Querying the Data Warehouse with the SQL Procedure SELECT Statement

Kirk Paul Lafler, Software Intelligence Corporation

ABSTRACT

To appreciate the wealth of information a data warehouse has to offer, you need to understand how to communicate with it. Information is typically accessed using an SQL query. The statement responsible for querying the many tables and returning the requested results in a data warehouse environment is the SELECT statement. It is an indispensable piece of the data retrieval process. This paper looks at the structure of the typical data warehouse, the types of information stored in it, and how users access and retrieve data from it.

INTRODUCTION

A data warehouse plays an important role in modern computer systems. It stores and organizes operational enterprise-wide data consisting of a mix of older unintegrated "legacy" application systems. Data warehouse data is accurate as of some moment in time. Its purpose is to support management and end-user analysis and decision making.

Operational data passes into a data warehouse after being captured and filtered. Only data meeting the needs of management and decision support activities is passed. As it is being passed from the operational environment, data usually undergoes a transformation process prior to being stored in the data warehouse. Structurally, data warehouse data bares no resemblance to data in the operational environment. Consequently, data redundancy is normally not a problem.

THE DATA WAREHOUSE STRUCTURE

A data warehouse has a unique structure as illustrated in Figure 1. It consists of current and older operational data combined with varying levels of summarizations and detailed vantage points. In Inmon and Hackathorn's book "Using the Data Warehouse", they describe data warehouse data as "data spanning a spectrum of time and the data relationships between two or more tables may be many." A data warehouse is organized with data consisting of major business areas such as customer, product, vendor, and activity.

Four types of data may be stored in a data warehouse: 1) older detail data, 2) current detail data, 3) lightly summarized data, and 4) highly summarized data.

Older detail data represents data that is not very recent, maybe as old as ten years or longer. It is voluminous and most frequently stored on mass storage such as tape, although more expensive disk storage may be used. Its level of detail is consistent with current detail data (see later description), but due to its longer time horizon is typically migrated to a "less-expensive" alternate storage medium.

Current detail data represents data of a recent nature and always has a shorter time horizon than older detail data. Although it can be voluminous, it is almost always stored on disk to permit faster access.

Lightly summarized data represents data distilled from current detail data. It is summarized according to some unit of time and always resides on disk.

Highly summarized data represents data distilled from lightly summarized data. It is always compact and easily accessible and resides on disk.

A final component of the data warehouse is that of ***metadata***. Metadata is best described as data about data. Simply put, it provides information about the structure of a data warehouse as well as the various algorithms used in data summarizations. It provides a descriptive view, or "blueprint", of how data is mapped from the operational level to the data warehouse.

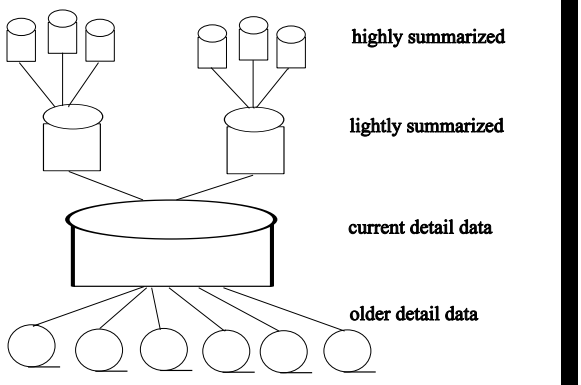

Figure 1. Data Warehouse Structure.

DATA WAREHOUSE FLOW OF DATA

A typical data warehouse is populated with data from the operational environment. Before data is physically stored in a data warehouse, it is first verified for correctness, then transformed through a process called integration. With integration, data becomes more organized and manageable by enforcing consistent data rules such as naming conventions, assigning consistent physical attributes, and adhering to consistent measurements.

It has been said that data entering a data warehouse represents a moment in time. Put another way, data warehouse data represents a collection of moments in time. No changes are usually permitted once data is captured in a data warehouse, as long as the underlying enterprise-wide operational data is correct. This assures a "safe" and nonvolatile environment to better enable analysis and decision making.

ADMINISTERING THE DATA WAREHOUSE

Administering and tuning activities in a data warehouse environment are sometimes necessary for improving access and performance. Indexing is a good example, since the number of indexes usually assigned to data at higher levels of summarization are much greater than those at lower levels of detail. This is due to the sheer volume of data found at lower levels. It is simply not desirable to assign indexes at lower levels.

Restructuring efforts, such as those found with indexing operations, require greater resources with lower levels of detail compared to data found at higher levels of summarization. Data administration activities are reserved mostly for data at higher levels of summarization. Consequently, data at lower levels of detail are usually left alone.

THE INFORMATION HIERARCHY

Information is vital to any organization. End-users access information for analysis and decision-making activities. As the needs of the enterprise require, access to a data warehouse range from infrequent and vague requests to frequent, repetitive, and specific requests. Questions and answers about end-users as well as the information most often requested appear in Figure 2.

Q. Who requests information from a data warehouse?
A. End-users with varying degrees of knowledge.

Q. At what levels of the organization do end-users support?
A. Top, middle, and line management.

Q. What type of information is provided to top management?
A. Summary and strategic information.

Q. What type of information is provided to middle management?
A. Specialized reports, spreadsheet analysis, and informational reports.

Q. What type of information is provided to line management?
A. Daily, weekly, and monthly reports.

Figure 2. Questions and Answers - End-Users.

QUERYING THE DATA WAREHOUSE

Access to data warehouse data is usually accomplished with a ***query***. In Inmon and Hackathorn's book, they define a query as, "a request for access to information in the data warehouse, with possibly some processing of that data before the results of the query are returned to the end user." They have classified end-user queries using the following taxonomy, see Figure 3.

Query	Looks at a lot of data. < or > Looks at only a record or two.
Query	Submitted repeatedly. < or > Submitted only once.

Figure 3. Taxonomy of Queries.

They provide greater insight by describing unique end-user characteristics and how queries are typically used in a data warehouse environment as illustrated in Figure 4.

Planned Query	Precanned. End-user knows what is being sought.
Ad-Hoc Query	Iterative process of submitting a query, looking at the results, modifying the query, and resubmitting it.

Figure 4. Planned versus Ad-hoc Queries.

EXAMPLES OF SQL QUERIES

Retrieve and Display Data

To extract and retrieve data, the SQL Procedure's SELECT statement is used. In the next example the desired columns or variables are displayed in the order indicated in the SELECT statement.

```
PROC SQL;
  SELECT   SSN,  SEX
         FROM  libref.PATIENTS;
QUIT;
```

In this example, the columns SSN (Social Security Number) and SEX from the PATIENTS data set are selected and displayed.

Sum and Display Data in Groups

To sum and display data in groups, the GROUP BY statement is used. The GROUP BY clause is used when a summary function is used in a query. The column being summed is WEIGHT (Patient's Weight) with the results of the query being grouped by SEX (Patient's Gender).

```
PROC SQL;
  SELECT SEX, SUM(WEIGHT) AS TOTWEIGH
      FROM  libref.PATIENTS /* Shortness of Breath */
         WHERE  SYMPTOM='10'
         GROUP  BY  SEX;
QUIT;
```

In this example, let's first examine the significance of the WHERE clause. It tells the SQL processor to extract only those rows (records) that contain a value of '10' (Shortness of Breath) in the SYMPTOM column. Rows not meeting this criteria are automatically excluded from the query. Then, the columns SEX (gender) and WEIGHT are selected from the PATIENTS data set. Next, the SQL processor groups rows (in ascending order) by the value found in the column SEX and totals each patient's weight storing the results in TOTWEIGH. Finally, the columns SEX and TOTWEIGH from the PATIENTS data set are selected and displayed.

Arrange Results in Ascending Order

To arrange results in ascending order, we will direct SQL by using the ORDER BY clause. One or more columns can be selected for sorting. One or more columns can be ordered in either ascending and/or descending order. The default sort order is ascending (lowest to highest). To override the default order (ascending order), you need to specify DESC (arrange in descending order) following the column-name that is specified.

```
PROC SQL;
  SELECT  LASTNAME, EDUC
         FROM libref.PATIENTS
         ORDER  BY  EDUC;
QUIT;
```

In this example, all rows are first arranged in ascending order by the column EDUC (patient's years of education). Then the columns LASTNAME and EDUC from the PATIENTS data set are selected and displayed.

Grouping Data

In the following query, the patients weight is summed and then are grouped according to the region of the country they live in.

```
PROC SQL;
   SELECT REGION, SUM(WEIGHT)
         FROM PERM.PATIENTS
         GROUP BY REGION;
QUIT;
```

Using Summary Functions

The following example illustrates how the MAX summary function can be used to obtain the maximum (largest) date value in each row within the Patients table.

```
PROC SQL;
   SELECT SSN, SEX,
         MAX(DOB,DOD,EXAM_DT,DIAG_DT,
                  ADMIT_DT,REL_DT) AS
                  ROWDATE FORMAT=MMDDYY8.
                  FROM PERM.PATIENTS;
QUIT;
```

Using the BETWEEN Condition

The BETWEEN condition selects rows having column values within a designated range of data values. The following example illustrates how patients residing in Region '1' through '3' can be selected.

```
PROC SQL;
   SELECT REGION, SSN
         FROM PERM.PATIENTS
         WHERE REGION BETWEEN '1' AND '3';
QUIT;
```

Using the IN Condition
The IN condition selects one or more rows based on the matching of a column value to a defined set of values. The following query selects patients who have a Symptom of '01' (Sleepiness), '05' (Weakness), or '10' (Shortness of Breath).

```
PROC SQL;
   SELECT SSN, DOB, SYMPTOM
         FROM PERM.PATIENTS
         WHERE SYMPTOM IN ('01','05','10');
QUIT;
```

Using the NULL Condition
Rows are selected with the IS NULL condition when a column value is missing (null). The following query illustrates patients that are NOT deceased.

```
PROC SQL;
    SELECT SSN, SEX, DOB
         FROM PERM.PATIENTS
         WHERE DOD IS NULL;
QUIT;
```

Using the LIKE Conditon
The LIKE condition selects rows from a table when a string search matches a predetermined pattern of characters. Since the LIKE condition is case-sensitive, it may be useful to specify the wildcard character "%" to match any number of characters. Another method of performing wildcard searches is to use the undescore "_" to represent an arbitrary character. The following query displays Clinics whose name begins with the uppercase character "N".

```
PROC SQL;
    SELECT CLINIC, REGION
         FROM PERM.PATIENTS
         WHERE CLINIC LIKE 'N%';
QUIT;
```

The following query displays Clinics whose firstname begins with 'M' and consists of a total of four characters.

```
PROC SQL;
     SELECT SSN, FRSTNAME
         FROM PERM.PATIENTS
         WHERE FRSTNAME LIKE 'M___' ;
QUIT;
```

Creating a Simple Index
A simple index is specifically defined for one column in a table and must be the same name as the column. The following example illustrates the process of creating a simple index.

```
PROC SQL;
   CREATE INDEX SSN ON PERM.PATIENTS(SSN);
QUIT;
```

Creating a Composite Index
A composite index is specifically defined for two or more columns in a table and must be a different name as the column. The following example illustrates the process of creating a composite index.

```
PROC SQL;
  CREATE INDEX REGSEX ON
       PERM.PATIENTS(REG, SEX);
QUIT;
```

Using the UNIQUE Keyword
The UNIQUE keyword prevents the entry of a duplicate value in a table. This keyword should be used with care since there may be times when more than one occurrence of a data value is necessary. Should multiple occurrences of the same value occur, the UNIQUE keyword is rejected and an index is not created for that column. The following example illustrates the process of defining the UNIQUE keyword for SSN in the Patients table.

```
PROC SQL;
  CREATE UNIQUE INDEX SSN ON
      PERM.PATIENTS (SSN);
QUIT;
```

Creating a View
Views behave in many ways as a table, even though they are not a table. A table stores data while a view is a set of instructions. Since a view is only a set of instruction, they cannot be updated as tables can. Consequently, views are "read-only".

The following example illustrates the process of creating an SQL view.

```
PROC SQL;
    CREATE VIEW  PERM.FEMALES  AS
         SELECT LASTNAME, SSN, DOB
         FROM PERM.PATIENTS
                  WHERE SEX = 'F'
```

```
                ORDER BY LASTNAME;
QUIT;
```

The View Contents

The DESCRIBE statement is used to describe the view in greater detail. The information is directed to the SAS System Log. The following example illustrates how the statement is specified.

```
PROC SQL;
   DESCRIBE VIEW PERM.FEMALES;
QUIT;
```

Using Views in Procedures

Views can act as input SAS System data sets in all SAS procedues. The following example illustrates using the view called PERM.FEMALES in the PRINT procedure.

```
PROC PRINT DATA=PERM.FEMALES NOOBS N;
RUN;
```

Joining Tables of Data

A join of two or more tables provides a means of gathering and manipulating data in a single SELECT statement. A "JOIN" statement does not exist in the SQL language. The way two or more tables are joined is to specify the tables names in a WHERE clause of a SELECT statement. A maximum of 16 tables can be joined.

The next example illustrates a simple two-way join.

```
PROC SQL ;
  SELECT  *
   FROM libref.PATIENTS, libref.COSTS
     WHERE PATIENTS.SSN  =
               COSTS.SSN ;
QUIT ;
```

Using the Pass-Through Facility

The SQL Pass-Through facility is a SAS software product that allows DBMS-specific SQL statements to be routed to a database management system (of choice) to enable direct data retrieval. It accomplishes this by using the appropriate SAS/ACCESS interface.

The next example sends an ORACLE SQL query to an ORACLE database for processing. MYORAC is used as the connection alias.

```
PROC SQL ;
  CONNECT TO ORACLE AS MYORAC
    (USER=ABC  ORAPW=PASSWORD
     PATH="@ORAC77") ;
  %PUT &SQLXMSG ;
  SELECT *
    FROM CONNECTION TO MYORAC
        (SELECT SSN, EMPID,
                LASTNAME, SEX, AGE
            FROM EMPLOYEES
                WHERE AGE > 65) ;
  %PUT &SQLXMSG ;
  DISCONNECT FROM MYORAC ;
QUIT ;
```

CONCLUSION

Learning to communicate with a data warehouse brings a wealth of information possibilities to an enterprise. The SQL procedure offers a comprehensive and powerful set of tools. It is an indispenable part of the data retrieval process.

REFERENCES

Inmon, W.H. and Richard D. Hackathorn, Using the Data Warehouse, Wiley-QED, 1994.

Kelly, Sean, Data Warehousing The Route to Mass Customization, John Wiley & Sons, 1994.

ACKNOWLEDGEMENTS

The author would like to thank the SUGI 23 Conference leaders especially Conference Chair Sally Goostrey and Section Chair Arthur Carpenter for asking me to present this topic in Advanced Tutorials.

CONTACT INFORMATION

The author welcomes any and all comments and suggestions. He can be reached at:

Kirk Paul Lafler
Software Intelligence Corporation
P.O. Box 1390
Spring Valley, CA 91979-1390
Tel: (619) 670-SOFT or (619) 670-7638
Email: KirkLafler@CompuServe.com