

# INFO SIZING

A.J. Mahajan  
Reference Architectures & CRS  
Sun Microsystems  
12 Network Circle  
Menlo Park, CA 94025

Paul Krneta  
Chief Technologist Sybase IQ  
Sybase Inc.  
5000 Hacienda Dr.  
Dublin, CA 94568

June 2, 2004

As per your request I verified the performance of the **Sun-Sybase DW Reference Architecture** and the record-breaking size of the fact table populated with **1 Trillion rows** representing **155 TB of raw input data**.

I verified the benchmark environment and the execution of the tests. I collected and analyzed the results of the measurements.

The attached report contains the details of my findings and is an attestation of the results obtained.

Respectfully Yours,



François Raab  
President

## Performance Benchmark Report

# One Trillion Rows

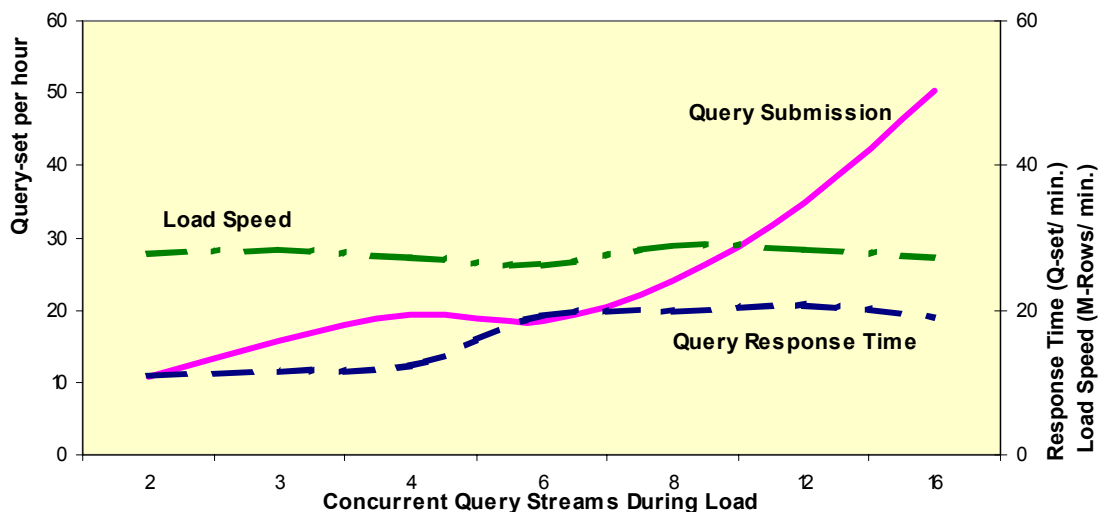
## Sun-Sybase DW Reference Architecture

### Highlights

In March 2004, at Sun Microsystem's request, I verified the performance of the **Sun-Sybase DW Reference Architecture** by defining and auditing the execution of a benchmark on a test platform assembled in Menlo Park, Ca.

The **Enterprise Data Warehouse Reference Architecture**, based on two Sun Fire servers and Sybase IQ, demonstrated several significant achievements:

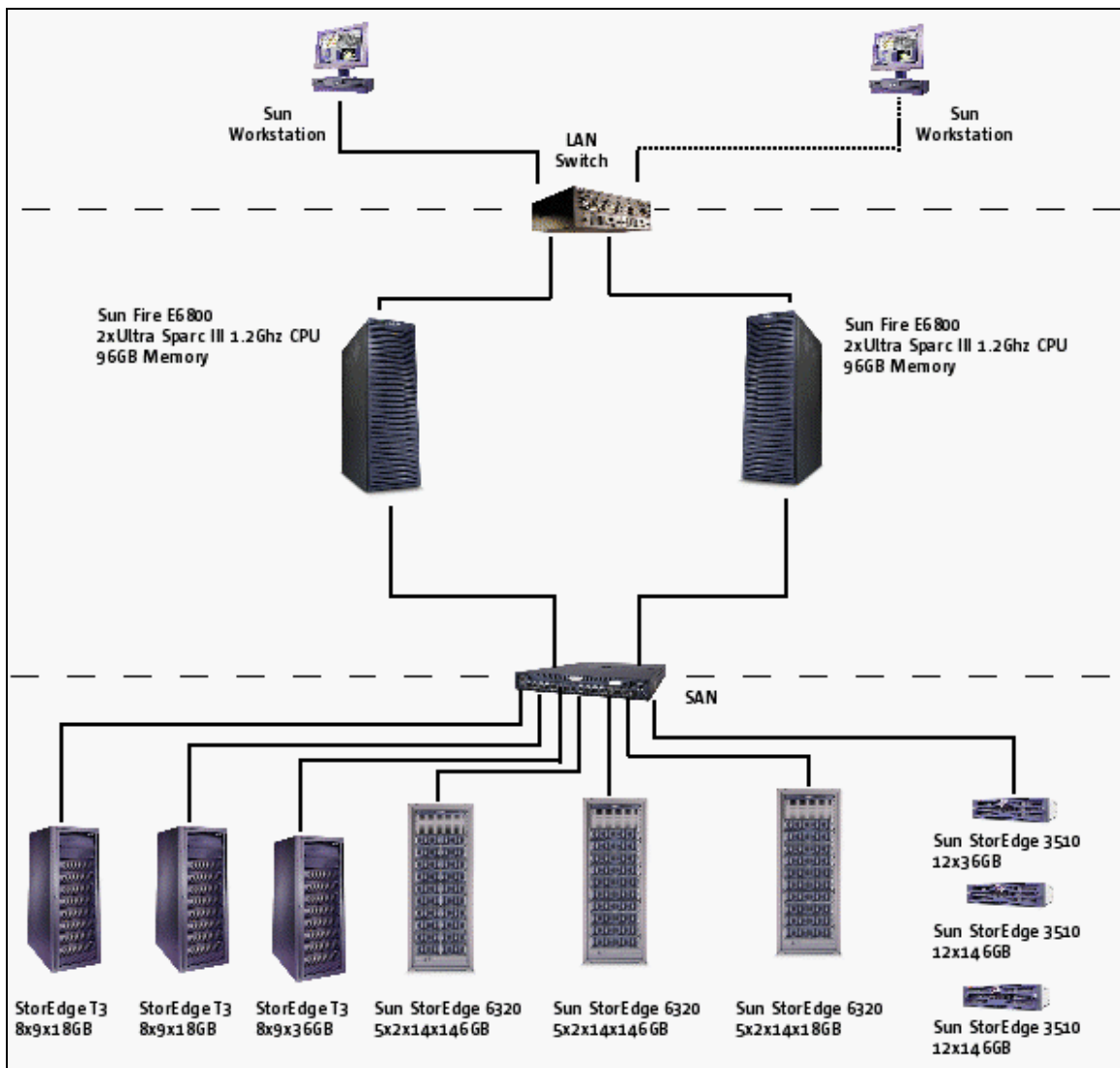
- Loaded with **one trillion rows** in the fact table, this is the largest database independently verified to date; enough to track every aluminum can sold in America over the last 20 years.
- A **deep data storage and compression ratio** allowing 155 terabytes of raw input data to be stored in a realistic data warehouse star schema using less than 55 terabytes of actual storage.
- The ability to simultaneously **maintain performance for query response times and data loading speed** while increasing the query submission rate up to 5 folds (see graph below.)



## Detailed Report

The Data Warehouse platform included two independent Sun server nodes networked in a shared-disk architecture, interconnected to an array of Sun storage sub-systems using fiber channels and holding a database managed by Sybase IQ version 12.5.0.

The following diagram details the layout of Sun's Enterprise Data Warehouse Reference Architecture:



In the process of scaling the database to one trillion rows different generations of Sun StorEdge products were used to demonstrate Sybase IQ's ability to run on older and newer storage systems.

## Platform Configuration Overview

<b>Database: Sybase IQ</b>	<b>Node A: Sun Fire F6800</b>
<ul style="list-style-type: none"> <li>○ 155 TB of Raw Data</li> <li>○ 64 to 120 Fact Tables</li> <li>○ 4 Dimension Tables</li> <li>○ 600 Billion to 1 Trillion Rows</li> </ul>	<ul style="list-style-type: none"> <li>○ 24 UltraSparc-III at 1.2Ghz</li> <li>○ 96 GB of Memory</li> <li>○ 3 Fiber Channels</li> </ul>
<b>Storage: Sun StorEdge</b>	<b>Node B: Sun Fire F6800</b>
<ul style="list-style-type: none"> <li>○ Sun StorEdge 6320 (20.7 TB)</li> <li>○ Sun StorEdge 6320 (20.7 TB)</li> <li>○ Sun StorEdge 6320 (2.5 TB)</li> <li>○ Sun StorEdge T3 (1.2 TB)</li> <li>○ Sun StorEdge T3 (2.6 TB)</li> <li>○ Sun StorEdge T3 (1.2 TB)</li> <li>○ Sun StorEdge 3510 (1.72 TB)</li> <li>○ Sun StorEdge 3510 (1.72 TB)</li> </ul>	<ul style="list-style-type: none"> <li>○ 24 UltraSparc-III at 1.2Ghz</li> <li>○ 48 GB of Memory</li> <li>○ 3 Fiber Channels</li> </ul>

## Sybase IQ Server Configuration

- *One IQ Writer residing on Server node A. The writer was bound to a 16 of the 24 processors and configured with 4 GB of Main IQ Cache and 4 GB of Temp IQ Cache.*
- *One IQ query server residing Server node A. This server was bound to 7 of the remaining 8 processors (leaving one for operation functions) and configured with 28 GB of Main Cache and 48 GB of Temp IQ Cache.*
- *One IQ query server residing Server node B. This server was bound to 7 of the 24 processors (leaving the remainder mostly idle) and configured with 20 GB of Main Cache and 20 GB of Temp IQ Cache.*

## Growing to One Trillion Rows

**Creating the database** - The Sybase IQ database was built around a generic star schema centered on a number of fact tables - each holding a month worth of fact data (25 columns per fact row.) Four dimension tables were created to drill into the fact data.

The four fact tables were populated to provide for a wide range of table cardinalities (from 5,000 rows to 500 million rows). Following are details on the population of the dimension tables:

Dimension Tables	Row Count	Columns	Row Size
Customer	500,000,000	11	246 bytes
Product	1,000,000	8	144 bytes
Channel	5,000	6	106 bytes
Location	30,000	7	108 bytes

**Initial Population** - Initially, a set of 60 fact tables was created. Each table was then populated with one month of fact data between 1999 and 2003. Data was generated using a set of 100 million seed records and replicating it multiple times while changing one or more fields to maintain uniqueness of rows. Using Sybase IQ's indexing technology, the fact table was fully inverted. An additional high-group index was defined on the primary key and on the four foreign keys to the dimension tables.

**First Validation** - Views were then defined to form a UNION ALL of the 12 fact tables for each year. An additional view was created to include all 5 years of fact data. The content of the database was validated and a series of performance tests (see details below) were executed.

**Making room for growth** - After the first validation, additional fact tables were created and similarly populated with additional years of fact data. The configuration of the Sun storage subsystem was augmented periodically to make space for the additional fact tables.

**Crossing the Trillion-row mark** - A total of 44 more fact tables were populated. A global view was then defined to form a UNION ALL of all 104 fact tables, covering the full 8.5 years of fact data. Following are details of the one trillion rows of fact data loaded into the fact tables:

Fact Table	Row Count	Columns	Raw Record size	Raw Data Size
ALL_FACTS	1,000 Billion	25	170 Bytes	154.6 Terabytes

**Final verification** – The trillion-row population was then validated and multi-table queries were then executed against the global view to verify that the database was still fully operational and that response times had not significantly degraded (see details below.)

## Performance Test Details

The following performance measurement tests were executed:

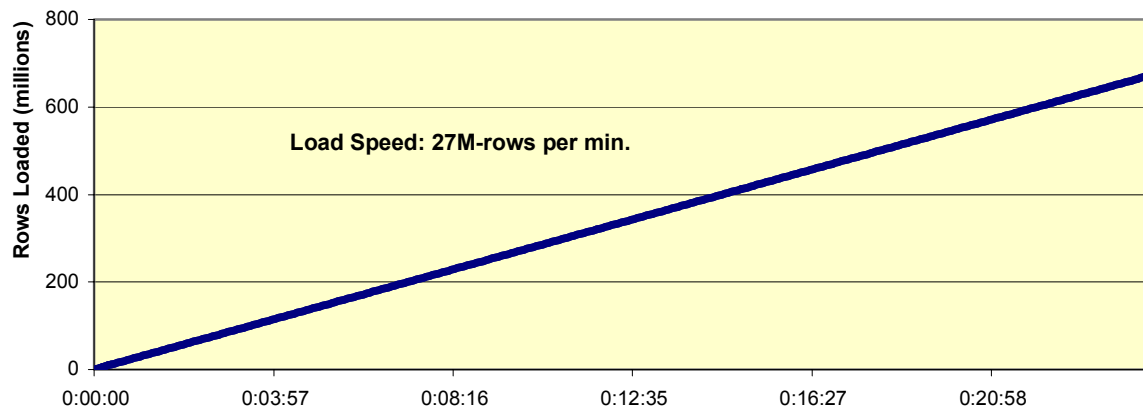
- *Multi-table joins against the 600 billion rows database.*
- *Load of additional fact data into the 600 billion rows database.*
- *Concurrent Load and queries against the 600 billion rows database.*
- *Multi-table joins against the 1 trillion rows database.*

**Multi-table joins** – A set of multi-table joins (see below) were defined and executed against the 600 billion rows database. The queries were executed one by one to obtain a baseline of response times. The measured response times were between 5 and 500 seconds.

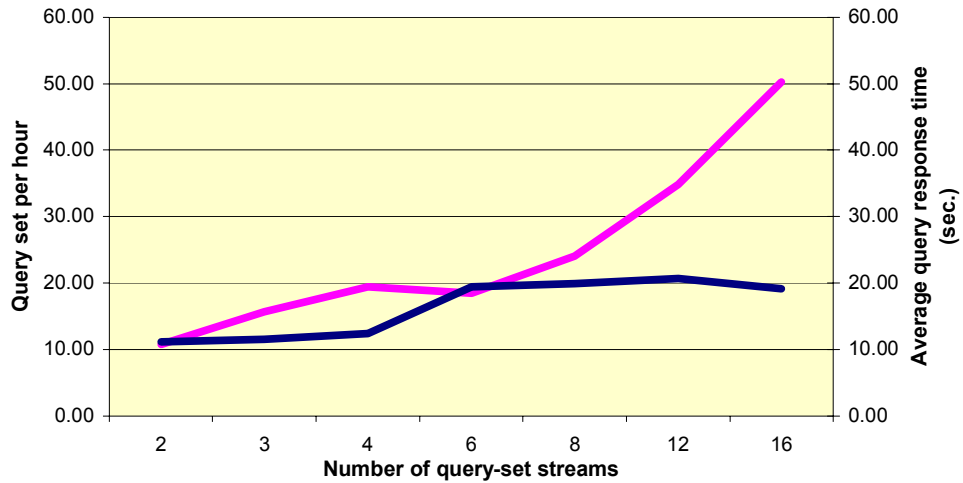
**Load of additional fact data** – An additional fact table was created on the 600 billion rows database. The new fact table was then populated with new data and the rate of this load operation was measured to establish a baseline. The new table was then dropped and the operation repeated with the creation and concurrent population of two new tables. Once more, the rate of this dual load operation was measured to establish a baseline. With two concurrent load streams the cumulative rate was measured at 30 million rows per minute.

**Concurrent load and queries** – While executing the dual load operation, an increasing number of query streams were execute. Up to eight query streams were ran concurrently on the same server node as the two load streams. Then up to eight more query streams were ran on a second server node sharing the same database.

The impact of the sixteen concurrent streams of join queries on the two concurrent load streams was minimal, as illustrated in the graph below tracking the load of 670 millions rows in under 25 minutes.



The impact of the sixteen concurrent streams of 10 join queries on each other's response times was also minimal (increased by less than 10 seconds) when compared with the 5-fold increase in query throughput, as illustrated in the graph below.



**Joins against the 1 trillion rows database** – After growing the fact table in the database to over 1 trillion rows, selected queries from the query set were executed. The time parameters in the “WHERE” clauses of these queries were adjusted to include the entire range of dates covered by the fact table.

A study of the execution plan chosen by Sybase IQ's query optimizer confirmed that all fact tables in the large “UNION ALL” view of one trillion rows were accessed by the queries. However, despite an increase in database size of over 65% to a record breaking one trillion rows, the query response time was no more impacted than by multiple concurrent query streams in the previous tests.

## Multi-Table Query Set

The query set used for most of the performance test was made of the following 10 queries. Parameter values in the “WHERE” clauses were substituted with different values in each query stream.

### Query 1 – Weak credit stores

This query computes the number of stores in a given location that serves customer with a weak credit score.

```
SELECT STORE_NAME,
       COUNT(*)
FROM ALL_FACTS,
     CUSTOMER,
     LOCATION
WHERE ALL_FACTS.LOCATION_ID = LOCATION.LOCATION_ID
      AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
      AND LOCATION.STORE_NUMBER = 29743
      AND CUSTOMER.CUSTOMER_SCORE < 5
      AND FACT_DATE BETWEEN '2003-01-25' AND '2003-02-03'
GROUP BY STORE_NAME
```

### Query 2 – Local customer activity

This query retrieves all the orders made by a selected customer at a given location over a specific period of time.

```
SELECT CUSTOMER_FNAME,
       STORE_NAME,
       ALL_FACTS.CUSTOMER_ID,
       PAYMENT_METHOD,
       FACT_DATE,
       ORDER_DATE,
       SHIP_DATE,
       DELIVERY_DATE,
       SHIPPING,
       DISCOUNT,
       TAX,
       TOTAL,
       QUANTITY
FROM ALL_FACTS,
     CUSTOMER,
     LOCATION
WHERE CUSTOMER.CUSTOMER_ID = 345123
      AND ALL_FACTS.LOCATION_ID = LOCATION.LOCATION_ID
      AND LOCATION.STORE_NUMBER = 29343
      AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
      AND FACT_DATE BETWEEN '2002-07-01' AND '2002-08-07'
```

**Query 3 – Average sales by location**

This query computes the average sales on a selected date, in terms of volume and income, for all stores serving specific sales channels in a selected location.

```
SELECT LOCATION.STORE_NAME,
       AVG(ALL_FACTS.TOTAL),
       AVG(ALL_FACTS.QUANTITY)
FROM ALL_FACTS,
     LOCATION,
     CHANNEL
WHERE ALL_FACTS.LOCATION_ID = LOCATION.LOCATION_ID
     AND ALL_FACTS.CHANNEL_ID = CHANNEL.CHANNEL_ID
     AND CHANNEL.CHANNEL_ID in ( 90, 61, 20)
     AND FACT_DATE = CAST('2002-11-01' AS DATE)
     AND LOCATION.STORE_NUMBER = 18242
GROUP BY
     LOCATION.STORE_NAME
ORDER BY SUM(ALL_FACTS.TOTAL) DESC
```

**Query 4 – Top daily sales**

This query computes the daily record sales, in terms of volume or income, for stores serving a selected channel in a given location.

```
SELECT LOCATION.STORE_NAME,
       MAX(ALL_FACTS.TOTAL),
       MAX(ALL_FACTS.QUANTITY)
FROM ALL_FACTS,
     LOCATION,
     CHANNEL
WHERE ALL_FACTS.LOCATION_ID = LOCATION.LOCATION_ID AND
     ALL_FACTS.CHANNEL_ID = CHANNEL.CHANNEL_ID AND
     CHANNEL.CHANNEL_ID = 59 AND
     ALL_FACTS.FACT_DATE = CAST('2003-06-01' AS DATE) AND
     LOCATION.STORE_NUMBER = 25341
GROUP BY _LOCATION.STORE_NAME
ORDER BY SUM(ALL_FACTS.TOTAL) DESC
```

**Query 5 – Sales analysis by channels**

This query computes the sales statistics on a selected date for all stores serving a specific sale channel in selected locations.

```
SELECT LOCATION.STORE_NAME,
       AVG(ALL_FACTS.TOTAL),
       MIN(ALL_FACTS.TOTAL),
       MAX(ALL_FACTS.TOTAL)
FROM ALL_FACTS,
     LOCATION,
     CHANNEL
WHERE ALL_FACTS.LOCATION_ID = LOCATION.LOCATION_ID
     AND ALL_FACTS.CHANNEL_ID = CHANNEL.CHANNEL_ID
     AND CHANNEL.CHANNEL_ID = 94
     AND FACT_DATE = CAST('2003-01-01' AS DATE)
     AND LOCATION.STORE_NUMBER IN ( 5401, 6319, 18950)
GROUP BY LOCATION.STORE_NAME
ORDER BY SUM(ALL_FACTS.TOTAL) DESC
```

**Query 6 – Credit exposure by product**

This query computes, over a specific period of time, the number of orders for a given product that originated from a specific location and were placed by customers with a weak credit rating.

```
SELECT STORE_NAME,
       COUNT(*)
FROM ALL_FACTS,
     CUSTOMER,
     LOCATION,
     PRODUCT
WHERE ALL_FACTS.LOCATION_ID = LOCATION.LOCATION_ID
     AND ALL_FACTS.PRODUCT_ID = PRODUCT.PRODUCT_ID
     AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
     AND LOCATION.STORE_NUMBER = 12744
     AND CUSTOMER.CUSTOMER_SCORE < 5
     AND PRODUCT.PRODUCT_TYPE = '0000000006'
     AND FACT_DATE BETWEEN '2001-05-25' AND '2001-06-03'
GROUP BY STORE_NAME
```

**Query 7 - Customer history**

This query retrieves all the orders made by a selected customer over a specific period of time.

```
SELECT CUSTOMER_FNAME,
       ALL_FACTS.CUSTOMER_ID,
       PAYMENT_METHOD,
       FACT_DATE,
       ORDER_DATE,
       SHIP_DATE,
       DELIVERY_DATE,
       SHIPPING,
       DISCOUNT,
       TAX,
       TOTAL,
       QUANTITY
FROM ALL_FACTS,
     CUSTOMER
WHERE CUSTOMER.CUSTOMER_ID = 2838456
      AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
      AND FACT_DATE BETWEEN '2002-06-01' AND '2002-07-07'
```

**Query 8 – Low credit exposure sales**

This query computes, for a selected product and over a period of time, the volume of sales generated by customers with high credit rating.

```
SELECT CUSTOMER_SCORE,
       SUM(TOTAL)
FROM ALL_FACTS,
     CUSTOMER,
     PRODUCT
WHERE ALL_FACTS.PRODUCT_ID = PRODUCT.PRODUCT_ID
      AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
      AND CUSTOMER_SCORE BETWEEN 85 AND 90
      AND PRODUCT_TYPE = '0000000016'
      AND FACT_DATE BETWEEN '2001-08-25' AND '2001-09-03'
GROUP BY CUSTOMER_SCORE
```

**Query 9 – Growth in low exposure sales**

This variant on Query 8 focuses on customers that show an increase in sales volume from the previous year.

```
SELECT CUSTOMER_SCORE,
       SUM(TOTAL)
FROM ALL_FACTS,
       CUSTOMER,
       PRODUCT
WHERE ALL_FACTS.PRODUCT_ID = PRODUCT.PRODUCT_ID
      AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
      AND CUSTOMER_SCORE BETWEEN 85 AND 90
      AND PRODUCT_TYPE = '0000000015'
      AND FACT_DATE BETWEEN '2001-08-25' AND '2001-08-31'
GROUP BY CUSTOMER_SCORE
HAVING SUM(TOTAL) > (
    SELECT SUM(TOTAL)
    FROM FACT_2000,
         PRODUCT
    WHERE FACT_2000.PRODUCT_ID = PRODUCT.PRODUCT_ID
          AND PRODUCT_TYPE = '0000000015'
          AND FACT_DATE = '2000-08-25'
    )
```

**Query 10 – Tracking customer growth**

This other variant on Query 8 focuses on a specific set of customers and, within that set, narrows it down to those who have recently ordered a selected product.

```
SELECT CUSTOMER_SCORE,
       SUM(TOTAL)
FROM ALL_FACTS,
       CUSTOMER,
       PRODUCT
WHERE ALL_FACTS.PRODUCT_ID = PRODUCT.PRODUCT_ID
      AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
      AND CUSTOMER_SCORE BETWEEN 75 AND 80
      AND PRODUCT_TYPE = '0000000041'
      AND FACT_DATE BETWEEN '2001-08-25' AND '2001-08-31'
      AND ALL_FACTS.CUSTOMER_ID IN (
    SELECT CUSTOMER_ID
    FROM FACT_2002
         CUSTOMER,
         PRODUCT
    WHERE ALL_FACTS.PRODUCT_ID = PRODUCT.PRODUCT_ID
          AND ALL_FACTS.CUSTOMER_ID=CUSTOMER.CUSTOMER_ID
          AND CUSTOMER_SCORE = 80
          AND CUSTOMER_ID BETWEEN 13000 AND 14000
          AND PRODUCT.PRODUCT_ID = 4323
          AND FACT_DATE = '2001-08-20'
    )
GROUP BY CUSTOMER_SCORE
```