iSeries

Business Integration using DB2 OLAP on iSeries

Experience Report

iSeries

Business Integration using DB2 OLAP on iSeries

Contents

Chapter 1. iCola Introduction

The iCola scenario is a test scenario developed by the IBM(R) Customer Solutions Test team. The Customer Solutions Test team designs, implements, evaluates, and deploys customer-like scenarios in a fashion similar to that used by information technology architects throughout the industry. The iCola scenario tests database functionality as well as business intelligence products. In particular, the current iCola scenario focuses on IBM DB2(R) OLAP Server(TM). In the future, iCola plans to incorporate additional database business intelligence functionality and potentially more tools. Additional future topics are examined in the Future Ideas section of this document.

This section provides an overview of Online Analytical Processing (OLAP) and details an overview of the iCola scenario.

This section is divided into two sub-sections:
- What is OLAP
- Overview of the iCola scenario

What is OLAP

Online Analytical Processing (OLAP) empowers users to ask intuitive and complex ad hoc questions about their business, such as, "What is the profitability for the third quarter across the southeast region for certain focus products?" Such a question requires multiple perspectives on the data, such as time, regions, and products. Each of these perspectives are called *dimensions*.

Relational data can be considered two-dimensional because each piece of data (a fact) correlates to one row and one column (a dimension). The dimensions in a multidimensional database are higher-level perspectives of the data that represents the core components of a business plan, such as Accounts, Time, Products, and Markets. In an OLAP application, these dimensions rarely change.

Each dimension has individual components called *members*. For example, the quarters of the year are members of the Time dimension and individual products are members of the Products dimension. There can be hierarchies of members in dimensions, such as months within the quarters of the Time dimension. Members tend to change over time, for example, as your business grows new products and customers are added.

OLAP has evolved into a mainstream information technology and is a major component of business intelligence solutions across all industries today. The increasing demands for analyzing large amounts of data and empowering a growing number of employees throughout the enterprise to make informed business decisions are pushing the limits of OLAP solutions.

IBM(R) DB2 OLAP Server(TM) is an OLAP product that can be used to create a wide range of multidimensional planning, analysis, and reporting applications. IBM DB2 OLAP Server delivers analytic applications for fast, intuitive multidimensional analysis, allowing users to ask questions in an intuitive business language. OLAP Server processes multidimensional requests that calculate, consolidate, and retrieve information from a multidimensional database, a relational database, or both.

DB2 OLAP Server is based on the OLAP technology that was developed by Hyperion Solutions Corporation. References to Hyperion Essbase and Hyperion Integration Server exist in the interface and throughout the documentation.

DB2 OLAP Server includes all of the capabilities of Hyperion Essbase. In addition, it offers the option of storing multidimensional databases as sets of relational tables. Regardless of the storage management option that is chosen, Essbase Application Manager and Essbase commands can be used to create an Essbase application and its associated databases. Furthermore, there are over 70 Essbase-ready tools provided by independent software vendors that can access multidimensional databases transparently.

Overview of the iCola scenario

iCola is a fictitious medium-sized beverage company that acts as a middleman between beverage manufacturers and customer suppliers. iCola has been in business for a few years and has collected sales and customer data; however, management has not been able to use this data to make business decisions because a mechanism to analyze the collected data was not available. Management at iCola has become aware that competitors are utilizing data to make marketing and sales decisions and are improving the quality of service as well as increasing revenue based on data analysis.

After planning sessions by management, the iCola company has decided to take advantage of business intelligence. Business intelligence is the concept of creating informational databases from operational data and using that data for analysis and decision making. Data is consolidated from multiple sources onto a single database server, and it is “transformed” as part of the consolidation process.After the data is consolidated, tools are used to analyze the data. Some of the tools for analysis are: Decision Support Systems (DSS), Executive Information Systems (EIS), Multidimensional Analysis Tools (MDA), Online Analytical Processing (OLAP), and Data Mining Tools. iCola has decided to start their business intelligence project by creating a data mart and using IBM(R)’s DB2(R) OLAP solution for analysis.

The iCola company owns iSeries(TM), pSeries(R), and xSeries(R) hardware. Current(R) operational systems run on iSeries and reporting operations are performed on pSeries and xSeries servers. During the first phase of implementation of the business intelligence solution, a data mart was created on an iSeries system. Furthermore, DB2 OLAP server was placed on a second iSeries system and DB2 OLAP Analyzer was installed on a Windows(R) xSeries server.

Figures 1 and 2 illustrate the key systems and software of the iCola scenario:

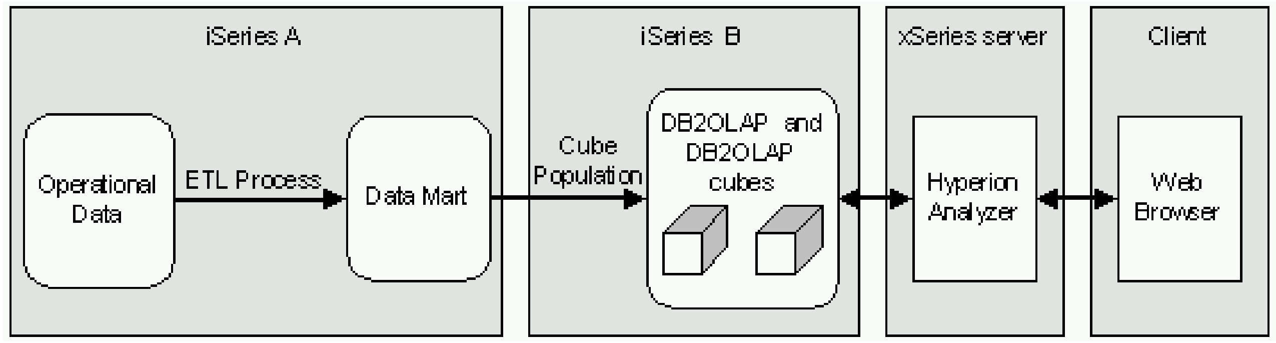


Figure 1: iCola Scenario Overview

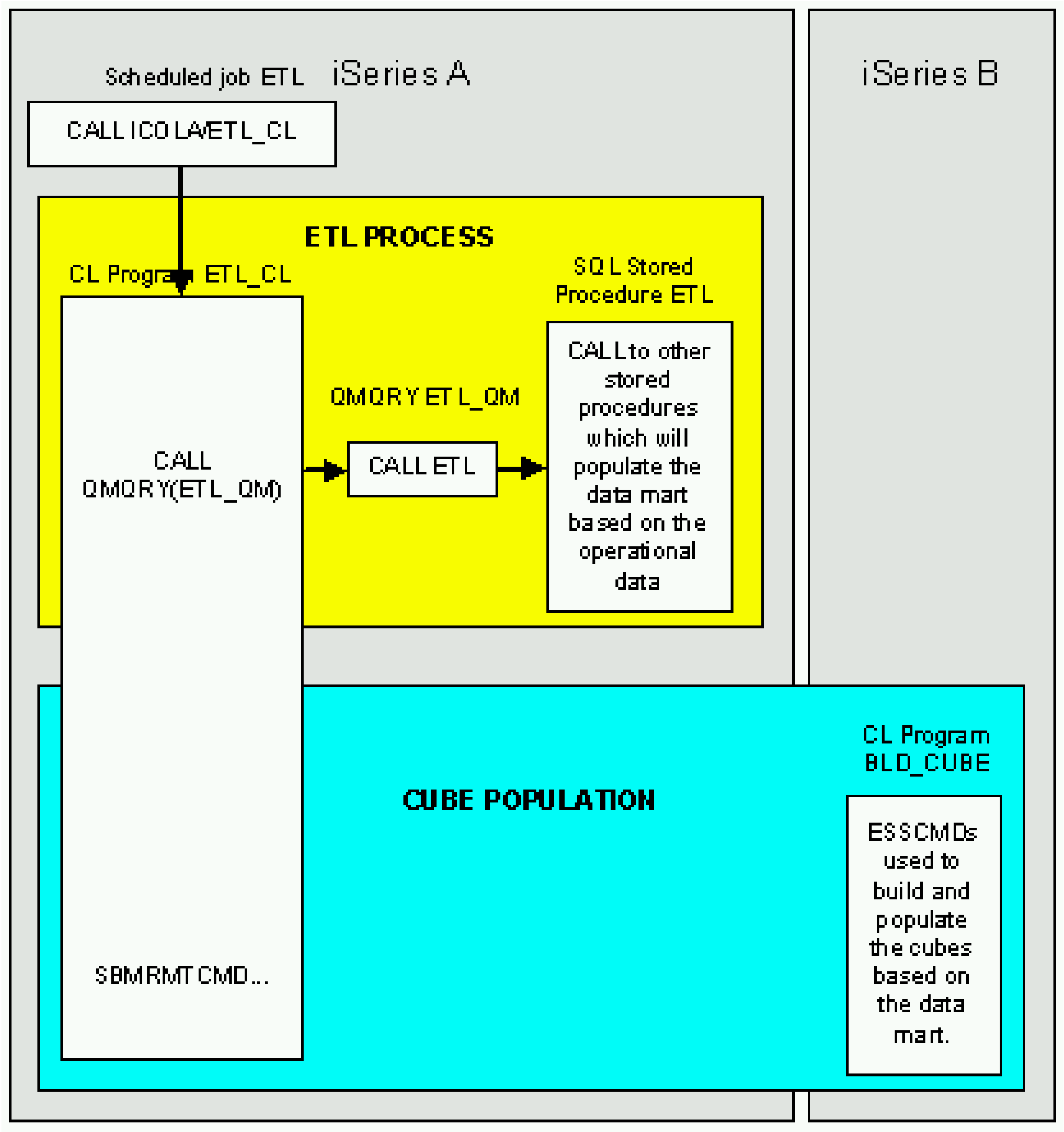


Figure 2: ETL Process and Cube Population in detail

iSeries A can be at either OS/400(R) V5R2 or V5R3. However iSeries B must be at OS/400 V5R2. Since the most recent iSeries version that IBM DB2 OLAP Server(TM) 8.1 runs on is OS/400 V5R2, iSeries B runs on OS/400 V5R2.

The remaining chapters detail the key systems and software:
- The data mart contains historical data for the iCola scenario
- The ETL process is the vehicle that moves the operational data into the data mart
- The data then is populated into a multidimensional cube, where it is stored in a format that business analysts can use
- Finally, iCola uses DB2 OLAP Server Analyzer to examine the data and build reports

Chapter 2. Data Mart

This section highlights the DB2 Universal Database(TM) for iSeries(TM) data mart setup for the iCola scenario along with the concept of data warehousing.

The data that is collected from the daily transactions of the business is known as operational data. This data may contain information that is useful to business analysts. For example, analysts can use information about what products were sold in which regions at what time of year to look for anomalies or to project future sales. However, there are several problems if analysts access the operational data directly:

- The analysts might not have the expertise to query the operational database. For example, querying Information Management System databases requires an application program that uses a specialized data manipulation language. In general, programmers who have the expertise to query the operational database have a full-time job in maintaining the database and its applications, not analyzing its contents.
- Performance is critical for many operational databases, such as databases for a bank. The system cannot handle analysts making ad hoc queries.
- The operational data generally is not in the best format for use by business analysts. For example, sales data that is summarized by product, region, and time is much more useful to analysts than raw data. Also, tools such as DB2(R) OLAP can create data cubes much easier with a data mart than with operational data.
- The information may be spread out of the organization. Analysts need to access centralized data instead of disparate sources of data.

Data warehousing solves these problems. Data warehousing is the design and implementation of processes, tools, and facilities to manage and deliver complete, timely, accurate, and understandable information for decision making. With data warehousing, stores of informational data are created. This data is extracted from the operational data and then transformed into a format business analysts can use. A data mart is a smaller version of data warehouse, and is used in the iCola scenario. There are various warehousing tools that can copy all the sales data from the operational database, perform calculations to summarize the data, and write the summarized data to a separate database from the operational data. Analysts can then query the separate database (the data mart) without impacting performance on the operational databases.

Application overview

The iCola company has an operational database for daily operation and a data mart for analysis. Both the operational database and the data mart reside in DB2 Universal Database(TM) for iSeries(TM).

The most common tasks that are required to set up a data mart include:

- Exploring the operational data to define which tables and fields to include in the data mart. These are the fields business analysts are most interested in.
- Defining a star schema model for the data in the data mart. A star schema is a specialized design that consists of multiple dimension tables, which describe aspects of a business (examples of dimensions include customers, products, and time), and one fact table, which aggregates fields from the dimension tables. The fact table typically contains sales figures as well.

An excellent resource for creating data warehouses and data marts is IBM(R) DB2 Universal Database Business Intelligence Tutorial, which can be found at http://www.ibm.com/software/data/db2/db2olap/docs/V71docs/db2tu/frame3.htm#db2tussw.

Operational Database

The operational database contains data from the daily transactions of the iCola business.

Examples of this data include:
- Products that iCola sells
- Customers that iCola sells to
- Suppliers that iCola buys their products from
- Shipping types that iCola uses

The operational data for iCola existed long before iCola decided to enhance their company with business intelligence. The data mart was added to the scenario without impacting the operational data or the transactions of the business.

Data Mart

Operational data generally is not in the best format for business analysis tools. A better format for the data is a star schema. A key difference between the operational data and the star schema for iCola is that the relations between the database tables have been simplified. The star schema is set up in such a way that data that is not useful to analysts is not included in the data mart. The data that is useful is then arranged in a hierarchical structure.

Application design points

Data mart design

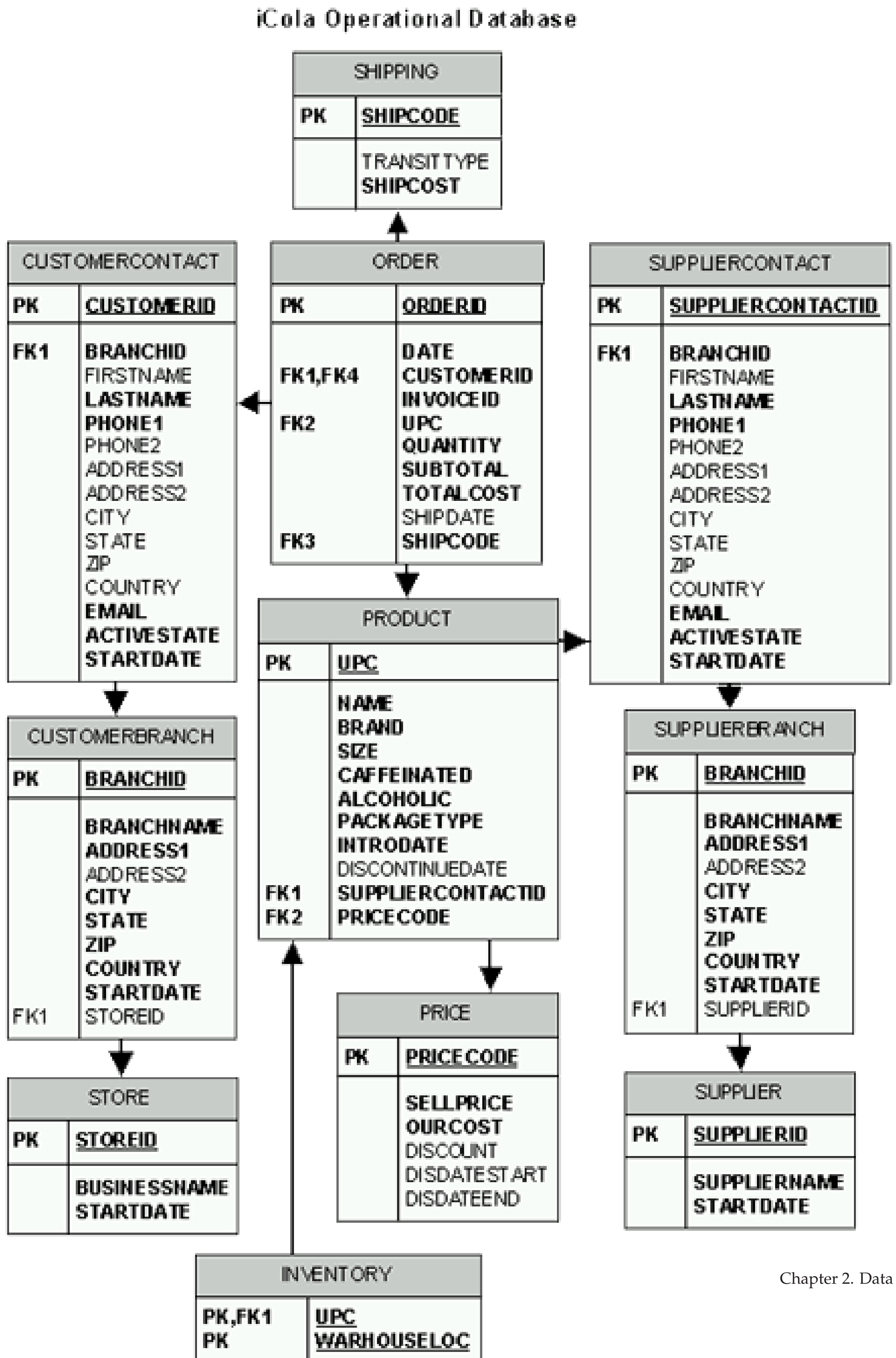


Figure 3. iCola Operational Database

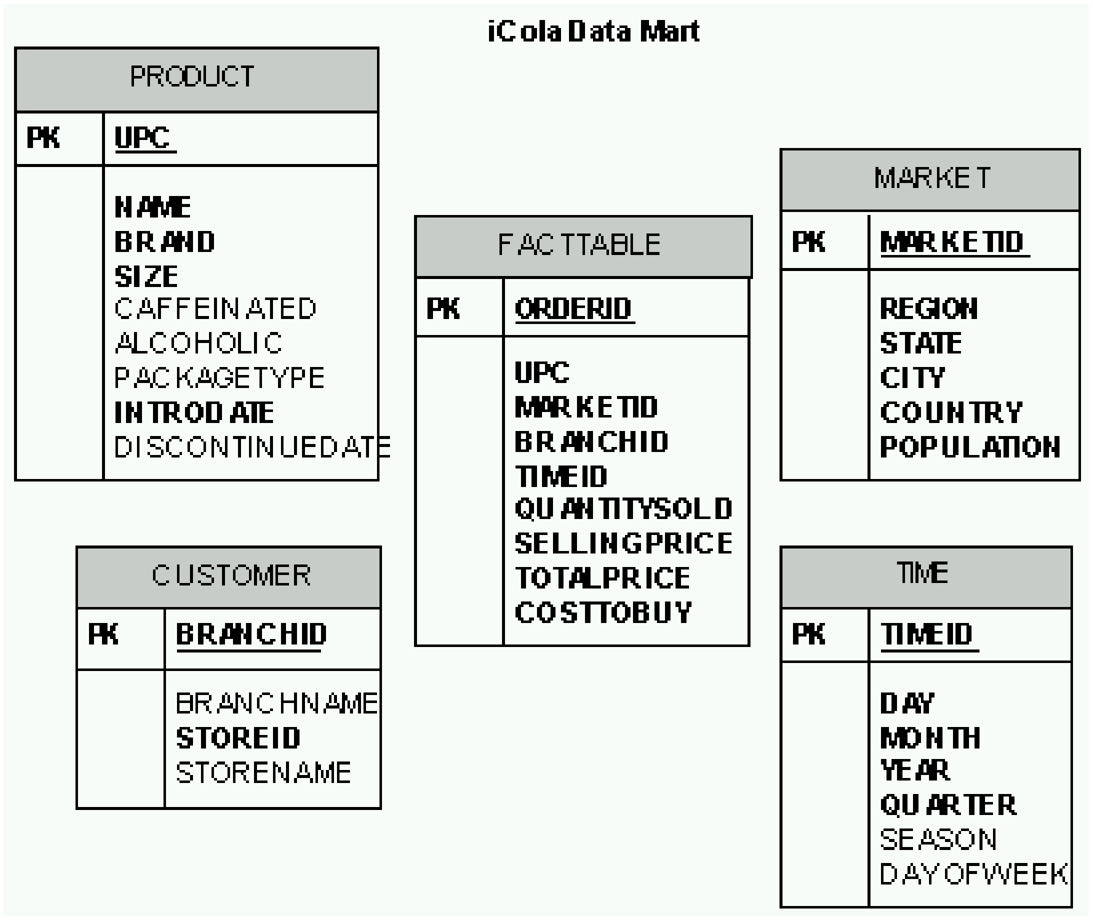


Figure 4. iCola Data Mart

Data mart design begins by analyzing the operational data, and looking for relationships among the data and determining what information might be of the most interest to users.

The tutorial *Designing the Star Schema* refers to the importance of determining measures and dimensions. Measures are numeric values that are measurable and additive. Examples of measures include:

- Sales data - every order generates
  - Cost of Goods Sold (COGS) value
  - Sales value
- The number of customers iCola has on a particular day
- The average number of products purchased on a particular day.

Dimensions affect how the measures are viewed. For example, when looking at sales figures, there is a time dimension involved indicating when the sales took place. It is also helpful to know sales by product or by customer.

The following were considered in the iCola data mart design:

- Measures
  - Costs of Goods Sold (COGS)

  - Sales (the amount the product sells for)
- Dimensions
  - Time
  - Product
  - Market
  - Customer

The fact table holds the measures, or facts, along with references to each dimension in the data mart. The measures do not provide any useful information without taking the dimensions into account.

iCola's fact table (called FACTTABLE) is based on the ORDER table in the ICOLA database. Rows in the ORDER table contain both measures and dimensions. The dimensions tie a time, location, and customer to the sale of a product.

The following table shows how the operational table ORDER and the FACTTABLE in the data mart are related:

ORDER	**FACTTABLE**
ORDERID	ORDERID
DATE	TIMEID
CUSTOMERID	BRANCHID
CUSTOMERID	MARKETID
QUANTITY	QUANTITYSOLD
SUBTOTAL/QUANTITY	SELLINGPRICE
SUBTOTAL	TOTALPRICE
UPC	UPC
OURCOST (PRICE table)	COSTTOBUY

Dimension tables should have a single-field primary key. This key is often an identity column, consisting of an automatically generated incrementing number. The reason a primary key is created is because it is used to map dimension elements with rows in the FACTTABLE. The other fields in the dimension table contain the full descriptions of relevant data for iCola. For example, the PRODUCT dimension is based on the PRODUCT table in ICOLA database. There are fields in it that contain the product name, the brand, the size, the package type, etc. These fields in the data mart do not contain codes that link to other tables other than the FACTTABLE.

A key difference between the fact table and the dimension tables is their shape:
- The fact table generally contains many records, but does not contain many columns. This is because the data that the fact table holds is either measures data or primary keys to the dimension tables. Consequently, the shape of this table is long and narrow.
- The dimension tables generally do not contain many records, but have many columns to detail the dimensions, since they are full descriptions of the dimensions. Consequently, the shape of this table is short and wide.

This hierarchy of dimensions allows us to perform "drill-down" functions on the data. Examples of queries that can be performed include:
- A query that calculate sums by brand.
- A query that calculates the quantity of products purchased by each market.
- Calculation of the sum of the individual products for a particular month.

The data mart ICOLADM contains one fact table and four dimension tables.

- FACTTABLE - This is the fact table which contains information about market, product, customer and time, along with the COGS and Sales data. This table is populated mainly from the ICOLA.ORDER table.
- PRODUCT - This is the dimension table which contains information about the products. All the information from this table is taken from ICOLA.PRODUCT.
- MARKET - This is the dimension table which contains information about the markets. This table is populated based on the table ICOLA.CUSTOMERBRANCH in the operational database.
- CUSTOMER - This is the dimension table which contains customer information. This table is populated based on the tables ICOLA.CUSTOMERBRANCH and ICOLA.STORE in the operational database.
- TIME - This is the dimension table which contains the timeline. This table is populated by a simple Java$^{(TM)}$ program that enters in one year's worth of dates.

There are also several other helper tables in ICOLADM:

- REGION - This contains all 50 states, along with their corresponding regions and countries. (Countries were included in case iCola expanded to international markets and needed to add states and providences from different countries). This table is used by the ETL process to populate the MARKET table.
- ETLPROCESS - This table is used to indicate when the last time the ETL process started and ended. The ETL process will be explained in a later section.

Application Setup

Data Mart

The data mart, ICOLADM can be created by running a set of SQL statements. Refer to Appendix A for details of the SQL statements.

Chapter 3. ETL process

To move the operational data to the data mart, administrators use an Extract, Transform, and Load (ETL) process. This ETL process is typically run on a scheduled interval when customer activity is low (the iCola scenario runs their ETL process daily at midnight). The ETL process is vital for keeping the data mart populated with recent data from the operational database. An ETL process can be fairly simple or extremely complex, depending on how much data needs to be moved and how much change the data must undergo.

Here are the steps of the ETL process in detail:

1. Extract - The relevant data must be selected from the operational data, and should correspond with the interval of time that the ETL process is scheduled. If the ETL process runs daily, then typically the application should select the last 24 hours worth of data. Choosing the relevant data here can save time and effort later.

   Also, the data mart might require more data than what the operational data provides. Here is an example in iCola's ETL process where additional data is needed:

   - iCola's data mart keeps track of regional information regarding customer orders. If a customer from Rochester, MN orders a product, iCola needs to know that the customer is in the Midwest. The data mart contains a table called REGIONS of all the U.S. states along with their regions. During the ETL process, this regional data is extracted along with the city and state data, and stored in the data mart.
2. Transform - Chances are the data that is selected from the operational data is not in the same format as the data that needs to be inserted into the data mart. The ETL process should handle any necessary transformations.
   - Here is an example of how iCola's ETL process transforms data:

     In the operational database, the data for each order is stored in the SQL "date" format. However, in the data mart, the date of each order must be in the format YYYYMMDD. To transform the operational data date to the data mart date, this SQL formula is used:

     ((YEAR(ICOLA.ORDER.DATE)*10000) + (MONTH(ICOLA.ORDER.DATE)*100) + DAY(ICOLA.ORDER.DATE))

   Often, the operational data must be "cleansed" by examining fields that might have multiple valid answers that refer to the same entity. For example, one customer might enter the city "Mt. Horeb", WI and another customer enters the city "Mount Horeb", WI. The ETL process should recognize that these cities are the same. In this simple scenario, data cleansing is not performed due to the nature of the data entered being in a controlled environment. However, data cleansing should be considered when evaluating how the data will be transformed.
3. Load - After transforming the data, it must be loaded into the data mart. This can be done with SQL insert statements.

There are products that administrators can use to aid in the implementation of the ETL process. One of these products is the IBM(R) DB2(R) Information Integrator. Because iCola is a simple scenario and requires an ETL process that runs on iSeries(TM), a tailored ETL processed was generated. In the future, iCola may consider using the IBM DB2 Information Integrator for their ETL process if the company expands.

In the following sections, the ETL process pertaining to the iCola store will be detailed further.

Application overview

Figure 5 illustrates the various components of iCola’s ETL process. The ETL process is only part of iCola’s scheduled process. In addition, the cube population process is also scheduled, and is described in the cube population section.

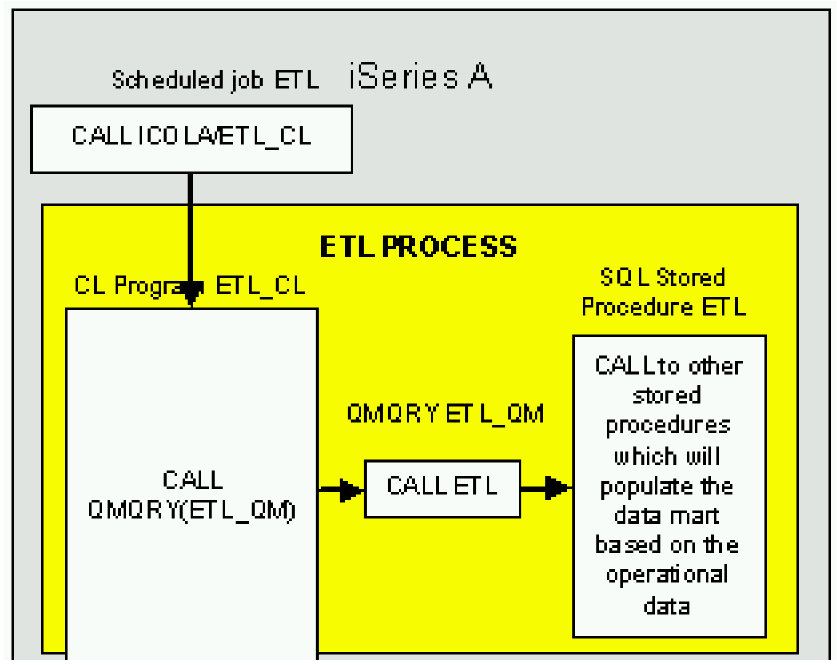


Figure 5: iCola’s ETL process (Note that this is a part of Figure 2 in the Overview section)

Scheduled job ETL: Using the Work with Job Schedule Entries (WRKJOBSCDE) menu, a scheduled job was created to enable the ETL process to start automatically every day at midnight. The application that is called by the scheduled job is the ETL program ETL_CL.

Command Language (CL) program ETL_CL: ETL_CL has two functions. The ETL portion of this program calls the Query Manager query ETL_QM. After the Query Manager query call is complete, ETL_CL also submits a command to the system that contains the cube (known as iSeries$^{(TM)}$ B) to start the cube population. Cube population is described in the Cube section.

Query Manager query ETL_QM: ETL_QM calls the SQL stored procedure ETL. The reason why iCola uses a Query Manager (QM) query to call the SQL stored procedure is because SQL stored procedures cannot be called directly from a command line (CL) program. However, SQL stored procedures can be called from a QM query, and a QM query can be called from a CL program.

SQL Stored Procedure ETL: The stored procedure ETL calls other SQL stored procedures, which are described in greater detail in the application design points:

- STARTETL
- CUSTOMER
- MARKET
- PRODUCT
- FACTTABLE
- ENDETL

These SQL stored procedures query the operational data and perform inserts into the data mart.

Application design points

- The SQL stored procedures are the heart of the ETL process. The following list describes the function of each SQL stored procedure:
  - ETL: The main stored procedure that calls the other stored procedures.
  - STARTETL: Inserts the time that the ETL process starts in the START column of the ETLPROCESS table. This table is used for logging purposes.
  - ENDETL: Inserts the time that the ETL process completes in the FINISH column of the ETLPROCESS table. This table is used for logging purposes.
  - CUSTOMER: Inserts information from the operational tables CUSTOMER and CUSTOMERBRANCH into the data mart's CUSTOMER table.
  - MARKET: Inserts regional information from the operational tables CUSTOMER and CUSTOMERBRANCH into the data mart's MARKET table.
  - PRODUCT: Inserts information from the operational table PRODUCT into the data mart's PRODUCT table
  - FACTTABLE: Inserts information from the operational table ORDER into the data mart's FACTTABLE

  If one compares the stored procedures above to the tables that compose the data mart, notice one of the tables is left out - the TIME table. Since time is constant and predictable, it is easiest to populate the TIME table with many entries at once by using a simple Java$^{(TM)}$ program instead of creating a stored procedure. The difficulty in creating a stored procedure to populate the TIME table is in handling situations where the TIME procedure is not able to run at its scheduled time. This could happen, for example, if a save of the system was being performed at the time when the scheduled ETL process usually runs.

Application Setup

The ETL process for the iCola scenario was set up by performing these steps:

1. Create the SQL stored procedures
2. Create the QM query
3. Create the CL program
4. Create the scheduled job
5. Ensure the profile running the ETL process has the proper authorities

Create the SQL stored procedures

The code used to create each of the SQL stored procedures can be found in Appendix B. In the SQL procedures, notice that the FACTTABLE, CUSTOMER, MARKET, and PRODUCT procedures use the keyword "EXCEPT" in the procedure. The EXCEPT keyword is new for DB2 Universal Database$^{(TM)}$ for iSeries$^{(TM)}$ V5R3. To illustrate how this keyword functions, refer to Figure 6:

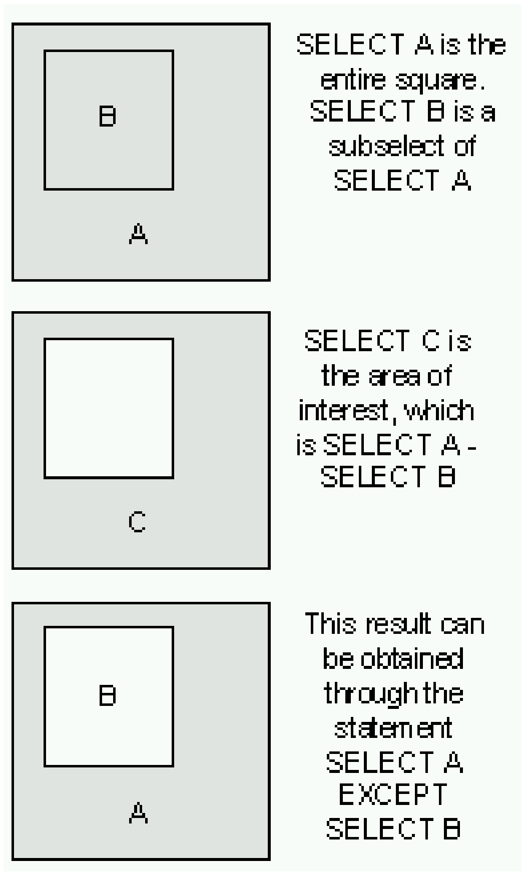

B
A
SELECT A is the entire square. SELECT B is a subselect of SELECT A
C
SELECT C is the area of interest, which is SELECT A - SELECT B
B
A
This result can be obtained through the statement SELECT A EXCEPT SELECT B

Figure 6: Illustration of the EXCEPT keyword

If the results produced from running *SELECT C* are the same results as running *SELECT A* - *SELECT B*, then to find the results of *SELECT C*, one would simply run: (*SELECT A*) EXCEPT (*SELECT B*). Likewise, to find the results of *SELECT B*, run (*SELECT A*) EXCEPT (*SELECT C*).

The EXCEPT keyword makes the creation of these SQL stored procedures easier. Because the operational data is comprehensive, only recently updated data needs to be inserted into the data mart, (i.e. data that is not already there). The EXCEPT keyword can be utilized to create these procedures.

The statements used to created the SQL procedures are located in Appendix B. Appendix B also contains statements that can be used to create the SQL procedures on OS/400(R) V5R2 (Since V5R2 does not support the EXCEPT keyword).

Create the QM query

To create the QM query to call the stored procedures, follow these steps:

1. Before creating the query, verify that the profile creating and calling the QM query has authority to use the appropriate SQL keywords. By default, profiles are only allowed to run the SELECT statement within QM. In the iCola scenario, the CALL statement needs to be run. To grant the authority to use other SQL keywords, sign on using a profile with *SECADM authority.
2. Start the QM interface by running STRQM.
3. To **Work with Query Manager Profiles**, select option 10.
4. From the list select the appropriate profile and choose option 2 to change the profile attributes.

5. Scroll down to the bottom of the parameters and for the option **Select allowed SQL statements**, choose Y.
6. The following screen displays various SQL statement keywords. To grant authority to use a particular keyword, type a "1" next to that keyword.
7. After verifying that the user profile has the authority to run the CALL statement, go back to the main Start QM screen and then select option 1.
8. Type in the appropriate library in which to create the query in the library parameter on the top of the screen and press enter. If there are any existing QM queries in that library, they will be displayed.
9. Under the library is the **Query Creation Mode** parameter. To create the query that runs a CALL statement, the **Query Creation Mode** should be SQL. If the **Query Creation Mode** is set to PROMPT, toggle to SQL by pressing F19 (shift + F7).
10. After verifying the **Query Creation Mode** is SQL, type option 1 on the top options field to create a new QM query.
11. A query creation screen should appear that is mostly blank. This is where the CALL statement is specified. To run the CALL statement, type `CALL ICOLA/ETL`.
12. When finished creating the QM query, press F3 to save changes and name the query.

Create the CL program

To create the CL program used to call the QM query, follow these steps:

1. First, create the source file to hold the CL program by running the following command:
   ```
   >CRTSRCPF FILE(ICOLA/QCLSRC) RCDLEN(92) TEXT('SOURCE FILE FOR CL')
   ```
2. Display the Program Development Manager by typing STRPDM. Select option 3 to work with members. In the **Specify Members** screen enter the QCLSRC file in the ICOLA library.
3. Once in the **Work with Members** screen, press F6 to create a new member. Give the member a title (ETL_CL) and specify to use CLLE as the source type.
4. A Source Entry Utility (SEU) screen will appear in which to enter the code to the CL program. In the iCola scenario, the following code was entered:
   ```
   PGM
   /* Call to QM Query */
   STRQMQRY QMQRY(ICOLA/ETL_QM)
   /* Populate the cube on iSeries B */
   SBMRMTCMD CMD('SBMJOB CMD(CALL PGM(ICOLA/BLD_CUBE))') DDMFILE(ICOLA/TO_SD)
   /* Print to the job log *:
   DSPJOBLOG OUTPUT(*PRINT)
   ENDPGM
   ```
5. When finished entering the CL program code, press F3 to save the changes.
6. To compile the CL program from the **Work with Members using PDM** screen, enter option 14 next to the member. To determine if the program compiled successfully, view the spooled files generated by using the WRKSPLF command.

Create the scheduled job

To create the scheduled job to run the CL program, follow these steps:

1. From the command line, work with job scheduled entries by running WRKJOBSCDE.
2. Enter F6 to create a new scheduled job. When the scheduled job was created for iCola, the following parameters were listed to create a scheduled job that runs every day at midnight:
   - Job name: ETL
   - Command to run: CALL PGM(ICOLA/ETL_CL)
   - Frequency: *WEEKLY

- Schedule Date: *NONE
- Schedule Day: *MON, *TUE, *WED, *THU, *FRI, *SAT, and *SUN
- Schedule Time: 00:00:00
- User: DB2OLAP

Authority

When creating the programs and procedures that make up the ETL process, it is important to verify that the profile running the ETL process has the proper authority to all the new objects. In the iCola scenario, the profile executing the scheduled job is DB2OLAP. Since all the objects that were created are in the same library (ICOLA), the following commands are run:

- This command grants DB2OLAP ownership and all authority to the iCola library:
  CHGOWN OBJ('/QSYS.LIB/ICOLA.LIB') NEWOWN(DB2OLAP)
- This command grants DB2OLAP ownership and all authority to all objects in the iCola library:
  CHGOWN OBJ('/QSYS.LIB/ICOLA.LIB/*') NEWOWN(DB2OLAP)

Key findings

The SQL keyword TRUNC

When creating SQL queries for the ETL process, information was needed on the price of the total order as well as the price per unit. The first attempt to accomplish this used this formula:

```
(ICOLA.ORDER.SUBTOTAL / ICOLA.ORDER.QUANTITY)
```

Division leaves multiple digits to the right of the decimal. However, the iCola data mart required there to be only two digits to the right of the decimal. In order to verify that only two digits were left to the right of the decimal after the calculation, this formula is now used instead:

```
TRUNC (ICOLA.ORDER.SUBTOTAL / ICOLA.ORDER.QUANTITY , 2 )
```

By specifying the number of places to the right of the decimal in the second parameter of the TRUNC keyword (in this case, 2), the formula leaves the specified number of decimal places and truncates all others.

Data selected by the FACTTABLE procedure

Another key finding involves the stored procedure FACTTABLE. Here is the SQL used to create the stored procedure:

```
CREATE PROCEDURE ICOLA.FACTTABLE ( )
LANGUAGE SQL
SPECIFIC ICOLA.FACTTABLE
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN INSERT INTO ICOLADM . FACTTABLE ( ORDERID , UPC , MARKETID , TIMEID,
QUANTITYSOLD , SELLINGPRICE , TOTALPRICE , COSTTOBUY , BRANCHID )
( ( SELECT ICOLA . ORDER . ORDERID , ICOLA . ORDER . UPC , ICOLADM . MARKET . MARKETID
, ( ( YEAR ( ICOLA . ORDER . DATE ) * 10000 ) + ( MONTH ( ICOLA . ORDER . DATE ) * 100 ) + DAY (
ICOLA . ORDER . DATE ) ) , ICOLA . ORDER . QUANTITY , TRUNC ( ICOLA . ORDER . SUBTOTAL /
ICOLA . ORDER . QUANTITY, 2 ) , ICOLA . ORDER . SUBTOTAL , ICOLA . PRICE . OURCOST , ICOLA
. CUSTOMERCONTACT . BRANCHID
FROM ICOLA . ORDER , ICOLA . PRODUCT , ICOLA . PRICE , ICOLA . CUSTOMERCONTACT,
ICOLA . CUSTOMERBRANCH , ICOLADM . MARKET
WHERE ICOLA . ORDER . CUSTOMERID = ICOLA . CUSTOMERCONTACT . CUSTOMERID
AND ICOLA . CUSTOMERCONTACT . BRANCHID = ICOLA . CUSTOMERBRANCH . BRANCHID
AND ICOLA . CUSTOMERBRANCH . CITY = ICOLADM . MARKET . CITY
AND ICOLA . CUSTOMERBRANCH . STATE = ICOLADM . MARKET . STATE
AND ICOLA . ORDER . UPC = ICOLA . PRODUCT . UPC AND ICOLA . PRODUCT . PRICECODE =
ICOLA . PRICE . PRICECODE
AND ICOLA.ORDER.DATE < CURRENT DATE)
EXCEPT
( SELECT ORDERID , UPC , MARKETID , TIMEID , QUANTITYSOLD , SELLINGPRICE, TOTALPRICE ,
COSTTOBUY , BRANCHID FROM ICOLADM . FACTTABLE ) ) ;
END ;
```

Originally, the procedure did not include the condition AND ICOLA.ORDER.DATE < CURRENT DATE in the where clause, but it was added for the following circumstance:

- During the ETL process which runs at midnight, and after the CUSTOMER procedure is run, a new customer registers with the site and places an order before the FACTTABLE procedure is run.

If this occurs, the customer information for that order will not be in the data mart. Consequently, when the data cube is built, errors will occur, since complete dimensional data is required. The solution to this problem was to add the where clause AND ICOLA.ORDER.DATE < CURRENT DATE to the SQL statement. Since the ETL process is scheduled to run at midnight, only orders between midnight and when the FACTTABLE procedure is run will not be included into the FACTTABLE. The amount of time between midnight and when the FACTTABLE procedure is run is only a few minutes, so only a few orders will not get into the FACTTABLE. Those orders that do not make it will be inserted the next day.

Chapter 4. Multidimensional Cube

At the heart of the iCola scenario are multidimensional database cubes used to hold historical summary data. These cubes can be created with IBM(R) DB2 OLAP Server. IBM DB2 OLAP Server(TM) provides fast, intuitive multidimensional analysis.

Database cubes are composed of various dimensions that sales analysts are concerned with such as products, time, and accounts. An example of a cube with these three dimensions is illustrated in Figure 7. The dimensions used to build a cube should be based on the data that will be extracted from the data mart.

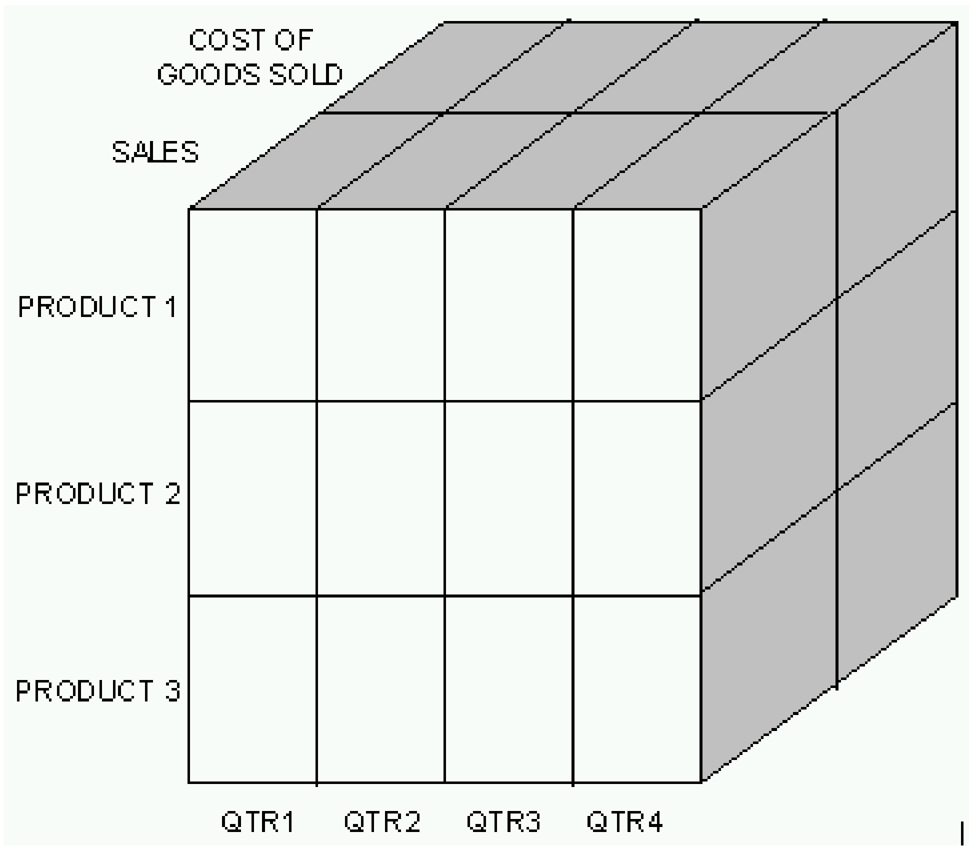


Figure 7: A multidimensional cube.

For example, by constructing a cube with the dimensions products, time, and accounts, the following are just some of the queries that can be extracted from the cube:

- Which products produced the greatest margin of profit (sales - expenses) during the first quarter?
- Which quarter did Product 1 have the least margin of profit?
- Which product during the course of the year had the greatest initial expense?

Take a closer look at the first example, determining which products produced the greatest margin of profit during the first quarter. To view this data, only one slice of the cube needs to be examined - the slice for QTR1. Likewise, to view sales figures for Product 1 during the course of the year, the Product 1 slice of the cube should be examined. To view the cost of goods sold for all products during the course of the year, the cost of goods sold slice should be examined. Fortunately, a user does not need to know which slice or section of the cube the data is stored in to generate the results for these queries. By using IBM DB2 OLAP Server Analyzer 8.1, the user indicates which data they are interested in and the results are automatically generated.

The queries listed previously can also be generated by using iCola's operational database (a relational database), since it contains the same data. However, the complexity of generating these and other queries decreases dramatically when using a multidimensional cube. Multidimensional databases have the power to quickly examine data from many different points of view. In many situations, the operational data is in disparate locations, so it is impossible to gather information by querying the operational data.

To glean the information for first quarter profits from the operational database, one would need to run these calculations in the query:

1. Determine which orders were placed in the first quarter.
2. For each first quarter order, determine the sales data by multiplying the sales cost of the item times the quantity sold.
3. For each first quarter order, determine the cost of goods sold data by multiplying the cost of the product times the quantity.
4. For each first quarter order, take the result of step 2 and subtract the result of step 3 to determine the margin of profit.
5. Add up the results of step 4 based on the product that was sold in the order to determine the margin of profit for each product.

By using IBM DB2 OLAP Server, the calculation work for this and many other queries is transparent to the end user.

Application overview

Choosing dimensions for the cube

The cubes for the iCola scenario are based on these five dimensions:

- TIME
- MEASURES (ACCOUNTS)
- PRODUCT
- MARKET
- STORE

Note: The TIME and MEASURES are standard cube dimensions. Because data cubes keep track of historical summary data, a TIME dimension is always present. Also, data cubes typically keep track of at least one numeric quantity (such as sales figures or quantity sold). This information is tracked in the MEASURES dimension.

The reason that the iCola cube contains the PRODUCT, MARKET, and STORE dimensions is because the company wanted to find the answers to questions such as:

- How do time of year and type of packaging effect sales of particular brands?
- What stores by region sell the most and least of a particular brand?
- Do certain products sell better in smaller stores vs. large chain stores?
- How do head to head sales compare between various products, by region and by store?

Expanding the dimensions in the cube outline

After choosing the dimensions for the cube, these dimensions must be expanded in the cube outline based on the data used to populate each dimension. Typically, each dimension contains several characteristics. For example, consider the MARKET dimension. Each market location has a region, state, and city. Since regions contain states, and states contain cities, the MARKET dimension has the following hierarchy.

- Market (Generation 1/Level 3)
  - Region (Generation 2/Level 2)
    - State (Generation 3/Level 1)
      - City (Generation 4/Level 0)

These layers are also known as generations or levels. Generation numbers refer to consolidation levels within a dimension, and increase in number starting from the root of the dimension working out toward the leaf. Level numbers also refer to a branch within a dimension; however, levels reverse the numerical ordering used for generations by counting up from the leaf member toward the root.

By defining these generations/levels in the MARKET dimension, data can be extracted for multiple markets. For example, to determine a product's success during the year based on location, the cube can calculate by city, state, and region.

Loading data into the cube

The multidimensional cube contains two categories of data which must be regularly updated. These categories are commonly known as the dimension build and the data load:

- Dimension build: this fills in the outline data for the PRODUCT, MARKET, and STORE dimensions. This data is needed because new products, markets, and stores may be added to the iCola scenario. For example, with the MARKET dimension, new cities may be added as iCola expands its market base.
- Data load: sales data taken from individual orders, such as:
  - What product was purchased?
  - How much of that product was purchased?
  - When was the product purchased?
  - Who purchased the product?

  This data is necessary to calculate the cube so that the sales questions previously listed can be answered.

To gather the values for these categories, load rules are created with the DB2 OLAP Server(TM) Application Manager. The load rules are what IBM(R) DB2(R) OLAP Server uses to populate the cube. When populating the cube, dimension build load rules must always be run before data load rules. This is because the cube must know dimension details of all possible data that can be entered during the data load. If the cube does not know about all possible data (For example, a new store's data is entered with a data load rule before the store is added via a dimension build rule), the cube will not be built properly.

Both categories of data are extracted from the data mart. To extract the data, the following load rules were created:

- Dimension build rules:
  - MRKT-RUL: Extracts geographic information from the MARKET table to complete the MARKET outline
  - PROD-RUL: Extracts product information from the PRODUCT table to complete the PRODUCT outline
  - STOR-RUL: Extracts store information from the STORE table to complete the STORE outline
- Data load rules:
  - EXP-2004: Extracts expenses information from the FACTTABLE for the data load
  - SAL-2004: Extracts sales information from the FACTTABLE for the data load

More information on these load rules can be found in the setup application section.

The data loading process is automated by using the same CL program used to perform the ETL process. After the ETL process is completed on iSeries(TM) A, a program is called on iSeries B to populate the cube using the load rules listed previously. (See Figure 2 for references to iSeries A and iSeries B.)

Application design points

Dimension considerations

- IBM(R) DB2 OLAP Server(TM) maximizes performance and minimizes storage by dividing the standard dimensions of a cube into two types: dense dimensions and sparse dimensions. A sparse dimension is a dimension that is likely to have a low percentage of its available data positions filled. Likewise, a dense dimension is a dimension that is likely to have a high percentage of its available data positions filled. This division allows IBM DB2(R) OLAP Server to cope with data that is not smoothly distributed, without losing the advantages of matrix-style access to the data.

  A good example of a dense dimension is the TIME dimension, since sales will likely occur every day. An example of a sparse dimension is the MARKET dimension, since not every product is sold in every market.

  iCola's dense dimensions are:

  – TIME

  – MEASURES (ACCOUNTS)

  iCola's sparse dimensions are:

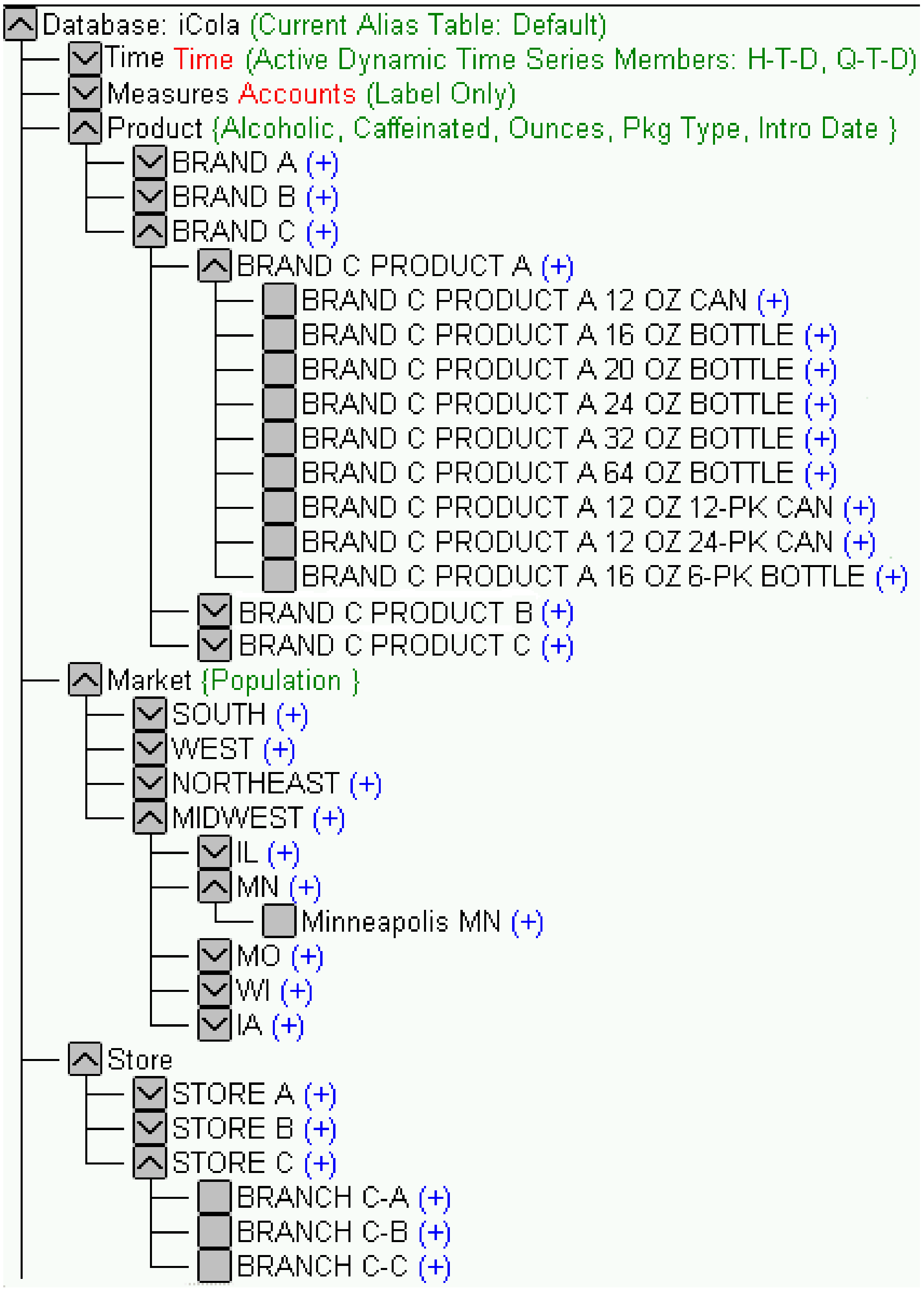
  – PRODUCT

  – MARKET

  – STORE

Outline considerations

- When creating the cube, an important point to consider is how much detail the cube must contain in order to answer the company's sales questions. For the TIME dimension, the smallest measure of time required to calculate the data had to be determined. In the iCola scenario, it was decided that a month would be the smallest amount of time that would satisfy the company's analysis needs. Consequently, the lowest level of the time dimension is month.
- Another decision that was made was what sort of numeric data was going to be in the cube. This is data such as sales figures, and is usually indicated by the ACCOUNTS dimension. Based on the queries that needed to be run, the company decided to use expenses and sales data.
- Figure 8 illustrates how the PRODUCT, MARKET, and STORE dimensions are expanded in the cube outline:



```
Database: iCola (Current Alias Table: Default)
  Time Time (Active Dynamic Time Series Members: H-T-D, Q-T-D)
  Measures Accounts (Label Only)
  Product {Alcoholic, Caffeinated, Ounces, Pkg Type, Intro Date }
    BRAND A (+)
    BRAND B (+)
    BRAND C (+)
      BRAND C PRODUCT A (+)
        BRAND C PRODUCT A 12 OZ CAN (+)
        BRAND C PRODUCT A 16 OZ BOTTLE (+)
        BRAND C PRODUCT A 20 OZ BOTTLE (+)
        BRAND C PRODUCT A 24 OZ BOTTLE (+)
        BRAND C PRODUCT A 32 OZ BOTTLE (+)
        BRAND C PRODUCT A 64 OZ BOTTLE (+)
        BRAND C PRODUCT A 12 OZ 12-PK CAN (+)
        BRAND C PRODUCT A 12 OZ 24-PK CAN (+)
        BRAND C PRODUCT A 16 OZ 6-PK BOTTLE (+)
      BRAND C PRODUCT B (+)
      BRAND C PRODUCT C (+)
  Market {Population }
    SOUTH (+)
    WEST (+)
    NORTHEAST (+)
    MIDWEST (+)
      IL (+)
      MN (+)
        Minneapolis MN (+)
      MO (+)
      WI (+)
      IA (+)
  Store
    STORE A (+)
    STORE B (+)
    STORE C (+)
      BRANCH C-A (+)
      BRANCH C-B (+)
      BRANCH C-C (+)
```

Figure 8: iCola's PRODUCT, MARKET, and STORE dimensions.

- Each product contains certain features such as size, package type, and an introduction date. The most effective way to portray this in the cube's outline is to make these attributes of the PRODUCT dimension. More information about creating attributes is described in the *IBM DB2 OLAP Server 8.1 Database* Administrator's Guide.

Performance considerations

- A performance point mentioned by the *IBM DB2 OLAP Server 8.1 Database Administrator's Guide* is to optimize **query** performance by ordering the dense dimensions before the sparse dimensions. Also, it mentions to order the most commonly queried sparse dimensions before the least commonly queried sparse dimensions. Following these guidelines for the iCola scenario, the dimensions would be ordered as follows:
  1. TIME (Dense)
  2. MEASURES (Dense)
  3. PRODUCT (Sparse)
  4. MARKET (Sparse)
  5. STORE (Sparse)

  However, another performance point mentioned is that to optimize **calculation** performance, order the dense dimensions before the sparse dimensions, but order the sparse dimension starting with the dimensions with the least amount of members to the greatest amount of members. Following these guidelines for the iCola scenario, the dimensions would be ordered as follows:
  1. TIME (Dense)
  2. MEASURES (Dense)
  3. MARKET (Sparse)
  4. STORE (Sparse)
  5. PRODUCT (Sparse)

  These two theories partially conflict with each other. The *IBM DB2 OLAP Server 8.1 Database Administrator's Guide* explains that the best outline ordering sequence is to prioritize whether fast queries or fast calculation is more important. For iCola, it was decided that query performance is more important than calculation performance, mainly due to the fact that calculations are performed once a day and queries are performed many times a day.
- Another performance consideration investigated was using multiple cubes. In the iCola scenario, one cube could be used to store several years worth of data. However, iCola is concerned with the amount of time required to calculate the cube and generate queries once the cube starts getting large. One solution would be to use a partitioned cube, which the iCola scenario may implement in the future. The current solution is to use a different cube for each year. Since information is stored in the cube by month, the company decided that having one cube for each year would be a practical solution to this issue. Currently, the iCola scenario has two cubes, one containing data from 2003 and one containing data from 2004.

Application setup

Install IBM(R) DB2 OLAP Server(TM)

For users familiar with the iSeries(TM) and iSeries operations, installing the IBM DB2 OLAP Server on the iSeries should be straightforward. Adequate information is provided in the installation guide that leads the user through the entire installation process. This guide was used extensively in the creation of the DB2 OLAP Server instance needed to house the OLAP cube within the iSeries that was used for the iCola scenario. The DB2(R) OLAP Server Installation Guide for iSeries can be found at: DB2 OLAP Server Installation Guide for iSeries. Chapter 2 ("Before you install") and Chapter 4 ("Installing Essbase/400 OLAP server components") are the key chapters in this document needed in order to install DB2 OLAP Server.

Install IBM DB2 OLAP Client

The DB2 OLAP Server Installation Guide provides the instructions on how to install the DB2 OLAP Client application on Windows(R). Chapter 6 ("Installing clients on Windows") helps the user through this process, and Chapter 7 ("Connecting to Essbase/400 OLAP server") assists the user in making the client useful by establishing the link between client and server applications. The installation guide can be found at: DB2 OLAP Server Installation Guide for iSeries.

Create the cube

To create a multidimensional cube, first verify the DB2 OLAP server is started. To give a profile the authority to run ESSBASE commands, enter:
- ESSBASE/GRTESSAUT USRPRF(*user_profile*)

Then use that profile to start DB2 OLAP by running this command:
- ESSBASE/STRESSSVR SERVER(*ALL) JOBD(OLAPBLDSVR/SCJOBD)

After the DB2 OLAP Server is started, the client application can be started by clicking Start > Programs > IBM DB2 OLAP Server 8.1 > Application Manager.

When working with IBM DB2 OLAP Server, some key terms to know are:
- Application: a project that contains databases, calculation scripts, load rules, and reports.
- Database: the *IBM DB2 OLAP Server 8.1 Database Administrator's Guide* refers to a cube as a database.

Details on how to create applications and databases can be found on page 151 in the *IBM DB2 OLAP Server 8.1 Database Administrator's Guide*

To add dimensions to the outline, refer to page 168 of the guide. The easiest way to create an outline is to copy one of the sample databases provided with IBM DB2 OLAP Server, such as Sample application's Basic database. iCola copied the Sample > Basic outline and made these changes:
- The Scenario dimension was removed, since this information is not required.
- The Store dimension was added, since the company wanted to keep track of store information.
- Since the market regions for iCola were different than Sample Basic's market regions, the regions were changed from East, Central, South, and West to NorthEast, Midwest, South, and West.
- The products under the product dimension were deleted, since iCola's products were different than in the Sample application.
- The attribute alcoholic was added to the list of existing attributes to denote whether a beverage was alcoholic. To add attributes to an outline, refer to page 249 of the guide.

The dimensions do not need to be expanded in great detail in the outline. The outline will be filled in once the dimension build load rules are created and run.**Create load rules - dynamic dimension building**

As mentioned previously, three dimension build load rules were created to ensure the outline stayed current as new markets, products, and stores are added. To create the load rules to dynamically build dimensions, follow these steps: (for additional help see Chapter 19 of the *IBM DB2 OLAP Server TM 8.1 Database Administrator's Guide*)
1. Create a new rules file by selecting the data load rules button on the DB2 OLAP Server client screen

   and clicking **New**.
2. To define the SQL statement used to build the rule file, select **File > Open SQL.**
3. Select the appropriate server, application, and database - then click OK.

4. Choose the appropriate datasource and enter in the SQL statement to extract the dimension build information from the data mart in the space provided. Click **OK/Retrieve** when finished. If user name and password information is required, type in the DB2 OLAP administrator user name and password for the DB2 OLAP Server.
5. The **Data Prep Editor** is then displayed. If the SQL statement is successful in retrieving data, this data is displayed in the editor.
6. The rule must be associated with an outline. To associate it with an outline, select **Options > Associate Outline** from the main menu. Select the appropriate server, application, and database; then click **OK**.
7. Complete the field properties for the outline by clicking on the **Define Properties**

   button in the **Data Prep Editor** screen.
   a. When creating a load rule for dynamically building a dimension, verify these properties for all columns under the **Global Properties** tab:
      - The **Data Field** box should be unchecked. This is unchecked because the rule is not for loading data.
      - The **Ignore field during dataload** box should be checked. This is checked because this rule is used for modifying a dimension, not for loading data.
      - The **Ignore field during dimension build** box should be unchecked. This is unchecked because this is a rule for modifying the dimension, and should not be ignored.
      - The **Drop leading/trailing whitespace** box should be checked. This ensures that if a field has extra whitespace, that it will not have an effect on the dimension build.

   b. Since the **Ignore field during dataload** box is unchecked, the user will not be able to select the **Data Load Properties** tab. However, the user can and should select the **Dimension Build Properties** tab to map all fields to the dimension outline.
      - For the **Field Type**, select Generation, and enter the appropriate generation number in the **Number** box.
      - For **Dimension**, choose the appropriate dimension from the **Dimension** drop down.

      If mapping an attribute to a field:
      - For the **Field Type**, select the attribute field name, listed under Attribute Dimensions in the **Field Type** drop down list.
      - Enter the lowest level generation number for that dimension in the **Number** box.
      - For **Dimension**, choose the dimension that the attribute defines.
8. Verify the **Dimension Build Settings** by clicking the

   button. Under the **Dimension Build Settings** tab, make sure that the **Build Method** is: **Use Generation References**.
9. Save the rules file.

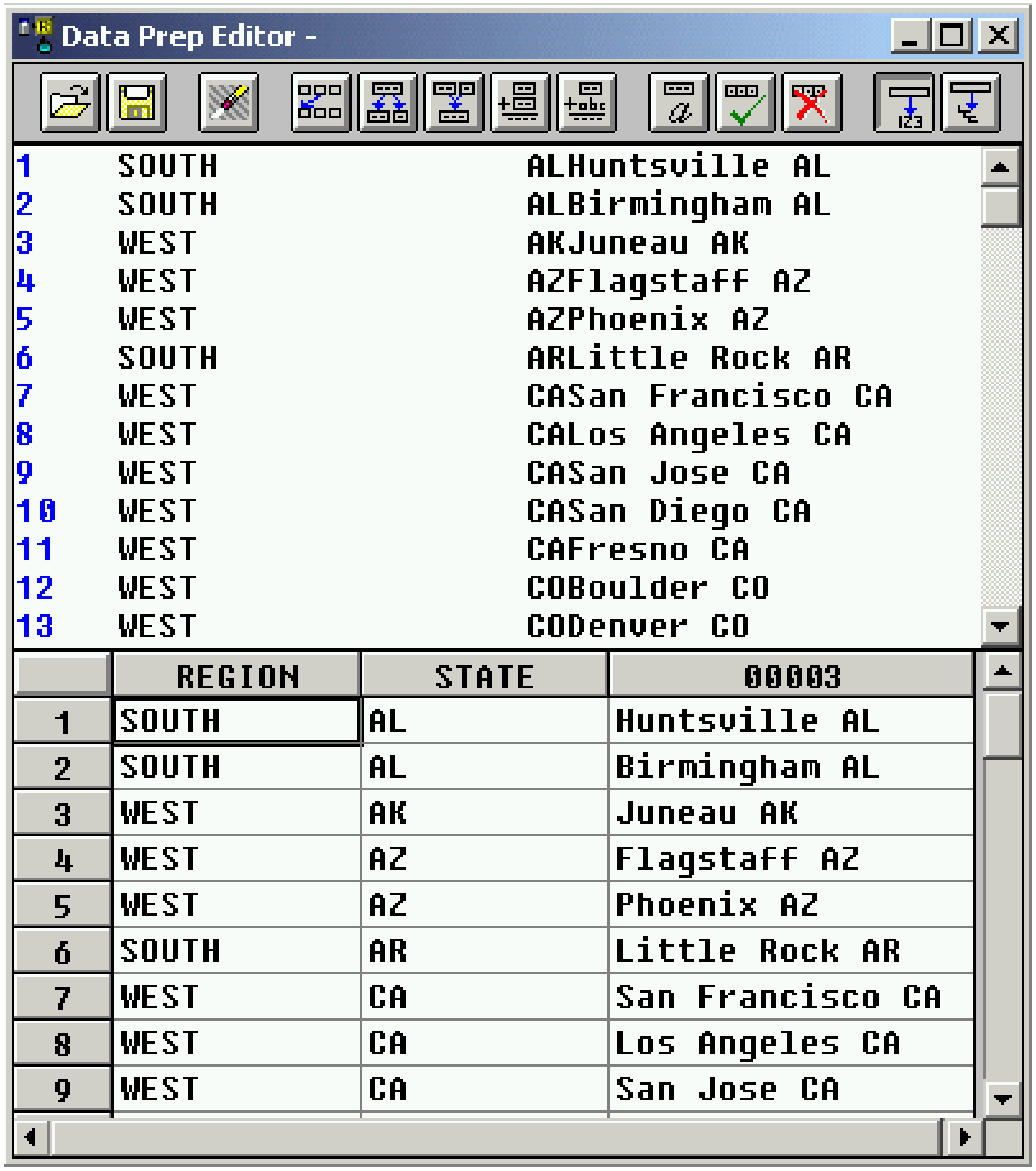
To help further understand how dimension build rules are created, the creation steps for one of the dimension build rules for iCola, MRKT-RUL, are shown here:

1. A new rules file was created by selecting the data load rules button on the DB2 OLAP server screen

   and clicking New.

2. Then the SQL statement used to build the rule file is defined by selecting **File > Open SQL.**
3. The server iSeries B (refer to Figure 2 for references to iSeries A and iSeries B), the application iCola, and database 2004 were selected.
4. For the datasource, iSeries A was chosen and this SQL statement was entered: SELECT
   ICOLADM.MARKET.REGION, ICOLADM.MARKET.STATE,
   CONCAT(RTRIM(ICOLADM.MARKET.CITY), CONCAT(' ', ICOLADM.MARKET.STATE))
   FROM ICOLADM.MARKET

   This SQL statement is used because it extracts information for each generation/level in the MARKET dimension.
5. After pressing **OK/Retrieve**, a prompt appears for a user name and password.
6. The **Data Prep Editor** is displayed, as shown in Figure 9. Notice that instead of extracting just the city, the city was concatenated with the state. This is because every dimension data element needs to be unique during a data load. For example, when extracting information for Rochester, MN, if only the city name Rochester is pulled, the DB2 OLAP Server would not know the difference between Rochester, MN and Rochester, NY when performing a data load. Concatenating the city and state field ensures that the city field is unique.

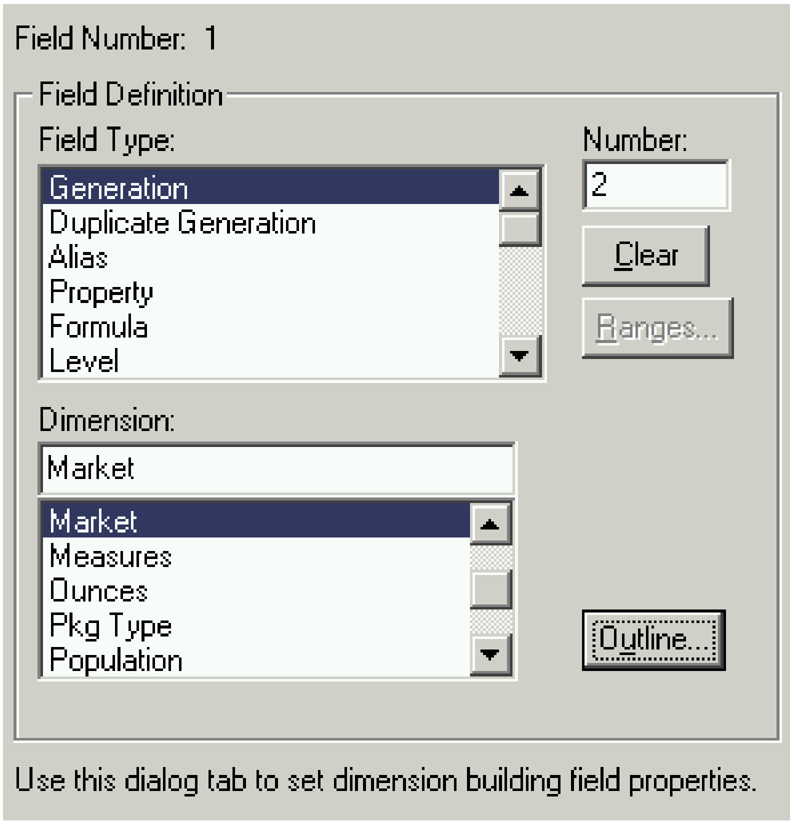

Data Prep Editor -

```
1       SOUTH                    ALHuntsville AL
2       SOUTH                    ALBirmingham AL
3       WEST                     AKJuneau AK
4       WEST                     AZFlagstaff AZ
5       WEST                     AZPhoenix AZ
6       SOUTH                    ARLittle Rock AR
7       WEST                     CASan Francisco CA
8       WEST                     CALos Angeles CA
9       WEST                     CASan Jose CA
10      WEST                     CASan Diego CA
11      WEST                     CAFresno CA
12      WEST                     COBoulder CO
13      WEST                     CODenver CO
```

	REGION	STATE	00003
1	SOUTH	AL	Huntsville AL
2	SOUTH	AL	Birmingham AL
3	WEST	AK	Juneau AK
4	WEST	AZ	Flagstaff AZ
5	WEST	AZ	Phoenix AZ
6	SOUTH	AR	Little Rock AR
7	WEST	CA	San Francisco CA
8	WEST	CA	Los Angeles CA
9	WEST	CA	San Jose CA

Figure 9: Dimension fields for the market dimension

7. Then the rule was associated with the 2004 outline.
8. Field Properties:
   - For the field properties, the appropriate fields under the **Global Properties** tab were checked.
   - Figures 10 - 12 provide the values that were entered for each of the generations under the **Dimension Build Properties** tab. Notice this corresponds with the generation information in the Application Overview section.


Field Number: 1

Field Definition

Field Type: Generation

Duplicate Generation

Alias

Property

Formula

Level

Number: 2

Dimension: Market

Market

Measures

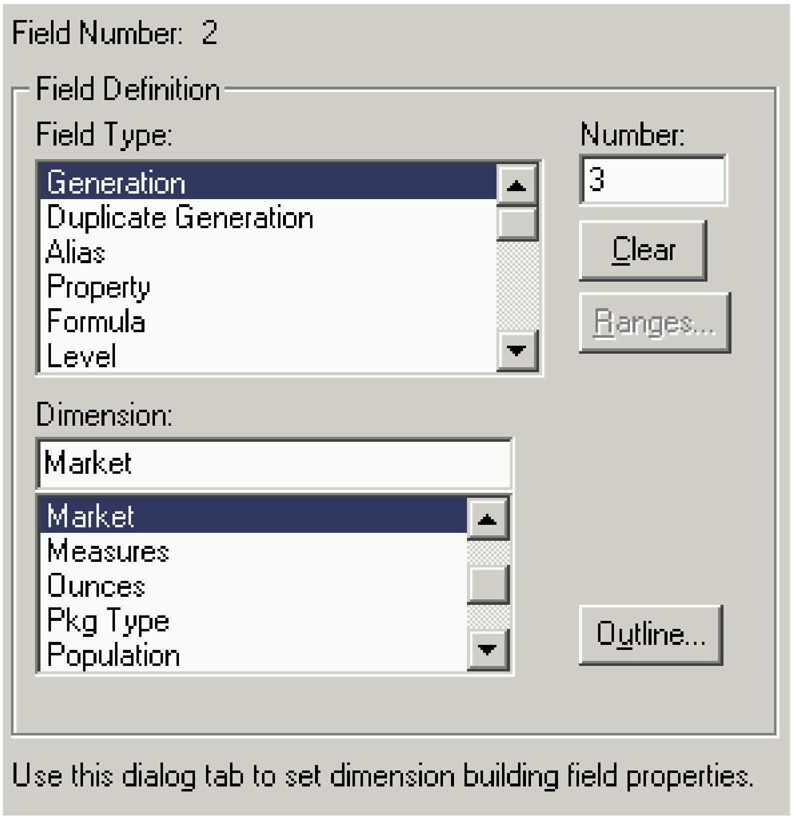
Ounces

Pkg Type

Population

Use this dialog tab to set dimension building field properties.

Figure 10: Definition for the region field


Field Number: 2

Field Definition

Field Type: Generation

Duplicate Generation

Alias

Property

Formula

Level

Number: 3

Dimension: Market

Market

Measures

Ounces

Pkg Type

Population

Use this dialog tab to set dimension building field properties.

Figure 11: Definition for the state field

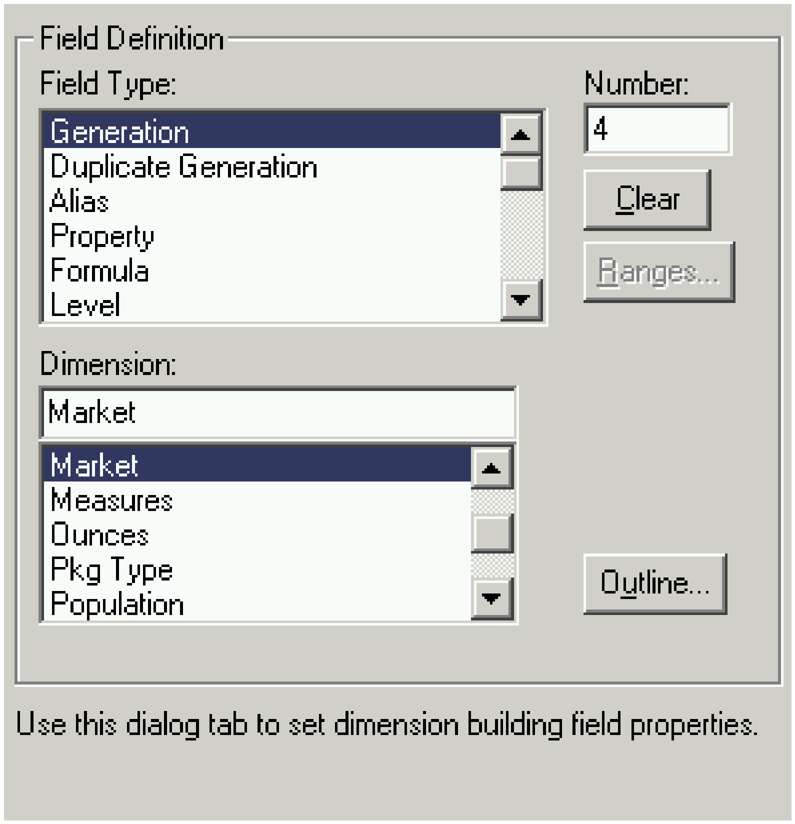


Figure 12: Definition for the city field

9. The setting **Use Generation References** was checked under the **Dimension Build Settings** tab.
10. The rules file was saved.

The SQL statements used to create iCola's dimension build rules are provided in Appendix C.

Create load rules - data load

Along with ensuring the dimensions are current, the data that the cube holds must be current. Load rules were created to load the current data. To create data load rules, follow these steps: (for additional info, see also chapter 20 of the *IBM DB2 OLAP Server 8.1 Database Administrator's Guide*)

1. Complete steps 1 - 6 of the dynamic dimension building section to build the load rule.

   The SQL statement used to load the current data contains the highest generation number (a.k.a. the lowest level number) for each dimension. For example, for iCola's data load rules, the SQL statement contains:
   - the month the sale was made
   - whether the figure was an expense or a sales figure
   - the product sold
   - the city the sale was made
   - the store the product was sold in

   The SQL statement also contains the total amount of the sale. This is the quantitative data amount which will be used in upcoming steps.

   See the Appendix C for the SQL statements used by iCola to create the data load rules.
2. Complete the field properties for the outline. In the **Data Prep Editor** screen, click on the **Define Properties**

button.

a. When creating a data load rule, verify these properties for all columns **except** for the field containing the quantitative data to load (usually a numeric quantity) under the **Global Properties** tab:
   - The **Data Field** box should be unchecked. This is unchecked because the particular field is not the field containing data.
   - The **Ignore field during dataload** box should be unchecked. This is unchecked because this rule is used for loading data, and should not be ignored.
   - The **Ignore field during dimension build** box should be checked. This is checked because this is a rule for loading data, not for building a dimension.
   - The **Drop leading/trailing whitespace** box should be checked. This ensures that if a field has extra whitespace, that it will not have an effect on the dimension build.

   For the field which does contain quantitative data, the **Data Field** box should be checked. All other fields remain the same as the previous list.
b. Since the Ignore field during dimension build box is unchecked, the user is not able to select the **Dimension Build Properties** tab. However, the user can select the **Data Load Properties** tab and name the fields of the data load. This step in not required, however.

3. Check the **Dimension Build Settings** by clicking the

   button. Under the **Dimension Build Settings** tab, verify that the **Build Method** is: **Use Generation References**.
4. Save the rules file.

To view the SQL statements used to create iCola’s dimension build rules, see Appendix C.

Clear data

Before loading data into the cube, the cube should be cleared to ensure proper calculation. To clear a cube, from the main menu select Database > Clear Data > All.

Run the dimension build and data load rules manually

After creating the dimension build and data load rules, the rules must be run to modify the database. To manually run the rules, follow these steps:

1. From the main menu, select **Database** > **Load Data**.
2. Select the server, application, and database to build the outline.
3. Select Type as SQL.
4. Under options, verify that **Modify Outline** is checked and **Load Data** is unchecked if modifying the outline. If loading data, verify that **Modify Outline** is unchecked and **Load Data** is checked.
5. For the SQL user and password, type the DB2 OLAP administrative user name and password.
6. Check the **Use Rules File** box and use the find button to select the appropriate rule.
7. Click OK. A prompt may appear asking if a new error file should be appended to or overwritten - select yes or no as appropriate.

Calculate Data

The data must be calculated after it is loaded into the cube. To calculate the data, select Database > Calculate. Verify that the appropriate server, application, and database, and calculation script are selected (A calculator script gives the cube information on how to calculate the data. The script can contain calculation commands, equations, and formulas. If no calculation scripts have been created, the default script can be used). Click OK to calculate the data.

Automate the dimension build and data load process

Since the processes of clearing the data, modifying the outline, loading the data, and calculating the data needed to be completed every day, a CL program called BLD_CUBE was created to automate these processes. The DB2 OLAP Server commands provided for the iSeries were used to create BLD_CUBE. BLD_CUBE is called after the ETL process completes.

Although the smallest measure in the TIME dimension is a month, the iCola scenario runs the BLD_CUBE program daily. It is run on a daily basis to allow the company to perform mid-monthly analysis on sales.

BLD_CUBE contains the following code.

```
PGM
/* Log onto DB2 OLAP with the DB2 OLAP administrator ID and password */
ESSBASE/LOGINESS SVRUSER(DB2OLAP) SVRPW(PASSW0RD)
/* Choose the 2004 database under the ICOLA application*/
ESSBASE/RUNESSCMDC COMMAND('SELECT "ICOLA" "2004"')
/* Clear the database */
ESSBASE/CLRESSDB APPNAME(ICOLA) DBNAME(2004)
/* Build the dimensions using the rules files */
ESSBASE/BLDESSDIM APPNAME(ICOLA) DBNAME(2004) RULEFILE('MRKT-ALL') ERRFILE(ERROR)
SQLUSER(DB2OLAP) +
SQLPW(PASSW0RD)
ESSBASE/BLDESSDIM APPNAME(ICOLA) DBNAME(2004) RULEFILE('PROD-ALL') ERRFILE(ERROR)
SQLUSER(DB2OLAP) +
SQLPW(PASSW0RD)
ESSBASE/BLDESSDIM APPNAME(ICOLA) DBNAME(2004) RULEFILE('STOR-ALL') ERRFILE(ERROR)
SQLUSER(DB2OLAP) +
SQLPW(PASSW0RD)
/* Load the data using the rules files */
ESSBASE/IMPESSSQL APPNAME(ICOLA) DBNAME(2004) RULEFILE('SAL-2004') +
ERRFILE('/ESSBASE/APP/ICOLA/2004/ERROR.TXT') SQLUSER(DB2OLAP) SQLPW(PASSW0RD)
ESSBASE/IMPESSSQL APPNAME(ICOLA) DBNAME(2004) RULEFILE('EXP-2004') +
ERRFILE('/ESSBASE/APP/ICOLA/2004/ERROR.TXT') SQLUSER(DB2OLAP) SQLPW(PASSW0RD)
/* Calculate the data */
ESSBASE/CLCESSDFT APPNAME(ICOLA) DBNAME(2004)



ENDPGM
```

Key findings

Calling the cube population process remotely

The program that loads the cube is part of the scheduled ETL process. Recall that the operational data and data mart are housed on iSeries$^{(TM)}$ A and the cube is housed on iSeries B. After the ETL process is completed on iSeries A, a program is called on iSeries B to populate the cube. Originally, there was a scheduled program on iSeries B to build the cube. However, it was difficult to determine when the ETL process had completed, and because the data mart would continually grow, the amount of time needed to complete the ETL process each night would vary. To solve the problem of determining when the ETL process was complete on iSeries A, the cube population was called remotely from iSeries A, so that the cube population would begin as soon as the ETL process was complete. This way, if something failed during the ETL process, the cube would not be repopulated.

To call the cube population process remotely from iSeries A, the following steps were completed:

1. iSeries A and iSeries B needed to be able to communicate with each other. In the iCola scenario, a host entry was added in iSeries A's host table to point to iSeries B.
2. A Distributed Data Management (DDM) file was created on iSeries A that referenced iSeries B. DDM controls remote file processing, and enables application programs running on an iSeries server to access data files stored on another server supporting DDM. Similarly, other systems that have DDM can access files in the database of the local iSeries server. DDM makes it easier to distribute file processing between two or more servers.

   This DDM file was created by running the following command on iSeries A:

   - CRTDDMF FILE(ICOLA/TO_SYSTM_B) RMTFILE(ICOLA/QCLSRC) RMTLOCNAME(*iSeries_B_Host_Name* *IP)

After creating the DDM file, the cube population program on iSeries B could be called via the submit remote command (SBMRMTCMD).

Calling the cube population process as a batch job

Originally, after the ETL process was complete on iSeries A, the program to build the cube on iSeries B was called directly:

- SBMRMTCMD CMD('CALL PGM(ICOLA/BLD_CUBE)') DDMFILE(ICOLA/TO_SYSTM_B)

However, calling the program directly meant that iSeries A would have to wait for iSeries B to finish populating the cube before its program could complete. This did not seem like an ideal solution, so instead the program call is submitted as a batch job:

- SBMRMTCMD CMD('SMBJOB CMD(CALL PGM(ICOLA/BLD_CUBE))') DDMFILE(ICOLA/TO_SYSTM_B)

Running the cube population program by submitting it as a batch job allows iSeries A to complete its program while iSeries B is populating the cube.

Allowing iSeries B to connect to the data mart on iSeries A

As mentioned previously, the data from the data mart on iSeries A is used to populate the cube on iSeries B. In order for iSeries B to be able to extract data from iSeries A, a Relational Database (RDB) entry was added on iSeries B to point to iSeries A. To add a Relational Database entry, use the Work with Relational Database Directory Entries command (WRKRDBDIRE).

Also, in order for iSeries B to access iSeries A's data mart, a user on iSeries A was needed with the same profile name and password as the user on iSeries B that was making a request for the data.(In the iCola scenario, this user was the DB2(R) OLAP administrator.) The matching profile on iSeries A must also have the authority to the database tables that are being accessed. If there is not a profile on iSeries A that matches the profile on the iSeries requesting the data (iSeries B), authority errors will occur.

Using SQL keywords as table names

The data load rules were initially created using these SQL phrases:

- SELECT ..., **TIME**.MONTH, ...)
  FROM ICOLADM.FACTTABLE, ICOLADM.CUSTOMER, ICOLADM.**TIME**, ICOLADM.PRODUCT,

ICOLADM.MARKET
WHERE... AND ICOLADM.FACTTABLE.TIMEID = **TIME**.TIMEID AND ...

When using this statement, errors were encountered pointing to the TIME table. In V5R3, TIME is a reserved word in SQL, which has special meaning. To run the query without having to rename the table, the table was aliased, as demonstrated below:

- SELECT ..., **MYTIME**.MONTH, ...)
  FROM ICOLADM.FACTTABLE, ICOLADM.CUSTOMER, ICOLADM.**TIME AS MYTIME**,
  ICOLADM.PRODUCT, ICOLADM.MARKET
  WHERE... AND ICOLADM.FACTTABLE.TIMEID = **MYTIME**.TIMEID AND ...

Utilizing lesser known SQL keywords

Some lesser known SQL keywords were discovered that were helpful when creating load rules:

- RTRIM: this keyword trims the blanks on the right side of the data field. For example, the string RTRIM('Rochester ') = 'Rochester'. The RTRIM keyword was useful for ensuring that extra white space was removed when concatenating strings.

  (There is a corresponding LTRIM function to trim left blanks)
- CONCAT: the CONCAT keyword binds two separate strings into one continuous string.
  - For example, CONCAT('String 1', 'String 2') = 'String 1String 2'.

  To add a space between the two strings, use CONCAT('String 1, CONCAT(' ', 'String 2')). This was useful for creating unique generation fields by binding two elements together.

Chapter 5. Analyzer

IBM(R) DB2 OLAP Server(TM) Analyzer 8.1 is a product designed to integrate with and leverage all the power of the DB2 OLAP Server application. It provides an intuitive Web-based interface for viewing the data contained within OLAP cubes generated by DB2 OLAP Server, and more importantly, allows for users to create analytical reports based on the data within the underlying OLAP cubes. These reports can then be utilized to make critical business decisions in order to improve upon past or current business practices and trends.

Analyzer allows users to transform complex multidimensional and relational data into insightful reports that can reflect product sales trends, profitability of the business, and other important business performance metrics. Analyses can then be shared and managed by sales, marketing, and other such departments within a business.

Various types of reports can be built using Analyzer: spreadsheets, line charts, pie charts, bar graphs, timelines, spatial graphs, etc. Creating a report is as simple as dragging and dropping graphical controls, but it takes some initial thought and design. Once a report has been built, the capability exists to further drill up and drill down within the data to create a new report from any of the newly generated views. Each report can then be saved within the Analyzer server, which then stores the report definition for further use. Any time the report definition is re-accessed by viewing an OLAP cube, the report data is refreshed based on the current data within the cube. Filtering options are also available to further restrict the analyses performed within a report.

Analyzer provides two main interfaces: A Java(TM)-based web client, and an HTML-based web client. When using the Java web client, users are allowed to create and save highly complex analytical reports through the use of an elaborate graphical interface. This part of the application is directed towards report designers and experienced users who need to be able to utilize more advanced functionality when working with reports. The HTML web client, on the other hand, is directed towards a different audience - those who are going to be using business analysis reports, not creating them. Business executives and project managers are typically the users of such an interface. Developers may also choose to extend the capabilities of IBM DB2 OLAP Server Analyzer 8.1 by using the Analyzer API toolkit, which allows for the creation of custom Web-based business analysis applications.

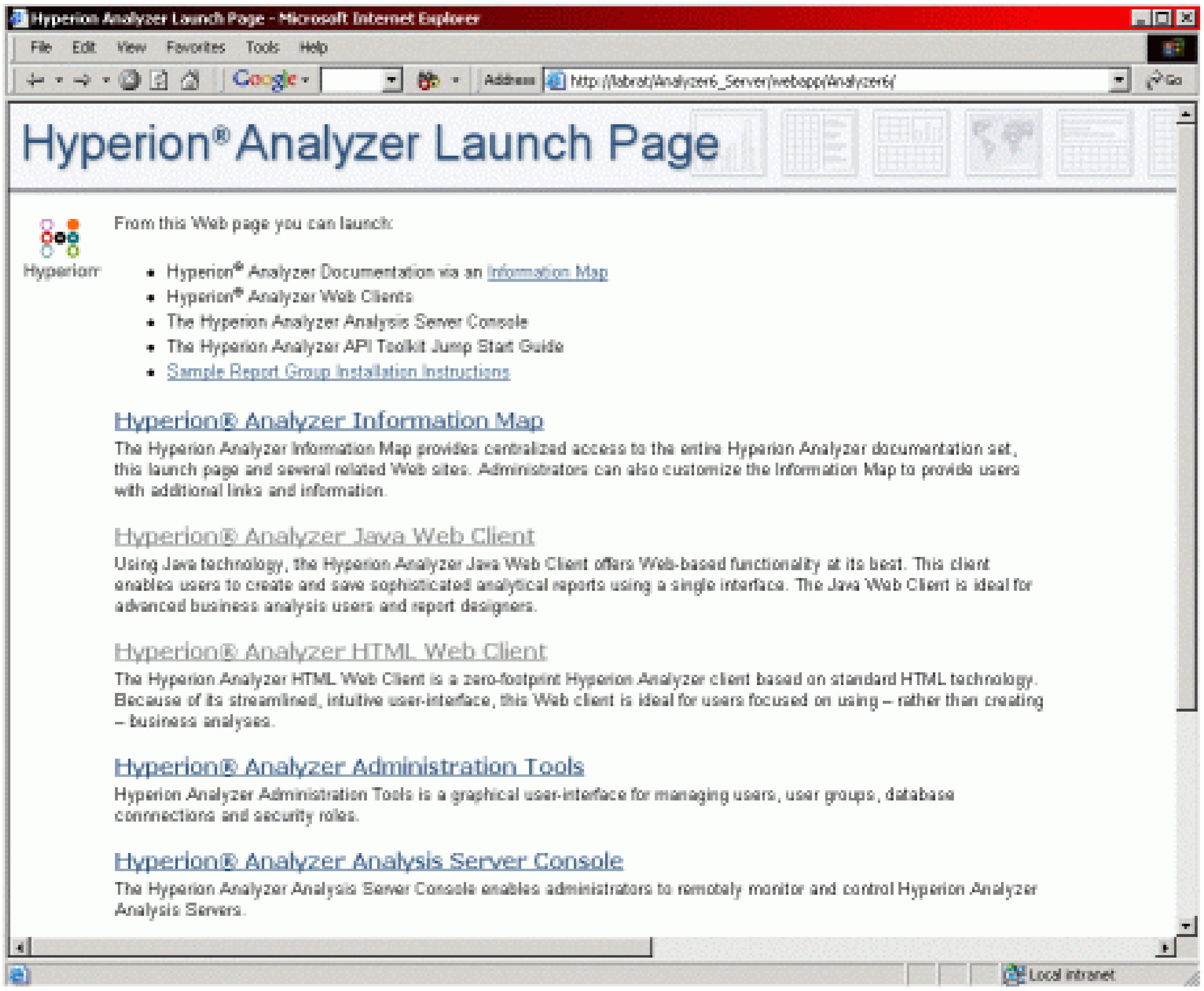
DB2 OLAP Server Analyzer is not a data mining tool. Rather, it is focused on business reporting based on the “queries” or questions posed by the report designer. The report designer usually creates a report that satisfies requirements needed by a manager or business executive. Data mining can be accomplished through use of other tools; Analyzer allows the user to create reports from OLAP data and determine business actions based on the trends seen within the reports.

IBM DB2 OLAP Server Analyzer 8.1 itself is based on Hyperion Analyzer 6.1, and by default utilizes the J2EE-compliant IBM WebSphere(R) Application Server. IBM DB2(R) Personal Edition is used as the default relational repository necessary for Analyzer-specific functionality. For the purposes of the iCola scenario, the two Analyzer Administration tools that are included with the Analyzer application were not used. Only one server, one user/group, and one database connection were needed for this application.

Application overview

In the iCola scenario, the primary mechanism for directly viewing and analyzing the data that is contained within a DB2(R) OLAP data cube is through the DB2 OLAP Analyzer product. The Analyzer server application is used in this scenario as the primary front-end to view the DB2 OLAP cube data and verifies that the analysis data changes after the cubes have been rebuilt. In the business world, Analyzer is the key component for generating reports and charts that guides the future business strategy for a company using DB2 OLAP Server(TM) as their business intelligence solution.

After installing Analyzer Server for Windows(R) and starting the application, the user is directed to the Analyzer launch page (Figure 13), which essentially contains links to all of the available tools provided by the application.


Hyperion Analyzer Launch Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://labrat/Analyzer6_Server/webapp/Analyzer6/
Go
Local intranet

Hyperion® Analyzer Launch Page

Hyperion

From this Web page you can launch:

- Hyperion® Analyzer Documentation via an Information Map
- Hyperion® Analyzer Web Clients
- The Hyperion Analyzer Analysis Server Console
- The Hyperion Analyzer API Toolkit Jump Start Guide
- Sample Report Group Installation Instructions

Hyperion® Analyzer Information Map

The Hyperion Analyzer Information Map provides centralized access to the entire Hyperion Analyzer documentation set, this launch page and several related Web sites. Administrators can also customize the Information Map to provide users with additional links and information.

Hyperion® Analyzer Java Web Client

Using Java technology, the Hyperion Analyzer Java Web Client offers Web-based functionality at its best. This client enables users to create and save sophisticated analytical reports using a single interface. The Java Web Client is ideal for advanced business analysis users and report designers.

Hyperion® Analyzer HTML Web Client

The Hyperion Analyzer HTML Web Client is a zero-footprint Hyperion Analyzer client based on standard HTML technology. Because of its streamlined, intuitive user-interface, this Web client is ideal for users focused on using – rather than creating – business analyses.

Hyperion® Analyzer Administration Tools

Hyperion Analyzer Administration Tools is a graphical user-interface for managing users, user groups, database connections and security roles.

Hyperion® Analyzer Analysis Server Console

The Hyperion Analyzer Analysis Server Console enables administrators to remotely monitor and control Hyperion Analyzer Analysis Servers.

Figure 13: Analyzer Launch Page

Both the Java(TM)-based Web client and the HTML-based Web client are used in the iCola scenario, although each one has its own purpose within the scenario. The Java-based Web client (Figure 14) was used during the design phase to create reports based upon a select set of business questions that were reviewed before being implemented.

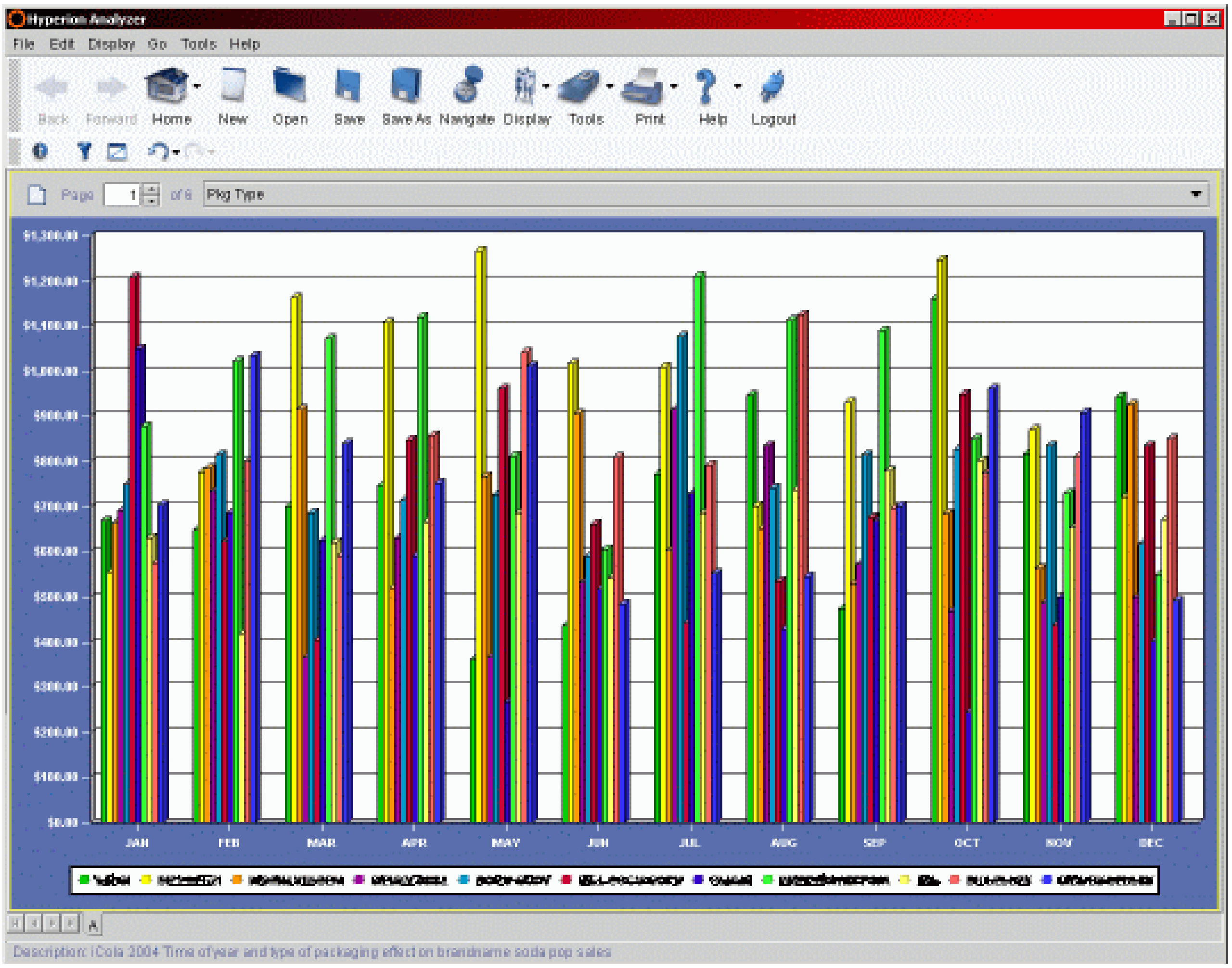

Hyperion Analyzer
File Edit Display Go Tools Help
Back Forward Home New Open Save Save As Navigate Display Tools Print Help Logout
Page 1 of 8 Pkg Type
JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
Description: iCola 2004 Time of year and type of packaging effect on brandname soda pop sales

Figure 14: Java Based Web Client

For the purposes of the iCola scenario, the chart style format was used for the reports, though various other capabilities exist for spreadsheet and combination (spreadsheet and chart) reports. The purpose of the inclusion of the Analyzer product within the iCola scenario is not so much to extensively test the Analyzer product itself, but rather to be an exerciser of the OLAP cubes and serve as a proof of concept for how the Analyzer product can be used.

The Analyzer application encompassed within the iCola scenario is one of the two items that is directly exercised by an automated workload application in the testing environment. The HTML-based Web client (Figure 15) provids an excellent interface for the workload (a workload is an automated script that simulates user action on a web interface) and makes logical sense, since constant changes to report definitions and design is not a realistic situation. Instead, the same views may be accessed at multiple points in time by multiple data reporters, who in turn would give these reports or a summary of them to their respective managers or business executives.

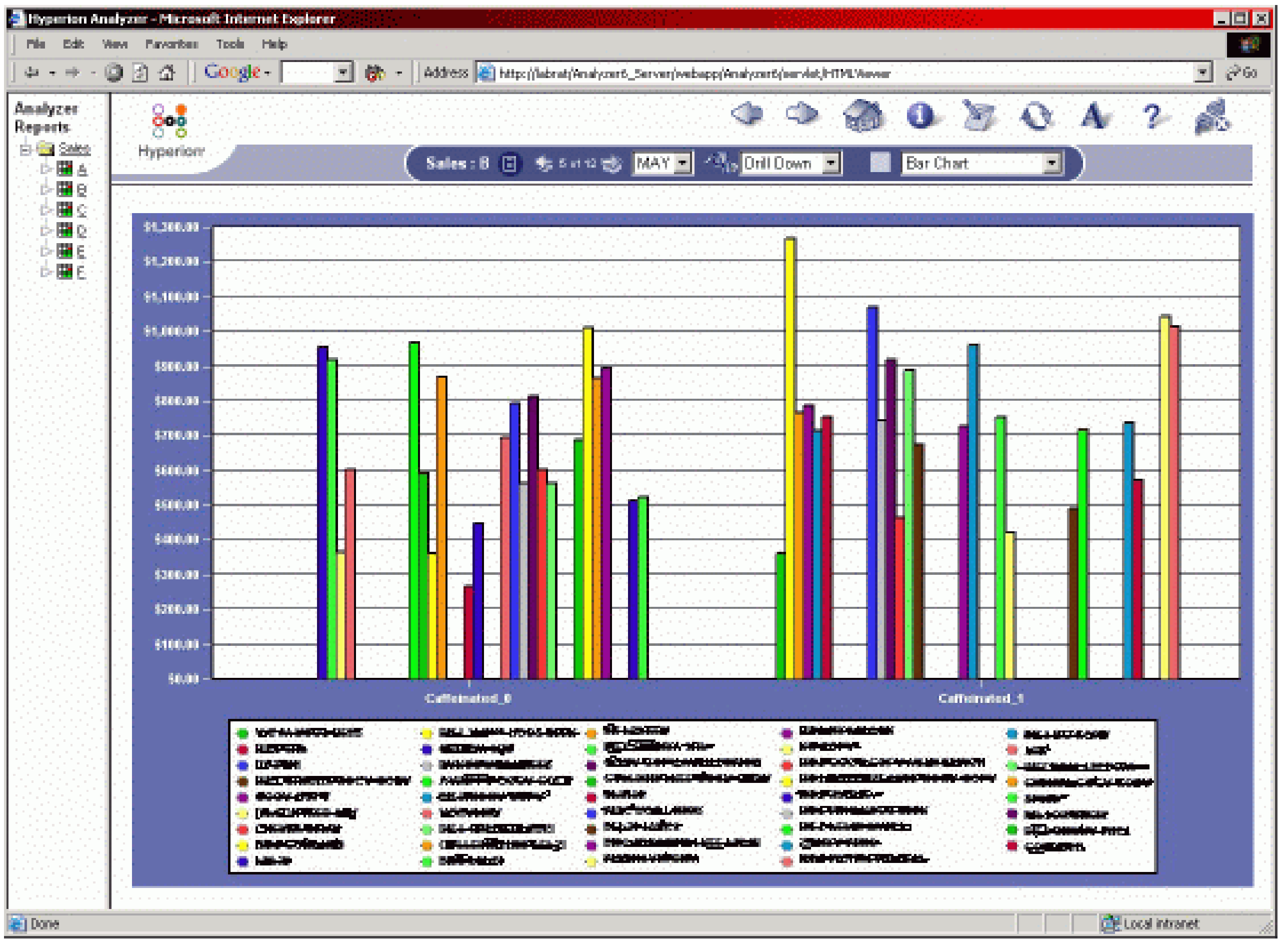


Figure 15: HTML Based Web Client

Application design points

There was not a lot of initial design for this portion of the scenario, since the product was used mostly "out of the box", without much additional configuration. With the IBM$^{(R)}$ DB2 OLAP Server$^{(TM)}$ Analyzer, there are a variety of customizations that can be done using the Java$^{(TM)}$ API that is provided, however this was not necessary because the basic needs of the scenario can be met by using the existing base product. After the Analyzer server application was installed, a few minor configuration steps needed to be completed for the application to correctly communicate with the OLAP Server. These steps are covered later on, when the application setup for Analyzer is discussed.

Since Analyzer itself did not require a specific design step within the scenario, the design process instead focuses on how the application is used as part of the daily iCola operations. Because the main purpose of Analyzer is to provide a reporting mechanism for the important business data, various business questions needed to be posed based on the available criteria. These Analyzer "queries" were formulated with a focus in mind on past and current business trends, so that better business practices and activities would be initiated in the present and future stages of the business.

For example, these queries can be used to examine in what months the amount of orders for certain brands of soda tend to lag, and then use that information to create a pricing incentive for buyers during those months. In an example of data sharing, the data can be used to start a coupon campaign during high-volume months with a partner, such as a potato chip manufacturer/distributor to help boost sales on both sides. The data gathered by the iCola company can also be gathered into report format and sold to the actual beverage manufacturers themselves. The manufacturers can then make educated business decisions on their products as well.

The following questions are currently executed over the OLAP cube data using the Analyzer product:
- How does the time of year and packaging type affect the sales of popular brand name soda
- What were the best and worst selling kinds of brand name soda in the previous year, by month, separated by caffeinated and caffeine-free
- What popular stores, by region, sold the most and least beer, by brand, in the current year
- How do cola sales compare head to head in the current year; Brand A versus Brand B
- How do beer sales compare head to head in the current year; Brand C versus Brand D
- What do the overall profit numbers look like for the current year, quarter, and month

Currently, the design also allows for the view to change dynamically, based on a selected year, so any of these queries can be directed over any previous years cube data for further trend analysis. Other valid questions also exist that can be used to analyze the data within the database, and more queries will be added in the future, as the scenario grows and expands.

Application setup

IBM(R) DB2(R) OLAP Analyzer 8.1 provides installation documentation for both Windows(R) and AIX(R), however after using it, the overall assessment was that it was hard to follow. Questions came up during the install process, some of which are not answerable by the documentation or on the Web.

Initially, an attempt was made to set up the application on an IBM pSeries(R) server running AIX(R). However, after contacting support, it was found that using DB2 OLAP Server(TM) Analyzer on AIX to access DB2 OLAP Server on iSeries(TM) V5R2 is not supported. Because of this, it was determined that using the Windows version of DB2 OLAP Server Analyzer running on an IBM xSeries(R) server would be a better choice.

The following is a high level summary of the install process needed to set up IBM DB2 OLAP Server Analyzer 8.1 for the Windows XP Professional platform on IBM xSeries:
1. Obtain the IBM DB2 OLAP Server Analyzer 8.1 for Windows CD-ROM.
2. Verify the xSeries server being used has a compatible version of the Windows operating system. Check the hardware recommendations for DB2 OLAP Server Analyzer to ensure it will perform adequately on the xSeries server.
3. Power on the xSeries (if not already powered on) and log into Windows using an administrative account.
4. Open up the Windows Control Panel and look for Java(TM) Plugin 1.3.0_02
5. If the Java Plugin 1.3.0_02 is not installed, install it now using the installer found on the Analyzer CD-ROM (use the .exe file in the Java_Plugin_1_3_0_02 directory). Complete the installation of the Java Plugin before moving on to the next step.
6. Verify installation of the Java Plugin 1.3.0_02 by re-opening the Windows Control Panel
7. Using the Analyzer CD-ROM, install DB2 Personal Edition Version 7.2 on the system. This file is found in the DB2 directory on the CD-ROM. During the install, choose all the default options; except fill in the check box for “Admin Client”, and use db2admin and “password” for all user ids and passwords. The user ids and passwords can be changed later if necessary. Complete the installation of DB2 before moving onto the next step.
8. Reboot the xSeries and log back into Windows using the same administrative account as previously used.

9. Stop all currently running DB2 services. To do this, open up the Windows Control Panel and select **Administrative Tools**, then select **Services**, then stop the three DB2 services that are currently running. Leave this window open.
10. Open up a DOS prompt, and execute the **usejdbc2.bat** file in your SQLLIB\java12 directory (usually C:\SQLLIB\java12). Once this file has finished executing, close the DOS prompt.
11. Return to the **Services** window that was previously open. Start all DB2 services that were running previously (Before step 9 was executed).
12. Create the ANALYZ60 database using the **Create DB** wizard within the DB2 Control Center application. Under the Start Menu, go to the IBM DB2 program folder and select **Control Center**. In the DB2 Control Center application, navigate under the system name, then under **Instance**, **DB2**, and **Databases**. Right click on the **Databases** folder and select Create -> Database Using Wizard. Type in ANALYZ60 as the database name and alias, then click the **Next** button. For the rest of the steps in the wizard, only take the default options (Click the **Next** button until the Summary page appears, then click **Finish**). Wait for the database to finish being created. Once this has completed, exit out of the DB2 Control Center application.
13. Install WebSphere(R) Application Server 3.5 from the Analyzer CD-ROM (located on the CD in the WebSphere35/NT directory). Take all the default options, except at the database screen. When choosing the database, select **InstantDB** as the database type for WebSphere to use for the install process to work properly. Continue to select the defaults until a summary screen is presented, then complete the installation process. Wait until the install is complete before moving onto the next step.
14. Reboot the xSeries and log back into Windows using the same administrative account as previously used.
15. Install WebSphere Application Server 3.5 Fixpak #5. This fixpak will change the application level of WebSphere from 3.5 to 3.5.5, which is necessary for Analyzer to correctly install. The fixpak file can be found on the root directory of the Analyzer CD-ROM (X:\was35_std_ptf_5.zip). Unzip this file on a local directory on the PC and then run the Setup file that gets extracted. Type in the parameters as the fixpak installer prompts for them. When prompted for the update to IBM HTTP Server (IHS), enter in the directory location for IHS on the local system to update this application.
16. Reboot the xSeries and log back into Windows using the same administrative account as previously used.
17. Install IBM DB2 OLAP Server Analyzer product from the DB2 OLAP Server Analyzer CD-ROM. Take all the defaults as they are presented (provided they match up with previously installed programs such as WebSphere and DB2 on the xSeries). Only deviation from the defaults is to select the RDBMS username to be **db2admin** instead of the default, **Analyzer**.
18. Finally, go to the URL (http://localhost/Analyzer6_Server/webapp/Analyzer6/index.html - replace **localhost** with the system host name for using this product remotely) to start using IBM DB2 OLAP Server Analyzer.

Key findings

The most noticeable finding during the entire process of working with IBM(R) DB2 OLAP Server(TM) Analyzer 8.1 was the difficulty of installing the application. The majority of this is due to the fact that the installation instructions provided with the Analyzer documentation were not precise. The instructions provided in the Setup Application section of this chapter were of great assistance when going through the install process. Once the product was installed, it proved to be a very useful and well structured application that fits the business intelligence needs of the iCola scenario.

Automated workload testing that simulated user accesses to the Analyzer application ran concurrently with the rest of the iCola scenario to exercise the backend OLAP cubes as well exercise the Analyzer Web application. When an automated, single user workload test was run against the application, it seemed to perform worse than when the application was accessed manually by a single human user. Other factors may have affected performance during a workload test run, however.

Windows$^{(R)}$ XP Professional was the operating system installed on the xSeries$^{(R)}$ box, although early testing was done on Windows 2000, which seemed to perform just fine. Another note on performance: The Java$^{(TM)}$ processes used by the Analyzer application were adjusted to run at realtime priority; this change can be made within the Windows Task Manager utility. This change was made to increase performance of the Analyzer application, and since it was the only real process necessary to run on the system, giving it highest priority would not affect other applications on the xSeries. If the xSeries being used is a multi-functional server and is not dedicated to just running Analyzer, changing the runtime priority may not be the best option for performance reasons.

One of the key discoveries that came from using Analyzer is the fact that the business intelligence data itself can be used as a profitable entity to sell as well. The data and reports gathered from using Analyzer could be used in an alternate way than just business intelligence data for the iCola company; target data specific to a particular beverage manufacturer could be sold as a product itself to that actual beverage manufacturer for their business use. In addition, certain stores themselves may not always keep track of every business trend for sales of certain products within their corporation. For companies like this, a "slice" of the cube could be sold as intellectual property that the corporation itself could use to boost sales against competitors depending on the results of their own analysis. Analyzer again would play a very important role in the business decisions of other companies using the data managed by the DB2 OLAP Server through the iCola beverage distribution company.

Overall, IBM DB2 OLAP Server Analyzer 8.1 is a valuable asset for the purposes of the iCola scenario, even though it was harder to install than expected. The product itself has a steep learning curve, but is easy to use once properly comprehended. Once users have adjusted to how the application works, it is very likely that the product will provide a positive impact on how business intelligence data is delivered to decision makers within a business situation. For anyone using DB2 OLAP Server, the Web interface of DB2 OLAP Server Analyzer is a must-have when it comes to successfully analyzing the business intelligence data being gathered through DB2$^{(R)}$ OLAP Server.

Chapter 6. Future ideas

As strategic financial and marking decision makers become familiar with the type of information that can be derived from business intelligence, their requests will grow. Future phases of the business intelligence project will involve creating more multidimensional cubes for data analysis. Furthermore, there are plans to take advantage of the data mining features included in DB2 OLAP Server$^{(TM)}$ to discover patterns which will help decision makers with strategic planning for their business.

In addition to taking advantage of advanced DB2 OLAP Server features and addressing the requests of strategic decision-makers for more data, the iCola scenario plans to take advantage of new iSeries$^{(TM)}$ database features, such as Materialized Query Tables.

As the company expands and data sources become more disparate, the need to expand the data mart into a data warehouse may become evident. If the need for a data warehouse arises, an analysis of available tools will be needed to assist with more complex extraction, transformation and loading processes. Furthermore, a single data warehouse may populate multiple subject oriented data marts.

Additional tools to provide more value from the data will also be reviewed. More advanced tools and techniques for data mining and decision making will be explored.

Chapter 7. Appendix A

SQL statements used to create the data mart, ICOLADM

The following SQL statement is used to create the ICOLADM library:

```
CREATE SCHEMA ICOLADM ;
```

The following SQL statements are used to create the ICOLADM fact table and dimension tables:

```
CREATE TABLE ICOLADM.CUSTOMER (
BRANCHID INTEGER NOT NULL ,
BRANCHNAME CHAR(25) CCSID 37 NOT NULL ,
STOREID INTEGER NOT NULL ,
STORENAME CHAR(25) CCSID 37 NOT NULL ,
CONSTRAINT ICOLADM.QSYS_CUSTOMER_00001 PRIMARY KEY( BRANCHID ) ) ;

CREATE TABLE ICOLADM.FACTTABLE (
ORDERID INTEGER NOT NULL ,
UPC INTEGER NOT NULL ,
MARKETID INTEGER NOT NULL ,
TIMEID INTEGER NOT NULL ,
QUANTITYSOLD FOR COLUMN QUANT00001 INTEGER NOT NULL ,
BRANCHID INTEGER NOT NULL ,
SELLINGPRICE FOR COLUMN SELLI00001 DECIMAL(9, 2) NOT NULL ,
TOTALPRICE DECIMAL(9, 2) NOT NULL ,
COSTTOBUY DECIMAL(9, 2) NOT NULL )
CONSTRAINT ICOLADM.QSYS_FACTTABLE_ORDERID_00001 PRIMARY KEY( ORDERID) ;

CREATE TABLE ICOLADM.MARKET (
MARKETID INTEGER GENERATED ALWAYS AS IDENTITY (
START WITH 1 INCREMENT BY 1
NO MINVALUE NO MAXVALUE
NO CYCLE NO ORDER
CACHE 20 ) ,
REGION CHAR(20) CCSID 37 NOT NULL ,
STATE CHAR(2) CCSID 37 NOT NULL ,
CITY CHAR(20) CCSID 37 NOT NULL ,
COUNTRY CHAR(2) CCSID 37 NOT NULL ,
POPULATION INTEGER NOT NULL ,
CONSTRAINT ICOLADM.QSYS_MARKET_00001 PRIMARY KEY( MARKETID ) ) ;

CREATE TABLE ICOLADM.PRODUCT (
UPC INTEGER NOT NULL ,
SIZE INTEGER NOT NULL ,
CAFFEINATED FOR COLUMN CAFFE00001 DECIMAL(1, 0) NOT NULL ,
ALCOHOLIC DECIMAL(1, 0) NOT NULL ,
INTRODATE DATE NOT NULL ,
DISCONTINUEDATE FOR COLUMN DISCO00001 DATE DEFAULT NULL ,
PACKAGETYPE FOR COLUMN PACKA00001 CHAR(13) CCSID 37 NOT NULL DEFAULT '' ,
NAME CHAR(80) CCSID 37 NOT NULL DEFAULT 'No default' ,
BRAND CHAR(40) CCSID 37 NOT NULL DEFAULT 'No default' ,
CONSTRAINT ICOLADM.Q_ICOLADM_PRODUCT_UPC_00002 PRIMARY KEY( UPC ) ) ;
```

```
CREATE TABLE ICOLADM.TIME (
TIMEID INTEGER NOT NULL ,
"DAY" SMALLINT NOT NULL ,
"MONTH" CHAR(3) CCSID 37 DEFAULT NULL ,
"YEAR" SMALLINT NOT NULL ,
QUARTER CHAR(4) CCSID 37 DEFAULT NULL ,
SEASON CHAR(6) CCSID 37 DEFAULT NULL ,
DAYOFWEEK CHAR(9) CCSID 37 DEFAULT NULL ,
CONSTRAINT ICOLADM.QSYS_TIME_00001 PRIMARY KEY( TIMEID ) ) ;
```

The following SQL statements are used to create the ICOLADM helper tables ETLPROCESS (used for logging) and REGIONS (used to help populate the MARKET table):

```
CREATE TABLE ICOLADM.ETLPROCESS (
START TIMESTAMP NOT NULL ,
FINISH TIMESTAMP DEFAULT NULL ) ;

CREATE TABLE ICOLADM.REGIONS (
STATE CHAR(2) CCSID 37 NOT NULL ,
REGION CHAR(15) CCSID 37 NOT NULL ,
COUNTRY CHAR(2) CCSID 37 NOT NULL DEFAULT 'US' ,
CONSTRAINT ICOLADM.QSYS_REGIOINS_00001 PRIMARY KEY( STATE ) ) ;
```

Chapter 8. Appendix B

SQL stored procedures for the ETL process

This is the code iCola uses to create each of the SQL stored procedures.

The ETL procedure calls all other SQL procedures:

```
CREATE PROCEDURE ICOLA.ETL ( )
LANGUAGE SQL
SPECIFIC ICOLA.ETL
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
CALL ICOLA . STARTETL ;
CALL ICOLA . CUSTOMER ;
CALL ICOLA . MARKET ;
CALL ICOLA . PRODUCT ;
CALL ICOLA . FACTTABLE ;
CALL ICOLA . ENDETL ;
END;
```

The STARTETL and ENDETL procedures log time information by recording the CURRENT TIMESTAMP:

```
CREATE PROCEDURE ICOLA.STARTETL ( )
LANGUAGE SQL
SPECIFIC ICOLA.STARTETL
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
INSERT INTO ICOLADM . ETLPROCESS ( START ) VALUES ( CURRENT TIMESTAMP ) ;
END ;
```

```
CREATE PROCEDURE ICOLA.ENDETL ( )
LANGUAGE SQL
SPECIFIC ICOLA.ENDETL
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
UPDATE ICOLADM . ETLPROCESS SET FINISH = ( CURRENT TIMESTAMP )
WHERE START = ( SELECT MAX ( START ) FROM ICOLADM . ETLPROCESS ) ;
END ;
```

The procedures CUSTOMER, MARKET, and PRODUCT insert information into their respective data mart tables. These versions of the procedures (which use the EXCEPT keyword) can be used only on OS/400 V5R3:

```
CREATE PROCEDURE ICOLA.CUSTOMER ( )
LANGUAGE SQL
SPECIFIC ICOLA.CUSTOMER
```

```
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
INSERT INTO ICOLADM . CUSTOMER ( BRANCHID , BRANCHNAME , STOREID , STORENAME )
( ( SELECT ICOLA . CUSTOMERBRANCH . BRANCHID , ICOLA . CUSTOMERBRANCH .
BRANCHNAME , ICOLA . CUSTOMERBRANCH . STOREID , ICOLA . STORE . BUSINESSNAME
FROM ICOLA . CUSTOMERBRANCH , ICOLA . STORE
WHERE ICOLA . CUSTOMERBRANCH . STOREID = ICOLA . STORE . STOREID )
EXCEPT
( SELECT BRANCHID , BRANCHNAME , STOREID , STORENAME FROM ICOLADM . CUSTOMER ) )
;
END ;

CREATE PROCEDURE ICOLA.MARKET ( )
LANGUAGE SQL
SPECIFIC ICOLA.MARKET
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
INSERT INTO ICOLADM . MARKET ( REGION , STATE , CITY , COUNTRY , POPULATION )
( ( SELECT ICOLADM . REGIONS . REGION , ICOLA . CUSTOMERBRANCH . STATE, ICOLA .
CUSTOMERBRANCH . CITY , ICOLADM . REGIONS . COUNTRY
FROM ICOLA . CUSTOMERBRANCH INNER JOIN ICOLADM . REGIONS ON ICOLA .
CUSTOMERBRANCH . STATE = ICOLADM . REGIONS . STATE )
EXCEPT
( SELECT REGION , STATE , CITY , COUNTRY , 0 FROM ICOLADM . MARKET ) ) ;
END ;

CREATE PROCEDURE ICOLA.PRODUCT ( )
LANGUAGE SQL
SPECIFIC ICOLA.PRODUCT
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
INSERT INTO ICOLADM . PRODUCT ( UPC , NAME , BRAND , SIZE , CAFFEINATED ,
ALCOHOLIC , PACKAGETYPE , INTRODATE )
( ( SELECT UPC , NAME , BRAND , SIZE , CAFFEINATED , ALCOHOLIC , PACKAGETYPE,
INTRODATE
FROM ICOLA . PRODUCT )
EXCEPT
( SELECT UPC , NAME , BRAND , SIZE , CAFFEINATED , ALCOHOLIC , PACKAGETYPE,
INTRODATE FROM ICOLADM . PRODUCT ) ) ;
UPDATE ICOLADM . PRODUCT SET DISCONTINUEDATE =
( SELECT DISCONTINUEDATE FROM ICOLA . PRODUCT
WHERE ICOLADM . PRODUCT . UPC = ICOLA . PRODUCT . UPC
AND ICOLA . PRODUCT . DISCONTINUEDATE IS NOT NULL ) ;
END ;
```

These versions of the CUSTOMER, MARKET, and PRODUCT procedures can be used on both OS/400 V5R2 and V5R3:

```
CREATE PROCEDURE ICOLA.CUSTOMER ( )
LANGUAGE SQL
```

```
SPECIFIC ICOLA.CUSTOMER
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
INSERT INTO ICOLADM . CUSTOMER (BRANCHID , BRANCHNAME , STOREID , STORENAME )
(SELECT ICOLA.CUSTOMERBRANCH.BRANCHID, ICOLA.CUSTOMERBRANCH.BRANCHNAME,
ICOLA .CUSTOMERBRANCH.STOREID, ICOLA.STORE. BUSINESSNAME
FROM ICOLA.CUSTOMERBRANCH, ICOLA.STORE
WHERE ICOLA.CUSTOMERBRANCH.STOREID = ICOLA.STORE.STOREID
AND ICOLA.CUSTOMERBRANCH.BRANCHID NOT IN
(SELECT ICOLADM.CUSTOMER.BRANCHID FROM ICOLADM.CUSTOMER)) ;
END ;

CREATE PROCEDURE ICOLA.MARKET ( )
LANGUAGE SQL
SPECIFIC ICOLA.MARKET
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
INSERT INTO ICOLADM . MARKET ( REGION , STATE , CITY , COUNTRY , POPULATION )
(SELECT DISTINCT ICOLADM.REGIONS.REGION, ICOLA.CUSTOMERBRANCH.STATE ,
ICOLA.CUSTOMERBRANCH.CITY, ICOLADM.REGIONS.COUNTRY, 0
FROM ICOLA.CUSTOMERBRANCH, ICOLADM.REGIONS, ICOLADM.MARKET
WHERE CONCAT(RTRIM(ICOLA.CUSTOMERBRANCH.CITY), CONCAT(' ',
ICOLA.CUSTOMERBRANCH.STATE)) NOT IN
(SELECT CONCAT(RTRIM(ICOLADM.MARKET.CITY), CONCAT(' ', ICOLADM.MARKET.STATE))
FROM ICOLADM.MARKET)
AND ICOLA.CUSTOMERBRANCH.STATE = ICOLADM.REGIONS.STATE ) ;
END ;

CREATE PROCEDURE ICOLA.PRODUCT ( )
LANGUAGE SQL
SPECIFIC ICOLA.PRODUCT
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN
INSERT INTO ICOLADM . PRODUCT ( UPC , NAME , BRAND , SIZE , CAFFEINATED ,
ALCOHOLIC , PACKAGETYPE , INTRODATE )
(SELECT UPC, NAME, BRAND, SIZE, CAFFEINATED, ALCOHOLIC, PACKAGETYPE, INTRODATE
FROM ICOLA.PRODUCT
WHERE UPC NOT IN (SELECT UPC FROM ICOLADM.PRODUCT) ) ;
UPDATE ICOLADM . PRODUCT SET DISCONTINUEDATE =
( SELECT DISCONTINUEDATE FROM ICOLA . PRODUCT
WHERE ICOLADM . PRODUCT . UPC = ICOLA . PRODUCT . UPC
AND ICOLA . PRODUCT . DISCONTINUEDATE IS NOT NULL ) ;
END ;
```

The FACTTABLE procedure inserts information into the data mart's FACTTABLE. The FACTTABLE procedure pulls order information from the operational data's ORDER table, with the help of other operational data tables to fill in additional details. This version of the FACTTABLE procedure (which uses the EXCEPT keyword) can be used only on OS/400 V5R3:

```
CREATE PROCEDURE ICOLA.FACTTABLE ( )
LANGUAGE SQL
SPECIFIC ICOLA.FACTTABLE
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN INSERT INTO ICOLADM . FACTTABLE ( ORDERID , UPC , MARKETID , TIMEID,
QUANTITYSOLD , SELLINGPRICE , TOTALPRICE , COSTTOBUY , BRANCHID )
( ( SELECT ICOLA . ORDER . ORDERID , ICOLA . ORDER . UPC , ICOLADM . MARKET . MARKETID
, ( ( YEAR ( ICOLA . ORDER . DATE ) * 10000 ) + ( MONTH ( ICOLA . ORDER . DATE ) * 100 ) + DAY (
ICOLA . ORDER . DATE ) ) , ICOLA . ORDER . QUANTITY , TRUNC ( ICOLA . ORDER . SUBTOTAL /
ICOLA . ORDER . QUANTITY, 2 ) , ICOLA . ORDER . SUBTOTAL , ICOLA . PRICE . OURCOST , ICOLA
. CUSTOMERCONTACT . BRANCHID
FROM ICOLA . ORDER , ICOLA . PRODUCT , ICOLA . PRICE , ICOLA . CUSTOMERCONTACT,
ICOLA . CUSTOMERBRANCH , ICOLADM . MARKET
WHERE ICOLA . ORDER . CUSTOMERID = ICOLA . CUSTOMERCONTACT . CUSTOMERID
AND ICOLA . CUSTOMERCONTACT . BRANCHID = ICOLA . CUSTOMERBRANCH . BRANCHID
AND ICOLA . CUSTOMERBRANCH . CITY = ICOLADM . MARKET . CITY
AND ICOLA . CUSTOMERBRANCH . STATE = ICOLADM . MARKET . STATE
AND ICOLA . ORDER . UPC = ICOLA . PRODUCT . UPC AND ICOLA . PRODUCT . PRICECODE =
ICOLA . PRICE . PRICECODE
AND ICOLA.ORDER.DATE < CURRENT DATE)
EXCEPT
( SELECT ORDERID , UPC , MARKETID , TIMEID , QUANTITYSOLD , SELLINGPRICE, TOTALPRICE ,
COSTTOBUY , BRANCHID FROM ICOLADM . FACTTABLE ) ) ;
END ;
```

This version of the FACTTABLE procedure can be used on both OS/400 V5R2 and V5R3:

```
CREATE PROCEDURE ICOLA.FACTTABLE ( )
LANGUAGE SQL
SPECIFIC ICOLA.FACTTABLE
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
BEGIN INSERT INTO ICOLADM . FACTTABLE ( ORDERID , UPC , MARKETID , TIMEID,
QUANTITYSOLD , SELLINGPRICE , TOTALPRICE , COSTTOBUY , BRANCHID )
(SELECT ICOLA.ORDER.ORDERID, ICOLA.ORDER.UPC , ICOLADM.MARKET.MARKETID,
((YEAR(ICOLA.ORDER.DATE) * 10000) + (MONTH(ICOLA.ORDER.DATE) * 100) +
DAY(ICOLA.ORDER.DATE)), ICOLA.ORDER.QUANTITY,
TRUNC(ICOLA.ORDER.SUBTOTAL/ICOLA.ORDER.QUANTITY, 2), ICOLA.ORDER.SUBTOTAL,
ICOLA.PRICE.OURCOST, ICOLA.CUSTOMERCONTACT.BRANCHID
FROM ICOLA.ORDER, ICOLA.PRODUCT, ICOLA.PRICE, ICOLA.CUSTOMERCONTACT,
ICOLA.CUSTOMERBRANCH, ICOLADM.MARKET
WHERE ICOLA.ORDER.CUSTOMERID = ICOLA.CUSTOMERCONTACT.CUSTOMERID
AND ICOLA.CUSTOMERCONTACT.BRANCHID = ICOLA.CUSTOMERBRANCH.BRANCHID
AND ICOLA.CUSTOMERBRANCH.CITY = ICOLADM.MARKET.CITY
AND ICOLA.CUSTOMERBRANCH.STATE = ICOLADM.MARKET.STATE
AND ICOLA.ORDER.UPC = ICOLA.PRODUCT.UPC AND ICOLA.PRODUCT.PRICECODE =
ICOLA.PRICE.PRICECODE
AND ICOLA.ORDER.DATE < CURRENT DATE
AND ICOLA.ORDER.ORDERID NOT IN (SELECT ORDERID FROM ICOLADM.FACTTABLE)
) ;
END ;
```

Chapter 9. Appendix C

SQL Statements used to create the rule files

MARKET dimension population: MRKT-RUL

```
SELECT ICOLADM.MARKET.REGION, ICOLADM.MARKET.STATE,
CONCAT(RTRIM(ICOLADM.MARKET.CITY), CONCAT(' ', ICOLADM.MARKET.STATE))
FROM ICOLADM.MARKET
```

PRODUCT dimension population: PROD-RUL

```
SELECT ICOLADM.PRODUCT.BRAND, ICOLADM.PRODUCT.NAME,
CONCAT(RTRIM(ICOLADM.PRODUCT.NAME), CONCAT(' ',
CONCAT(CAST(ICOLADM.PRODUCT.SIZE AS CHAR(2)), CONCAT(' OZ ',
RTRIM(ICOLADM.PRODUCT.PACKAGETYPE))))), ICOLADM.PRODUCT.SIZE,
ICOLADM.PRODUCT.PACKAGETYPE, ICOLADM.PRODUCT.CAFFEINATED,
ICOLADM.PRODUCT.ALCOHOLIC
FROM ICOLADM.PRODUCT
```

STORE dimension population: STOR-RUL

```
SELECT ICOLADM.CUSTOMER.STORENAME,
CONCAT(RTRIM(ICOLADM.CUSTOMER.STORENAME), CONCAT(' ',
RTRIM(ICOLADM.CUSTOMER.BRANCHNAME)))
FROM ICOLADM.CUSTOMER
```

Query used to load the expenses data: EXP-2003

```
SELECT CONCAT(RTRIM(ICOLADM.CUSTOMER.STORENAME), CONCAT(' ',
RTRIM(ICOLADM.CUSTOMER.BRANCHNAME))), MYTIME.MONTH,
CONCAT(RTRIM(ICOLADM.PRODUCT.NAME), CONCAT(' ',
CONCAT(CAST(ICOLADM.PRODUCT.SIZE AS CHAR(2)), CONCAT(' OZ ',
RTRIM(ICOLADM.PRODUCT.PACKAGETYPE))))), CONCAT(RTRIM(ICOLADM.MARKET.CITY),
CONCAT(' ', ICOLADM.MARKET.STATE)), 'COGS', SUM(ICOLADM.FACTTABLE.COSTTOBUY *
ICOLADM.FACTTABLE.QUANTITYSOLD)
FROM ICOLADM.FACTTABLE, ICOLADM.CUSTOMER, ICOLADM.TIME AS MYTIME,
ICOLADM.PRODUCT, ICOLADM.MARKET
WHERE ICOLADM.FACTTABLE.BRANCHID = ICOLADM.CUSTOMER.BRANCHID AND
ICOLADM.FACTTABLE.TIMEID = MYTIME.TIMEID AND ICOLADM.FACTTABLE.UPC =
ICOLADM.PRODUCT.UPC AND ICOLADM.FACTTABLE.MARKETID =
ICOLADM.MARKET.MARKETID AND MYTIME.YEAR = 2003
GROUP BY CONCAT(RTRIM(ICOLADM.CUSTOMER.STORENAME), CONCAT(' ',
RTRIM(ICOLADM.CUSTOMER.BRANCHNAME))), MYTIME.MONTH,
CONCAT(RTRIM(ICOLADM.PRODUCT.NAME), CONCAT(' ',
CONCAT(CAST(ICOLADM.PRODUCT.SIZE AS CHAR(2)), CONCAT(' OZ ',
RTRIM(ICOLADM.PRODUCT.PACKAGETYPE))))), CONCAT(RTRIM(ICOLADM.MARKET.CITY),
CONCAT(' ', ICOLADM.MARKET.STATE))
```

Query used to load the sales data: SAL-2003

```
SELECT CONCAT(RTRIM(ICOLADM.CUSTOMER.STORENAME), CONCAT(' ',
RTRIM(ICOLADM.CUSTOMER.BRANCHNAME))), MYTIME.MONTH,
CONCAT(RTRIM(ICOLADM.PRODUCT.NAME), CONCAT(' ',
CONCAT(CAST(ICOLADM.PRODUCT.SIZE AS CHAR(2)), CONCAT(' OZ ',
RTRIM(ICOLADM.PRODUCT.PACKAGETYPE))))), CONCAT(RTRIM(ICOLADM.MARKET.CITY),
CONCAT(' ', ICOLADM.MARKET.STATE)), 'SALES', SUM(ICOLADM.FACTTABLE.TOTALPRICE)
FROM ICOLADM.FACTTABLE, ICOLADM.CUSTOMER, ICOLADM.TIME AS MYTIME,
ICOLADM.PRODUCT, ICOLADM.MARKET
WHERE ICOLADM.FACTTABLE.BRANCHID = ICOLADM.CUSTOMER.BRANCHID AND
ICOLADM.FACTTABLE.TIMEID = MYTIME.TIMEID AND ICOLADM.FACTTABLE.UPC =
ICOLADM.PRODUCT.UPC AND ICOLADM.FACTTABLE.MARKETID =
ICOLADM.MARKET.MARKETID AND MYTIME.YEAR = 2003
GROUP BY CONCAT(RTRIM(ICOLADM.CUSTOMER.STORENAME), CONCAT(' ',
RTRIM(ICOLADM.CUSTOMER.BRANCHNAME))), MYTIME.MONTH,
CONCAT(RTRIM(ICOLADM.PRODUCT.NAME), CONCAT(' ',
CONCAT(CAST(ICOLADM.PRODUCT.SIZE AS CHAR(2)), CONCAT(' OZ ',
RTRIM(ICOLADM.PRODUCT.PACKAGETYPE))))), CONCAT(RTRIM(ICOLADM.MARKET.CITY),
CONCAT(' ', ICOLADM.MARKET.STATE))
```

The only change that was made between the EXP-2003 and the EXP-2004 query (as well as between the SAL-2003 and SAL-2004 query) was to change the year in the last where clause:

- AND MYTIME.YEAR = 2004

Chapter 10. Disclaimer

Information is provided “AS IS” without warranty of any kind. Mention or reference to non-IBM products is for informational purposes only and does not constitute an endorsement of such products by IBM. Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Chapter 11. References

- Designing the Star Schema

  (http://www.ciobriefings.com/whitepapers/starschema.asp)
- IBM(R) DB2 OLAP Server(TM) 8.1 Database Administrators Guide
- IBM DB2 Universal Database Business Intelligence Tutorial

  (http://www-306.ibm.com/software/data/db2/db2olap/docs/V71docs/db2tu/frame3.htm#db2tussw)
- Hyperion(R) Analyzer Release 6.1 Installation Guide

Printed in USA