Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases

An important issue with handling large multidimensional databases was the ability to query, view and analyze it in order to find trends and causal relationships. Past systems have treated data as data cubes that could have different dimensions encoded into the rows and columns. Polaris was a new tool which extended the PivotTable idea by building different views using a formal algebra that could interact with a relational database. Polaris combined work from visual specification languages, table based displays and data exploration tools to produce a product that could drill down and roll up data in a fashion that gave a complete overview.

The main mode of operation of Polaris was the table, which encoded how records would be viewed. These could be minified and easily compared with other tables. They were familiar to statisticians and they were able to display high dimensional data. By dragging and dropping fields from the relation onto a table, the view changed and created a new visual specification, the underlying language used to retrieve and manipulate data. The data in the table could then be grouped, aggregated, displayed with different visual marks, sorted and partitioned.

In order to create tables, a visual specification made of the table configurations, the type of graphics displayed and the detail of the encoding was generated. A formal algebra was proposed in which valid expressions were ordered sequences of symbols with operators. Operators like concatenation, cross product and nesting were used (nesting is very similar to intersection). Users could then choose the type of graphical display. The system presented a list of graphical marks which defined the type of view. For example, choosing bars resulted in a bar graph. They also defined three types of graphics that were helpful in understanding different kinds of data. Ordinal-Ordinal graphics were used for understanding trends in some function mapping \( O \rightarrow O \). Ordinal-Quantitative graphics were used when a quantitative value was dependent on the ordinal value and the properties of the function mapping \( O \rightarrow Q \) were being studied. Quantitative-Quantitative graphics were used to display the distribution of data and discover causal relationships between two separate data sources. Records were displayed in a variety of visual marks aimed to maximize user efficiency. Shape, size, orientation, and color were all carefully chosen.

Allowing users to change the data after being displayed was a key goal. Generating aggregation and statistical summaries was a common task for statisticians. Moreover, users could also perform partitioning of quantitative data, counting of distinct values, thresholding aggregation and ad-hoc grouping. Sorting and filtering were both implemented as well as brushing, which highlighted data selected in one display across all displays.

In order to retrieve data, a three step process is employed in which each step was mapped to a SQL command. First data was selected using initial user input. Then the records were partitioned into panes based on further criteria and finally, the user could transform data after it had been displayed with methods mentioned above. The only caveat to the system was the lack of performance tuning. Although large databases were used with relative success, a full quantitative exploration of performance issues was lacking.
Flush with a large and unique set of data, the authors realized that existing solutions for processing lacked key qualities, such as allowing for irregular grid structure, good performance, and a defined algebra that could be used to simplify and optimize. Like many datasets, their data was in the form of a grid. Data tuples were bound to the cells (n-dimensional) of the grid and when data was transformed both grid and cells were updated. Their approach moved away from weaknesses imposed by RDBMS systems and data dependency brittleness introduced by visualization libraries. In RDBMS systems, data would need to be duplicated and pre-computation would be need to be stored. The authors even felt that schema changes might be necessary upon insertions. Visualization software worked with one grid at a time, did not offer the full range of functions required, did not allow cross products and could not represent time as a dimension. Their new system aimed to address these issue. Their main contribution was defining an algebra for operating on grids that separated topology data from geometric data. By doing so irregular grids were supported, operations were performed on grids instead of intermediate data representations, and operations could be optimized.

The gridfield algebra is comprised of grids made of k-dimensional cells. When data is bound to cells, it becomes a gridfield. The basic unit is the 0-dimensional node which allows for encoding of topological relationships. Each node is part of a namespace. Higher dimensional objects are created through a cross product of grids. What this means is that non-standard geometries can be captured and operations can be performed on gridfields that transform the grid and the data. For example, the Restrict operation takes a grid and outputs another grid where certain data has been pruned from the cells. Merge will find common cells of two grids and put them together in a new grid. Aggregate takes data from many cells potentially and maps it onto a new cell. There were numerous benefits to this approach that included representing geometry as data, allowing for extensibility through aggregation and not limiting the number of dimensions. Their implementation of a grid field relied on C++ standard maps and vectors. Hashing within the maps allows retrieving cell data in constant time.

A number of optimizations were available to the authors because of the formal algebra created. Forward binding pre-computed ordinal positions of cells which could then be retrieved quickly, allowing for a restrict to operate on gridfields before a cross product operation was done. The same applied for lowering dimensionality, where dimensions were removed in the restrict phase before cross producting was performed. The authors noted that working in 2-D was computationally less expensive than 3-D and was to be preferred. Another optimization was performed when grids were merged. Specifically, if two sets of cells were defined over the same grid, then restrict did not need to compute the intersection of all elements. Instead, restricting in sequential order which gave the same result, but in constant time.