Paper 124

What's All This Metadata Good For, Anyway?
Using Metadata to Dynamically Generate SQL

Jonathan Stokes, JJT Inc., Austin, Texas

Metadata is becoming increasingly popular for data warehouse administration, but typically reporting and analysis applications utilize the same metadata in very limited ways. This paper discusses the *metadata model* and methods designed by JJT Inc. to create an *extraction engine*, using object-oriented SAS/AF® software, that dynamically generates very efficient SQL code. Such an engine provides enterprise-wide performance enhancements by ensuring that all database extractions are optimized, a particularly important feature when using remote database systems.

This paper gives an overview will be given of the programmatic interface to this *metabase engine*, including its effect on a typical end-user interface. It also explains how the SQL procedure's pass-through facility, available with base SAS® software and SAS/ACCESS® products, boosts the efficiency of metadata-driven operations on remote databases.

The information in this paper can benefit anyone interested in innovations in data warehousing and data analysis. The concept of dynamic, pass-through extract generation greatly increases the potential of stored metadata, and changes the role of the metabase in data warehousing.

What Is Metadata?

Metadata, or *data about data*, describes the contents of a database. Administrators can use this information for data management purposes; end-users and end-user applications can use it for locating and identifying information needed for decision support analysis.

Typically, metadata is used in the form of documentation. Charts display table ownership, column data types, and relationships between columns for reference purposes. However, stored metadata has thus far been under-utilized by most applications.

Optimal use of metadata is achieved when applications are metadata-driven. In this approach, data-driven applications can rely on a metabase to provide the locations of needed data, rather than depending on specific columns in particular tables. Accurate metadata becomes as important as the data itself. As a result, administrative tasks and documentation also benefit from this accuracy.

Metabase Organization

The structure of metadata, or the metadata model, can vary between implementations. JJT Maestro Metabase™ stores and accesses metadata using *object-oriented* technology. While detailed subclasses handle complexity, metadata "objects" fall into two general categories: Data Sources and Data Items.

Data Sources

Data Source objects represent data providers. In relational databases each table or view is a Data Source. The Data Source object representing a table or view contains details on where the table is located (remote path, username, password, etc.) and information about the columns within the table (i.e. column names and data types), in addition to descriptive information about the table and its ownership.

The metabase administrator defines Data Source objects in the Metabase. Analyst users, however, do not need to know what sources are defined; they define reports using only Data Items.

Data Items

Data Items represent columns or fields of data. These metadata objects define *virtual columns* or variables. Any specific field in a database has one corresponding Data Item, regardless of how many tables (Data Sources) that field appears in. In addition to descriptive information about the field, a Data Item object contains "pointers" to each column within each Data Source in which it appears. These links to Data Sources can be used in SQL generation to determine how tables can be joined together.

Examples of Metadata

Following is a simple example of metadata using the Data Item-Data Source model.

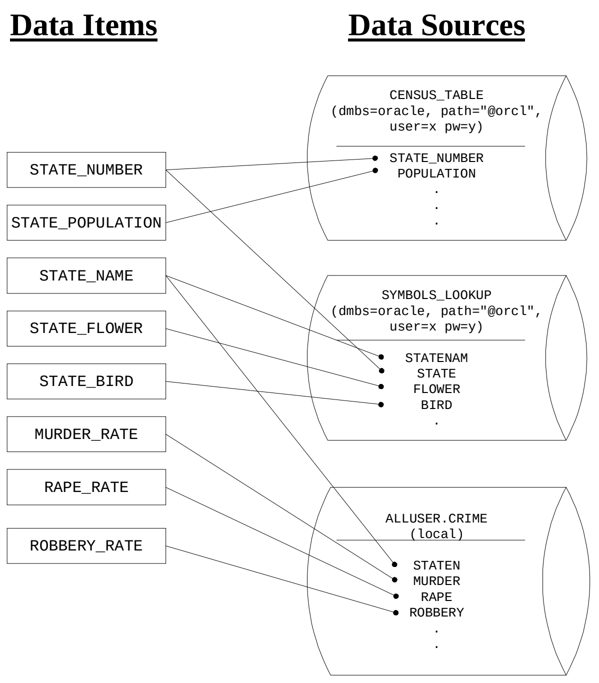

Figure A

This example demonstrates three Data Sources: two Oracle® tables and one SAS data set. The CENSUS_TABLE contains U.S. State populations by state number, the SYMBOLS_LOOKUP table contains state flowers and state birds by state name, and the CRIME data set contains various crime rates by state name and state number.

There are eight (8) Data Item objects listed. Because *state number* is a joining-key between sources, only one Data Item is defined, STATE_NUMBER, which points to the state number column in both Data Sources. The same applies to the STATE_NAME Data Item. Each of the other six Data Items points to a single column, and serves mainly to provide a unique, global name to that field, and to store data type information about that column.

User Interface

Because of the SQL Generation Engine (discussed in following sections), only Data Items need to be fronted to the end-user. Reports can be defined using the Data Items available in the Metabase, without concern for where the underlying tables are located or how they should be joined together.

In the JJT Metabase, each Data Item is uniquely named with long (up to 200 character) names. This allows the user to pick from a customized *hierarchical tree* of Data Items organized into categories understood by the user community.

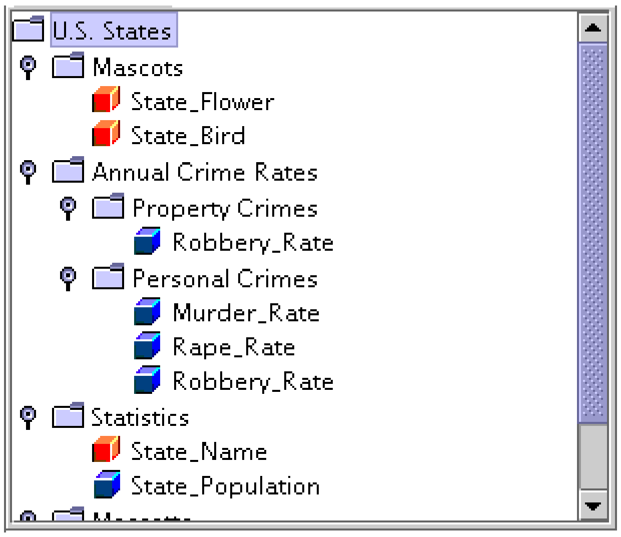

Figure B

Figure B shows an example of a Data Item selection tree. Any Data Item can appear anywhere in the tree, and can be placed in more than one branch of the tree. The structure of the categories in the hierarchy is arbitrary, and does not need to reflect column or table location in the database. This interface is simply an easy way for the user to navigate through a vast list of fields available for extract from the database, without concern for the physical or logical location of the data.

In this example, various icons are used to denote different data types, such as numeric analysis Data Items and character grouping Data Items.

Note that some Data Items may not appear at all in the user interface. In the example shown in Figures A and B, the State_Number Data Item represents a numeric code assigned to each state. The code is used simply as an identity field in certain tables, and for joining between tables. Such Data Items are not used for reporting, but are defined in the metabase purely for generating SQL.

SQL Generation

The "next step" in applying metadata technology is extract generation. Rather than each end-user application querying the metadata for "where do I find columns X, Y, and Z so I can extract them?" an *extraction engine* can be asked, "give me the code to extract X and Y where Z > 2." This approach frees up reporting application development, and especially ad hoc reporting, to concentrate on reporting without concern for data extraction.

Why SQL

An extract engine has several advantages if the extraction code generated is Structured Query Language (SQL), particularly when remote Database Management Systems such as Oracle or Sybase are analyzed using SAS Software. The most important advantages are versatility and performance.

Versatility

Because SQL is a standard supported at some level by all relational database software, it provides common syntax that can be used regardless of the DBMS. PROC SQL provides SQL access to SAS data sets and views, as well as foreign DBMS tables via various SAS/ACCESS engines.

Performance

A major reason the Maestro Metabase extract engine uses SQL is because of the power of PROC SQL's *pass-through* facility. When accessing foreign databases, such as Oracle, pass-through provides the ability to delegate `WHERE` clause constraints and even table joins to the remote database. This delegation offers decreased network I/O and increased, enterprise-wide efficiency.

Programmatic Interface

When the metabase extract engine is accessed programmatically by applications, it is asked: "Give me the code (SQL) to create a table called _______, that contains Data Items A, B, and C, using constraints C = 'Y' and D < 5"

A, B, C, and D are all Data Items which, in the metadata, point to their respective columns within Data Sources. The calling application does not need to know which Data Sources are involved. Instead, the extraction engine determines which Data Source(s) will most efficiently provide the given Data Items, and returns the SQL that would be used to create the result set.

Example

The following example demonstrates how an application would use an SQL-generating metabase:

An analyst, using SAS/AF ad hoc reporting tool, wants to determine if there is a correlation between the state bird of a U.S. state and its annual robbery incident count. (His hypothesis is that states with predator mascots may have higher incidents of violence.) To explore his theory, he wishes to extract two columns, `state_bird` and `robbery_rate`, without really knowing in which tables or databases they reside. He selects these two Data Items from a selection list in the analysis application. In addition, the analyst wishes to limit his query to include only states with populations greater than two million. The application then calls on the metabase to generate SQL that will extract the desired results.

What Is Given

An application requesting such data might call a metabase object in this way:

```
call send( metabaseID
         , 'extract'
         , dataItemNameList
         , 'WORK.MYDATA'
         , 'State_Population > 2000000'
         , generatedSQLList
         );
```

where the `dataItemNameList` parameter is an SCL list in the following form:

```
dataItemNameList = ( 'State_Bird'
                     'Robbery_Rate'
                   )
```

This method call would either invoke an SQL generation engine within the SAS/AF application, or call on a remote metabase server to perform the SQL generation.

What Is Used

Internally, the metabase must determine which Data Source, or Data Sources, can be combined to provide the necessary Data Items. In a real-world metabase, this involves complex algorithms that consider hundreds or thousands of Data Sources. In our simple example, all three Data Sources described in Figure A will be used to create the desired result set.

Data Source	Data Items Included	Joining Column?
Crime_Rates_Last_Year	State_Name	Yes
	Robbery_Rate	No
Census_Table	State_Number	Yes
	State_Population	No
State_Symbols	State_Name	Yes
	State_Number	Yes
	State_Bird	No

Figure C

What Is Returned

The Maestro Metabase would respond to the above request by returning the following SQL code:

```
proc sql;
  connect to oracle( path="@orcl"
                   , user=x
                   , orapw=y
                   );
  create table WORK.MYDATA as (
    select STATE_BIRD
         , ROBBERY_RATE
    from ( select STATEN
                  as STATE_NAME
                , ROBBERY
                  as ROBBERY_RATE
           from ALLUSER.CRIME
         ) sas
       , connection to oracle(
           select STATE_NAME
           from ( select STATE_NUMBER
                  from CENSUS_TABLE
                  where POPULATION
                           > 2000000
                ) census
              , ( select STATENAM
                         as STATE_NAME
                       , STATE
                         as STATE_NUMBER
                       , BIRD
                         as STATE_BIRD
                  from SYMBOLS_LOOKUP
                ) symbols
           where census.STATE_NUMBER
                 = symbols.STATE_NUMBER
         ) orcl
    where sas.STATE_NAME
          = orcl.STATE_NAME
  );
  disconnect from oracle;
quit;
```

This SQL brings the minimal amount of data from the remote DBMS into SAS, creating a SAS data set with only the desired two columns. Note that the `where` clause is applied and the two remote tables are joined by Oracle before the resulting subset is returned (across the network) to SAS for further joining. This approach distributes machine-load effectively and reduces over-all network traffic by providing the most efficient queries possible

The above SQL might create the following SAS data set.

STATE_BIRD	ROBBERY_RATE
Mockingbird	74
Pelican	61
Mockingbird	33
Yellowhammer	43
Brown Thrasher	55
Mockingbird	82

If desired, the analyst can also include any number of additional Data Items to aid in reporting.

Benefits

Utilizing a code-generating metabase can have phenomenal advantages:

The User Does Not Need To Know SQL

The analyst can extract or subset data for analysis without extensive training in SQL programming. Even the competent SQL database programmer can focus on tasks more important than data retrieval, knowing that the most efficient SQL possible is programmatically generated.

The Data Model Can Be Very Complex

Because of the efficiency of the pass-through facility, an SQL-generating metabase can contain metadata that spans database management systems. The extraction engine can easily generate SQL code to automatically join Oracle and Sybase tables with SAS data sets into a SAS data set or view for analysis. Any heterogeneous mix of database systems can be used as long as a SAS/ACCESS engine is available for that system. This flexibility provides for legacy systems while allowing integration of newer data warehouses.

The Data Model Can Change Radically

Databases are always in flux. Any metabase can provide the flexibility of renaming columns and moving tables without affecting metadata-driven applications; but an extraction engine metabase allows more drastic changes (which are common in real-world data systems) without ill effect. Columns can be moved to or duplicated in other tables, thus the data model can be normalized for maintainability or de-normalized for performance, and all applications and reports immediately apply the new data schema.

The User Does Not Need To Know the Data Model

Because the users interact only with Data Items, they need not be concerned with the database schema. While they must be familiar with the fields stored in the database, they do not need to know which fields are in which tables in which database instance on which server.

Reporting Applications Can Concentrate On Reporting

The Maestro Metabase engine "objectifies" the sub-setting or extraction of data, which allows reporting applications to simply call on the metabase for data. This has enabled analysis and reporting tools to concentrate on charts and graphs without concern for data specifics.

Advancements

Throughout three years of metabase development, JJT has made numerous enhancements to stored metadata, increasing its ability to handle the ever-changing needs of the data analysis community. After all, data is rarely simple and straightforward. While the Data Item - Data Source metadata model described in this paper affords basic SQL generation, small additions to the metadata objects can provide significant features. Some of these advancements are briefly described below.

SQL Generation Aids

In some cases, when metadata is used to dynamically generate SQL, data complexities require more information to accurately determine the most efficient SQL possible. This can occur, for example, when the same data is available in different DBMSs and/or on different host machines. In such cases it may be beneficial to define which is more efficient. Metadata objects can be expanded to include "efficiency" and join-key specifics, which enable the metabase to make more intelligent decisions when generating SQL code.

Views Plus

Constraining and extracting data in terms of abstract Data Items leads to a logical extension: calculated functions of Data Items. This level of abstraction allows calculated columns to be created based on "virtual" columns that may span tables, hosts, or even database systems. Data Item functions can be created to provide scalar (row-based) calculations or summary (grouping) functions.
Also, various *Filtered* Data Sources can be created to front manipulated data, such as transposed data sets. These calculations can be defined in the metadata, so that they are made during extraction and remain independent of table location.

Architecture

As the volume of metadata grows with each new table, performing run-time operations such as SQL generation can become costly for an application process. For very large metadata stores, the Maestro Metabase has been adapted for a *client-server* environment. The metabase server process runs alongside any DBMS servers and is contacted by any client application via TCP sockets. This process can take advantage of caching for much better performance.

User Interface Enhancements

When utilized well, metadata can be used to enhance a data-driven user interface. Details about specific fields or Data Items are extremely valuable in ad hoc reporting interfaces, where data type information can be used to determine the role of various fields in a report.

Furthermore, Data Items containing *distinct value* information add value to Item-Source metadata. If the metadata contains the range of values for a field, or a distinct list of values in that field, then constraints (`where` clauses) can be built using elegant interfaces that call upon that knowledge.

As more metabase products are developed, other advancements will continue to evolve metadata exploitation into a mature technology.

Conclusion

Global metadata is increasingly popular for data warehouse administration, but application use has been limited. Metadata-driven analysis reports and applications are shielded from data model changes and restructuring. Generating SQL code dynamically from metadata is the *next step* in metadata exploitation. It shields users from having to keep up with details of the data model schema, and allows analysts to extract or subset data without knowing SQL programming. The Data Item-Data Source metadata model provides the abstraction necessary to adapt a robust, object-oriented metabase to the growing needs of the data analysis community.

Acknowledgements

The author would like to express his appreciation to Vivian Badillo and Tim Feeley for their assistance in this paper, as well as John T. Stokes and the staff at JJT for making the Maestro Metabase possible.

Contact

Jonathan Stokes
JJT Inc.
1610 West Ave.
Austin, TX 78701
(512) 474-6743x21
jonathan@jjt.com