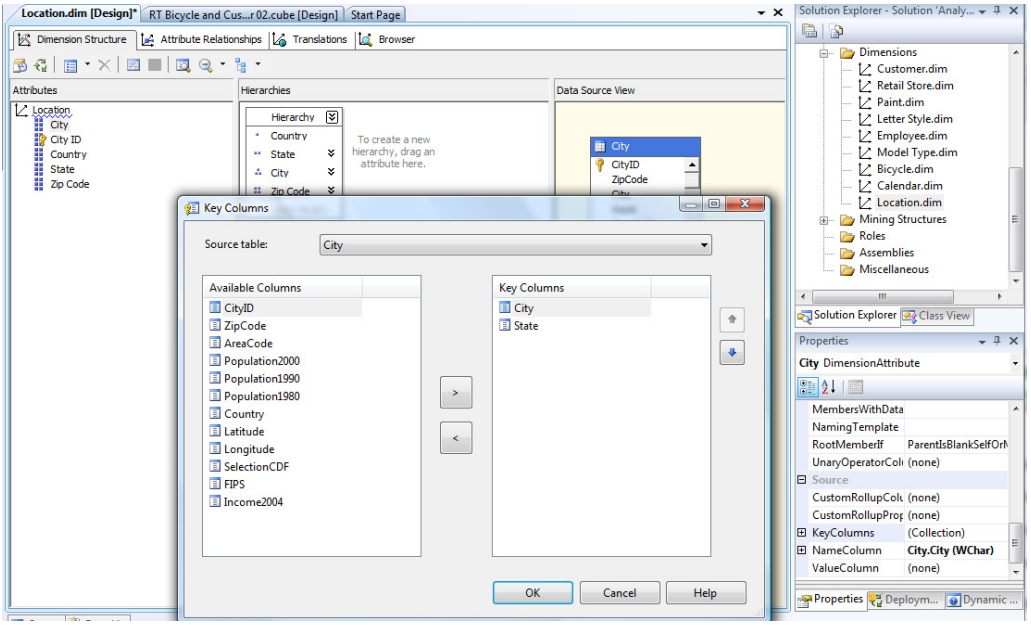
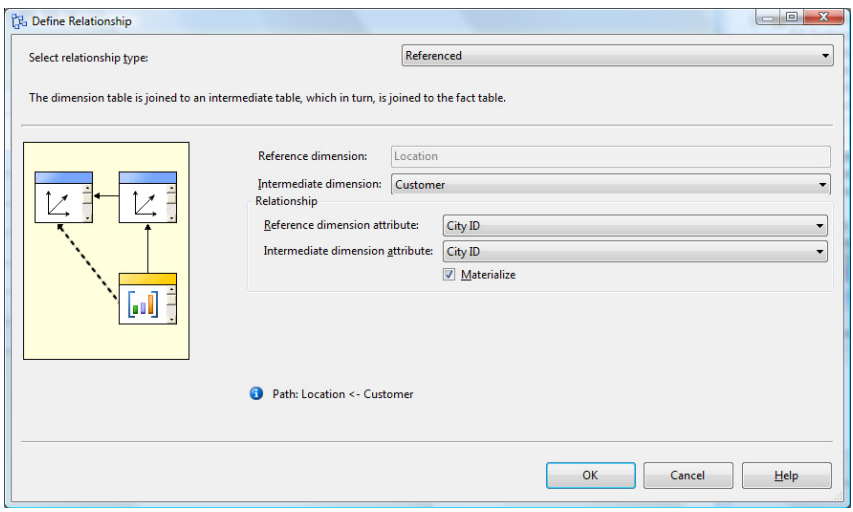


Figure 3.31

Adding key columns to create a one-to-many relationship. Because a City name can exist in many states, use the KeyColumns property to add State to the key so that State has a one-to-many relationship with City+State. With multiple columns in the Key, the Name column also must be set (to City).

Figure 3.32

Hierarchy relationship to data is referenced through Customer. Because of the snowflake design, a referenced relationship is needed to specify the intermediate Customer table.



Bicycle Count	Column L					
Row Labels	Mountain	Mountain full	Race	Road	Tour	Grand Total
⊞	2			2	1	5
⊟USA	7515	13065	11103	9717	2397	43797
⊞	1	1	1			3
⊞AK	22	54	47	55	10	188
⊞AL	136	240	148	180	36	740
⊞AR	94	172	110	134	26	536
⊞AZ	84	150	128	110	30	502
⊟CA	508	981	1216	709	180	3594
⊞Acton	3	1		1		5
⊞Adelanto		1	1			2
⊞Agoura Hills			1	1		2
⊞Alameda	6	10	5	7		28
⊞Alamo				1		1
⊞Albany	1	1				2
⊞Alhambra	1	1	3	4		9
⊞Aliso Viejo	2	3	8	1		14
⊞Allendale CDP				1		1
⊞Alondra Park	1				1	2
⊞Alpine	1		1			2
⊞Alta Sierra			1			1
⊞Altadena		2	2			4
⊞Alturas	1		1	1		3
⊞Anaheim	2	3	9	2		16
⊞Anderson	1					1

Figure 3.33

Sample cube with location hierarchy. The heading drop-down lists can be used to remove little-used data such as Italy, Unknown, and Blank for Country.

is the name of the city, and clearly the same city name can exist in many different states or countries (for instance, Paris is in Texas, Tennessee, and France).

Figure 3.31 shows how to solve this problem by adding a second column to the key for City (and any other attribute with the same problem). In the dimension editor, switch to the Dimension Structure tab. Select the City attribute in the left pane. Open the Properties window in the bottom right pane. Scroll down and select the KeyColumns entry. Click the ellipses button. If you are unsure of which values are many-to-many, try to process the dimension and watch for error messages, then modify the specified column. In this case, State needs Country added, and Zip Code needs the City.

The City table has now been converted into a dimension hierarchy. The final step is to attach that dimension to the cube. Close the dimension editor and open the cube editor. Switch to the Dimension tab, right-click the main window and choose to add a cube dimension. Select the new Location dimension which adds it to the display. Click the ellipses button in the gray box to the right of the Location dimension to open the relationship editor. The process for creating this relationship is slightly different than it was for the time dimension. Remember that the CityID is in the Customer table, not the Bicycle measure table. This link from City to Customer to Bicycle is a feature of the snowflake diagram.

As shown in Figure 3.32, change the relationship type from "No relationship" to "Referenced." Set Customer as the intermediate dimension. The reference dimension attribute is CityID and the intermediate dimension attribute is also CityID.

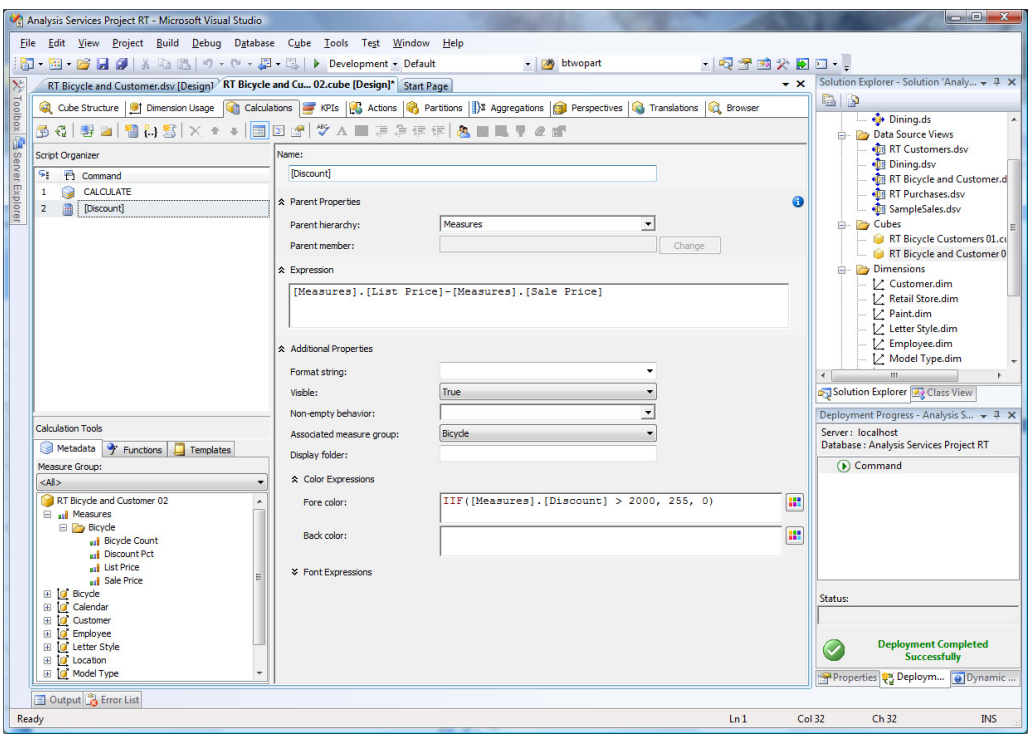
Finally, save everything, process the cube, build a PivotTable, and browse it. Use Location as the row dimension and ModelType as the column dimension. Use Bicycle Count as the value. The heading Country has a drop-down arrow that can be used to remove small entries for Unknown, Blank, and Italy. Expand a state and check the cities and ZIP codes. Almost all of the cities have only a single Zip code in the database—a consequence of the data generator relying on a single ZIP code when generating city data. Figure 3.33 shows the final cube.

Fine Tuning the Cube

How can the cube provide more information? Think of an OLAP cube as an easy way for managers to explore the data by examining various subtotals or groupings of the data. Of course, the filters are also useful for showing slices of the cube to examine specific details or categories. But, the cube browser is restricted to the data measures and dimensions that have been predefined. Consequently, as the designer, you need to think about all of the possible things that

Figure 3.34

Calculated measure. Right-click the Script Organizer window and create a new calculation. Enter a name (Discount) and assign it to the Measure hierarchy. Drag attributes from the list on the lower left and drop them in the Expression window. Add the minus sign. Assign it to the Bicycle group. Use the IIF function to set the color to Red for discount totals over 2000.



	A	B	C	D	E	F	G
1	Discount	Column Labels					
2	Row Labels	Mountain	Mountain full	Race	Road	Tour	Grand Total
3	⊞Calendar 1994	3638.39		3361.51	2993.84	2108.4	12102.14
4	⊞Calendar 1995	1349.04		523.2	1933.02	1577.82	5383.08
5	⊞Calendar 1996	4167.95		2886.1	743.04	471	8268.09
6	⊞Calendar 1997	1576.8	3561.99	3380.12	1625.12		10144.03
7	⊞Calendar 1998	2948.79	6138.03	1416.24	3111.83	450.79	14065.68
8	⊞Calendar 1999	4551.26	4015.33	2779.61	1898.54	741.26	13986
9	⊞Calendar 2000	2787.18	2757.42	1107.14	1092.88		7744.62
10	⊞Calendar 2001	1860.35	2136.92	1668.99	1411.26	143.19	7220.71
11	⊞Calendar 2002	2463.05	3937.2	2591.46	2659.83	472.82	12124.36
12	⊞Calendar 2003	2988.88	5053.02	1891.96	3450.06	941.28	14325.2
13	⊞Calendar 2004	1486.24	4509.17	2131.16	2750.25	938.08	11814.9
14	⊞Calendar 2005	1479.42	3837.19	3460.91	3143.24	778.35	12699.11
15	⊞Calendar 2006	1432.01	2941.79	2932.47	2472.16	813.92	10592.35
16	⊞Calendar 2007	1319.33	3674.88	3721.76	2920.11	309.79	11945.87
17	⊞Calendar 2008	955.8	3109.3	3551.64	2487.34	447.79	10551.87
18	⊞Calendar 2009	758.84	2773.95	2589.18	2315.95	210.88	8648.8
19	⊞Calendar 2010	864.03	3278.91	3076.01	2430.78	454.31	10104.04
20	⊞Calendar 2011	727.57	3652.42	3405.22	2821.64	630.33	11237.18
21	⊟Calendar 2012	773.07	3236.72	3516.8	2817.57	454.54	10798.7
22	⊞Quarter 1, 2012	112.12	633.33	680.17	567.28	99.86	2092.76
23	⊞Quarter 2, 2012	217.28	781.19	811.33	750.22	117.52	2677.54
24	⊞Quarter 3, 2012	214.3	667.67	705.53	485.53	70.52	2143.55
25	⊞Quarter 4, 2012	229.37	1154.53	1319.77	1014.54	166.64	3884.85
26	⊞Calendar 2013	698.57	3091.21	4027.3	2757.39	561.43	11135.9

Figure 3.35

Cube results for Discount. Notice that the color conditions and the calculations apply to the total values as they are displayed. With VS 2012, these colors transfer to Excel but do not display in the VS cube browser.

managers will want to see. Note that it is important for the designers to actually talk with the managers while defining cubes.

But, what happens if managers need values that are not in the underlying database? Specifically, managers and analysts often need calculated values such as Profit = Revenue – Cost or Discount = ListPrice – SalePrice. **Calculations** and queries

Also, what happens when a cube gets too big with dozens of measures and attributes? **Perspectives** are used to create multiple views of a cube so that each perspective shows a smaller set of the dimensions and attributes—specifically for one group or type of problem.

What about companies that operate in multiple countries and languages? Internationalization is important, and much of the work has to be carried through the entire database. However, cubes have some functions that provide support for translations. Note that some functions described in this section are supported only on the Enterprise and Developer versions of SQL Server. The Standard version provides limited support for some types of calculations and most optimizations. If some examples do not work on your configuration, verify the version you are using.

Calculations and Queries

Many problems require computing new values based on existing data. The cube designer provides the ability to specify calculations. Because the cube is based on

	X	Y	X*Y
	100	2	200
	5	10	50
Sum	105	12	1,260 or 250

Figure 3.36

The order of operations. It makes a big difference if numbers are added first and then multiplied or multiplied first and then summed.

a data source view which behaves as a query, it is also possible to compute new columns using SQL statements as named calculations within the data source view. Always remember that the cube is designed to display aggregate values—usually totals. Combining calculations with totals can cause serious errors. It is critical to understand how calculations work—and the lack of documentation makes it a challenge.

Cube Calculations

Consider an easy example first, where managers want to explore the value of the discount given on each bicycle. Discount = ListPrice – SalePrice. The reason this example is easy is because it is not the percentage discount. The calculation involves simple subtraction. As shown in Figure 3.34, select the Calculations tab and right-click the Script Organizer window to add a New Calculated Member. Name it Discount and assign it to the Measures hierarchy. Drag the ListPrice and SalePrice columns from the list in the lower-left window into the Expression window and add the minus sign. Assign the new measure to the Bicycle group. Just for fun, open the Color Expressions properties. For the foreground color, enter the **immediate if function (IIF)** to set the color to red if the total discount exceeds 2000:

```
IIF([Measures].[Discount] > 2000, 255,  0)
```

The IIF function takes three parameters and is similar to the IF function in Excel. The first is a conditional test. The second is the value returned if the condition is true. The third is the value returned if the condition is false. The color values 255 and 0 represent Red and Black. Use the color picker next to the box to obtain more complex color numbers.

Save everything and process the cube. Transfer the cube to Excel and create a new display cube with the Discount as the value totals, ModelType as the column dimension, and Calendar as the row dimensions. Figure 3.35 shows the results. Notice that the color conditions apply to the totals as they are shown. This result makes sense, but it does complicate the choice of the conditional value (2000). As the user drills down by quarter and month, the values will likely fall below the critical number. Conditional color adds some emphasis to the cube display, but it will only work at a certain level.

A more important problem arises but is invisible with this example. It turns out that the defined calculation (Discount) is also applied to the totals or visible data. The Discount value is computed at each displayed level as: Sum(ListPrice) – Sum(SalePrice) for that level. Because the calculation involves only subtraction (or addition), this method of calculation is fine. From basic arithmetic:

```
Sum(ListPrice - SalePrice) = Sum(ListPrice) -
Sum(SalePrice)
```

But this method of calculation can be seriously wrong if the calculation uses multiplication or division. Figure 3.36 shows a simple example with two columns and two rows of data. Because of the **order of operations**, it makes a big difference if the numbers are added first and then multiplied or multiplied first and then added. In the example, the two calculations are 1,260 versus 250. It is absolutely critical to remember that calculations defined within the cube are based on first computing the total and then performing the calculation. In the example, if the cube calculation is X*Y, the result would be 1,260 which is Sum(X)*Sum(Y). Essentially, the calculations are performed on the values as they are shown in the cube—not on the detailed rows. Once in a while, this order of calculation makes sense—for example, to compare one subtotal to another. Much of the time, this level of calculation leads to huge errors in interpretation. So, remember a simple rule: Only use addition and subtraction in calculations defined on the cube. Never use multiplication or division. The discount calculation shown in this section is fine because it used only subtraction.

But some problems require the use of multiplication and division! The answer is to define those at the row level of the data—in the data source view or at the database level in a query. For example, a cube-level calculation computes the sum first and then divides, while a query-level calculation computes the division first and then sums (or averages)

$$\frac{\sum SalePrice}{\sum ListPrice} \neq \sum \frac{SalePrice}{ListPrice}$$

Figure 3.37

Standard cube using the Discount Percent. But the totals are greater than one. By default, the cube totals all data at lower levels. Totals do not make sense for percentages.

			Model Type					
			Mountain	Mountain full	Race	Road	Tour	Grand Total
Year	Quarter	Month	Discount Pc	Discount Pc	Discount Pc	Discount Pc	Discount Pc	Discount Pc
⊞ Calendar 2000			0.97759999	0.98159999	0.4197	0.42900000		2.8079
⊞ Calendar 2001			0.6066	0.69659999	0.5936	0.5054	0.0564	2.4586
⊞ Calendar 2002			0.86849999	1.3564	0.8655	0.9756	0.217	4.28299999
⊞ Calendar 2003			0.84939999	1.46229999	0.5007	1.0559	0.3738	4.24209999
⊞ Calendar 2004			0.5544	1.3801	0.523	0.74439999	0.3069	3.5088
⊞ Calendar 2005			0.5292	1.1938	0.85019999	0.88199999	0.2532	3.70839999
⊞ Calendar 2006			0.3842	0.711	0.568	0.5522	0.2176	2.433
⊞ Calendar 2007			0.348	0.88139999	0.69139999	0.59449999	0.0813	2.5966
⊟ Calendar 2008	⊞ Quarter 1, 2008		0.0682	0.1872	0.1968	0.1234	0.0236	0.5992
	⊞ Quarter 2, 2008		0.0573	0.1934	0.1885	0.1568	0.0204	0.6164
	⊞ Quarter 3, 2008		0.053	0.1779	0.1556	0.105	0.0261	0.5176
	⊞ Quarter 4, 2008		0.0627	0.1833	0.161	0.1562	0.0332	0.5964
	Total		0.2412	0.74179999	0.7019	0.54140000	0.1033	2.3296
Grand Total			13.9648	14.9075	11.2047	11.1553	4.0707	55.30299999

9	⊞Calendar 2000	0.35%	0.36%	0.25%	0.25%	0.78%
10	⊞Calendar 2001	0.26%	0.29%	0.28%	0.26%	0.70%
11	⊞Calendar 2002	0.36%	0.46%	0.36%	0.37%	0.28%	1.19%
12	⊞Calendar 2003	0.32%	0.47%	0.23%	0.41%	0.25%	1.18%
13	⊞Calendar 2004	0.28%	0.45%	0.22%	0.27%	0.23%	0.96%
14	⊞Calendar 2005	0.26%	0.38%	0.28%	0.30%	0.21%	1.02%
15	⊞Calendar 2006	0.18%	0.24%	0.19%	0.21%	0.19%	0.67%
16	⊞Calendar 2007	0.18%	0.28%	0.23%	0.21%	0.12%	0.72%
17	⊞Calendar 2008	0.15%	0.25%	0.22%	0.20%	0.11%	0.64%
18	⊞Calendar 2009	0.12%	0.16%	0.13%	0.15%	0.08%	0.38%
19	⊞Calendar 2010	0.11%	0.17%	0.14%	0.14%	0.09%	0.43%
20	⊞Calendar 2011	0.11%	0.17%	0.14%	0.15%	0.08%	0.44%
21	⊟Calendar 2012	0.09%	0.14%	0.16%	0.16%	0.09%	0.43%
22	⊞Quarter 1, 2012	0.10%	0.11%	0.14%	0.15%	0.07%	0.34%
23	⊞Quarter 2, 2012	0.08%	0.14%	0.14%	0.16%	0.09%	0.43%
24	⊞Quarter 3, 2012	0.09%	0.13%	0.14%	0.12%	0.08%	0.35%
25	⊞Quarter 4, 2012	0.09%	0.19%	0.22%	0.21%	0.10%	0.62%
26	⊞Calendar 2013	0.07%	0.13%	0.18%	0.16%	0.10%	0.44%

Figure 3.38

Discount Percent as an average. The discount percentage is computed at row-level in the data source view and the cube averages the values from the lower levels.

Query-Level Calculations

Most business managers looking at discounts prefer to examine percentages. But the calculation for discount percent uses division: 1 – SalePrice / ListPrice. Which is the reason this computation was introduced in the earlier section about data source views. You should have already defined this Discount Percent as a named calculation within the Bicycle table. If not, open the data source view and add it to that table. Calculations performed within a query are handled at the row level, performed for each row of data before any aggregations take place.

To see the effect, return to the cube browser. Drag the Discount calculation out of the cube and replace it with the Discount Percent. Figure 3.37 shows the result. Look at the grand totals and notice the problem. How can percentages be greater than one? The answer is that by default the cube browser computes the sum of all lower-level values.

The sum of percentages does not make much sense to a business analyst. It would be better to use averages. Fortunately, the cube browser knows how to compute averages. Unfortunately, because it is an aggregation, it means the cube has to be reprocessed. Switch to the Cube Structure tab and select the Discount Pct measure. Open the Properties window and change the Aggregate Function from Sum to AverageOfChildren. Also, set the FormatString to Percent to make the results easier to read.

Figure 3.38 shows the final cube. It might be nice to add conditional color to highlight larger discounts. But, conditional color is available only for cube calculations. It can be added by calculating a new cube measure that is exactly equal to the Discount Pct column, and apply color to the new value.

Calculations are useful, but it is crucial that you understand the difference between computing values row-by-row in a query and performing cube calculations

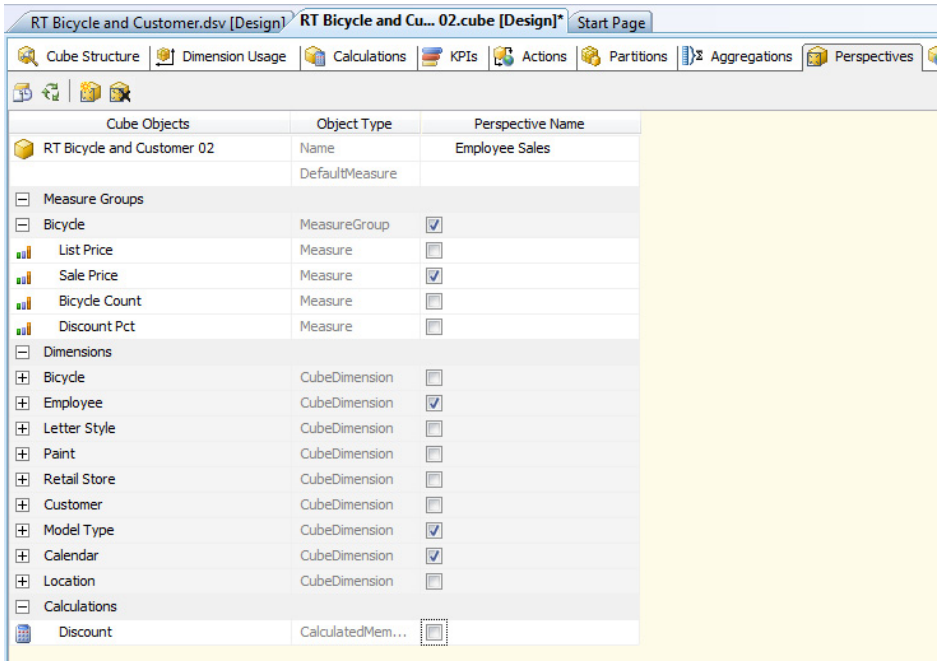


Figure 3.39

Creating a perspective. Right-click the main screen to create a new perspective. Name it (Employee Sales) and check only the values that should be available.

at the level of the totals. And it is important to document your calculations. Managers and analysts using the cube need to know which method was used to perform the calculations.

Perspectives

Data and cubes can quickly become huge when multiple measures and many dimensions are available. Putting every possible attribute into one cube is going to make the cube hard to understand. A common solution to handling complex problems is to break them into smaller pieces. Analysis services provides several mechanisms for segmenting a problem. One is to create separate data source views for each specific problem. Related to that approach, separate OLAP cubes can be defined. Each cube can be built for a specific problem or group of managers. It is straightforward to assign security permissions to each cube, and when data changes, each cube can be rebuilt individually. Hence, cubes can be built for specific tasks to be managed by different groups. On the flip side, if the cubes predominantly use the same data, rebuilding each cube wastes processor time by rebuilding the same data multiple times.

Consider the case where basic data needs to be shared among several different groups, but some dimensions should be seen only by a few users. For instance, perhaps HRM and some top managers should be the only ones to see employee sales data. The answer is to define perspectives on the cube. A perspective is a view of the cube that contains a subset of the available measures and attributes. Figure 3.39 shows the basic steps to create a perspective. Open the Perspectives

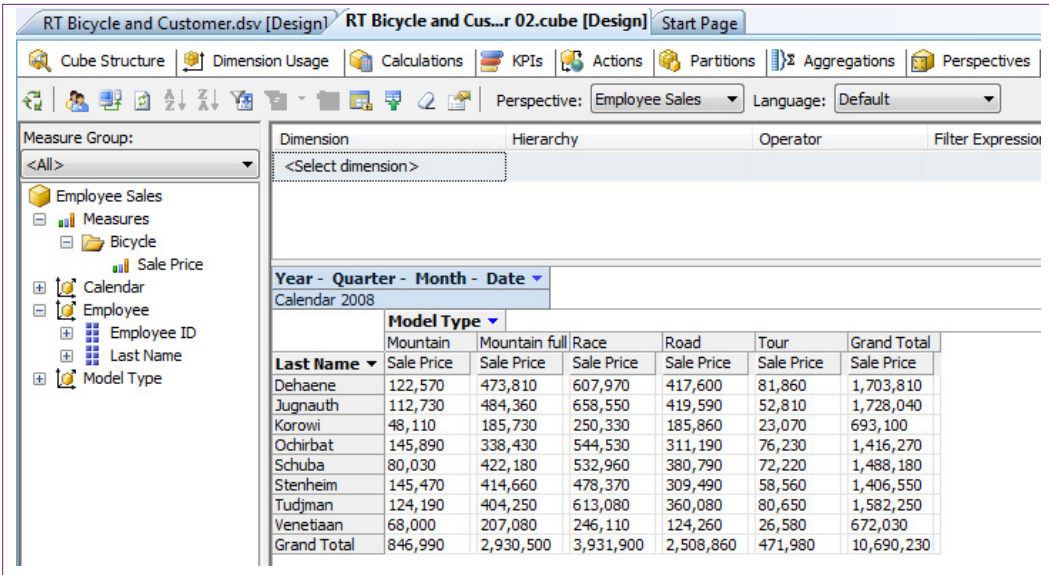


Figure 3.40

A perspective reduces the measures and dimensions visible to the user. Use the drop-down box to select the desired perspective.

Figure 3.41

Translating dimensions. Attributes stored in tables is translated and stored in new columns in the same table.

PaintID	ColorName	SpanishName
1	Neon Blue	Neón Azul
2	Arctic White	Blanco Ártico
3	Sea Green Fade	Verde Mar
4	Black Speckle	Salpicaduras Negras
5	Candy Stripe	Caramelo Banda
6	Fire and Smoke	Fuego y el Humo
7	Mountain Green	Montañas Verdes
8	Purple Accent	Morado Acento
9	Hazard Flame	Llama Peligro
10	Morning Sun	Sol por la Mañana
11	Grey Granite	Granito Gris
12	Copper Haze	Neblina Cobre
13	Sky Fire	Cielo de Bomberos
14	Wine Country	Vino País
15	Black Hole	Agujero Negro

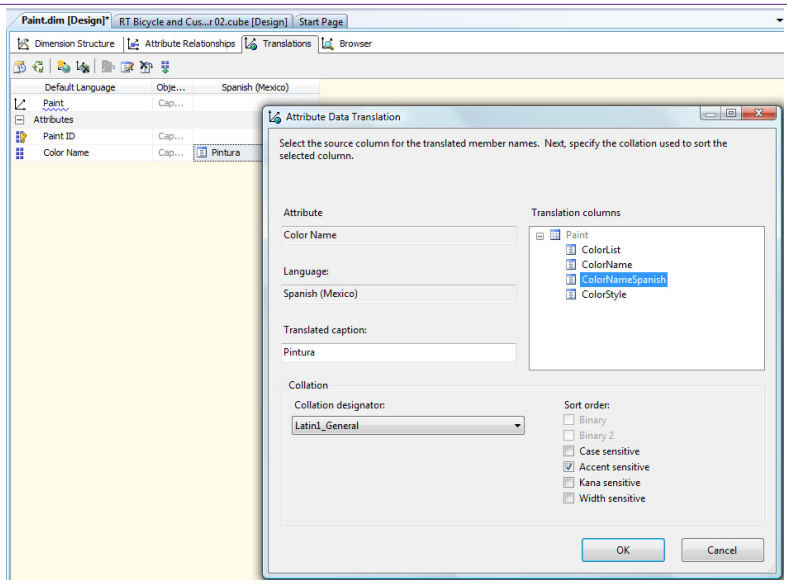


Figure 3.42

Assigning dimension translations. Open the dimension and click the translation tab. Right-click the screen to add a new translation and pick the language. Click the ellipses button in the gray box next to the Color Name attribute. Select the matching column name. Translate the title.

tab and right-click the main screen area to add a new perspective. Enter a name for the perspective that all users will recognize. Set the check boxes to select only the measures and attributes that should be included in the new perspective. By default, all available items are checked, so it becomes a process of removing the items not needed.

Save everything, process the cube, switch to the browser and reconnect. Figure 3.40 shows the new cube. The drop-down list in the center of the toolbar is used to choose a perspective. When Employee Sales is chosen, note the limited number of options available for use in the cube.

Internationalization and Translations

Many organizations today extend across national boundaries and need to provide data in multiple languages. Analysis Services has several tools to facilitate handling multiple languages. Keep in mind there is no magic bullet—someone still has to translate all of the terms. The purpose of the tools is to display the data and metadata in a specified language.

Two types of information need to be translated and displayed correctly: (1) Dimension data stored in the tables and (2) Metadata such as the titles of dimensions. The difference might seem subtle, but the two translations are handled with different techniques.

Dimension data requiring translation would include the different model types, color names, and so on. The original data exist in rows in the table (Paint), so the translation is handled by adding new columns to that table—one column for each language. Figure 3.41 shows sample data that might be used to translate the color names into Spanish. A marketing manager would likely want to improve the

names to be more appealing in the local language. The process is to add a column to the Paint table with a column name that indicates the language and then enter the translations row-by-row. The same process would be followed for the other dimensional attributes.

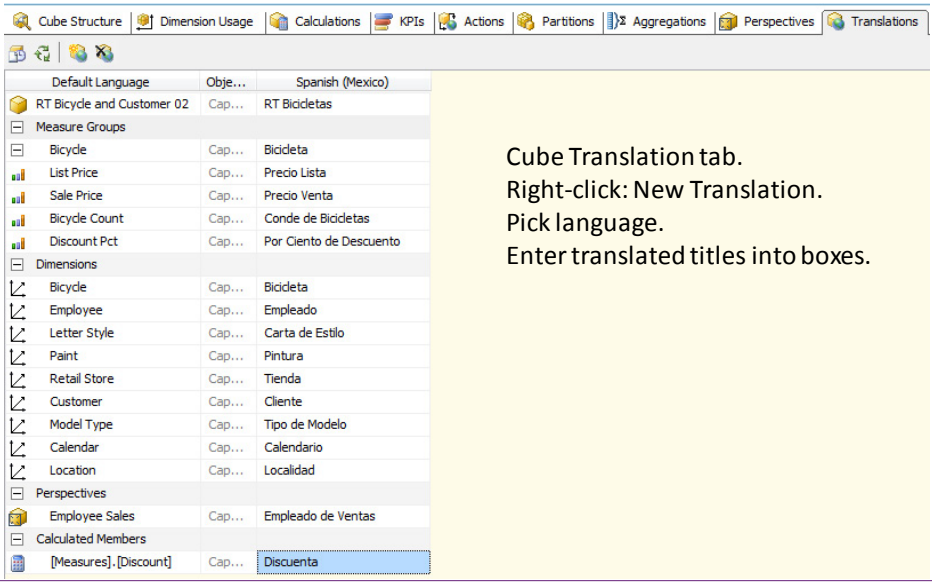
Once the translations exist, they can be assigned within the dimension. Open the dimension from the Solution Explorer and select the Translations tab. Right-click the main screen and choose the option to create a new translation. Choose the desired language. Click the ellipses button in the gray box next to the Color Name attribute to open the edit window. Select the new column name for the correct language. Enter the translation for the title. Note that all of the dimensions will need to be translated. It will take quite a bit of time just to enter the data and assign all of the dimension attributes—not including the time it takes to translate the actual words. When planning a schedule, remember to multiply the time by the number of languages needed.

A second step remains. The metadata for the cube needs to be translated—all of the titles that were entered must also be translated. Fortunately, the cube keeps track of them and organizes them in one location on the Translation tab. As shown in Figure 3.43, the editing process is simpler for metadata. Switch to the Translation tab in the cube editor and right-click to add a new language. Choose the language to create a column next to the original data. Enter the translated titles.

Figure 3.44 shows the cube in the new language. Remember to process the cube first then open the browser. Select the new language from the drop-down list. Notice the dimension names are mostly in Spanish in the left window. The paint names are correct, but the model types have not yet been translated. If a translation does not exist, the cube falls back to the default (initial) values. The point of

Figure 3.43

Translating cube metadata. Select the Translation tab on the cube editor. Right-click to add a new language. Enter the translated names for all of the titles.



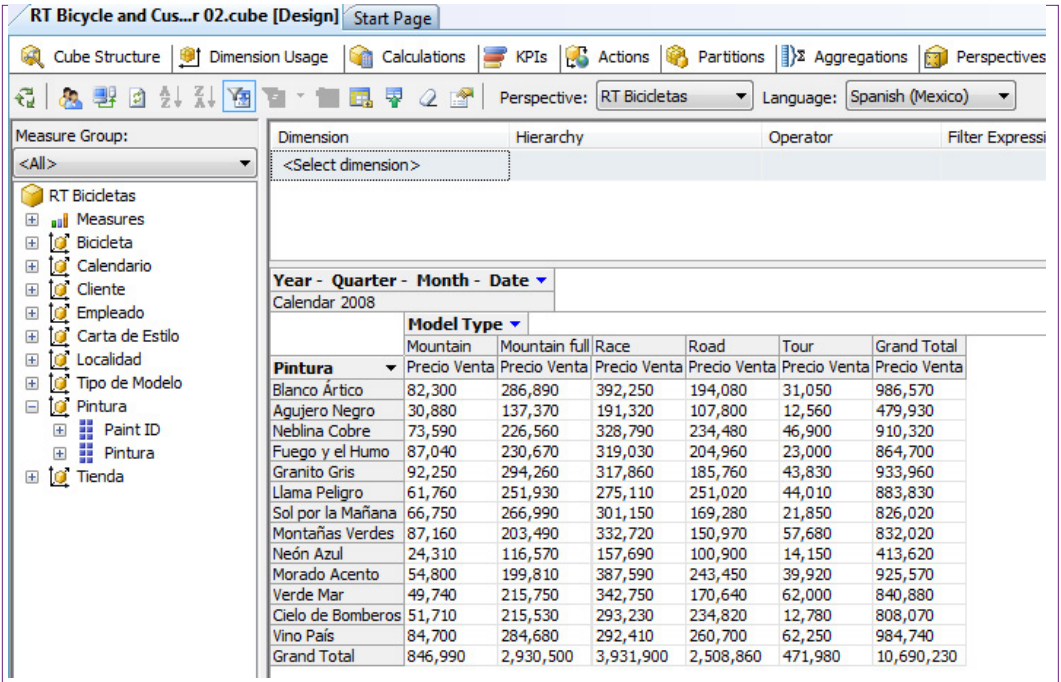


Figure 3.44

The cube partially in Spanish. Browsing the cube in a different language is accomplished by electing the desired language from the drop-down list.

the demonstration is that it takes quite a bit of time to identify all of the data and metadata and to enter the translated values. However, once the translation and data-entry is done, the users simply select a language and the cube picks up the appropriate values automatically.

Of course, countries generally have different currencies. The cube browser has a limited ability to convert monetary values to different currencies. Currency conversions are handled by creating a giant table of exchange rates. The Microsoft Adventure Works demonstration database has a sample table. It lists daily exchange rates for several currencies. Tables of historical exchange rates can be found online (for example, www.oanda.com). Setting up the table correctly and obtaining the data is the most complicated step. It is probably best to import the table from the Adventure Works database and create an ExchangeRate measure by copying Microsoft's example. Once the exchange rates are specified, a BI wizard helps apply them to the cube. Right-click the cube name in the Solution Explorer and choose the option to Add Business Intelligence. One of the options is "Define currency conversion." This wizard steps through the process of defining the exchange rate conversions. However, the details are not covered in this book. See (Harinath et al. 2009) for the specific steps using the Adventure Works example.

Performance: Partitions and Aggregations

Even with advanced hardware, huge problems can lead to performance problems. Analysis Services provides several options that can be added when performance begins to decline. One of the important tools is partitions. A partition is a physical

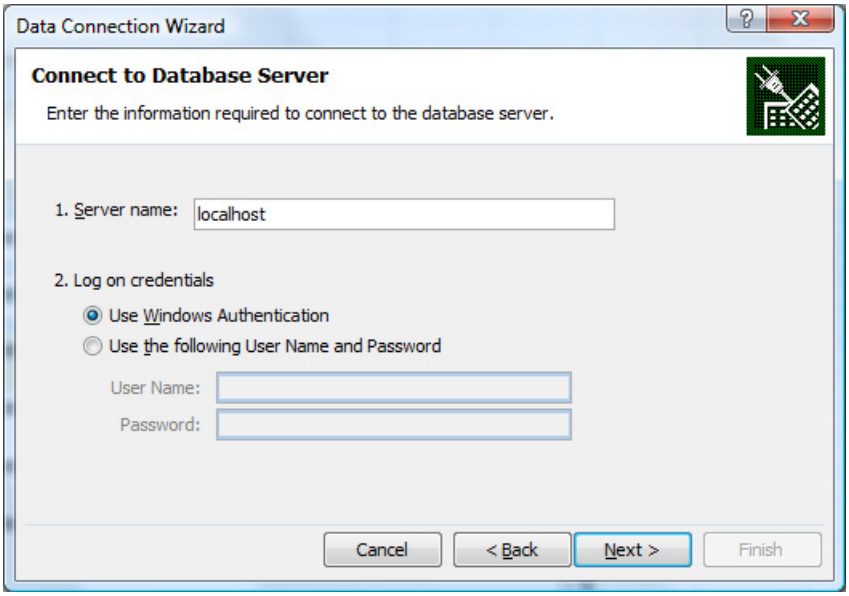


Figure 3.45

Connecting to an OLAP data cube. The first time a PivotTable is created, follow the wizard steps to connect to an external data source and create a new connection. Enter the server name and login information on this screen.

separation of data in the cube. By default, all data is stored in a single partition. Microsoft recommends that a partition should contain no more than 20 million rows. For problems with substantially more rows, new partitions should be added that split the data into these smaller sets. Each partition can be stored in different locations, can be assigned different storage structures (MOLAP, ROLAP, or hybrid), and can be assigned different security permissions.

The partitions are useful because they can be indexed and searched separately. They can also be processed on remote servers—spreading the processing load across multiple servers reduces the overall load and process time. Also, if a query retrieval requires only part of the data, the system automatically pulls data only from the needed partitions.

Aggregations are the totals and averages computed on the measures to form the subtotals of the cube. Complex cubes can end up with dozens or hundreds of aggregations. Each aggregation requires processing, storage, and indexing. One way to improve performance is to cut back on the number of aggregations needed. The remaining aggregations can be spread across different partitions, again taking advantage of parallel processing.

Partitions and aggregations can be examined and modified using the respective tabs in the cube editor. However, detailed performance analysis is beyond the scope of this book. Analysis Services does provide a wizard to help design aggregations and optimize the storage structure based on usage patterns. But, the process of optimization requires knowledge of high performance computing and experience with analyzers and the cube processing. You should know that these tools exist and that experts can be found to help tune the cube browser to handle extremely large problems.

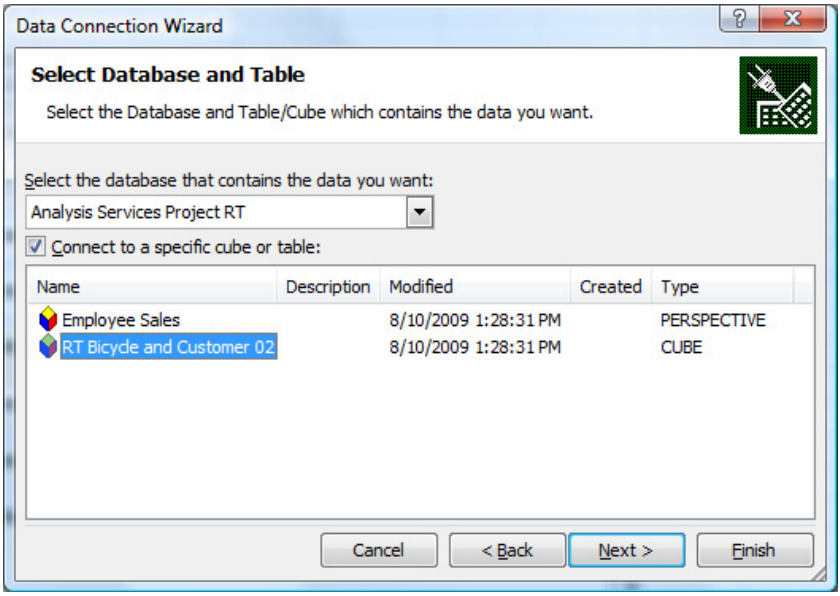


Figure 3.46

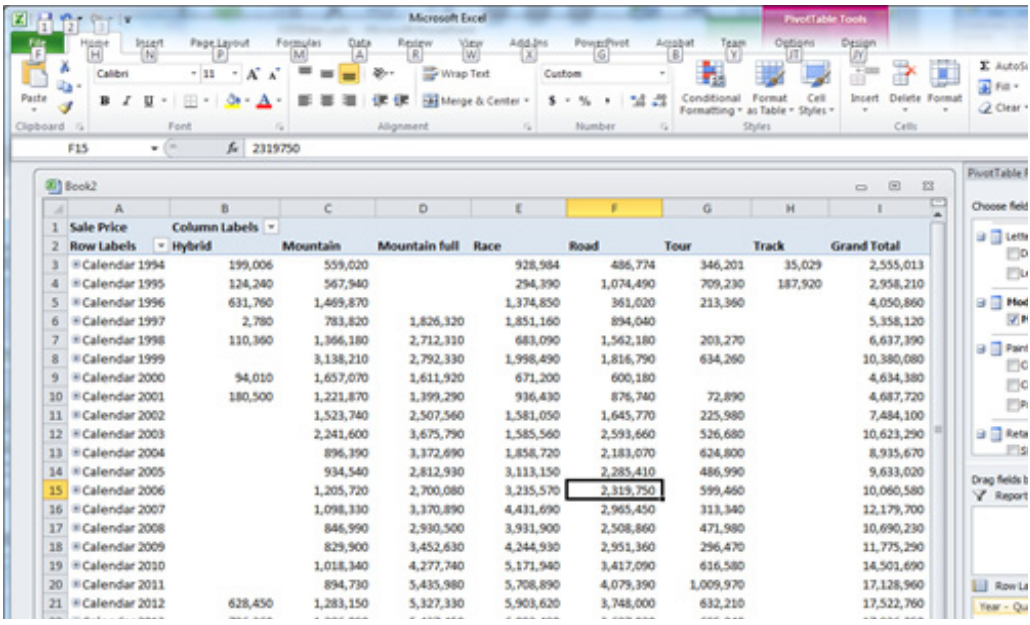
Choosing an analysis project and OLAP cube. Projects and cubes are displayed only if the user has permission to use them.

Excel PivotTables

How can the cube be accessed outside of Analysis Services?

Analysis Services and the Developer Studio are good for designing and processing OLAP cubes. However, the user interface for exploring the cubes is a little clunky. It is not something that should be given to managers. Fortunately, the Analysis Services is just that—a service. It processes and provides cube data to any front-end tool that knows how to use Microsoft's data connection methods. At the top of that list is Microsoft Excel and SQL Server Reporting Services, which means that cubes can be distributed easily through SharePoint servers as well using Power Pivot. To illustrate the process and the benefits, this section builds an Excel spreadsheet to connect to the cube, explore the data, and create a chart to highlight trends.

For many years, Excel has supported the PivotTable and PivotChart objects. These tools are a cube browser and a graphical display tool for cube data. The tools accept many types of data for input, including spreadsheets, queries to Access and SQL Server, and direct connections to SSAS cubes. Once managers understand the purpose and flexibility of OLAP cubes, the tools are relatively easy to use. The only challenge lies in connecting to the data—simply because of the number of steps and pop-up windows. Even this process can be simplified for managers. Once the connection is defined, it can be distributed to managers as a file that opens the connection immediately. Microsoft also has a PowerPivot add-in for Excel that can be downloaded (www.powerpivot.com). The main strength of PowerPivot is the ability to handle millions of rows of data and publish results to SharePoint servers. The tool can pull data from Analysis Services, but there is not much gain over a simple PivotTable.



Sale Price	Column Labels							
Row Labels	Hybrid	Mountain	Mountain full	Race	Road	Tour	Track	Grand Total
Calendar 1994	199,006	559,020		928,984	486,774	346,201	35,029	2,555,013
Calendar 1995	124,240	567,940		294,390	1,074,490	709,230	187,920	2,958,210
Calendar 1996	631,760	1,469,870		1,374,850	361,020	213,360		4,050,860
Calendar 1997	2,780	783,820	1,826,320	1,851,160	894,040			5,358,120
Calendar 1998	110,360	1,366,180	2,712,310	683,090	1,562,180	203,270		6,637,390
Calendar 1999		3,138,210	2,792,330	1,998,490	1,816,790	634,260		10,380,080
Calendar 2000	94,010	1,657,070	1,611,920	671,200	600,180			4,634,380
Calendar 2001	180,500	1,221,870	1,399,290	936,430	876,740	72,890		4,687,720
Calendar 2002		1,523,740	2,507,560	1,581,050	1,645,770	225,980		7,484,100
Calendar 2003		2,241,600	3,675,790	1,585,560	2,593,660	526,680		10,623,290
Calendar 2004		896,390	3,372,690	1,858,720	2,183,070	624,800		8,935,670
Calendar 2005		934,540	2,812,930	3,113,150	2,285,410	486,990		9,633,020
Calendar 2006		1,205,720	2,700,080	3,235,570	2,319,750	599,460		10,060,580
Calendar 2007		1,098,330	3,370,890	4,431,690	2,965,450	313,340		12,179,700
Calendar 2008		846,990	2,930,500	3,931,900	2,508,860	471,980		10,690,230
Calendar 2009		829,900	3,452,630	4,244,930	2,951,360	296,470		11,775,290
Calendar 2010		1,018,340	4,277,740	5,171,940	3,417,090	616,580		14,501,690
Calendar 2011		894,730	5,435,980	5,708,890	4,079,390	1,009,970		17,128,960
Calendar 2012	628,450	1,283,150	5,327,330	5,903,620	3,748,000	632,210		17,522,760

Figure 3.47

Excel PivotTable. Drag dimensions to the four boxes on the lower right. The PivotTable functions are similar to those for the cube browser in SSAS.

To create a PivotTable from scratch, open Excel and use the Insert/Tables ribbon entry to start the PivotTable wizard. Select the option to "Use an external data source," and click the Choose Connection button. Once the connection file exists, it can be found in the list displayed on the next pop-up window. For now, click the Browse for More button. On the selection window, click the New Source button.

Figure 3.45 shows the first step in the data connection wizard. You need to enter the server name and login information to find the OLAP cube hosted by Analysis Services. If the Analysis Services is running on the client computer, the default server is simply localhost. Windows login is the easiest connection method. However, once the cube is created and built for managers, and often for students, Analysis Services will be running on a standalone server and login information will be controlled by the system administrator (or instructor).

When the connection has been established and security verified, you will be asked to choose the project and the desired cube or perspective from the list within that project. Figure 3.46 shows the basic selection screen. Large projects can have dozens of cubes. Ultimately, projects and cubes are displayed only if the user has permission to use them. Security and other administrative issues are not covered in this chapter, but the process of applying user permissions is relatively standardized. Clicking the Next button leads to the final connection screen. That page enables you to name the file that will hold the connection information as well as provide a description of the connection. Once created, this file can be used to open the cube in the future with a few clicks. Finishing the data connection wizard returns you to the initial PivotTable setup wizard. Simply click the OK button to start the PivotTable.

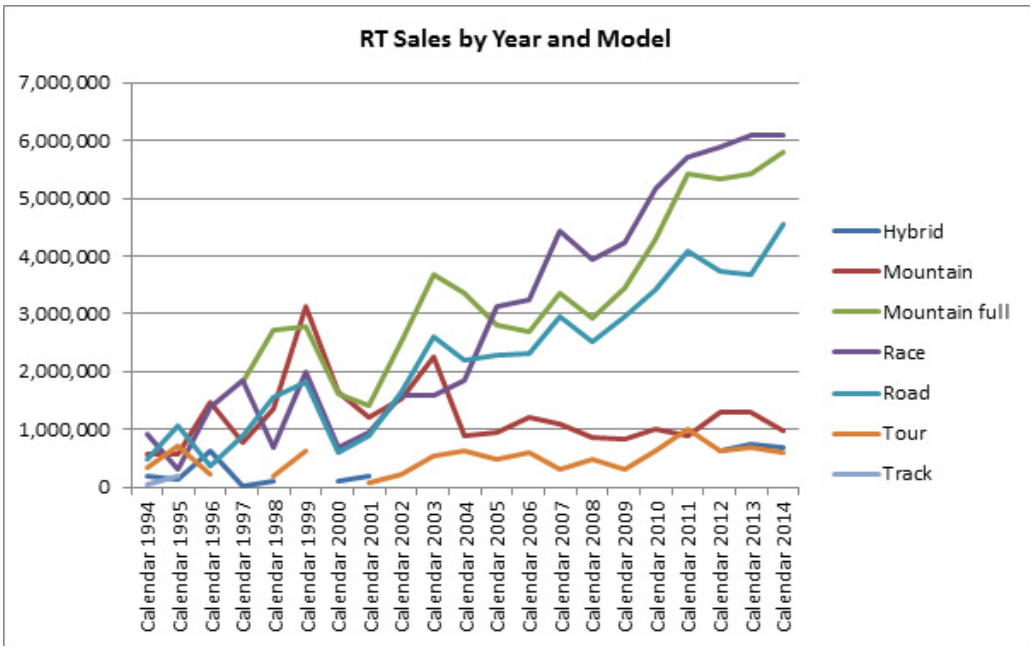


Figure 3.48

Excel PivotChart. Simply click the Chart button on the PivotTable menu. The chart automatically adjusts as the data in the table is filtered or changed.

The base PivotTable form in Excel is similar to the blank OLAP cube browser. It contains four basic locations: column dimensions, row dimensions, filter dimensions, and the main value form. These four areas are shown on the initial screen and as four display boxes in the PivotTable popup toolbar. Figure 3.47 shows one version of the cube. It is straightforward to drag dimensions to different locations and to drill down to details in the hierarchy. Standard Excel formatting options are available.

One of the nicer features of using Excel to explore the data is that a PivotChart can be created simply by clicking the Chart button in the PivotTable Options ribbon. Figure 3.48 shows one version of the chart. The chart itself is connected to the table. Changing the filters for the page, such as selecting one state, updates the table and the chart. Removing an attribute or changing a dimension automatically updates the chart display. The chart provides a visual presentation of the data that updates as the manager explores the various dimensions. It is a convenient way to let managers explore the data and see relationships. Similar versions can be deployed on in-house Web servers and through SharePoint to enable teams to share data.

Actions

How can the cube connect to external data such as Web sites and maps? [Note: This section probably will not work with the newer VS 2012 Cube Browser.] The basic cube operations display various subtotals, enabling users to drill down to see details and compare data across dimensions. But it is possible to do more. Actions can be assigned to various elements of the cube. For

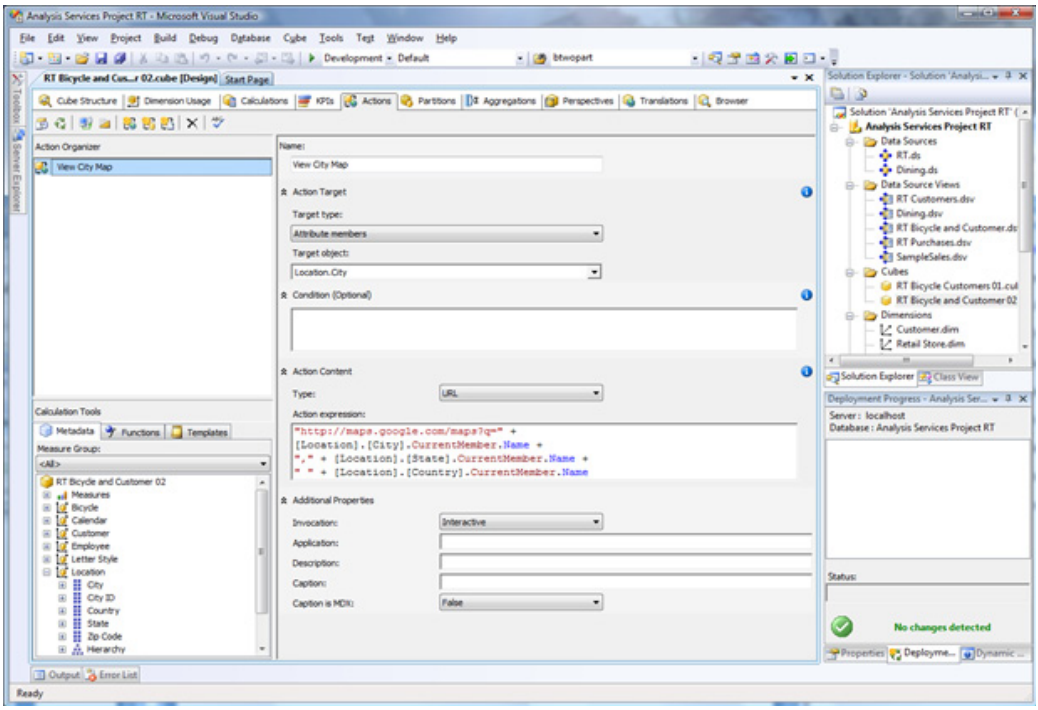


Figure 3.49

Create a new action. Right-click the top-left pane and select New Action. Enter a recognizable name, choose Attribute Members as the target type and Location.City as the target object. Create the URL link to the Web site (Google).

example, clicking a value can call a **drill through** action that displays all of the detail rows that make up the selected subtotal. Data columns can be designed to provide specific information that might be needed to answer questions.

Actions can also be defined to call external programs or open a Web site. Data values from the cube can be passed to the Web site so it can respond with matching data. This technique is useful for displaying maps and for opening SQL Server reports that carry specific information that matches the cube values. To illustrate the process, consider a simple link that opens a Google map for a selected city.

Figure 3.49 shows the basic process for creating a new action. Select the Actions tab in the cube browser. Right-click the top-left Action Organizer panel and choose the New Action option to open the editor. It is important to choose a recognizable action name—this name will be displayed to users. To create a URL mapping action, select Attribute members as the target type and Location.City as the target object. Notice that many other cube objects can be chosen for other purposes, including cell, level, dimension, and hierarchy. Choose URL as the action type and create the URL action that opens a Google map. The Google site has instructions for passing parameters on the URL line. The simplest is to send the City, State, and Country values as if they were entered as a query. The basic format is of the form:

```
http://maps.google.com/maps?q=Sacramento, CA USA
```

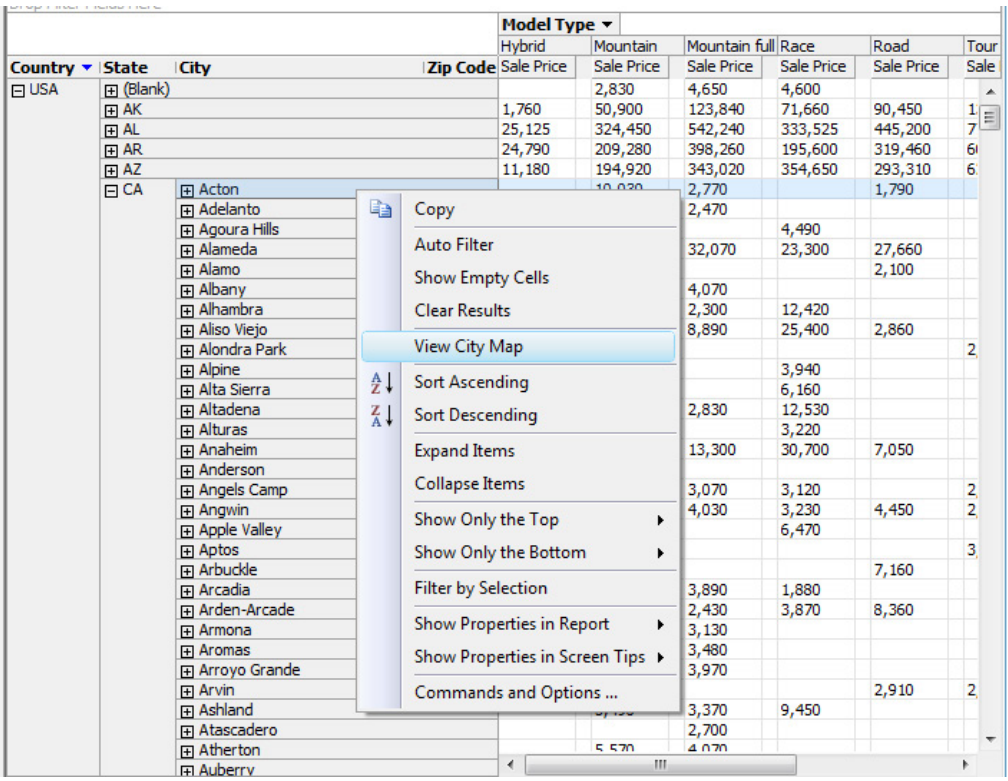


Figure 3.50

Activate an action in a cube. Add location to the cube and expand down to a list of cities. Right-click a city and open the new View City Map action in the popup menu.

The http portion of the URL is passed as a text string so it needs to be enclosed in quotation marks. The City, State, and Country are extracted from the currently selected city. You can drag the [Location].[City] and other values into the expression box, but you have to add the .CurrentMember.Name by hand. The complete expression is

```
"http://maps.google.com/maps?q=" +
[Location].[City].CurrentMember.Name +
"," + [Location].[State].CurrentMember.Name +
" " + [Location].[Country].CurrentMember.Name
```

Google maps (or Bing maps) support additional parameters, enabling the map to be displayed in a variety of ways but they are not important here.

Save everything, process the cube, and browse it. Add the location hierarchy to the rows and expand down to a city. As shown in Figure 3.50, right-click a city to see the pop-up menu. Note the new action: View City Map. Select the action and a browser should open and display that city in Google maps. Relatively sophisticated actions can be built using Web sites and custom programming, but they are beyond the scope of this book. The main point is to remember that a cube can be extended to support many different actions. Complex problems will require the assistance of a programmer.

Key Performance Indicators

How can simple data be provided to managers on a daily basis?

The OLAP cube is a useful tool but it requires active exploration by managers and analysts. It is a useful way to examine data from a variety of perspectives. It is useful when searching for specific comparisons—because it does not require writing SQL and because results are almost immediate. But, cube browsing could be time consuming if managers had to use it to look up similar items every day. For example, perhaps executives simply want to know what happened to sales for the year, quarter, month, or week. Or, regional managers want a quick look at salary expenses for the past month. These types of values could be found through the cube, and the cube can show comparisons to the last period or the same period a year ago. But managers will quickly grow tired of having to fire up the cube browser, select the basic filters, and search for the results. These numbers are so common and so basic that they want a simpler way to see the values quickly.

Definition

A key performance indicator (KPI) is a piece of data that informs managers about a specific measure. The item represents some aspect of the organization that is viewed as important. The item provides a measure of progress, and its value over time is an important indicator of future results. In business, sales revenue and various expenses are often useful KPIs. In line management, perhaps quality, quantity, and cost numbers are more important. The point is that each level of management, and even each manager, has a different collection of KPIs.

Managers want a way to see the current values of KPIs, to see them on a daily basis, and to be warned of patterns and changes. One solution is the digital dashboard, which provides gauges showing the values and trends of various KPIs. Similar in concept to the dashboard of a car (or cockpit of an airplane), the KPI gauges help managers evaluate the status, direction, and trends of various business factors to guide the company.

Analysis Services provides a method to define KPIs and store those definitions on the server. Various client tools can then query the server, which automatically retrieves the data, runs the KPI code, and returns values that can be used to set the values of gauges. KPIs are defined in terms of several expressions to compute: Value, Goal, Status, and Trend. The Value is the current value of the measure, such as annual sales. A Goal is a target value for the measure. It might be a value specifically entered by managers—providing a target level of production or sales for each region. Or, it might be expressed as a percentage change value—the amount the company wants to increase sales. Status is a value designed to be used by gauges and other indicators. It returns a value between -1 and 1, with zero a neutral indicator. Technically, status values are continuous and can return any level between -1 and 1 (inclusive). However, many systems simplify the results and return one of only three values: -1 (poor), 0 (neutral), and 1 (good). Some client gadgets can handle only those three states. For example, a traffic-light icon uses red, yellow, and green lights to show the status value. The fourth item of Trend is similar. It is also used to set an indicator icon. Typically, the icon is an arrow (up, down, horizontal) to show three trend values (increasing, decreasing, and neutral).

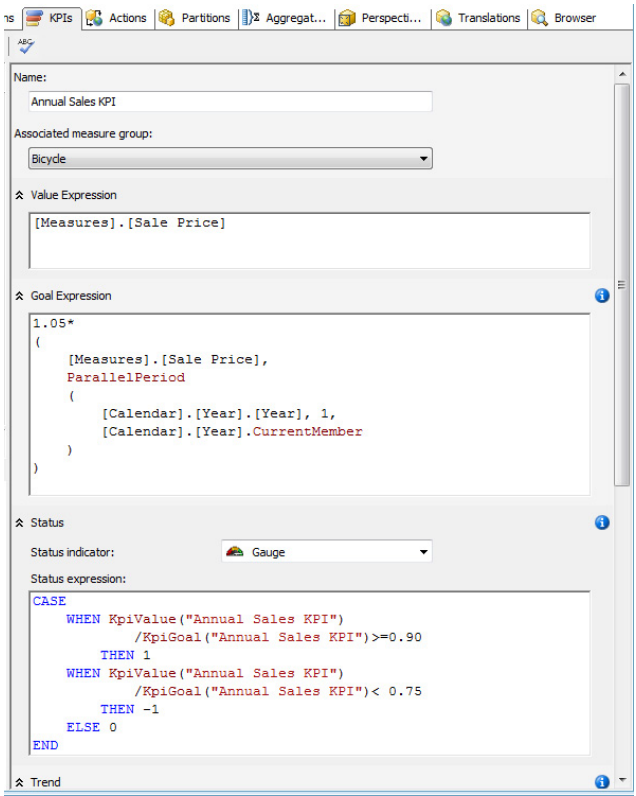


Figure 3.51

KPI edit form. The name is critical. The value expression returns the actual value. This Goal expression sets the goal as a percent of the prior period. The Status and Trend expressions return -1, 0, 1 for Bad, Neutral, Good values to gauges.

Creating KPIs

The benefit to creating a KPI in Analysis Services is that the definitions are stored centrally, so the same data is available to everyone who has access to the KPI. When the measures need to be changed, they are changed in one location. Also, the KPIs use the aggregated cube data, so the results are computed quickly and efficiently.

Creating a KPI has one important challenge: The definitions of the KPI values are written in the **multidimensional expressions (MDX)** query language. MDX is a query language for OLAP cubes first defined by Microsoft, but several vendors support it in their products—including various design tools for client-side displays. MDX is powerful, bears no relationship to SQL, and has some useful functions that make it relatively easy to create KPIs. But, solving complex problems requires an in-depth knowledge of MDX. Instead of trying to cover all of the details of MDX in this chapter, the examples focus on commonly used types of queries. These examples have some limitations, but they are easy to set up and can be modified for many common situations.

To create a new KPI, click the KPIs tab in the cube editor. Right-click the KPI Organizer panel in the top-left of the editor and select New KPI. As shown in

Figure 3.51, the edit form shows the sections that need to be created: Name, Value Expression, Goal Expression, Status Expression, and Trend Expression. Figure 3.51 shows most of the edit form. The Trend expression and some optional parameters at the bottom of the form are not shown.

The Name is critical because it must indicate the purpose of the KPI to managers so they choose the right KPI for their desired purpose. This KPI will use the Bicycle Sale Price total to measure annual sales. KPIs can also be stored in designated folders—making it possible to group related KPIs together. The Additional Properties section at the bottom of the form has options for setting a Display folder and a Parent KPI for KPIs that are hierarchically related (such as Annual, Quarterly, Monthly, and Weekly sales). The section also contains a box for Description to better explain the KPI.

The Value Expression is the main definition of the KPI. This is the value that will be returned to any client that calls the KPI. To obtain sales revenue, open the Measures section in the Metadata list and drag the Bicycle Sale Price measure into the expression box to set the MDX formula:

```
[Measures].[Sale Price]
```

Goal Expressions are useful because they can be used to set the status charts. The presumption is that some Goal exists for the Value. Perhaps each sales region has been given a target goal or each factory a production quota. If these goals are set by management, a separate column attribute needs to be defined in the database to hold these numbers for each time period. In that case, the expression is as simple as the Value expression—simply drag the goal measure into the expression box. Another approach to setting goals is to compute the target as a percentage of the sales in the prior period. A special MDX function called **ParallelPeriod** is used to retrieve the value from the prior period. The function has three parameters: (1) The hierarchy level, (2) The number of periods, and (3) The current value. The syntax for the 5% increase is:

```
1.05*
(
    [Measures].[Sale Price],
    ParallelPeriod
    (
        [Calendar].[Year].[Year], 1,
        [Calendar].[Year].CurrentMember
    )
)
```

The 1.05 value at the top sets the percentage increase. The [Measures].[Sale Price] sets the measure to use, and the ParallelPeriod function returns the matching value for the prior calendar year. The function can be used to compare values using a fiscal year calendar instead—as long as the fiscal year calendar is defined as a hierarchy dimension. Note that the [Calendar] name must match the name of the hierarchy dimension in the OLAP cube.

More importantly, note that the goal expression is only defined for years. Interestingly, the KPI will display output when applied to quarterly data, but the numbers would not make much sense. The goal value is always computed in terms of the annual data, even if the display is based on a different level. Hence, it is important to include "Annual" in the name of the KPI.

The Status expression must return values between -1 and 1 because the values will be used to set values for visual icons. The edit form provides the ability to choose the type of indicator. A gauge is the default choice. Typically, the status indicates how close the actual value is to the goal. Often, it is convenient to use only three categories: poor (-1), neutral (0), and good (1). Some gauges support continuous measures, but it can be challenging to convert real-world numbers into an appropriate scale. A basic CASE statement can be used to assign three values in the Status Expression:

```
CASE
    WHEN KpiValue("Annual Sales KPI")
            /KpiGoal("Annual Sales KPI")>=0.90
        THEN 1
    WHEN KpiValue("Annual Sales KPI")
            /KpiGoal("Annual Sales KPI")< 0.75
        THEN -1
    ELSE 0
END
```

Notice that this formula uses the existing definitions for Value and Goal to compute a simple percentage. If the actual sales are 90 percent of the goal or higher, it is considered good. Sales less than 75 percent of the goal are bad, and everything in the middle is neutral. Again, note that these numbers are subjective, and they are hard-wired into the expressions. A few tricks exist to put the numbers into measures that can be edited by managers, but the tricks require considerable knowledge of MDX.

The Trend Expression also drives an indicator icon—typically an arrow. So it should return values -1, 0, or 1. The most common approach to show a trend is to compute the percentage change from the prior period. If the change is sufficiently positive, the arrow should point up (1). If the change is negative, the arrow should point down (-1). Anything in the middle should be displayed as a horizontal arrow (0). The formula for percent change is (new-old)/old, but it can be simplified to just new/old – 1. This version has the benefit of needing to use the prior value only once instead of twice. Remember that the prior value is obtained using the ParallelPeriod function and using it twice makes the expression difficult to read. The Trend expression becomes:

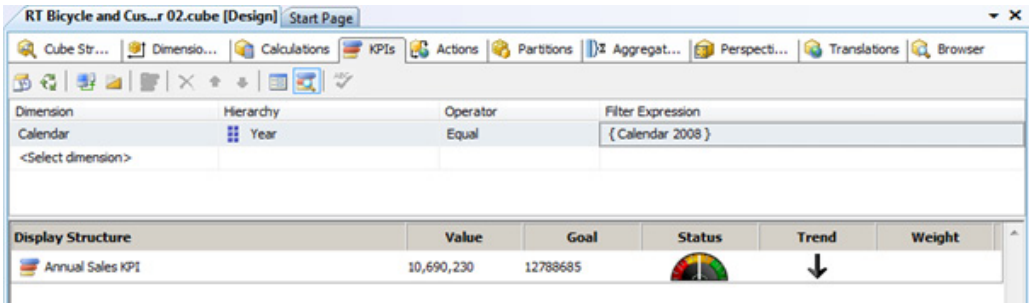


Figure 3.52

KPI browser. The two icons in the tool bar under the KPIs tab switch between edit and browser mode. Testing works best when the Calendar dimension is dragged to the filter bar near the top of the form. Select Year as the Hierarchy level and pick a specific year from the Filter Expression list.

```
CASE
    WHEN
      (KpiValue("Annual Sales KPI")/
        (KpiValue("Annual Sales KPI"),
         ParallelPeriod(
            [Calendar].[Year].[Year], 1,
            [Calendar].[Year].CurrentMember
         )) -1
      ) <= -0.05
    THEN -1
    WHEN
      (KpiValue("Annual Sales KPI")/
        (KpiValue("Annual Sales KPI"),
         ParallelPeriod(
            [Calendar].[Year].[Year], 1,
            [Calendar].[Year].CurrentMember
        )) -1
      ) > 0.05
    THEN 1
    ELSE 0
END
```

Again, two "WHEN" statements set the high and low values, leaving everything else in the middle at zero. The syntax for the division (new/old) is a little hard to read because of the parentheses. Just remember that the old value is the term in parentheses that uses the ParallelPeriod function. This example sets the low and high levels to negative and positive 5 percent.

Browsing a KPI

It is important to understand that Analysis Services stores the MDX Expressions for a KPI. It does not actually compute the values until the KPI is queried. KPIs are designed to be used by other applications running on client computers. For example, a Web page or SharePoint server page might include a reference to several KPIs. In many cases, a KPI can be embedded on a desktop through a specific

gadget. Periodically, the gadget requeries the server to get current values for all of the KPIs. This timing control is built on the client computer. Analysis Services is passive and simply returns the values when asked.

It is also important to test the KPI expressions. It is easy to misplace a parentheses or comma. Fortunately, the Analysis Services KPI editor includes a simple browser to test and display the values. A small icon hiding on the toolbar switches the editor to browser mode. A similar icon next to it returns to edit mode.

Figure 3.52 shows the browser form for the Annual Sales KPI in Calendar year 2008. The KPI icons will display with the default values of all years, but the goal cannot be computed, and the icons do not make much sense. Instead, it is best to select a single year to evaluate the KPI. Drag the Calendar hierarchy dimension from the metadata list onto the filter section at the top of the form. Select Year as the level in the Hierarchy column. If the KPI was built for Quarter, Month, or Week, choose the appropriate level. The operator defaults to Equal, so choose a single year in the Filter Expression list. Be sure to click on the Display list after selecting or changing a year value. The display is updated only after it is clicked. It is good practice to choose a couple of adjacent years; record the values and the goals and manually check the calculations to ensure the goal is computed correctly. It is also interesting to test a few years to see what happens to the gauge and trend icons.

KPIs have additional properties that can be used for more complex cases. The Weight is used for groups of KPIs—typically a set of KPIs that have the same parent. For instance, Sales might be divided into sales by region. The regional KPIs could be assigned percentage weights that represent the relative size of each region. Parent assignments and weights are set in the "Additional Properties" section of the edit form.

Summary

Online analytical processing was created because storing data efficiently and retrieving data quickly lead to conflicts. The indexing, pre-computed totals, and duplication of data needed to retrieve data in seconds or less causes problems with saving new data. OLAP cubes are designed conceptually and physically around the concepts of dimensions and measureable facts. The star design connects dimensional attributes directly to the fact table. The snowflake design allows links from dimensional tables to other tables. Many dimensions have a hierarchy of values. Time or dates are perceived at levels from years to quarters to days. Geographic hierarchies are commonly used to group locations.

To managers and analysts, a hyper cube is essentially a large collection of subtotals (or averages) based on many possible dimensions. Hierarchies provide support for drill down and rollup to see details or summaries at any level. Filters are used to slice the cube and compare values for different attributes.

Creating a cube is straightforward with Analysis Services and its wizards. The primary steps are: (1) Connect to at least one data source, (2) Create a data source view that connects the fact table to the dimension tables, (3) Choose measures and dimensions for a cube, (4) Improve the dimensions and add hierarchies. Data source views are the key method of providing data to the data mining tools.

Cubes are designed to perform summary calculations—usually sums of numbers. However calculations can be defined at two levels: (1) Aggregate calculations applied to the subtotals, or (2) Row-by-row calculations performed at the query (view) level. Any calculation requiring multiplication or division should

take place at the query level. But, if the detail values are percentages, the aggregation should be changed to averages across the children instead of sums.

Analysis Services is a server—it defines the cubes and efficiently retrieves the data. It might not be the best browser. Excel and Reporting Services provide additional options for browsing cubes, including options to place cubes on internal Web sites. For instance, interactive charts are easier to create in Excel than in Analysis Services. Actions can be defined to provide more detailed tools and options to browsers. For example, it is straightforward to add Web hyperlinks to data so users can link to more detailed data or even Web pages with maps and descriptions. Key performance indicators can be defined on the server and accessed by client applications to display gauges and other icons that indicate trends in important data measures.

Key Words

aggregation
attribute relationship
Business Intelligence Development Studio
calculation
cube browser
data warehouse
digital dashboard
dimension
drill down
drill through
extraction, transformation, and loading (ETL)
fact
hierarchy
hybrid (HOLAP)
immediate if function (IIF)
index
key performance indicators (KPI)
measure
metadata
multidimensional expressions (MDX)
multidimensional OLAP (MOLAP)
named calculation
named query
online analytical processing (OLAP)
online transaction processing (OLTP)
order of operations
ParallelPeriod
perspective
relational OLAP (ROLAP)
rolled up
snowflake
star design
view

Review Questions

1. Why is MOLAP better than ROLAP for most large applications?
2. Why does setting up extraction, transformation, and loading often take a long time?
3. How is the snowflake design different from the star design?
4. What are the primary steps in creating an OLAP cube in Analysis Services.
5. When should time hierarchies be built on the server?
6. What are the major benefits to storing hierarchy data in tables in the original database?

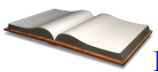
7. What is the purpose of attribute relationships in a hierarchy? What constraint or issue is imposed when building them?

8. What problem arises when defining calculations at the cube level? How are these resolved?

9. What is the point of adding actions to a cube?

10. What are four main components of key performance indicators

Exercises



Book

1. Create the main Rolling Thunder Sales cube with the time and geography hierarchies.

2. Create the Percent Discount column in the record source view and add it to the existing cube using an average instead of sum.

3. Define two perspectives on the existing cube. One that focuses on sales by employee and one that focuses on customer purchases without the employee data.

4. Modify the model type table to add a second language. Translate the model types (use a Web translator if necessary). Add the new language to the cube.

5. Create an Excel PivotTable that connects to the cube and build a PivotChart.

6. Add the URL action to open Google maps for the location.

7. Define and test a key performance indicator for monthly sales.

Rolling Thunder Database

8. Create a new cube that focuses on purchases from manufacturers.

9. Create a new cube that focuses on component inventory. Use the cube to identify any problems the firm has with respect to inventory.

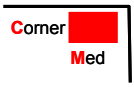
10. Identify at least three KPIs other than total sales that could be useful to managers of Rolling Thunder Bicycles.

Diner

11. Build a cube for the Diners database.

12. Create an Excel PivotTable and PivotChart that will help managers. Identify the primary decisions the chart is designed to improve.

13. What KPIs would be useful for managers of the restaurant?



Corner Med

14. Build a cube for the Corner Med database that focuses on the patient visit, diagnoses, and treatments.

15. Build a cube for the Corner Med database that focuses on the work output and revenue generated by the employees.

16. Getting payments is always a problem at medical offices. Build a cube, PivotChart, and KPIs that can be used by managers to monitor payments by insurance companies.

Basketball

17. Build a cube that helps coaches of individual teams track the performance of their players.

18. Build a cube that helps coaches evaluate players across the league to help decide who to recruit. Create an Excel PivotTable.

19. Identify at least three KPIs that might be useful for a coach during the season.

Bakery

20. Create a cube for the Bakery database that helps managers explore sales. Use an Excel PivotChart to make it easier for managers to understand the data.

21. Note that Sales Date for the bakery also includes the time of day. Create or modify the sales cube to include a dimension for time of day that splits the day into three time periods: morning, lunch, and afternoon.

22. Identify at least three KPIs that would be useful for managers of the bakery.

Cars

23. Create a cube that enables potential car buyers to evaluate the cars.

24. Modify the cube to include an action link that brings the user to a Web page for the manufacturer's Web site. It is probably not possible to link to a specific vehicle, but check at least one of the independent Web sites to see if there is a way to link to a specific vehicle.

Teamwork

25. Split the team into subgroups of two people and assign a language to each subgroup. Modify the Rolling Thunder Bicycles database to incorporate each of the new languages, translate the dimension data, and modify the cube to handle each language.

26. Have each team member choose a different basketball team. Using a cube and PivotCharts for analysis, identify the best players on the team. Note, these evaluations must be based on data, not personal opinion. Combine the individual results and make a case for choosing the best player among the group.

27. Using the bakery data, assign product categories so that each team member evaluates at least one category and all of the categories are evaluated. Using the cube analysis and charts, identify any patterns, trends, or problems with sales of each category. Combine the results and identify any categories that should be used to highlight an advertising campaign.

Additional Reading

Harinath, Sivakumar, Matt Carroll, Sethu Meenakshisundaram, Robert Zare, Denny Guang-Yeu Lee, 2009, *Professional Microsoft SQL Server Analysis Services 2008 with MDX*, Wrox/Wiley: Indianapolis. [Detailed how-to discussion of many steps in SSAS. Most of the authors were Microsoft employees working on SSAS, so the book is loosely the documentation Microsoft did not create.]

Jacobs, Adam, 2009, "The Pathologies of Big Data," *Communications of the ACM*, 52(8), 36-44. [One commentary on problems with relational databases. He tried to load 6.5 billion rows into a relational database.]