Surviving Client/Server: Data Mining With TDecisionCube

by Steve Troxell

Data mining is a hot buzzword in the database world. Data mining generally refers to collecting, categorizing and summarizing a system's data to gain some useful insight about the operation: sales performance, shipping distribution, marketing penetration, any number of possibilities. Actually, "data mining" is just a fancier term for "decision support," which is a fancier term for the even older "data analysis." You can sell more products by convincing customers they are behind the perceived latest industry trends, while you simply repackage, simplify, and dress up the same techniques that have been used for years.

Most RDBMS vendors have begun to provide specialized "summarizing" capabilities in their SQL languages. These SQL extensions typically produce crosstabulation, or crosstab, result sets in a more simplified and efficient manner than is possible with straight SQL syntax (see *More Data Processing with SQL* in Issue 15). With Delphi 3 we now have a suite of decision support components, centered around the `TDecisionCube` component, which we can add to our client applications. With these components, we can add powerful decision support capabilities to our applications. Data analysis with crosstabs has been around for decades. `TDecisionCube` is just an elegant, simplified user interface for this statistical analysis technique. Note that it is included only with the Client/Server edition of Delphi 3.

A Data Analysis Example

A typical order tracking system might contain data like that shown in Figure 1. Here we track every order that was made by date and by method of payment for a six month period. The software system uses this data, and more not shown, to process the order and ensure proper accounting of inventory and accounts receivable. However, there is more value to this data beyond the simple processing of each order.

As an employee concerned about the sales performance of our company, let's ask ourselves some reasonable questions about this data. What were the total sales for the 2nd Quarter? How did 2nd Quarter sales compare to the previous quarter? How did the MasterCard, Visa, and American Express sales compare for the period? What is the average order paid by credit versus the average order paid by check? What were the total sales by month and was there any trend?

Such questions turn ordinary transaction data into useful analytical results. We are now in the realm of data analysis, or data mining to use the more popular buzzword of recent times. If these questions are routinely asked, we will usually provide predefined reports or displays to show exactly the results we need. If the types of questions are widely varied or ad hoc, then we can obtain answers from the data with a calculator or a few SQL queries. However, with the `TDecisionCube` component and its supporting components introduced in Delphi 3, we can add a more elegant user interface for some types of ad hoc data analysis.

What Will TDecisionCube Do For Me?

There is a suite of Decision Cube components, each handling a specific task in the operation. The `TDecisionCube` component manages the crosstab data in memory, it does not in itself display anything to the user. The raw data is most often acquired from a database through a `TDecisionQuery` component, but a regular `TQuery` or even `TTable` component could be used.

Next come the user interface components: `TDecisionPivot`, `TDecisionGrid` and `TDecisionGraph`. We'll explain the usefulness of these components as we go along. Finally, `TDecisionSource` is the "pipeline" between the user interface components and the actual `TDecisionCube` component; in much the same way as `TDataSource` lies

➤ *Figure 1*

```
SELECT SaleDate, PaymentMethod, COUNT(*) AS Num, SUM(ItemsTotal) AS Amount
  FROM Orders
  WHERE SaleDate BETWEEN '1/1/94' AND '6/30/94'
  GROUP BY Saledate, PaymentMethod
```

SaleDate	PaymentMethod	Num	Amount
1/1/1994	Credit	1	$3,087.00
1/9/1994	Credit	1	$10,054.00
1/17/1994	Credit	1	$4,229.80
2/13/1994	Check	1	$15,052.00
3/3/1994	Visa	1	$5,011.00
3/14/1994	MC	1	$12,900.75
3/25/1994	AmEx	1	$7,671.90
4/7/1994	Credit	1	$97,698.60
4/16/1994	Credit	1	$3,860.85
5/1/1994	Visa	1	$13,226.80
5/5/1994	Credit	1	$13,935.95
5/9/1994	MC	1	$12,367.00
5/22/1994	Credit	1	$9,793.55
6/1/1994	Check	1	$2,206.85
6/4/1994	Credit	1	$102,453.60
6/9/1994	Credit	1	$3,153.00
6/14/1994	Credit	1	$342.00
6/26/1994	Visa	1	$2,692.85

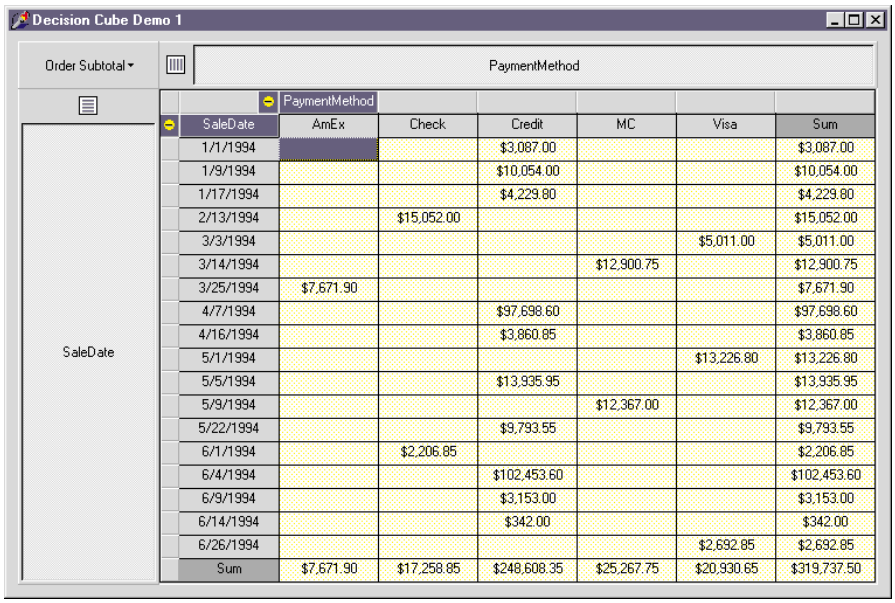
between data-aware controls and dataset components.

Before we look at how the decision cube suite of components work, let's look at their runtime behavior so we know what we're trying to achieve. Figure 2 shows a screen we might set up to represent our data. It shows a `TDecisionGrid` component surrounded by three `TDecisionPivot` components, one above (the large `PaymentMethod` button), one to the left (the large `SaleDate` button) and one in the upper left corner (the `Order Subtotal` dropdown list).

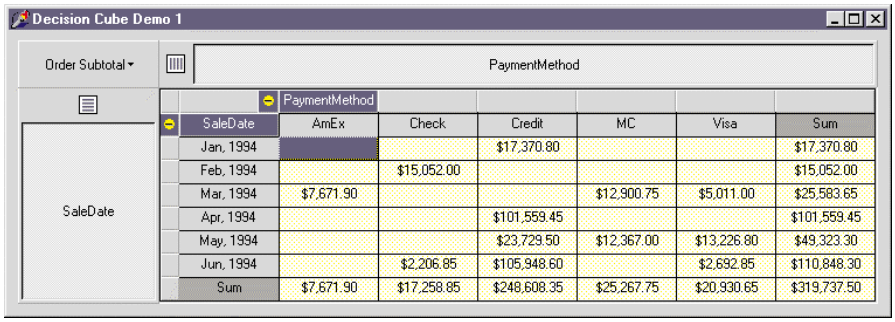
The data is represented in this example by the `TDecisionGrid` as a crosstab of two dimensions. The columns represent the unique values for `PaymentMethod` (the `PaymentMethod` dimension) and the rows represent the unique values for `SaleDate` (the `SaleDate` dimension). Crosstabs such as this are not limited to two dimensions: we can have a single dimension or any number dimensions in the decision cube. The values in the interior of the grid are summary values representing the total amount of sales for the intersection of a given date and a given payment method. The edges of the grid conveniently show us the totals for each row and column or, more precisely, totals for each value in each dimension.

So a decision cube is a crosstab consisting of 1 to n dimensions with a summary value at each intersection. The summary value is an aggregation of some value in all the records having the same data values as the corresponding dimensions. For example, there happens to be one record with `SaleDate` of `6/4/1994` and `PaymentMethod` of `Credit` and its summary value is $102,453.60. This aggregation aspect is hard to see in this example since there is only one record per date in `SaleDate`.

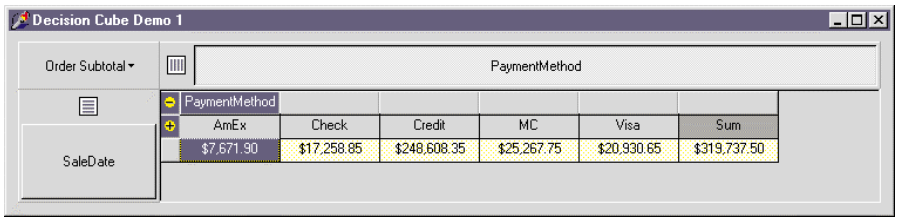
We can give the user the ability to change the granularity of date dimensions to single values, months, quarters, or even years. Figure 3 shows the same data rolled up into months. Now we see the three June credit orders aggregated together into the single summary value $105,948.60.



➤ *Figure 2*



➤ *Figure 3*



➤ *Figure 4*

The `TDecisionGrid` component allows the user a large degree of control over the display of the data. Each dimension in the display can be expanded or collapsed by clicking the small yellow button next to the dimension header label. We can also expand or collapse a dimension by clicking the corresponding button in the `TDecisionPivot` control.

We can effectively change the number of dimensions in the display, and hence the resolution of the data analysis, by opening and closing one or more dimensions. When collapsing a dimension, only the totals for that dimension are visible, as shown in Figure 4 with the `SaleDate` dimension collapsed.

With a collapsed dimension, we can also allow the user to "drill into" a specific value of that dimension. By right-clicking on a pivot button, we can select `Drilled In` from the context menu. Once drilled in, click on the pivot button again to get a list of values in that dimension and select any one we wish to focus on. In Figure 5, we've drilled in on the `March 1994` value for the `SaleDate` dimension.

With the pivot controls, we can also alter the arrangement of dimensions in the display. By right-clicking on the `PaymentMethod` button, one of the context menu options is `Move to Row Area`. By selecting this, the entire dimension is redisplayed with row orientation as shown in Figure 6. Likewise, we can move any row dimension to column orientation by selecting `Move to Column Area` from the pivot button's context menu.

We can drag and drop a dimension button within a pivot control to change its relative position within the display. In Figure 7, we dragged the `SaleDate` button above the `PaymentMethod` button and obtained a very different view of the data.

We can provide any number of summary values, but can only display one set of summary values at a time. The user can select from among the summary values we provide through the summary button in the pivot control. In these examples, this is the `Order Totals` button in the upper left corner. Examples of summaries we can provide for this data are: number of orders, total amount of order including freight, taxes, and discounts, average order amount, etc. Any summary calculation we can express using a `SELECT GROUP BY` statement in SQL can be provided as a display option to the user. Note that this does not limit us solely to SQL aggregate function results, although they almost always participate in the calculations.

Finally, we also have the `TDecisionGraph` component at our disposal to graphically represent our crosstab data (Figure 8). We use the same pivot controls to alter the dimensions and summary values. `TDecisionGraph` is merely an alternative display of the decision cube data and is actually a variation of the `TeeChart` component by David Berneda. It contains a plethora of options for customizing a graph's appearance. The standard `TeeChart` component is also available in Delphi 3 as `TChart` on the `Additional` page.

TDecisionQuery

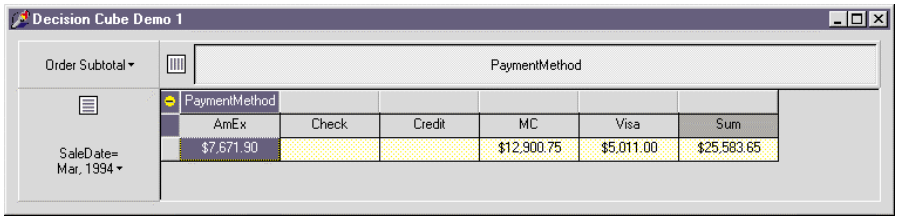
Now that we know what Decision Cubes can do for us, what do we have to do to set it all up? First we start with the `TDecisionQuery` component to define the values we need from the database. `TDecisionQuery` is a specialized form of `TQuery`.

With the Decision Query Editor, you can select one or more tables and identify which columns will be the dimensions and what values will be the summaries for those dimensions. It automatically generates an SQL `SELECT` statement with the appropriate `GROUP BY` clause and select list to support the decision cube data you've defined.

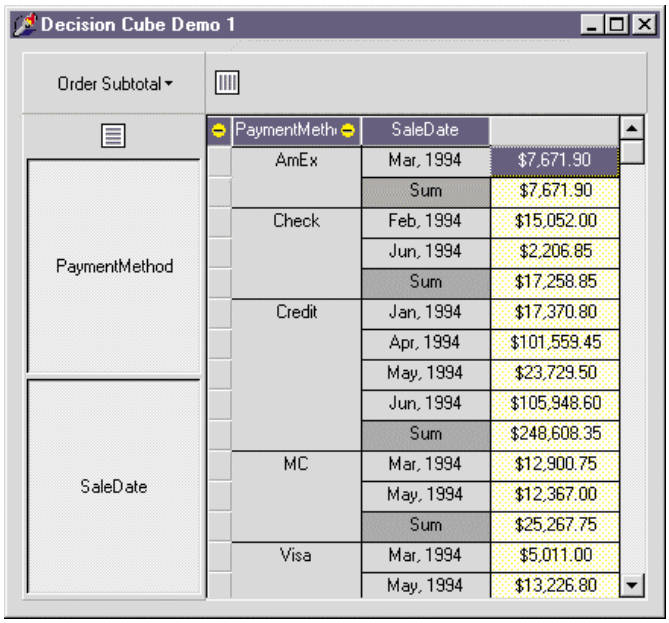
Listing 1 shows the SQL generated for the decision cube we've been using in our examples. The dimensions are `SaleDate` and `PaymentMethod` and there are three sets of summary values representing by the three aggregate functions. Notice that we can have as many different summary values as we want, but only one set can be displayed in `TDecisionCube` at one time.

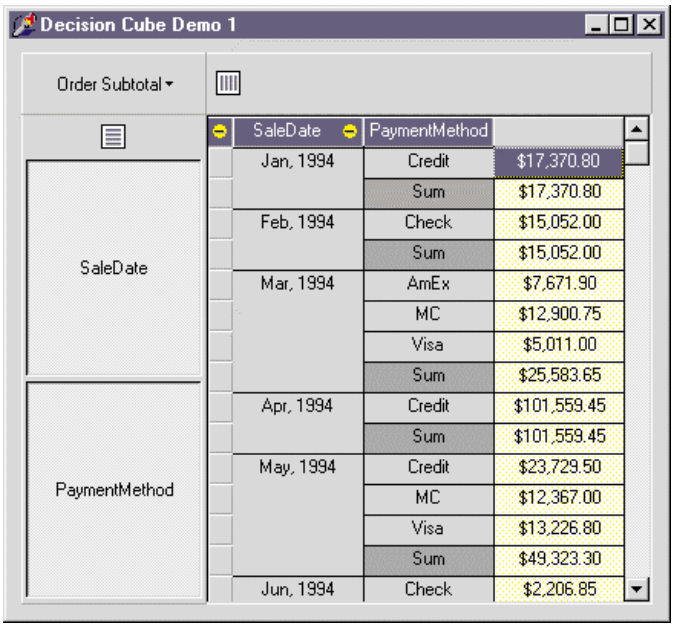
The `WHERE` clause was added by hand to filter the range of records down. Actually if we wanted summary values or dimensions that were not simple fields from the data, we can modify the SQL by hand to add the values we need. For example, we might want to see the total of the order amount and

➤ *Figure 5*



➤ *Figure 6 (below left)*
➤ *Figure 7 (below right)*





the freight charges but not tax, so we might add `SUM(ItemsTotal) + SUM(Freight)` to our `SELECT` list as an additional summary value.

TDecisionCube

Now that we have the data defined, we come to the actual `TDecisionCube` component and attach our `TDecisionQuery` component to it via the `Dataset` property. The significant property of this component is the Decision Cube Editor, which is used to setup some parameters of the decision cube.

The `Display Name` is just that, the name that appears in the visual controls. The pivot, grid, and graph controls all inherit the display name for each field from `TDecisionCube`. The `Type` field tells us whether the value is a dimension or summary value. You only have to set this if a `TTable` is used to provide the data.

`Active Type` can be set to one of three values: `Active`, `Inactive` or `As Needed`. `Active` means the dimension or summary is always loading into `TDecisionCube`. `Inactive` means it is never loaded and is not available through the display controls. Note that the data still gets loaded from the database when the decision cube refreshes itself, it is just not stored in `TDecisionCube` memory. When large decision cubes are built, you might want to turn on and off the loading of some dimensions or summaries so that only a certain amount are available at the same time. `As Needed` means the value can be swapped in and out of memory with other `As Needed` values to stay within a predefined threshold. This threshold is defined using the `Memory Control` tab of the Decision Cube editor or the `MaxDimensions` and `MaxSummaries` properties.

For dimensions comprising date values, the `Grouping` and `Initial Value` settings can be used to roll up dates into more useful sets of dates: years, quarters, months or explicit values (compare Figure 2 and Figure 3). When a date grouping has been defined, `Initial Value` can be used to set the boundaries of the grouping. To tell the truth, I have had a number of problems getting this feature to work correctly. I am still baffled as to how to get date values to bin into the proper groups for anything other than normal calendar groups and have seen erroneous attribution of years in some Quarter groupings. Nevertheless, I will attempt to explain how this feature is intended to work.

`Initial Value` is most commonly needed for software that, for accounting purposes, use fiscal date groupings that rarely line up with the calendar. For example, the company's fiscal year may run from November 1st through October 31st. Therefore, its first quarter would run from Nov 1st through Jan 31st. If the company handles monthly recurring billing, like most public utility companies, pay-television subscription services, Internet Service Providers etc, then their "billing month" may run from the 20th of the previous month to the 19th of the current month, for example. In this case, their fiscal year and quarters most likely would also start of the 20th of their respective months instead of the 1st.

All the `TDecisionCube` parameters may be setup at runtime through the `DimensionMap` property. This may allow you to provide users with runtime selection of fields and values to analyze in the Decision Cube.

TDecisionSource

User interface controls are connected to the decision cube through the `TDecisionSource` component, much the same as data-aware controls are connected to data-access components via a `TDataSource` component. `TDecisionSource` acts as the intermediary between `TDecisionCube` and any `TDecisionPivot`, `TDecisionGrid` or `TDecisionGraph` controls you may be using.

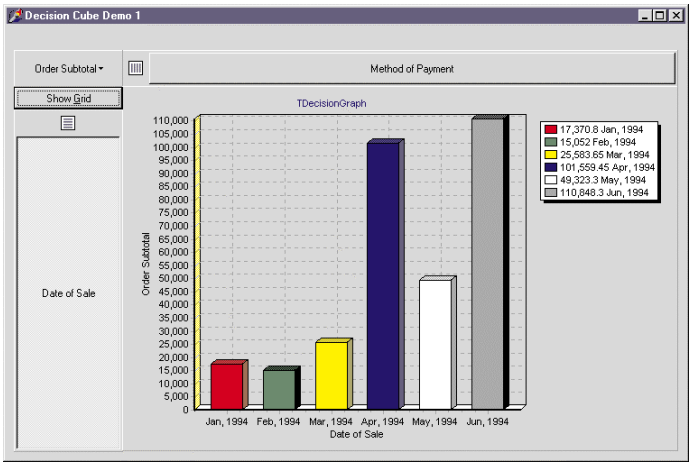
TDecisionPivot

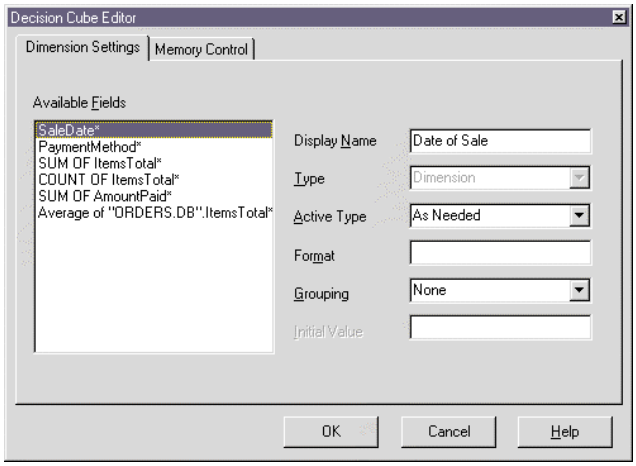
As we saw earlier, `TDecisionPivot` controls provide a way for users to rearrange the dimensions that

➤ *Listing 1*

```
SELECT SaleDate, PaymentMethod, SUM(ItemsTotal),
       COUNT(ItemsTotal), SUM(AmountPaid)
FROM "ORDERS.DB"
WHERE   (SaleDate >= '01/01/1994')
   AND  (SaleDate <= '06/30/1994')
GROUP BY SaleDate, PaymentMethod
```

➤ *Figure 8 (below left)*
➤ *Figure 9 (below right)*





constitute the "rows" and "columns" of the decision cube. These controls do not actually display any data in the cube, but merely control its representation. They can be omitted if you want to limit the degree of control your users have over the cube display.

Pivot controls have buttons to represent each dimension value and a single button to select from the available summary values. All of these can be grouped into a single pivot control, or split out into separate controls as we've done in the previous examples. The `Groups` property allows us to specify which controls are handled by a specific pivot (rows, columns or summaries).

TDecisionGrid

The most common way cube data will be displayed in an application is through a `TDecisionGrid` control. The `Dimensions` property brings up the Cube Dimension Settings Editor (much like the Fields Editor for a `TDataSet` component). This editor lists all the dimension and summary fields of the cube, and when any one of the listed values is selected, the Object Inspector reveals the display properties of that value. At this level, you can change the alignment, color, format, or display name of any of the values in the grid. Note that it is here where you would change the bothersome default center-alignment behavior of the summary cells. Annoyingly, you cannot change the alignment of the subtotal cells though.

You have some degree of control over the painting of each cell with the `OnDecisionDrawCell`. However, this is unlike the `OnDrawCell` event handler for regular grids: you don't have full control over the entire rectangle to be drawn. You can only affect the display string, color, and font. So, even though you can set right-alignment for cell values through the `Dimensions` property, you cannot override the cell drawing to prevent the data from being painted just one pixel away from the cell edge (as is the default for right-aligned data).

`TDecisionGrid` contains a protected `DrawCell` method that you could override to have more control over cell painting.

Conclusion

The Decision Cube suite of components in Delphi 3 can provide useful decision support functions to your applications. Decision Cubes should not be used as a substitute for fixed reporting and displays if the user needs specific information on a recurring basis. However, as a means to provide an ad hoc analysis tools, decision cubes may be very helpful. Be wary of the grouping feature of date dimensions.

Next month we'll look at several fuzzy logic algorithms for locating words in text when the exact spelling may not be known.

Steve Troxell is a Senior Software Engineer with TurboPower Software. He can be reached by email at stevet@turbopower.com or on CompuServe as STroxell.