Eldar Radovici **December 1st, 2003**
Advanced Databases **Database and Tuning Project**

Database Qualitative Benchmarks for Finance

Abstract…

This paper uses the FinTime queries to gauge the usefulness of the big three commercial database systems. This document provides research and reference to topics ranging from tuning a database system, the qualifications of the database language, and understanding how these commercial databases execute statistical queries. The purpose is to understand the obvious and subtle differences among these databases and select the best choice for a financial analysis project.

Table of Contents

1. Introduction

Time-series databases are important for financial modeling, analysis, and related applications. There are an abundance of database applications, but it's difficult to determine which is best for a specific statistical need. And the smallest advantage in terms of execution speed that one database may offer may be the competitive edge that you need to win in the financial world. What's needed to gauge and compare database systems are a set of querying benchmarks that ask the important financial questions.

Kaippallimalil J. Jacob and Dennis Shasha proposed the FinTime benchmark which makes up the queries in this paper. The questions and queries of the FinTime benchmark model practical uses of time-series databases in financial applications. The benchmark has two distinct models [historical and tick] that span different parameters that influence a time-series database system. Those parameters are:

1. Periodicity of data (regular such as daily and irregular such as real-time ticks)
2. Density of data (such as dense of sparse)
3. Schedule of updates (periodic such as during the week or continuous)
4. Complexity of queries (simple to complex)
5. Time interval between queries (ad hoc or batch queries)
6. Number of concurrent users (few or many)

For the purpose of this analysis all the queries [of both models] were conveniently changed to reflect the following:

Attribute	**Specification**
Periodicity of data	Periodic
Density of data	Dense
Schedule of updates	Periodic updates (i.e. at the end of a business day or on the weekends)
Complexity of queries	Simple and complex (i.e. decision support queries)
Time interval between queries	Ad hoc and batch
Number of concurrent	Low (i.e. meant for exclusive financial analysis)

These assumptions are ok because the theoretical research and application client [i.e. Preft] assumes this model. The Preft application is discussed in more detail in the **Background, Preft Project** section.

For further information on the FinTime benchmark, please refer to "A financial time series benchmark."[1]

[1] http://www.cs.nyu.edu/cs/faculty/shasha/fintime.d/bench.html

2. Objective

Our primary goal is to study the internals of a database system for the purpose of research and as a basis for rational performance tuning. The purpose of this research is to understand the big three commercial databases. The big three commercial databases studied in this document are:

1. IBM DB2 v8.1.3.132
2. Oracle v9.2.0.1.0 Production
3. Microsoft SQL Server 2000.

3. Background

3.a. Motivation. This paper stems off of the product of previous research that is "Predicting Financial Trends" [i.e. Preft]. The motivation to pursue this research is three fold.

1. Economic incentive
2. Combination of interests
3. Burgeoning industry

The economic incentive is straight-forward. By researching trends and the financial markets, it is likely that I'll develop a better understanding of the financial markets which is beneficial in itself. This research adds another tool to the growing bag of market applications that could be used to analyze investments. And the process of understanding the market and how it works allows us to grow as investors.

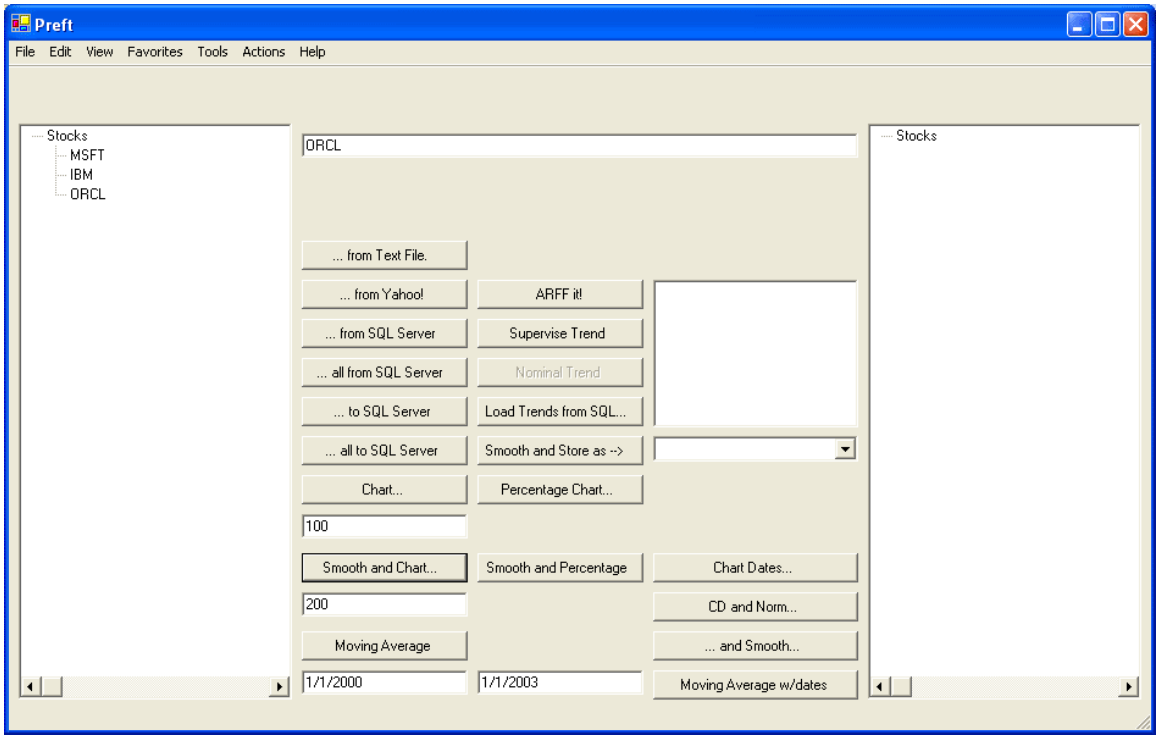
The research branches out across several industries and academic fields, all of which I'm interested in. Finance and economics play a vital role in understanding the domain. Data mining and data warehousing are burgeoning fields and are central to my area of interest as a computer scientist. And mathematics and statistics overlap many of these fields and provide tools for verification, validation and algorithms that are readily used in this application.

Analytical research in financial markets is not a new concept, but the burgeoning technologies and sciences such as data mining and warehousing are helping revolutionize and pioneer this industry.

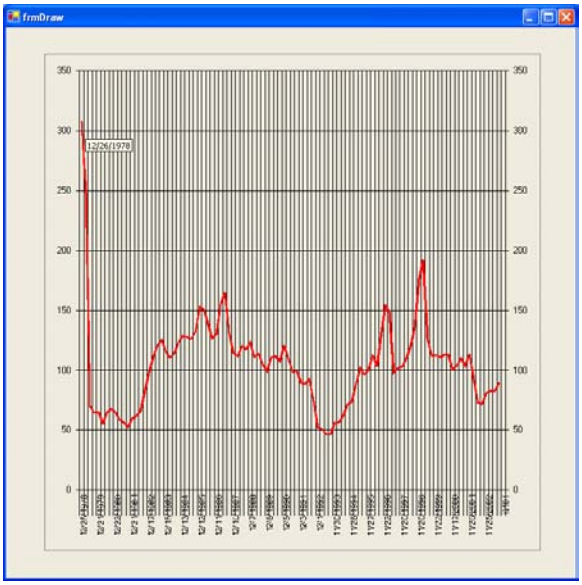
3.b. Preft Project

The four diagrams below are of:

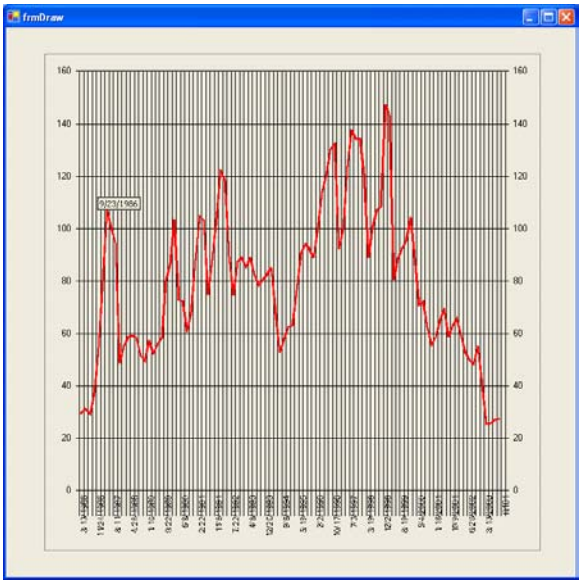
1. The Preft financial analysis project. This research was meant to increase this application's efficiency by way of tuning methods, increasing the SQL functionality and power of expression, and better understanding how time series databases work and how the commercial databases compare against each other for this purpose. This project interface has several options that allow:
    a. Loading and storing financial quotes from several sources [i.e. online quote source such as Yahoo Finance].
    b. Normalized, smoothed and/or moving average charts.
    c. Produce WEKA compatible ARFF file for machine learning and data mining.
2. Three associated 'moving averages' charts for IBM, Oracle and Microsoft respectively. Each moving average chart is calculated individually. These calculations are detailed in the **Background - Preft Project** section of this paper.


Preft
File Edit View Favorites Tools Actions Help
Stocks
MSFT
IBM
ORCL
ORCL
... from Text File.
... from Yahoo!
ARFF it!
... from SQL Server
Supervise Trend
... all from SQL Server
Nominal Trend
... to SQL Server
Load Trends from SQL...
... all to SQL Server
Smooth and Store as -->
Chart...
Percentage Chart...
100
Smooth and Chart...
Smooth and Percentage
Chart Dates...
200
CD and Norm...
Moving Average
... and Smooth...
1/1/2000
1/1/2003
Moving Average w/dates
Stocks

Preft


frmDraw
12/26/1978

IBM



Oracle [Orcl]

Microsoft [MSFT]

3.c. System Design and Infrastructure

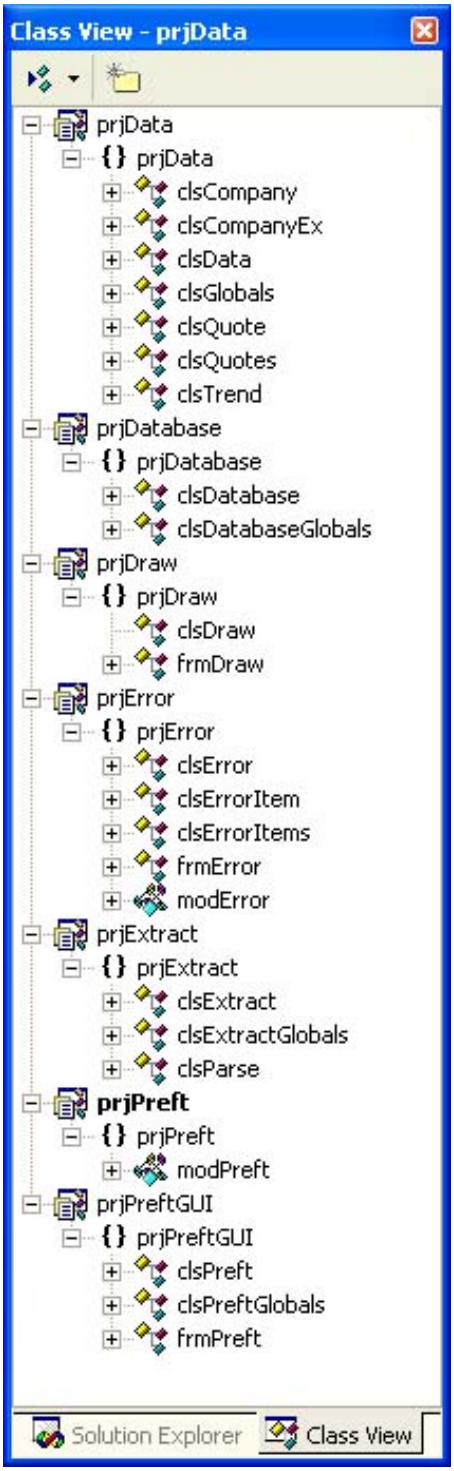
This project requires a proprietary infrastructure to extract, clean and transform financial data. The data acquisition, cleaning and preparing are significant parts of the data warehousing and pre-mining process. The following are [just some of] those tasks supported by this application:

1. Extract data from several sources such as Yahoo! Finance. [See 'prjExtract']
2. Warehouse data using SQL Server 2000. [See 'prjDatabase']
3. Provide the mathematical functionality to transform and smooth financial data.
    a. Binning allows us to smooth the data on the x-axis [default of 10 positions]

    b. Normalization on the y-axis adjusts all equity prices to a unit percentage scale
4. Charting functionality.
5. Supervise and select trends.
6. Output artificial datasets [i.e. ARFF artificial dataset generation].

The infrastructure: The design is implemented with Microsoft Visual Studio [VB.NET] and supported initially by SQL Server 2000. It makes use of SQL Analysis Manager, Query Analyzer and now IBM DB2 and Oracle database technology.

Code snippets: The following is a list of components and code that highlights the major tasks of the financial analysis project.



The above figure is that of the project class view [April 29, 2003]

PrjData: This component maintains all the data and supervised trends. The mathematical functions are contained in each class in an object orientated [and inherited] environment.

1. ClsTrend: Supervised dataset. This is a selection of clsData marked as a trend.
2. ClsCompany: Equity information such as ticker symbol, enumerated market, sector and industry.
3. ClsData: The highest-level data object. This contains the clsCompany and clsQuotes.
4. ClsQuotes: Maintains a collection of single day quotes or clsQuote.
5. ClsQuote: One day's quote including the open and close, low and high and volume.

```
Public Class clsCompany

'////////////////////////////////////////////////////////////////////////
'//
'// File:            clsCompany.vb
'//
'// Description:   Maintains the concept of an equity
'//
'// Author:        Eldar Radovici, 03/21/03
'//
'////////////////////////////////////////////////////////////////////////

Public Class clsQuote

'////////////////////////////////////////////////////////////////////////
'//
'// File:          clsQuote.vb
'//
'// Description:    Maintains the concept of a quote
'//
'//                              Statistical math functions for a single quote object
'//
'// Author:        Eldar Radovici, 03/21/03
'//
'////////////////////////////////////////////////////////////////////////

Public Function GetMovingAverage(ByVal oQuotes As prjData.clsQuotes, _
                                 Optional ByVal iMovingAverage As Integer = 50, _
                        Optional ByVal nwWeight As enumMathNormalizeWeight = nwBack) _
                                 As prjData.clsQuote

    Dim oResultantQuote As clsQuote
    oResultantQuote = New clsQuote
    oResultantQuote.propDate = propDate
    Dim dtDate As Date
    dtDate = propDate
    Dim iCount As Integer
    If nwWeight = enumMathNormalizeWeight.nwBack Or nwWeight = enumMathNormalizeWeight.nwFront
Then
        For iCount = 1 To iMovingAverage
            While Not oQuotes.InQuotes(dtDate)
                If DateDiff(DateInterval.Day, dtDate, oQuotes.propDateStart) > 0 Or
DateDiff(DateInterval.Day, dtDate, oQuotes.propDateEnd) < 0 Then
                    GetMovingAverage = Nothing : Exit Function
                End If
                dtDate = dtDate.AddDays(CDbl(nwWeight))
            End While
            Call oResultantQuote.Append(oQuotes.GetQuote(dtDate))
            dtDate = dtDate.AddDays(CDbl(nwWeight))
        Next
```

```
            Else '// nwWeight = enumNormalizeWeight.nwCenter
                For iCount = 1 To (iMovingAverage / 2)
                    While Not oQuotes.InQuotes(dtDate)
                        If DateDiff(DateInterval.Day, dtDate, oQuotes.propDateStart) > 0 Or
        DateDiff(DateInterval.Day, dtDate, oQuotes.propDateEnd) < 0 Then
                            GetMovingAverage = Nothing : Exit Function
                        End If
                        dtDate.AddDays(enumMathNormalizeWeight.nwBack)
                    End While
                    Call oResultantQuote.Append(oQuotes.GetQuote(dtDate))
                    dtDate.AddDays(enumMathNormalizeWeight.nwBack)
                Next
                For iCount = 1 To (iMovingAverage / 2)
                    While Not oQuotes.InQuotes(dtDate)
                        If DateDiff(DateInterval.Day, dtDate, oQuotes.propDateStart) > 0 Or
        DateDiff(DateInterval.Day, dtDate, oQuotes.propDateEnd) < 0 Then
                            GetMovingAverage = Nothing : Exit Function
                        End If
                        dtDate.AddDays(enumMathNormalizeWeight.nwFront)
                    End While
                    Call oResultantQuote.Append(oQuotes.GetQuote(dtDate))
                    dtDate.AddDays(enumMathNormalizeWeight.nwFront)
                Next
            End If
            oResultantQuote.Divide(CDbl(iMovingAverage))
            GetMovingAverage = oResultantQuote
            oResultantQuote = Nothing

End Function

Public Function GetRatio(ByVal oQuotes As prjData.clsQuotes, _
                         Optional ByVal iRatio As Integer = 50, _
                Optional ByVal nwWeight As enumMathNormalizeWeight = nwBack) _
                         As prjData.clsQuote

Public Class clsQuotes

'////////////////////////////////////////////////////////////////////////////
'//
'// File:          clsQuotes.vb
'//
'// Description:    Maintains the concept of a collection of quotes
'//
'//                          Statistical math functions for a quotes object
'//
'// Author:        Eldar Radovici, 03/21/03
'//
'////////////////////////////////////////////////////////////////////////////

'////////////////////////////////////////////////////////////////////////////
'//
'// Method:         Function GetStandardize() As prjData.clsQuotes
'//
'// Parameters:     None
'//
'// Returns:        clsQuotes
'//
'// Scope:          Public
'//
'// Description:    Normalizes the values [the y-axis]
'//
'////////////////////////////////////////////////////////////////////////////

'////////////////////////////////////////////////////////////////////////////
'//
'// Method:         Function GetStretch(ByVal dblSplit As Double, _
```

```
'//                      Optional ByVal bmEnum As enumMathBinMethods = bmMean) _
'//                      As prjData.clsQuotes
'//
'// Parameters:     (IN) Double - the number of partitions desired [i.e. def 10]
'//                [IN] enumMathBinMethods - leverage of the binning
'//                   i.e., mean, median, mode, weighted, etc.,
'//
'// Returns:        clsQuotes
'//
'// Scope:          Public
'//
'// Description:    Stretches [or constricts] the data [the x-axis]
'//
'////////////////////////////////////////////////////////////////////////////

Public Class clsData

'////////////////////////////////////////////////////////////////////////////
'//
'// File:           clsData.vb
'//
'// Description:    Maintains the concept of an entire equity
'//
'// Author:         Eldar Radovici, 03/21/03
'//
'////////////////////////////////////////////////////////////////////////////
```

PrjDatabase: Interaction with the development SQL Server 2000 database

1. Communicate with SQL Server and OLAP services.
2. Save/load data and trend data structures..

```
Public Class clsDatabase

'////////////////////////////////////////////////////////////////////////
'//
'// File:          clsDatabase.vb
'//
'// Description:    SQL 2000 functionality. Saving/loading data/trends.
'//
'// Author:        Eldar Radovici, 03/21/03
'//
'////////////////////////////////////////////////////////////////////////

'////////////////////////////////////////////////////////////////////////
'//
'// Method:        Function LoadDataStore() As Collection
'//
'// Parameters:     -
'//
'// Returns:       Collection  - collection of clsData
'//
'// Scope:         Public
'//
'// Description:    Returns a collection of all clsData members in the
'//               SQL Server datawarehouse.
'//
'////////////////////////////////////////////////////////////////////////

'////////////////////////////////////////////////////////////////////////
'//
'// Method:        Function LoadQuotes(ByVal iCompanyKey As Integer) As prjData.clsQuotes
'//
'// Parameters:     (IN) Integer       - Unique company identification [associated with the database]
'//
'// Returns:       prjData.clsQuotes   - Quotes object
'//
'// Scope:         Public
'//
'// Description:    Loads the data for a given stock
'//
'// Assumptions:    None
'//
'// Author:        Eldar 03/15/03
'//
'////////////////////////////////////////////////////////////////////////

Private Function LoadQuotes(ByVal iCompanyKey As Integer, Optional ByVal strStart As String = vbNullString,
Optional ByVal strEnd As String = vbNullString) As prjData.clsQuotes

    Dim dtStart As Date
    Dim dtEnd As Date
    On Error GoTo DateError
    Dim strCommandText As String
    If strStart = vbNullString Then
       strCommandText = "SELECT " & _
          SQL_QUOTES_DATE_KEY & ", " & _
          SQL_QUOTES_HIGH & ", " & _
          SQL_QUOTES_LOW & ", " & _
          SQL_QUOTES_PRICE_CLOSE & ", " & _
          SQL_QUOTES_PRICE_OPEN & ", " & _
          SQL_QUOTES_VOLUME & _
          " FROM " & SQL_QUOTES & " WHERE " & SQL_COMPANY_KEY & " = " & iCompanyKey & _
```

```
            " ORDER BY " & SQL_QUOTES_DATE_KEY & " DESC"
        Else
            dtStart = CDate(strStart)
            If strEnd = vbNullString Then
                dtEnd = Date.Today
            Else
                dtEnd = CDate(strStart)
            End If
            If DateDiff(DateInterval.Day, dtStart, dtEnd) > 0 Then
                strCommandText = "SELECT " & _
                    SQL_QUOTES_DATE_KEY & ", " & _
                    SQL_QUOTES_HIGH & ", " & _
                    SQL_QUOTES_LOW & ", " & _
                    SQL_QUOTES_PRICE_CLOSE & ", " & _
                    SQL_QUOTES_PRICE_OPEN & ", " & _
                    SQL_QUOTES_VOLUME & _
                    " FROM " & SQL_QUOTES & " WHERE " & SQL_COMPANY_KEY & " = " & iCompanyKey & _
                    " AND " & _
                    "(" & _
                    SQL_QUOTES_DATE_KEY & " > " & dtStart & _
                    " AND " & _
                    SQL_QUOTES_DATE_KEY & " < " & dtEnd & _
                    ")" & _
                    " ORDER BY " & SQL_QUOTES_DATE_KEY & " DESC"
            End If
        End If
        '// Create a Command object
        Dim dbCmd As SqlCommand
        dbCmd = m_dbConn.CreateCommand
        dbCmd.CommandText = strCommandText
        m_dbReader = dbCmd.ExecuteReader(CommandBehavior.Default)
        Dim oQuotes As prjData.clsQuotes
        oQuotes = New prjData.clsQuotes()
        While (m_dbReader.Read())
            Dim oQuote As prjData.clsQuote
            oQuote = New prjData.clsQuote()
            oQuote.propDate = m_dbReader.GetDateTime(enumDatabaseQuotes.DateKey)
            oQuote.propHigh = m_dbReader.GetDouble(enumDatabaseQuotes.High)
            oQuote.propLow = m_dbReader.GetDouble(enumDatabaseQuotes.Low)
            oQuote.propClose = m_dbReader.GetDouble(enumDatabaseQuotes.PriceClose)
            oQuote.propOpen = m_dbReader.GetDouble(enumDatabaseQuotes.PriceOpen)
            oQuote.propVolume = m_dbReader.GetInt64(enumDatabaseQuotes.Volume)
            oQuotes.AddQuote(oQuote)
            oQuote = Nothing
        End While
        '// Close the reader and not the database connection
        m_dbReader.Close()
        LoadQuotes = oQuotes
        oQuotes = Nothing

DateError:

End Function
```

4. Project Outline

4.a. The Approach.

1. Setup databases.
2. Extract and manage historical data.
    a. Extract data.
    b. Load data.
        i. SQL statements and where to get them [including SQLLDR]
        ii. Chunks
3. Compare and test database systems and technologies.
    a. How these where achieved.
        i. Profiler and execution plan
        ii. Oracle nominal timing, script and manual access plan
        iii. IBM and access plan
    b. Compare retrieval benchmarks.
    c. Compare in terms of speed and complexity math related benchmarks [moving averages].
4. Tune the databases for this project's overall purpose.

4.b. Databases. The three databases below have been selected for this project's querying benchmarks and comparisons.

1. Microsoft SQL 2000
2. IBM DB2 version 8.1
3. Oracle 9

4.c. Server. The machine running these databases as well as all the querying has the following hardware configuration:

1. Windows 2000 Server
2. 2.4 G-Hertz P4
3. 512 MB SD Ram
4. 2x80 GB 7200 RPM 8 MB Buffer

4.d. Data. The data is acquired and prepared by an application written in VB.NET 2003. The data source is Yahoo Finance and includes the following data attributes:

Data Attributes [Quotes Table]:

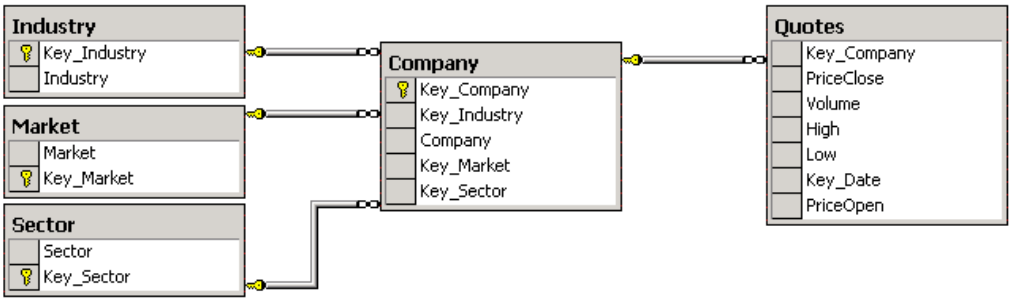
0. Index
1. Open
2. Volume
3. High
4. Low
5. Date

The data 'universe' is the Standard and Poors 1500 domestic index.[2]

4.e. Schema. The Quotes table in diagram below is the fact table that warehouses all the data information. The following is the extended star schema [snowflake] structured relationship.

The relationship:

[2] Refer to the accompanying file S&P1500.xls


Industry
Key_Industry
Industry
Market
Market
Key_Market
Sector
Sector
Key_Sector
Company
Key_Company
Key_Industry
Company
Key_Market
Key_Sector
Quotes
Key_Company
PriceClose
Volume
High
Low
Key_Date
PriceOpen

5. FinTime Proposed Queries

Queries[3]

Historical

1. Get the closing price of a set of 10 stocks for a 10-year period and group into weekly, monthly and yearly aggregates. For each aggregate period determine the low, high and average closing price value. The output should be sorted by id and trade date.

2. Adjust all prices and volumes (prices are multiplied by the split factor and volumes are divided by the split factor) for a set of 1000 stocks to reflect the split events during a specified 300 day period, assuming that events occur before the first trade of the split date. These are called split-adjusted prices and volumes.

3. For each stock in a specified list of 1000 stocks, find the differences between the daily high and daily low on the day of each split event during a specified period.

4. Calculate the value of the S&P500 and Russell 2000 index for a specified day using unadjusted prices and the index composition of the 2 indexes (see appendix for spec) on the specified day

5. Find the 21-day and 5-day moving average price for a specified list of 1000 stocks during a 6-month period. (Use split adjusted prices)

6. (Based on the previous query) Find the points (specific days) when the 5-month moving average intersects the 21-day moving average for these stocks. The output is to be sorted by id and date.

7. Determine the value of $100,000 now if 1 year ago it was invested equally in 10 specified stocks (i.e. allocation for each stock is $10,000). The trading strategy is: When the 20-day moving average crosses over the 5-month moving average the complete allocation for that stock is invested and when the 20-day moving average crosses below the 5-month moving average the entire position is sold. The trades happen on the closing price of the trading day.

8. Find the pair-wise coefficients of correlation in a set of 10 securities for a 2 year period. Sort the securities by the coefficient of correlation, indicating the pair of securities corresponding to that row. [Note: coefficient of correlation defined in appendix]

9. Determine the yearly dividends and annual yield (dividends/average closing price) for the past 3 years for all the stocks in the Russell 2000 index that did not split during that period. Use unadjusted prices since there were no splits to adjust for.

[3] http://www.cs.nyu.edu/cs/faculty/shasha/fintime.d/bench.html

Tick

1. Get all ticks for a specified set of 100 securities for a specified three hour time period on a specified trade date. [Granularity adjustment: We assume three hours to be three years].

2. Determine the volume weighted price of a security considering only the ticks in a specified three hour interval. [Granularity adjustment: We assume three hours to be three years].

3. Determine the top 10 percentage losers for the specified date on the specified exchanges sorted by percentage loss. The loss is calculated as a percentage of the last trade price of the previous day.

4. Determine the top 10 most active stocks for a specified date sorted by cumulative trade volume by considering all trades. [Granularity adjustment: We assume one year of trades].

5. Find the most active stocks in the "COMPUTER" industry (use SIC code)

6. Find the 10 stocks with the highest percentage spreads. Spread is the difference between the last ask-price and the last bid-price. Percentage spread is calculated as a percentage of the mid-point price (average of ask and bid price). [Changed: Find the 10 stocks with the highest percentage spreads in terms of the day's high and day's low.]

6. Setup

The database systems were set up for the purpose of monitoring performance and analyzing queries.

6.a. Oracle Access Plans[4]. Oracle uses an "Explain Plan" to record and show the execution plan that the Oracle database devises when executing a particular query. The explain plan offers valuable information on how the Oracle database attempts to optimize your query in their native language. With an explain plan, the database administrator is able to determine if the database is correctly picking the right indexes and joining tables in the most efficient manner.

Please refer to **evolt.org** article on "Use Oracle's Explain Plan to Tune your Queries" in the references section.

6.b. IBM DB 2 Access Plans[5]

Similarly to Oracle's access plan and Microsoft SQL Server's execution plan, IBM DB2 produces a flow-chart style and user friendly access plan. Please refer to **seas.ucla.edu** article on "Creating an Access Plan Graph" in the references section.

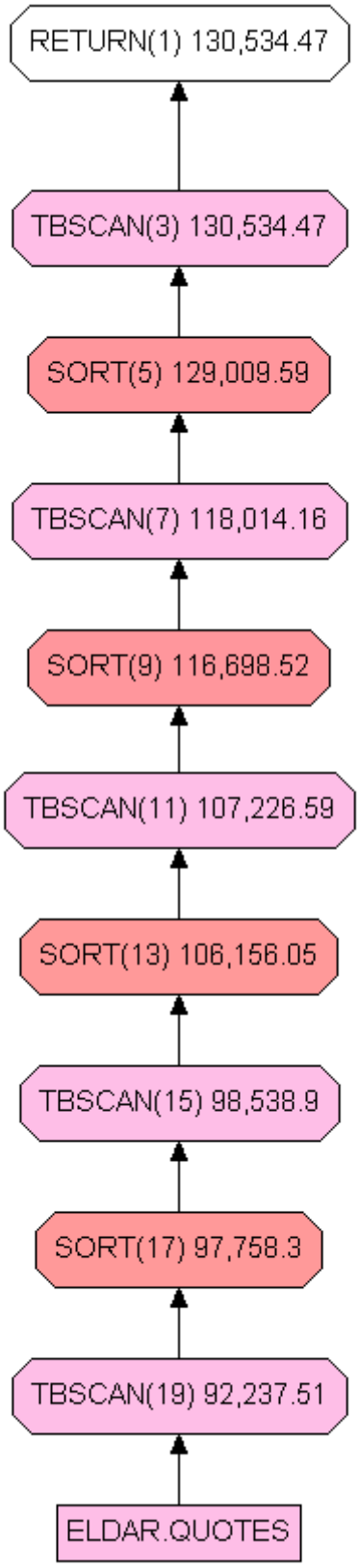
The following access plan is the first historical query and shows how the colors are associated with particular database and querying functionality. As per the diagram below, the color scheme is:

a. White: Return results
b. Pink: Table scan
c. Red: Sort

Other color associations are:

d. Brown: Join
e. Green: Group by
f. Beige: Implementing indexing / fetch

[4] http://www.evolt.org/article/Use_Oracle_s_Explain_Plan_to_Tune_Your_Queries/17/2986/
[5] http://www.seas.ucla.edu/db2/db2help/index.htm#tve02



Hist.Query.1.Years.10.IBM.Plan.Legend.gif

7. Results

The results for each database system are displayed and analyzed separately. The access plan and accompanying diagrams are displayed immediately after their associated queries. The naming convention under each diagram is as such:

<TYPE>.<Query>.<#>.<Data Attribute Name>.<Value>[.Misc].<Company>.<Output Type>

For example: **Hist.Query.1.Years.10.Rollup.IBM.Plan.gif** means that the above diagram is for the historical query number 1 on the IBM DB2 database. It specifies that the where clause is between 10 years of quote data and has the added specification that it uses the Rollup OLAP functionality. The output type is that of an access plan [or execution plan].

The three database systems that are covered in this analysis are:

1. IBM DB2 v8.1.3.132
2. Oracle v9.2.0.1.0 Production
3. Microsoft SQL Server 2000.

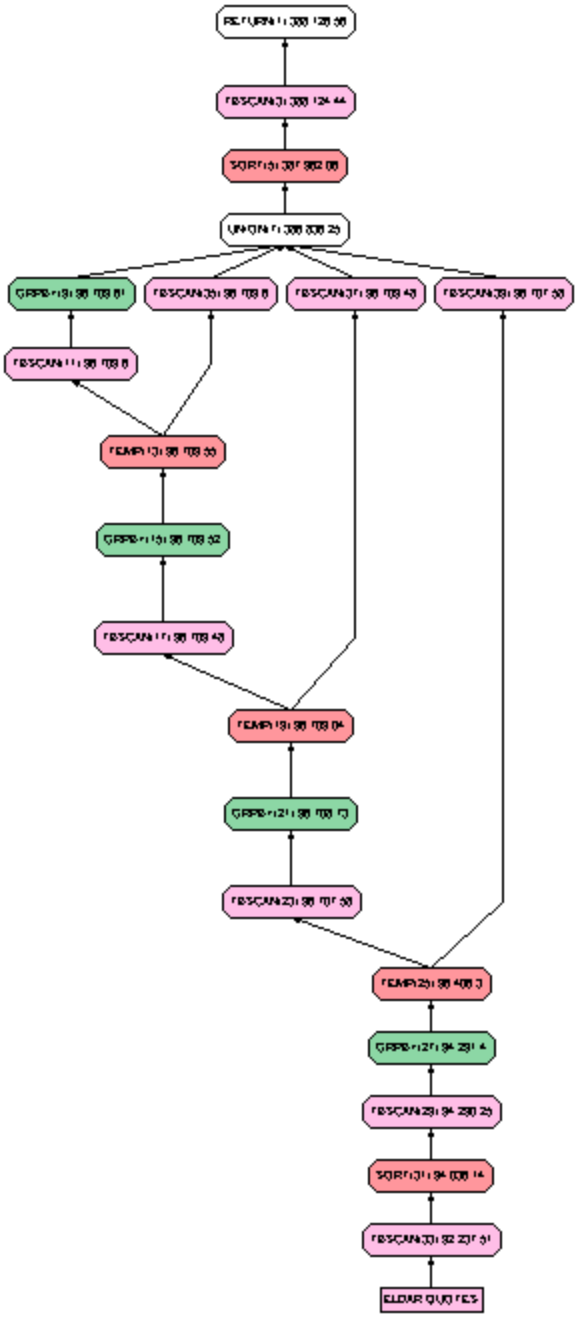
7.a. IBM DB2 v8.1.3.132

Historical.

Query 1. DB2 with Rollup

a. 10 Years. This query executed in approximately **4** seconds. The total cost (timerons): 388,126.56

```
SELECT
 Key_Company,
 year(KeyDate) AS year,
 month(KeyDate) AS month,
 week(KeyDate) AS week,
 max(PriceClose) AS price_max,
 min(PriceClose) AS price_min,
 avg(PriceClose) AS price_avg
 FROM
 Quotes
 WHERE
   Key_Company > 100 AND Key_Company <= 110 AND
   KeyDate > '1990-01-01 00:00:00.000000'
   AND
   KeyDate < '2000-01-01 00:00:00.000000'
 GROUP BY rollup (
  (Key_Company,year(KeyDate)),
  (Key_Company,month(KeyDate)),
  (Key_Company,week(KeyDate)))
 ORDER BY Key_Company, year(KeyDate), month(KeyDate), week(KeyDate);
```



Hist.Query.1.Years.10.Rollup.IBM.Plan.gif

Interactive | Script | Results | Access Plan

Perform required changes and then click the commit update button.

KEY_COM...	YEAR	MONTH	WEEK	PRICE_MAX	PRICE_MIN	PRICE_AVG
101	1990	1	1	36.13	35.75	35.94
101	1990	1	2	35.88	33.75	35.376
101	1990	1	3	33.13	31.37	32.05
101	1990	1	4	31.75	30.75	31.15
101	1990	1	5	30.75	30.62	30.707
101	1990	1		36.13	30.62	33.125
101	1990	2	5	31.5	30.75	31.125
101	1990	2	6	31.5	30.62	30.972
101	1990	2	7	32	30.75	31.448
101	1990	2	8	31.87	30.62	31.37
101	1990	2	9	31.25	30.87	31.08
101	1990	2		32	30.62	31.214
101	1990	3	9	31.12	31	31.06
101	1990	3	10	32.5	30.87	31.698
101	1990	3	11	33.25	31.5	32.074
101	1990	3	12	34	33.25	33.578
101	1990	3	13	33.88	33.63	33.778
101	1990	3		34	30.87	32.625
101	1990	4	14	34.5	32.5	33.778
101	1990	4	15	33.88	33.38	33.628
101	1990	4	16	34	33.63	33.852
101	1990	4	17	33	32.63	32.878
101	1990	4	18	33.38	33.38	33.38
101	1990	4		34.5	32.5	33.522
101	1990	5	18	33.88	33.38	33.628
101	1990	5	19	35.5	34	34.726

Hist.Query.1.Years.10.Rollup.IBM.Results.gif

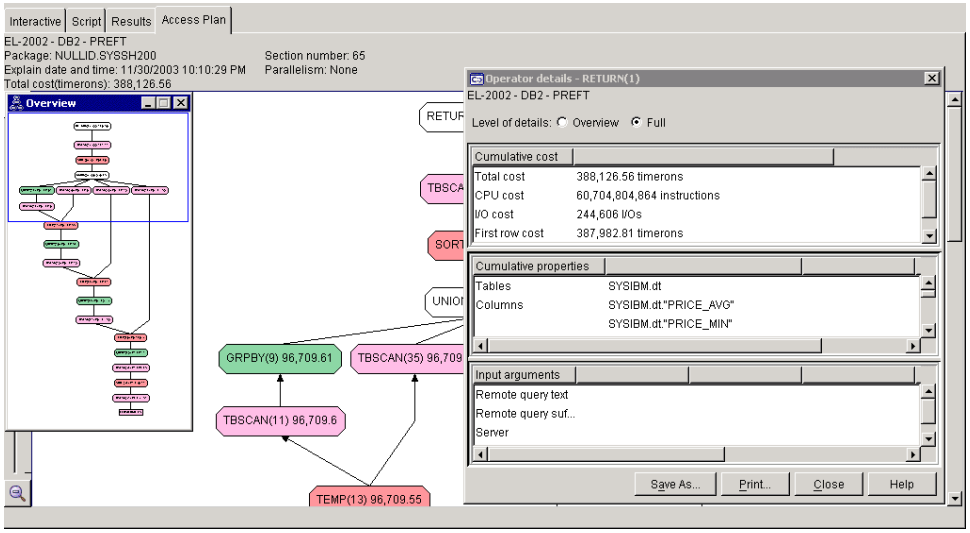
Query 1. DB2

a. 10 Years. This query executed in approximately **5-7** seconds. Surprisingly, with an index it executed slower completing in approximately **15-17** seconds. The access plans are the same indicating that the optimizer decided to forego the indexes and execute the table scan. This may imply that the majority of the execution time was spent deciding how to optimize the query, in which case the decision was to use the original execution plan. Total cost (timerons): 130,534.47

b. 10 Days. This query executed in approximately **3-4** seconds.

```
SELECT
 Key_Company,
 KeyDate,
 max(PriceClose) OVER (PARTITION BY Key_Company, month(KeyDate)) AS monthMax,
 min(PriceClose) OVER (PARTITION BY Key_Company, month(KeyDate)) AS monthMin,
 avg(PriceClose) OVER (PARTITION BY Key_Company, month(KeyDate)) AS monthAvg,
 max(PriceClose) OVER (PARTITION BY Key_Company, year(KeyDate)) AS yearMax,
 min(PriceClose) OVER (PARTITION BY Key_Company, year(KeyDate)) AS yearMin,
 avg(PriceClose) OVER (PARTITION BY Key_Company, year(KeyDate)) AS yearAvg,
 max(PriceClose) OVER (PARTITION BY Key_Company, week(KeyDate)) AS weekMax,
 min(PriceClose) OVER (PARTITION BY Key_Company, week(KeyDate)) AS weekMin,
 avg(PriceClose) OVER (PARTITION BY Key_Company, week(KeyDate)) AS weekAvg
 FROM
 Quotes
 WHERE
   Key_Company > 100 AND Key_Company <= 110 AND
   KeyDate > '1990-01-01 00:00:00.000000'
   AND
   KeyDate < '2000-01-01 00:00:00.000000'
```

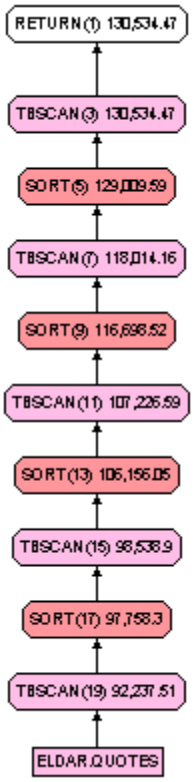
ORDER BY Key_Company, KeyDate;



Hist.Query.1.Years.10.IBM.Info.gif

KEY_COM...	KEYDATE	MONTHMAX	MONTHMIN	MONTHAVG	YEARMAX	YEARMIN	YEARAVG	WEEKMAX	WEEKMIN	WEEKAVG
101	Jan 2, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	40.13	26.37	31.676
101	Jan 3, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	40.13	26.37	31.676
101	Jan 4, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	40.13	26.37	31.676
101	Jan 5, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	40.13	26.37	31.676
101	Jan 8, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	49.88	26	35.118
101	Jan 9, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	49.88	26	35.118
101	Jan 10, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	49.88	26	35.118
101	Jan 11, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	49.88	26	35.118
101	Jan 12, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	49.88	26	35.118
101	Jan 15, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.38	26	34.946
101	Jan 16, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.38	26	34.946
101	Jan 17, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.38	26	34.946
101	Jan 18, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.38	26	34.946
101	Jan 19, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.38	26	34.946
101	Jan 22, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.5	34.726
101	Jan 23, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.5	34.726
101	Jan 24, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.5	34.726
101	Jan 25, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.5	34.726
101	Jan 26, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.5	34.726
101	Jan 29, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.62	34.84
101	Jan 30, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.62	34.84
101	Jan 31, 1990 12:00:00 AM	49.88	25.5	34.792	37.25	26.87	32.452	48.5	25.62	34.84
101	Feb 1, 1990 12:00:00 AM	49.13	23.06	34.845	37.25	26.87	32.452	48.5	25.62	34.84
101	Feb 2, 1990 12:00:00 AM	49.13	23.06	34.845	37.25	26.87	32.452	48.5	25.62	34.84
101	Feb 5, 1990 12:00:00 AM	49.13	23.06	34.845	37.25	26.87	32.452	49.13	26.62	35.237
101	Feb 6, 1990 12:00:00 AM	49.13	23.06	34.845	37.25	26.87	32.452	49.13	26.62	35.237

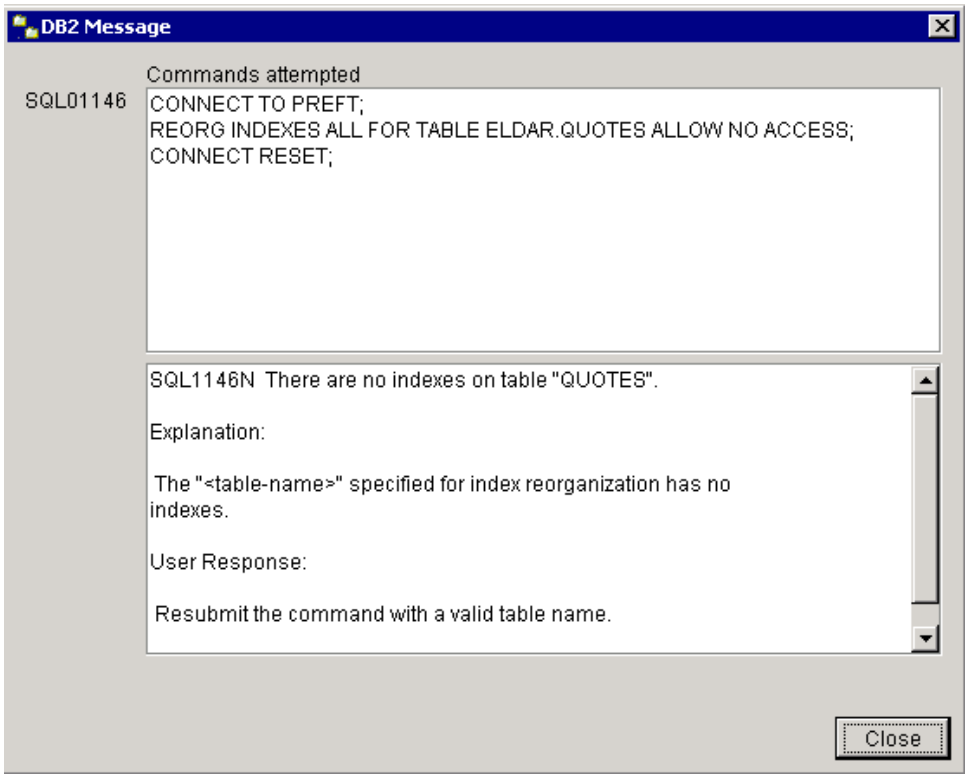
Hist.Query.1.Years.10.IBM.Results.gif



Hist.Query.1.Years.10.Index.1.IBM.Plan.gif

Notes: I was surprised that the original query executed so quickly. The DB2 query executed in 5 seconds. SQL Server 2000 didn't complete the 10 year query in an hour and was cancelled before it could finish. However it's important to note that the SQL statements are very different. DB2 utilized 'partition by' and/or the rollup functionality making its SQL query cleaner and more efficient. SQL Server 2000 supports a different SQL dialect language and as such [I] was unable to express an efficient solution to this query.

The diagram below shows that no index existed at this time. The index reorganization was implemented to check if IBM DB2 had [by default] inserted an index on the Quotes table.

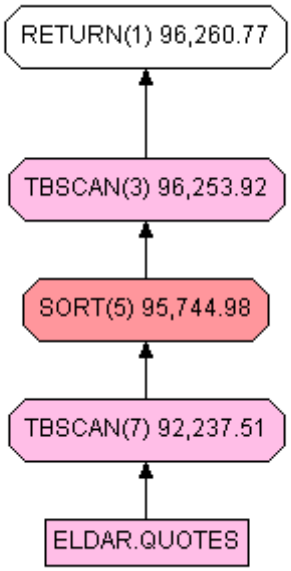


Hist.Query.1.Years.10.Index.No.IBM.gif

Query 3.

a. This query executed in approximately **4-5** seconds. Similarly to the previous query, this one executed slower [est. **7** seconds] with indexes. The access plan diagram is the same for the index and non-indexed tables. Total cost (timerons): 96,260.77

```
SELECT (High-Low) AS DailyDifferential, PriceClose, Key_Company
FROM QUOTES
WHERE
   KeyDate >= '2002-01-01 00:00:00.000000'
   AND
   KeyDate <= '2002-01-31 00:00:00.000000'
   AND KEY_COMPANY > 100 AND Key_Company <= 1100
 ORDER BY Key_Company;
```



Hist.Query.3.IBM.Plan.gif

Interactive | Script | Results | Access Plan

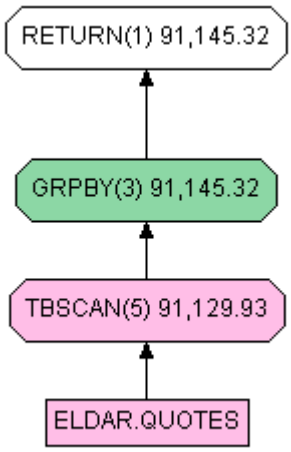
Perform required changes and then click the

DAILYDIFF...	PRICECL...	KEY_COM...
0.87	14.05	101
0.24	14.06	101
0.48	14.52	101
0.73	15.15	101
0.7	15.5	101
0.5	15.88	101
0.52	15.52	101
0.34	15.38	101
0.24	15.42	101
0.52	15.44	101
0.54	14.97	101
1.12	13.87	101
0.71	13.74	101
1.35	13.02	101
0.85	12.8	101
0.6	12.4	101
0.8	12.95	101
0.4	12.86	101
0.73	13.56	101
1.1	12.25	101
0.61	12.46	101
2.05	30.55	102
2.09	29.87	102
1.14	29.47	102
1.18	28.48	102
1.86	28.99	102

Hist.Query.3.IBM.Results.gif

Query 4.
a. This query executed in approximately **3-4** seconds. With an index the execution time was similar [**3-4** seconds]. The access plan diagram is the same for the index and non-indexed tables. The result to this query is **31.175397093652236.** Total cost (timerons): 91,145.32

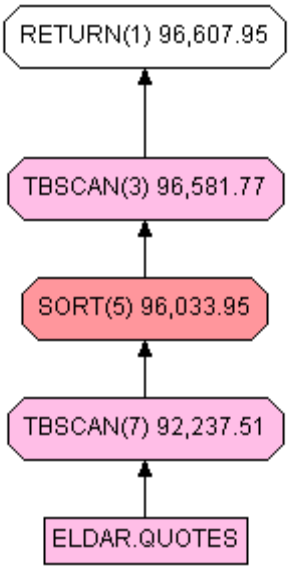
```
SELECT AVG(PriceClose) AS SandP1500Index
 FROM
 Quotes
 WHERE KeyDate ='2002-01-31 00:00:00.000000';
```



Hist.Query.4.IBM.Plan.gif

Query 5.
a. Execution without index: **3-5** seconds. Execution with index: **3-5** seconds. The access plan diagram is the same for the index and non-indexed tables. Total cost (timerons): 96,607.95

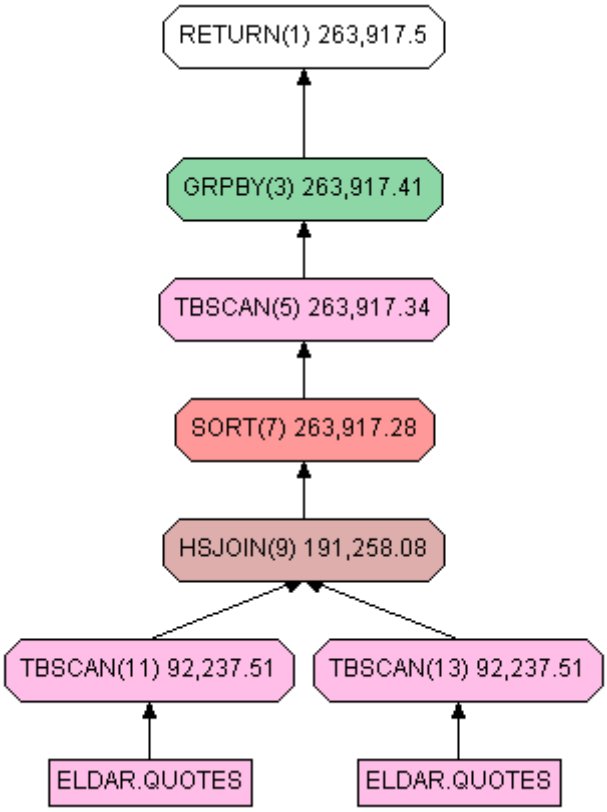
```
SELECT
 Key_Company,
 KeyDate,
 avg(PriceClose)
  OVER (PARTITION BY Key_Company ORDER BY KeyDate asc ROWS between 20 preceding
AND current row) mAvg_21,
 avg(PriceClose)
  OVER (PARTITION BY Key_Company ORDER BY KeyDate asc ROWS between 4 preceding AND
current row) mAvg_5
 FROM Quotes
 WHERE
  Key_Company > 100 AND Key_Company <= 1100 AND
  KeyDate >= '2002-07-01 00:00:00.000000'
  AND
  KeyDate < '2003-01-01 00:00:00.000000';
```



Hist.Query.5.IBM.Plan.gif

Query 8.
a. Execution without index: **8-9.5** seconds. Execution with index: **9** seconds. The access plan diagram is the same for the index and non-indexed tables. Total cost (timerons): 263,917.5

```
SELECT Q1.Key_Company, Q2.Key_Company, correlation(Q1.PriceClose, Q2.PriceClose) AS
CorrCoef
 FROM Quotes Q1, Quotes Q2
 WHERE
  Q1.KeyDate=Q2.KeyDate
  AND Q1.Key_Company > 100 AND Q1.Key_Company <= 110
  AND Q2.Key_Company > 100 AND Q2.Key_Company <= 110
  AND Q1.KeyDate BETWEEN '2000-01-01 00:00:00.000000'
  AND '2002-01-01 00:00:00.000000'
GROUP BY Q1.Key_Company, Q2.Key_Company;
```

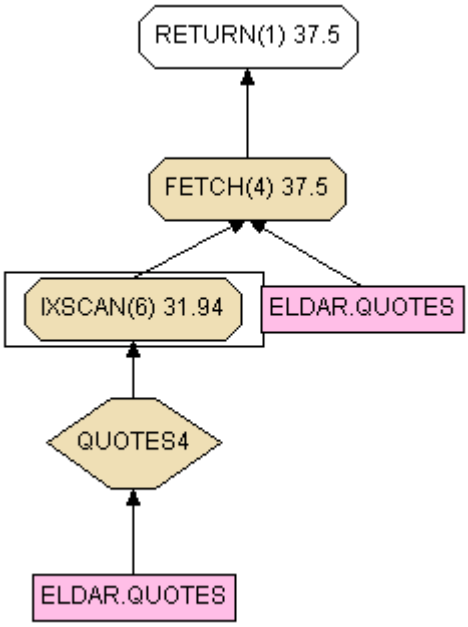


Hist.Query.8.IBM.Plan.gif

Index Query

a. The purpose of this query was to show the existence of the index. The query intuitively works best with an index on KeyDate and/or Key_Company. The access plan shows that the index was used to scan for the KeyDate value.

```
SELECT
 *
 FROM Quotes
 WHERE (KeyDate = '2003-01-01 00:00:00.000000') AND
 Key_Company = 532;
```



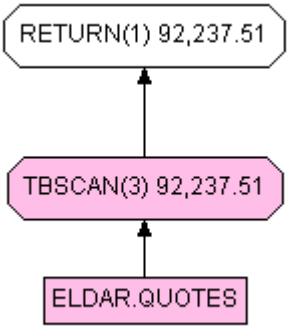
Index.Test.IBM.Plan.gif

Tick.

Query 1. The two queries answer the same question. However the second query returns the 'name' of the ticker symbol and is more readable then the Key_Company result. Although the access plan is more complicated for the second query, the total cost is nearly identical.

a. Execution time: **11** seconds. Total cost (timerons): 92,237.51

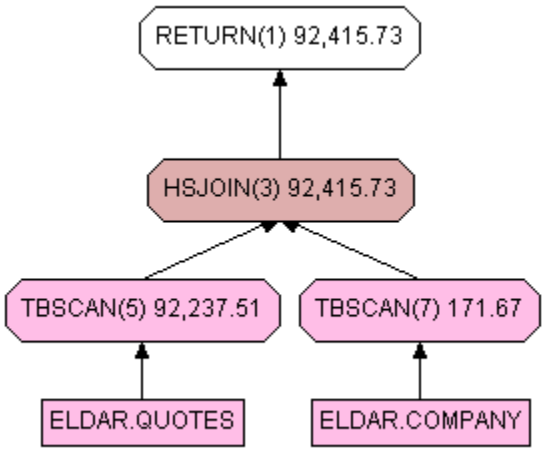
```
SELECT *
 FROM Quotes
 WHERE
 Key_Company > 100 AND Key_Company <= 200  AND
 KeyDate >= '2000-01-01 00:00:00.000000' AND KeyDate < '2003-01-01 00:00:00.000000';
```



Tick.Query.1.IBM.Plan.gif

b. Execution time: **12** seconds. Total cost (timerons): 92,415.73

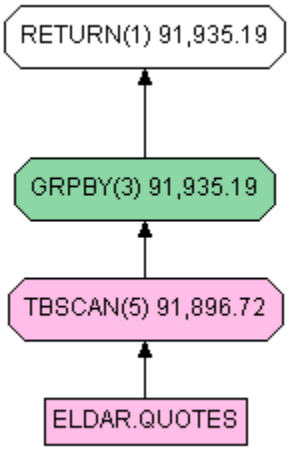
```
SELECT c.Company, q.*
 FROM Quotes q, Company c
 WHERE
 q.Key_Company = c.Key_Company AND
 q.Key_Company > 100 AND q.Key_Company <= 200 AND
 q.KeyDate >= '2000-01-01 00:00:00.000000' AND q.KeyDate < '2003-01-01
00:00:00.000000';
```



Tick.Query.1.Company.IBM.Plan.gif

Query 2.
a. Execution time: **5** seconds. Total cost (timerons): 91,935.19

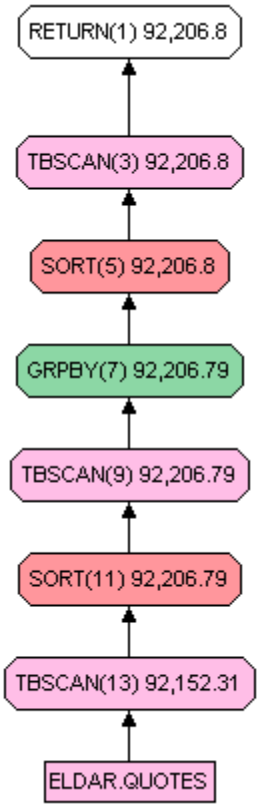
```
SELECT SUM(PriceClose*Volume)/SUM(Volume) VolPrice
 FROM Quotes
 WHERE
  KeyDate >= '2000-01-01 00:00:00.000000' AND KeyDate < '2003-01-01 00:00:00.000000';
```



Tick.Query.2.IBM.Plan.gif

Query 4.
a. Execution time: **4** seconds. Total cost (timerons): 92,206.8

```
SELECT SUM(Volume) AS cumalVolume, Key_Company
 FROM Quotes
 WHERE (Key_Company > 1450) AND KeyDate >= '2002-01-01 00:00:00.000000' AND
KeyDate <= '2002-01-31 00:00:00.000000'
 GROUP BY Key_Company
 ORDER BY cumalVolume DESC;
```

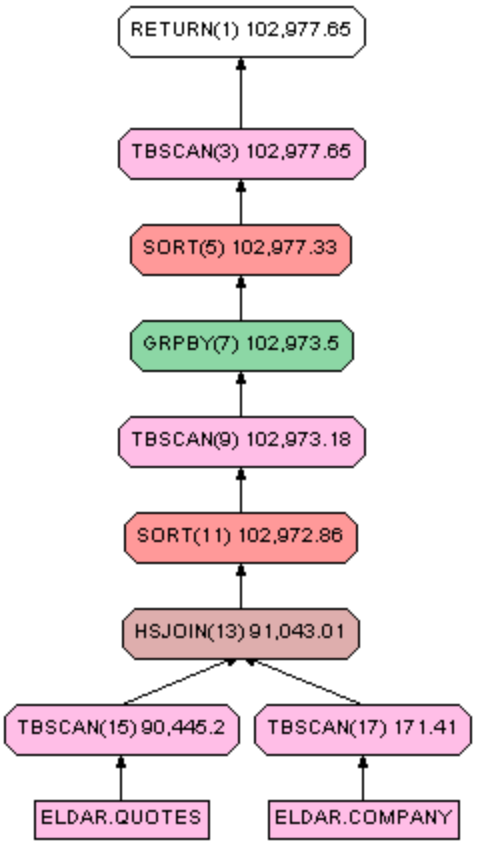


Tick.Query.4.IBM.Plan.gif

Query 5.

a. Execution time: **5** seconds. Total cost (timerons): 102,977.65

```
SELECT
 AVG(q.Volume) AS Activity, q.Key_Company, c.Company
 FROM Quotes q, Company c
 WHERE q.Key_Company = c.Key_Company
 AND c.Key_Sector = 1
 GROUP BY q.Key_Company, c.Company
 ORDER BY Activity DESC;
```

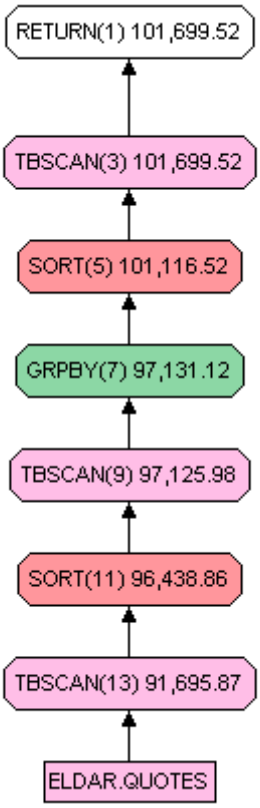


Tick.Query.5.IBM.Plan.gif

Query 6.

a. Execution time: **4-5** seconds. Total cost (timerons): 101,699.52

```
SELECT
 (QUOTES.PRICECLOSE/((QUOTES.HIGH+QUOTES.LOW)/2)) AS pSpread, Quotes.Key_Company
 FROM QUOTES WHERE Key_Company > 1450 AND QUOTES.KeyDate = '2002-01-31
00:00:00.000000'
 GROUP BY Key_Company, PriceClose, High, Low
 ORDER BY pSpread DESC, Key_Company;
```



Tick.Query.6.IBM.Plan.gif

7.b. Oracle v9.2.0.1.0 Production

Historical.

Query 1.

```
SELECT Key_Company,
 TRUNC(Key_Date,'W') AS week,
 TRUNC(Key_Date,'MM') AS month,
 TRUNC(Key_Date,'YY') AS year,
 MAX(PriceClose) AS price_max,
 MIN(PriceClose) AS price_min,
 AVG(PriceClose) AS price_avg
 FROM Quotes
  WHERE
   Key_Company > 100 AND Key_Company <= 110 AND
   Key_Date BETWEEN
   TO_DATE('January 01, 1990', 'Month dd, YYYY', 'NLS_DATE_LANGUAGE = American')
   AND
   TO_DATE('January 01, 2000', 'Month dd, YYYY', 'NLS_DATE_LANGUAGE= American')
 GROUP BY Key_Company,
 ROLLUP
  (
  TRUNC(Key_Date,'YY'),
  TRUNC(Key_Date,'MM'),
  TRUNC(Key_Date,'W')
  )
 ORDER BY Key_Company;
```

a. Without indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	GROUP BY ROLLUP	
TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	GROUP BY ROLLUP	
TABLE ACCESS	BY INDEX ROWID	QUOTES
INDEX	RANGE SCAN	QUOTES1

Query 3.

```
SELECT (High-Low) AS DailyDifferential, PriceClose, Key_Company
FROM QUOTES
WHERE
   Key_Date >= TO_DATE('01-January-2002') AND Key_Date <= TO_DATE('31-January-2002')
   AND KEY_COMPANY > 100 AND Key_Company <= 1100
 ORDER BY Key_Company;
```

a. Without indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	ORDER BY	

TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
TABLE ACCESS	BY INDEX ROWID	QUOTES
INDEX	RANGE SCAN	QUOTES1

Query 4.

```
SELECT AVG(PriceClose) AS SandP1500Index
 FROM
 QUOTES
 WHERE Key_Date = TO_DATE('31-January-2002');
```

a. Without indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	AGGREGATE	
TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	AGGREGATE	
TABLE ACCESS	FULL	QUOTES

Query 5.

```
SELECT
 Key_Company,
 Key_Date,
 AVG(PriceClose)
  OVER (PARTITION BY Key_Company ORDER BY Key_Company, Key_Date ROWS 20
PRECEDING) AS mAvg_21,
 AVG(PriceClose)
  OVER (PARTITION BY Key_Company ORDER BY Key_Company, Key_Date ROWS 4
PRECEDING) AS mAvg_5
 FROM Quotes
 WHERE
  Key_Company >= 100 AND Key_Company <= 1100 AND
  Key_Date BETWEEN
   TO_DATE('July 01, 2002', 'Month dd, YYYY', 'NLS_DATE_LANGUAGE= American')
   AND
   TO_DATE('January 01, 2003', 'Month dd, YYYY', 'NLS_DATE_LANGUAGE= American');
```

a. Without indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
WINDOW	SORT	
TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	OPTIONS	OBJECT_NAME
SELECT STATEMENT		
WINDOW	BUFFER	
TABLE ACCESS	BY INDEX ROWID	QUOTES
INDEX	RANGE SCAN	QUOTES1

Query 6.

```
CREATE TABLE tempAvg
AS SELECT
 Key_Company,
 Key_Date,
 AVG(PriceClose)
  OVER (PARTITION BY Key_Company ORDER BY Key_Company, Key_Date ROWS 20
PRECEDING) AS mAvg_21,
 AVG(PriceClose)
  OVER (PARTITION BY Key_Company ORDER BY Key_Company, Key_Date ROWS 4
PRECEDING) AS mAvg_5
 FROM Quotes
 WHERE
  Key_Company >= 100 AND Key_Company <= 1100 AND
  Key_Date BETWEEN
   TO_DATE('July 01, 2002', 'Month dd, YYYY', 'NLS_DATE_LANGUAGE= American')
   AND
   TO_DATE('January 01, 2003', 'Month dd, YYYY', 'NLS_DATE_LANGUAGE= American');

Table created.

SELECT Key_Company, Key_Date
 FROM
 (
  SELECT
   Key_Company AS Key_Company,
   Key_Date,
   mAvg_5-mAvg_21 AS diff,
   LAG(mAvg_5-mAvg_21) OVER (PARTITION BY Key_Company ORDER BY Key_Company,
Key_Date) AS pre_diff
  FROM tempAvg
 )
 WHERE
  pre_diff * diff <= 0
  AND
  NOT (pre_diff=0 and diff=0)
 ORDER  BY Key_Company, Key_Date;
```

a. Without indexes

OPERATION	OPTIONS	OBJECT_NAME
CREATE TABLE STATEMENT		
LOAD AS SELECT		
WINDOW	SORT	
TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	OPTIONS	OBJECT_NAME
CREATE TABLE STATEMENT		

LOAD AS SELECT		
WINDOW	BUFFER	
TABLE ACCESS	BY INDEX ROWID	QUOTES
INDEX	RANGE SCAN	QUOTES1

Query 8.

```
SELECT Q1.Key_Company, Q2.Key_Company, CORR(Q1.PriceClose, Q2.PriceClose) AS CorrCoef
 FROM Quotes Q1, Quotes Q2
 WHERE
  Q1.Key_Date=Q2.Key_Date
  AND Q1.Key_Company > 100 AND Q1.Key_Company <= 110
  AND Q2.Key_Company > 100 AND Q2.Key_Company <= 110
  AND Q1.Key_Date BETWEEN TO_DATE('January 1, 2000', 'Month dd, YYYY',
'NLS_DATE_LANGUAGE= American')
  AND TO_DATE('January 1, 2002', 'Month dd, YYYY', 'NLS_DATE_LANGUAGE= American')
GROUP BY Q1.Key_Company, Q2.Key_Company;
```

a. Without indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	GROUP BY	
MERGE JOIN		
SORT	JOIN	
TABLE ACCESS	FULL	QUOTES
SORT	JOIN	
TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	GROUP BY	
TABLE ACCESS	BY INDEX ROWID	QUOTES
NESTED LOOPS		
TABLE ACCESS	BY INDEX ROWID	QUOTES
INDEX	RANGE SCAN	QUOTES1
INDEX	RANGE SCAN	QUOTES1

Tick.

Query 1.

```
SELECT *
 FROM Quotes
 WHERE
 Key_Company > 100 AND Key_Company <= 200 AND
 Key_Date >= TO_DATE('01-January-2000') AND Key_Date < TO_DATE('01-January-2003');
```

a. Without indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
TABLE ACCESS	BY INDEX ROWID	QUOTES
NESTED LOOPS		
TABLE ACCESS	FULL	COMPANY
INDEX	RANGE SCAN	QUOTES1

b. Company

```
SELECT c.Company, q.*
 FROM Quotes q, Company c
 WHERE
 q.Key_Company = c.Key_Company AND
 q.Key_Company > 100 AND q.Key_Company <= 105 AND
 q.Key_Date >= TO_DATE('01-January-2000') AND q.Key_Date < TO_DATE('01-January-
2003');
```

OPERATION	**OPTIONS**	**OBJECT_NAME**
MERGE JOIN		
SORT	JOIN	
TABLE ACCESS	FULL	COMPANY
SORT	JOIN	
TABLE ACCESS	FULL	QUOTES

Query 2. The Quotes table was created with Volume as a "Long" datatype. And Oracle's use of the "Long" datatype isn't the same as IBM DB2 or Microsoft SQL Server's interpretation. In fact it is "up to 2 gigabytes, or 231 -1 bytes" and doesn't allow for aggregate math functions.[6] Unfortunately, at this time I was unwilling to rectify this [reinstalled Oracle twice and loaded the Quotes table too many times]. As such the following tick queries [2 through 5] aren't included in Oracle's analysis.

```
SELECT SUM(PriceClose*Volume)/SUM(Volume) VolPrice
 FROM Quotes
 WHERE
  Key_Date >= TO_DATE('01-January-2000') AND Key_Date < TO_DATE('01-January-2003');
SELECT SUM(PriceClose*Volume)/SUM(Volume) VolPrice
```

[6] Oracle 9i Limitations. http://www.iselfschooling.com/mc4articles/OracleLimitation.htm

Result:
ERROR at line 1:
ORA-00932: inconsistent datatypes: expected NUMBER got LONG

Query 6.

```
SELECT
 (PriceClose/((High + Low) / 2)) AS pSpread, Key_Company
 FROM Quotes
 WHERE (Key_Date = TO_DATE('31-January-2002'))
 GROUP BY Key_Company, PriceClose, High, Low
 ORDER BY pSpread DESC, Key_Company;
```

a. Without indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	ORDER BY	
SORT	GROUP BY	
TABLE ACCESS	FULL	QUOTES

b. With indexes

OPERATION	**OPTIONS**	**OBJECT_NAME**
SELECT STATEMENT		
SORT	ORDER BY	
SORT	GROUP BY	
TABLE ACCESS	FULL	QUOTES

7.c. Microsoft SQL Server 2000.

Historical.

Query 1. Simply looking at this query screams something is wrong. This query is unreadable and inefficient. However, at the time of this analysis, there wasn't another known method to accomplish this query's task.

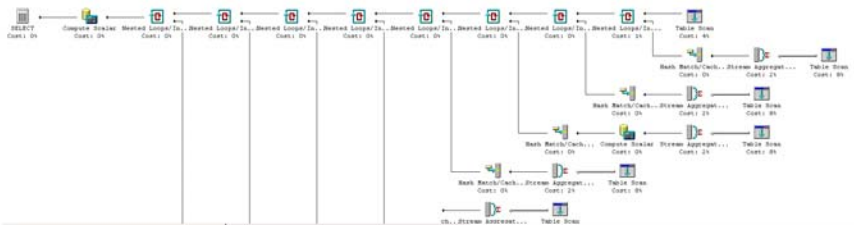
Query Type	Indexed	CPU	Reads	Writes	Duration[7]	Client Process ID
Historical						
Query 1. 10 Years.					1 Hour +	Cancelled
Query 1. 1 Month.					1 Hour +	Cancelled
Query 1. 10 Days		224937	9733812	0	228703	4188
Query 1. 10 Days.	Yes	2031	11548	0	2046	1508
Query 1. 10 Years.	Yes	962422	5649904	0	978766	1508

```
SELECT
 Q1.Key_Company,
 Q1.Key_Date,
 (
  SELECT MAX(PriceClose)
   FROM [Preft].[dbo].[Quotes] AS QWeekMax WHERE
    QWeekMax.Key_Company = Q1.Key_Company AND
    DateDiff(dd, QWeekMax.Key_Date, Q1.Key_Date) Between 0 AND 5
 ) AS weekMax,
 (
  SELECT  MIN(PriceClose)
   FROM [Preft].[dbo].[Quotes] AS QWeekMin WHERE
    QWeekMin.Key_Company = Q1.Key_Company AND
    DateDiff(dd, QWeekMin.Key_Date, Q1.Key_Date) Between 0 AND 5
 ) AS weekMin,
 (
  SELECT AVG(PriceClose)
   FROM [Preft].[dbo].[Quotes] AS QWeekAvg WHERE
    QWeekAvg.Key_Company = Q1.Key_Company AND
    DateDiff(dd, QWeekAvg.Key_Date, Q1.Key_Date) Between 0 AND 5
 ) AS weekAvg,
 (
  SELECT MAX(PriceClose)
   FROM [Preft].[dbo].[Quotes] AS QmonthMax WHERE
    QmonthMax.Key_Company = Q1.Key_Company AND
    DateDiff(dd, QmonthMax.Key_Date, Q1.Key_Date) Between 0 AND 20
 ) AS monthMax,
 (
  SELECT  MIN(PriceClose)
   FROM [Preft].[dbo].[Quotes] AS QmonthMin WHERE
    QmonthMin.Key_Company = Q1.Key_Company AND
    DateDiff(dd, QmonthMin.Key_Date, Q1.Key_Date) Between 0 AND 20
 ) AS monthMin,
 (
  SELECT AVG(PriceClose)
   FROM [Preft].[dbo].[Quotes] AS QmonthAvg WHERE
    QmonthAvg.Key_Company = Q1.Key_Company AND
    DateDiff(dd, QmonthAvg.Key_Date, Q1.Key_Date) Between 0 AND 20
 ) AS monthAvg,
 (
```

[7] Duration is in thousandth of a second. For example, Query 3 [1033] is 1.033 seconds.

```
 SELECT MAX(PriceClose)
  FROM [Preft].[dbo].[Quotes] AS QyearMax WHERE
   QyearMax.Key_Company = Q1.Key_Company AND
   DateDiff(dd, QyearMax.Key_Date, Q1.Key_Date) Between 0 AND 260
) AS yearMax,
(
 SELECT  MIN(PriceClose)
  FROM [Preft].[dbo].[Quotes] AS QyearMin WHERE
   QyearMin.Key_Company = Q1.Key_Company AND
   DateDiff(dd, QyearMin.Key_Date, Q1.Key_Date) Between 0 AND 260
) AS yearMin,
(
 SELECT AVG(PriceClose)
  FROM [Preft].[dbo].[Quotes] AS QyearAvg WHERE
   QyearAvg.Key_Company = Q1.Key_Company AND
   DateDiff(dd, QyearAvg.Key_Date, Q1.Key_Date) Between 0 AND 260
) AS yearAvg
FROM [Preft].[dbo].[Quotes] AS Q1
WHERE
 Key_Company > 100 AND Key_Company <= 110 AND
 Key_Date >= '01/01/1990'
 AND
 Key_Date <= '01/01/2000';
```

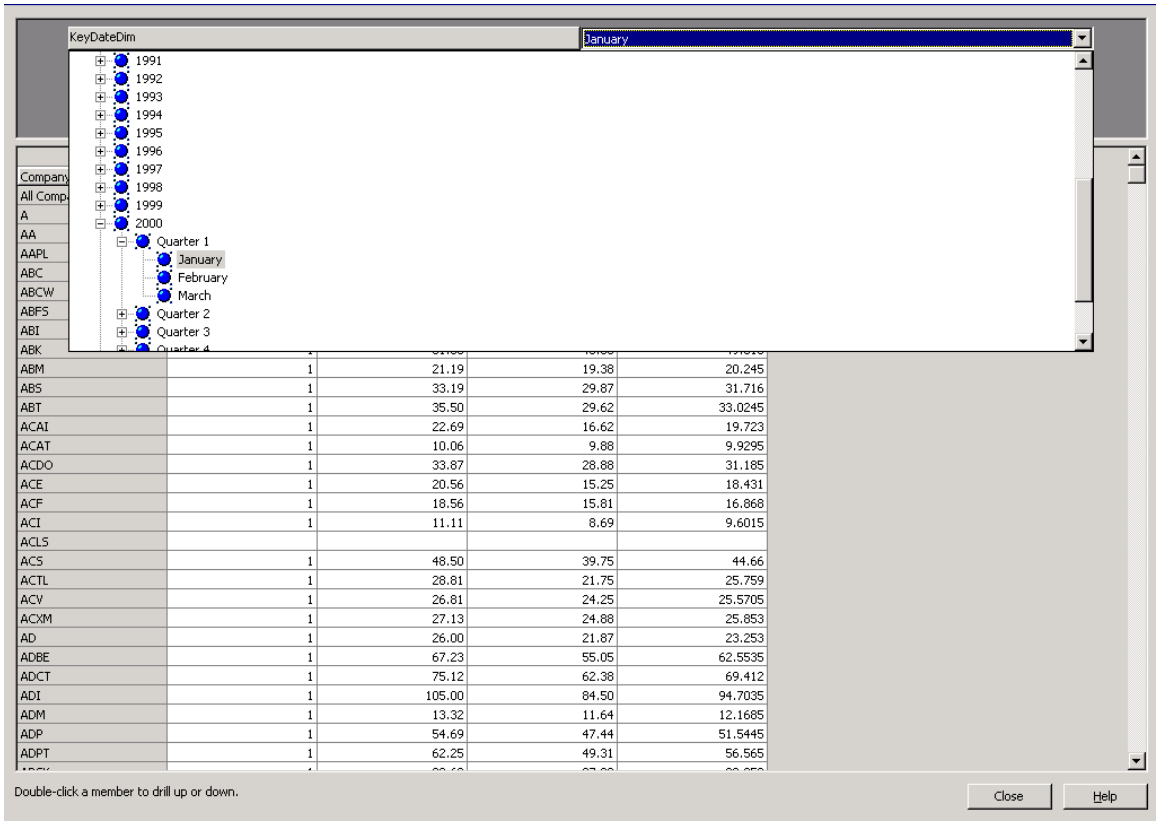
The diagram below is the execution plan for this query. Although the diagram is small and unreadable, it shows the unnecessary complexity of this query.

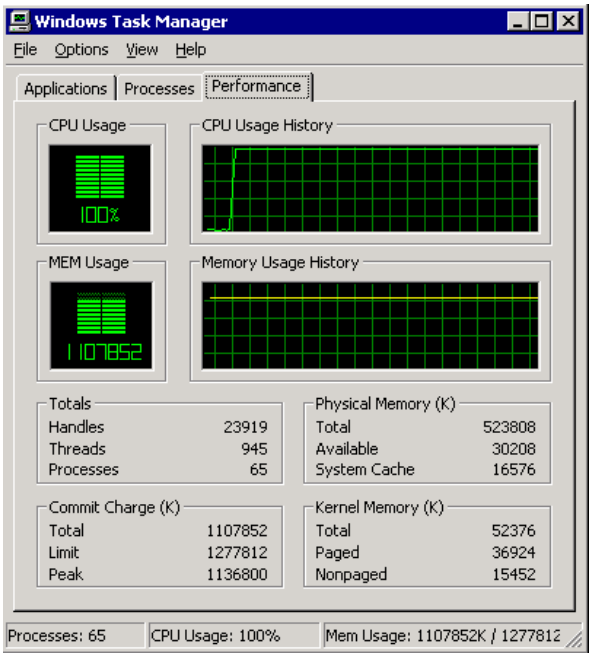


Hist.Query.1.MSFT.Plan.gif

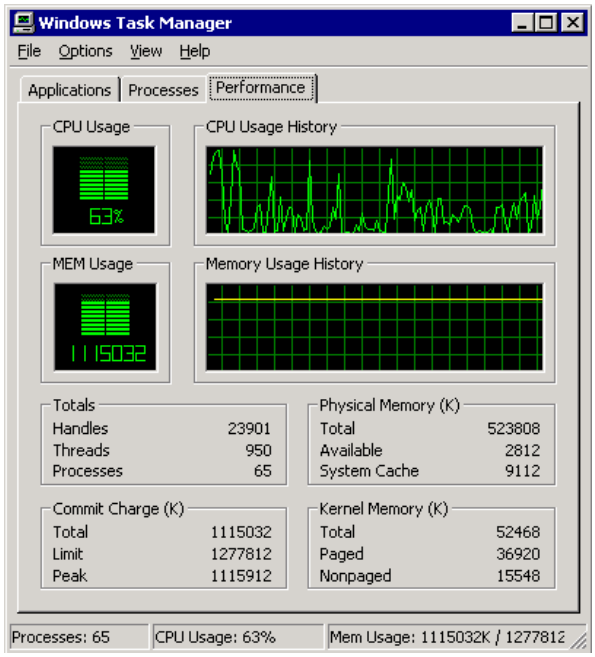
A Better Method: Analysis Package. Microsoft SQL Server 2000's dialect of SQL is unable to recognize some OLAP functionality used in similar queries with the other two commercial databases. However, there are other means such as Microsoft's analysis software. The following cube processed all the Quotes data for MAX, MIN and AVG mathematical measures against two dimensions:
1. Time [i.e. the granularity is Year, Quarter, Month]
2. The equity [i.e. IBM, ORCL, MSFT]

The following diagram displays a list of equities [alphabetically ordered] in the month of January 2000.

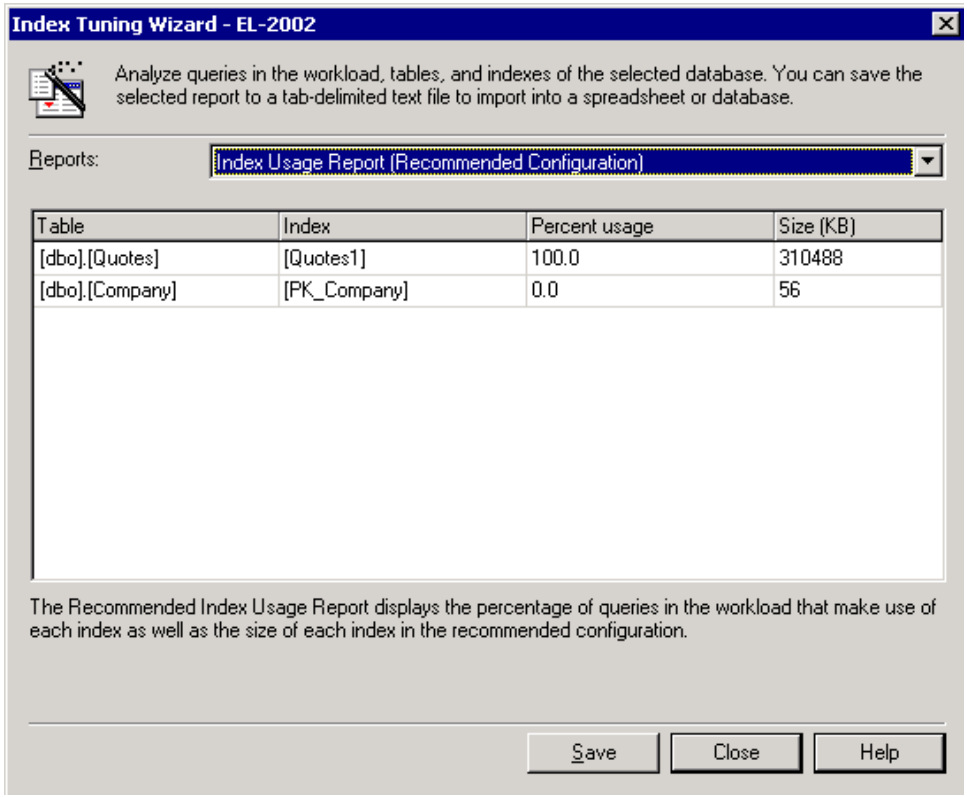




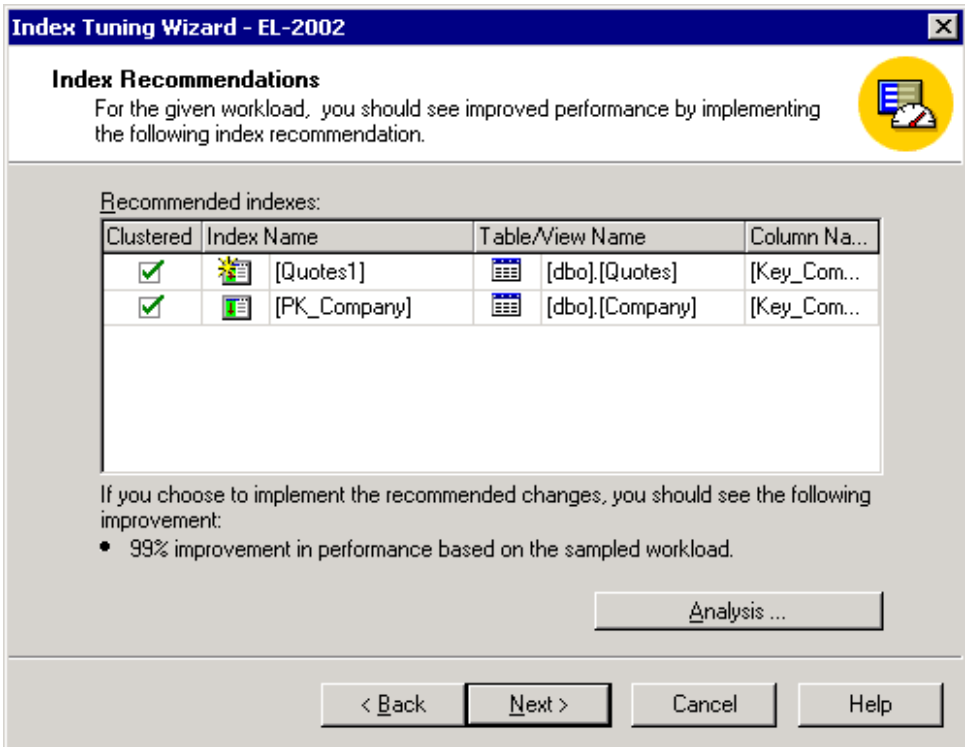
Hist.Query.1.Days.10.MSFT.Task.gif



Hist.Query.1.Years.10.MSFT.Task.gif



Hist.Query.1.Index.MSFT.Wizard.Analysis.gif



Hist.Query.1.Index.MSFT.Wizard.gif

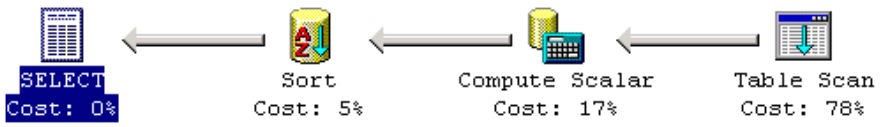
Query 3.

a. Execution without index: **> 2** second. Execution with index: **< 1** second.

```
SELECT (High-Low) AS DailyDifferential, PriceClose, Key_Company
FROM Preft.dbo.QUOTES
WHERE
   Key_Date >= '01/01/2002'
   AND
   Key_Date <= '01/31/2002'
   AND KEY_COMPANY > 100 AND Key_Company <= 1100
 ORDER BY Key_Company;
```

```
SELECT (High-Low) AS DailyDifferential, PriceClose, Key_Company
FROM Preft.dbo.QUOTES
WHERE
   Key_Date >= '01/01/2002'
   AND
   Key_Date <= '01/31/2002'
   AND KEY_COMPANY > 100 AND Key_Company <= 1100
 ORDER BY Key_Company;
```

```
Query 1: Query cost (relative to the batch): 100.00%
Query text: SELECT (High-Low) AS DailyDifferential, PriceClose, Key
```



Hist.Query.3.MSFT.Plan.gif

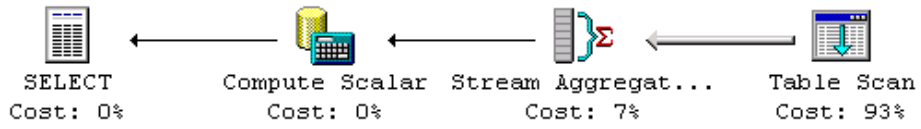
Query 4.

a. Execution without index: **> 1.5** seconds. Execution with index: **< 1** seconds.

```
SELECT AVG(PriceClose) AS SandP1500Index
 FROM
 [Preft].[dbo].[Quotes]
 WHERE Key_Date ='01/31/2002';
```

```
SELECT AVG(PriceClose) AS SandP1500Index
 FROM
 [Preft].[dbo].[Quotes]
 WHERE Key_Date ='01/31/2002';
```

```
Query 1: Query cost (relative to the batch): 100.00%
Query text: SELECT AVG(PriceClose) AS SandP1500Index    FROM     [
```

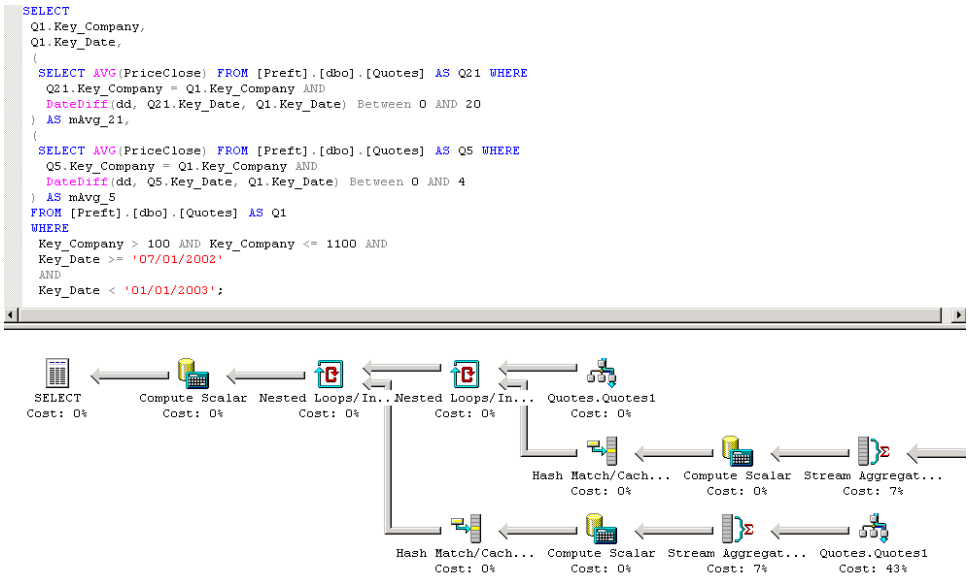


Hist.Query.4.MSFT.Plan.gif

Query 5.

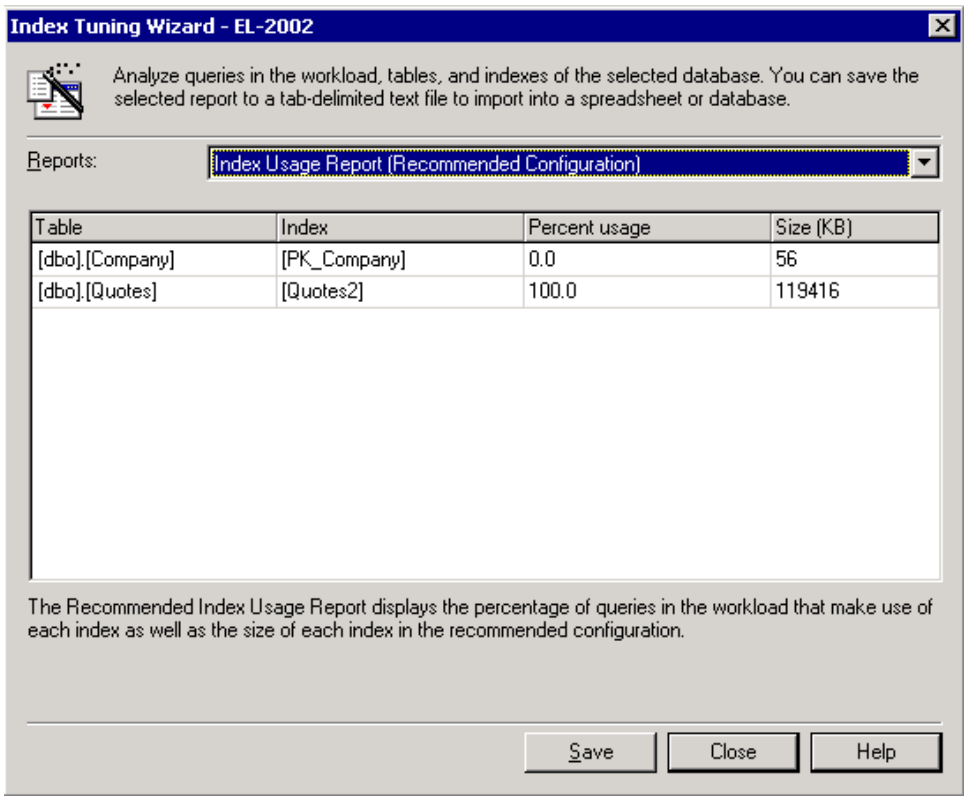
Query Type	Indexed	CPU	Reads	Writes	Duration[8]	Client Process ID
Query 5. 10 Stocks.		999328	3.3E+07	0	1 Hour +	Cancelled
Query 5. 1 Stock.		230281	9887460	0	234046	4188
Query 5. 10 Stocks.	Yes	0	190	0	126	1508
Query 5. 1000 Stocks.	Yes	1287313	8913321	0	1322016	1508

```
SELECT
 Q1.Key_Company,
 Q1.Key_Date,
 (
  SELECT AVG(PriceClose) FROM [Preft].[dbo].[Quotes] AS Q21 WHERE
   Q21.Key_Company = Q1.Key_Company AND
   DateDiff(dd, Q21.Key_Date, Q1.Key_Date) Between 0 AND 20
 ) AS mAvg_21,
 (
  SELECT AVG(PriceClose) FROM [Preft].[dbo].[Quotes] AS Q5 WHERE
   Q5.Key_Company = Q1.Key_Company AND
   DateDiff(dd, Q5.Key_Date, Q1.Key_Date) Between 0 AND 4
 ) AS mAvg_5
FROM [Preft].[dbo].[Quotes] AS Q1
WHERE
 Key_Company > 100 AND Key_Company <= 1100 AND
 Key_Date >= '07/01/2002'
 AND
 Key_Date < '01/01/2003';
```

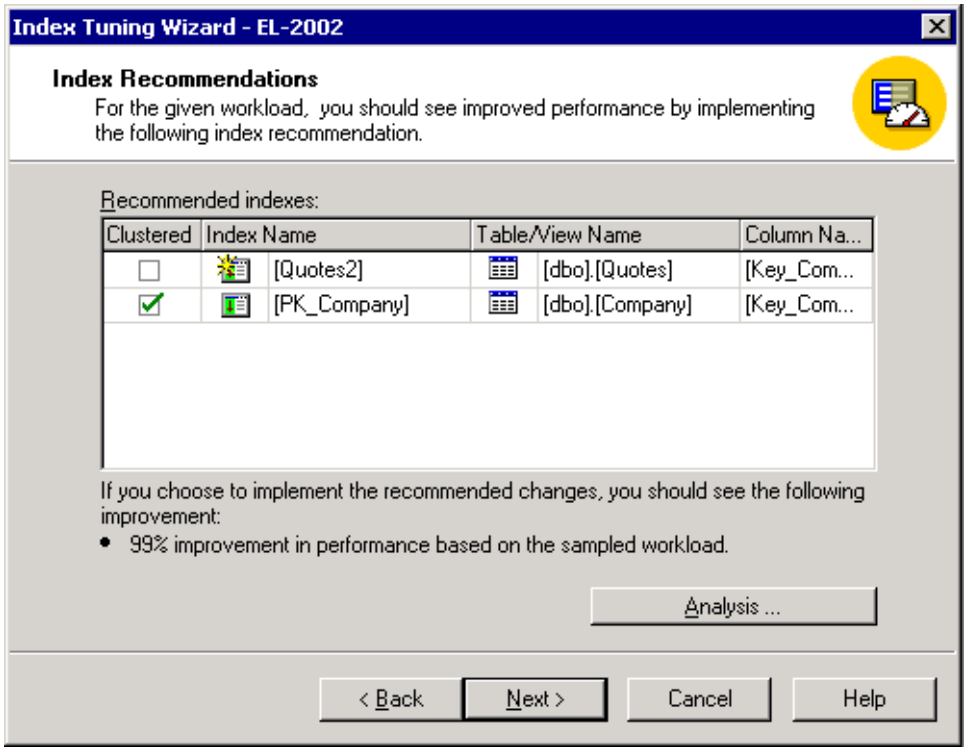


Hist.Query.5.Index.MSFT.Plan.gif

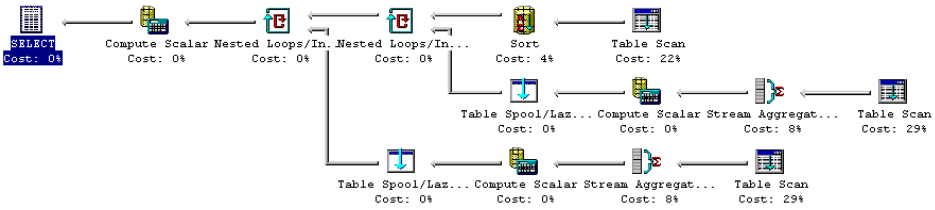
[8] Duration is in thousandth of a second. For example, Query 3 [1033] is 1.033 seconds.



Hist.Query.5.Index.MSFT.Wizard.Analysis.gif



Hist.Query.5.Index.MSFT.Wizard.gif



Hist.Query.5.MSFT.Plan.gif

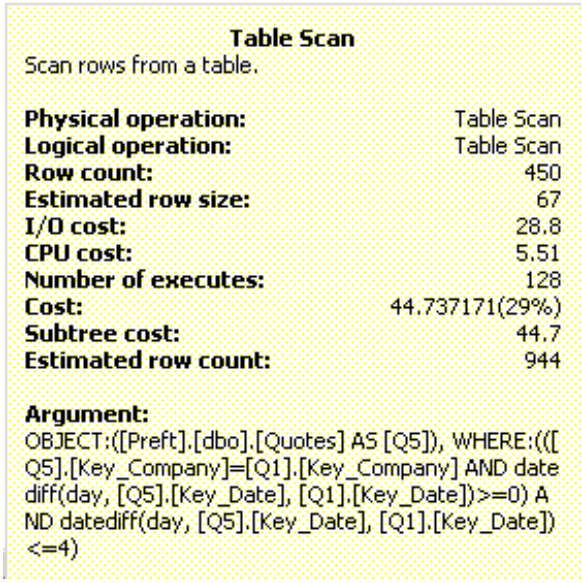


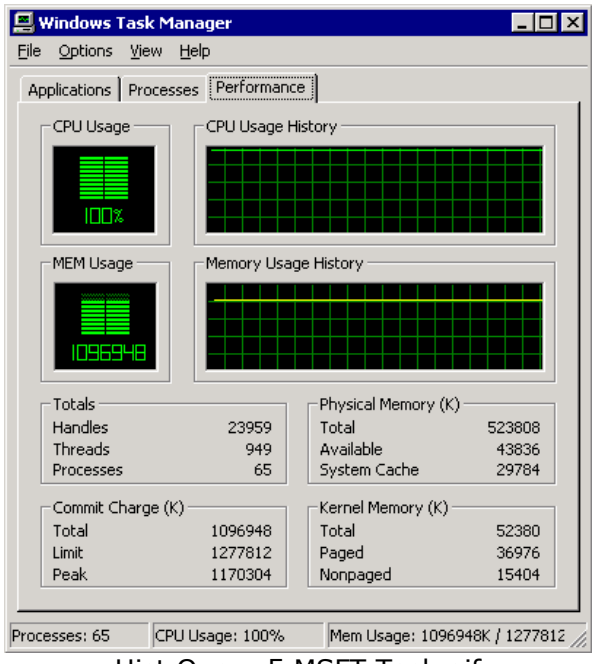
Table Scan

Scan rows from a table.

Physical operation:	Table Scan
Logical operation:	Table Scan
Row count:	450
Estimated row size:	67
I/O cost:	28.8
CPU cost:	5.51
Number of executes:	128
Cost:	44.737171(29%)
Subtree cost:	44.7
Estimated row count:	944

Argument:
OBJECT:([Preft].[dbo].[Quotes] AS [Q5]), WHERE:(([Q5].[Key_Company]=[Q1].[Key_Company] AND datediff(day, [Q5].[Key_Date], [Q1].[Key_Date])>=0) AND datediff(day, [Q5].[Key_Date], [Q1].[Key_Date])<=4)

Hist.Query.5.MSFT.Plan.Info.gif



Hist.Query.5.MSFT.Task.gif

Summary:

Query Type	Indexed	CPU	Reads	Writes	Duration[9]	Client Process ID
Historical						
Query 1. 10 Years.					1 Hour +	Cancelled
Query 1. 1 Month.					1 Hour +	Cancelled
Query 1. 10 Days		224937	9733812	0	228703	4188
Query 1. 10 Days.	Yes	2031	11548	0	2046	1508
Query 1. 10 Years.	Yes	962422	5649904	0	978766	1508
Query 3. Index.		375	49690	0	720	4240
Query 4. Index.		750	75642	0	12173	4240
Query 3.		625	38482	0	1033	4188
Query 4.		562	38516	0	576	4188
Query 5. 10 Stocks.		999328	3.3E+07	0	1 Hour +	Cancelled
Query 5. 1 Stock.		230281	9887460	0	234046	4188
Query 5. 10 Stocks.	Yes	0	190	0	126	1508
Query 5. 1000 Stocks.	Yes	1287313	8913321	0	1322016	1508

[9] Duration is in thousandth of a second. For example, Query 3 [1033] is 1.033 seconds.

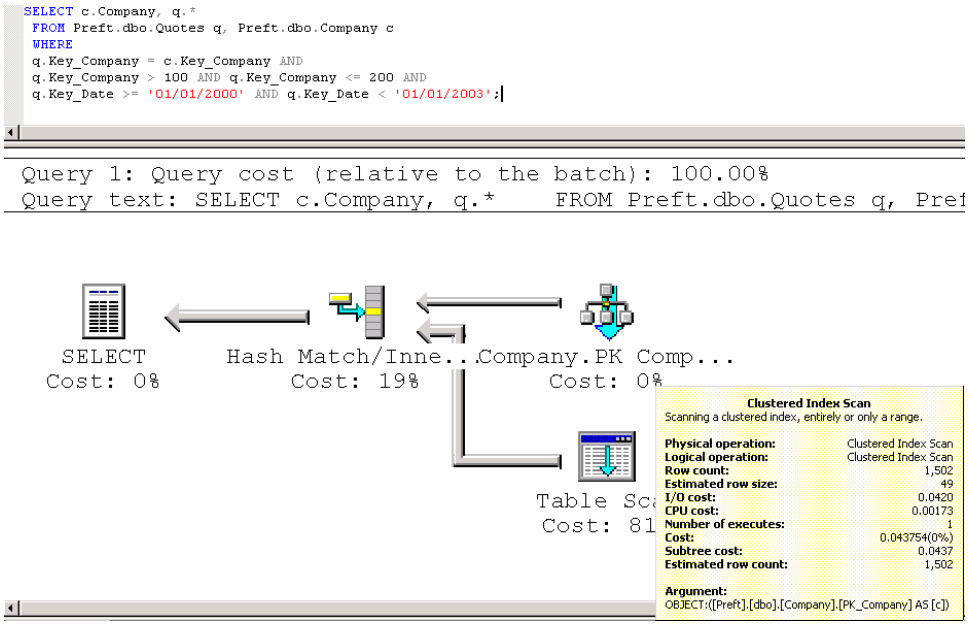
Tick.

Query 1. Query 'b' does a join with the Company table and returns the ticker symbol for expressed readability.
a. Execution: **< 6** seconds.

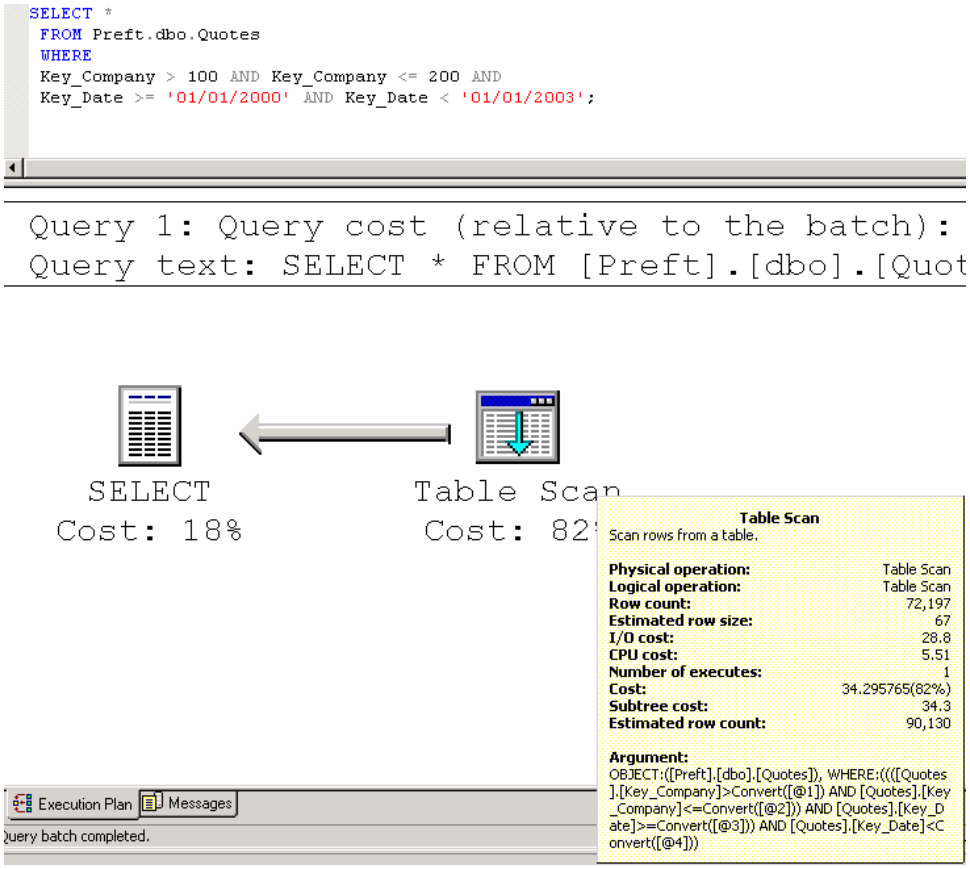
```
SELECT *
 FROM Preft.dbo.Quotes
 WHERE
 Key_Company > 100 AND Key_Company <= 200 AND
 Key_Date >= '01/01/2000' AND Key_Date < '01/01/2003';
```

b. Execution: **> 6** seconds.

```
SELECT c.Company, q.*
 FROM Preft.dbo.Quotes q, Preft.dbo.Company c
 WHERE
 q.Key_Company = c.Key_Company AND
 q.Key_Company > 100 AND q.Key_Company <= 200 AND
 q.Key_Date >= '01/01/2000' AND q.Key_Date < '01/01/2003';
```



Tick.Query.1.Company.MSFT.Plan.gif

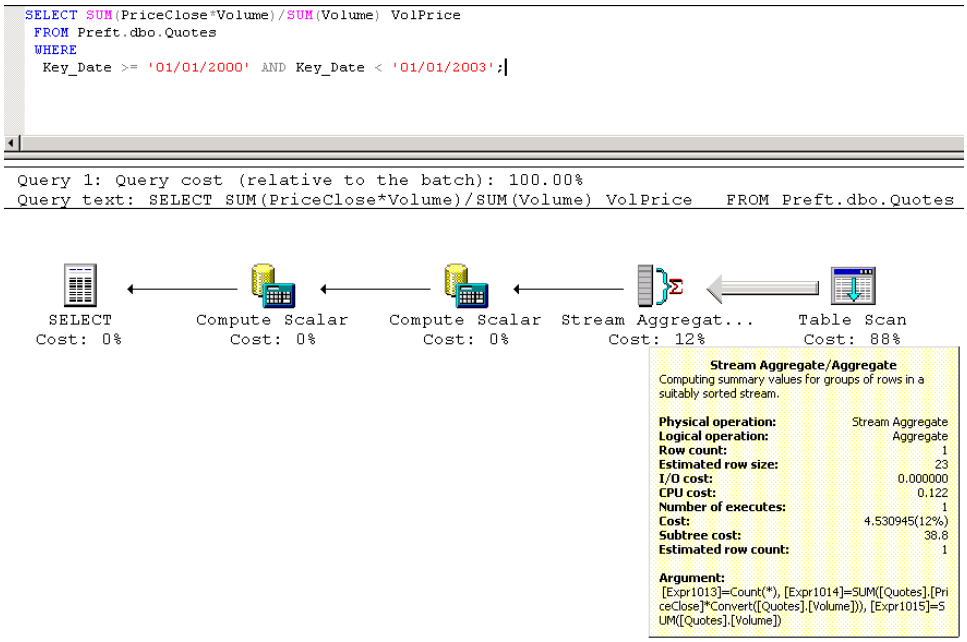


Tick.Query.1.MSFT.Plan.gif

Query 2.

a. Execution: **< 2** seconds.

```
SELECT SUM(PriceClose*Volume)/SUM(Volume) VolPrice
 FROM Preft.dbo.Quotes
 WHERE
  Key_Date >= '01/01/2000' AND Key_Date < '01/01/2003';
```

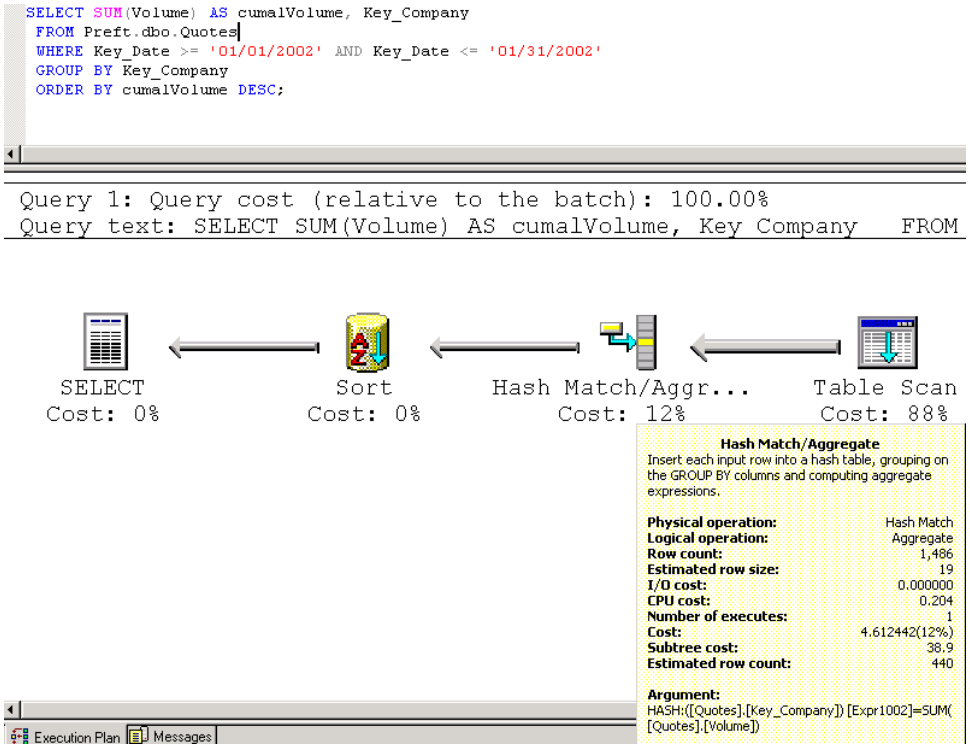


Tick.Query.2.MSFT.Plan.gif

Query 4.

a. Execution: **< 1** second.

```
SELECT SUM(Volume) AS cumalVolume, Key_Company
 FROM Preft.dbo.Quotes
 WHERE Key_Date >= '01/01/2002' AND Key_Date <= '01/31/2002'
 GROUP BY Key_Company
 ORDER BY cumalVolume DESC;
```

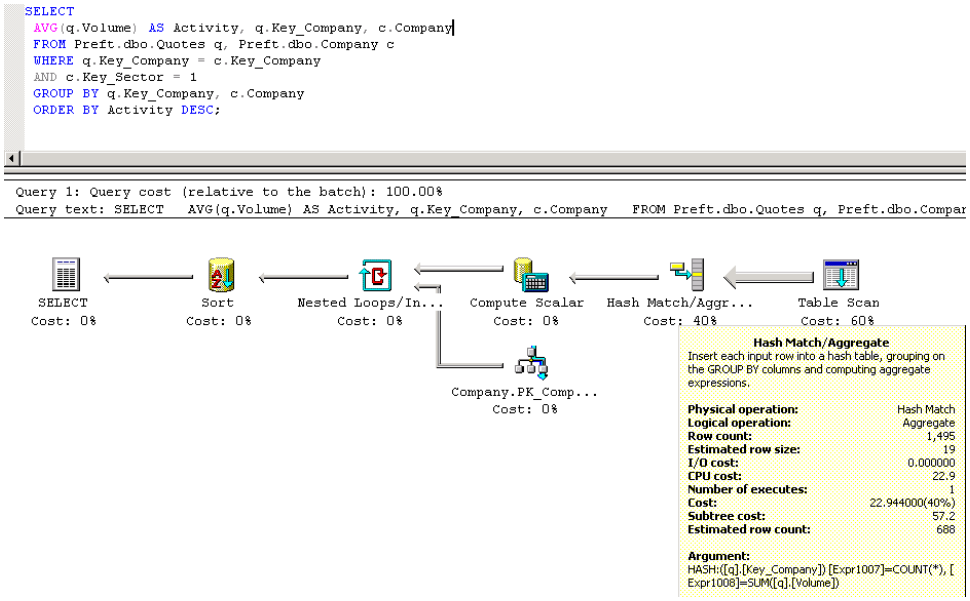


Tick.Query.4.MSFT.Plan.gif

Query 5.

a. Execution: **> 6** seconds.

```
SELECT
 AVG(q.Volume) AS Activity, q.Key_Company, c.Company
 FROM Preft.dbo.Quotes q, Preft.dbo.Company c
 WHERE q.Key_Company = c.Key_Company
 AND c.Key_Sector = 1
 GROUP BY q.Key_Company, c.Company
 ORDER BY Activity DESC;
```

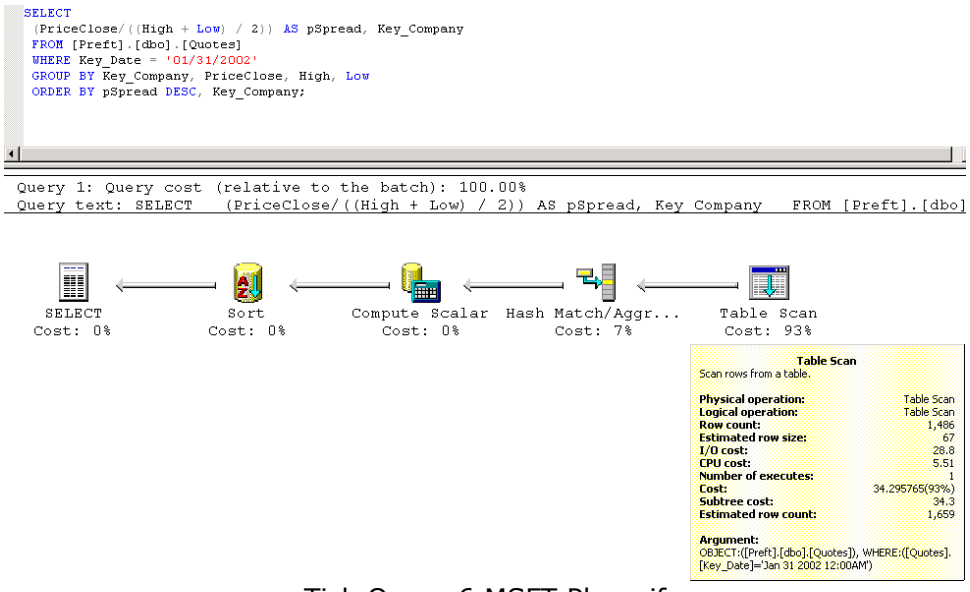


Tick.Query.5.Company.MSFT.Plan.gif

Query 6.

a. Execution: **< 1** second.

```
SELECT
 (PriceClose/((High + Low) / 2)) AS pSpread, Key_Company
 FROM [Preft].[dbo].[Quotes]
 WHERE Key_Date = '01/31/2002'
 GROUP BY Key_Company, PriceClose, High, Low
 ORDER BY pSpread DESC, Key_Company;
```



Tick.Query.6.MSFT.Plan.gif

Tick Summary:

Query Type	Indexed	CPU	Reads	Writes	Duration[10]	Client Process ID
Tick						
Query 1. Company.		890	38580	0	6203	4188
Query 1.		812	38534	0	5953	4188
Query 2.		2688	38542	0	2733	4188
Query 4.		594	38534	0	670	4188
Query 5.		6484	41579	0	6530	4188
Query 6.		546	38562	0	576	4188

[10] Duration is in thousandth of a second. For example, Query 1 [5953] is 5.983 seconds.

8. Analysis and Conclusion

The evidence in this paper and the subjective and intangible preference established with the use of each database leads me to believe that IBM DB2 has the most to offer for our financial analysis needs. That is IBM DB2 may be the better database system of the three.

It offers the greatest arsenal of tools for tuning while maintaining a friendly user interface and ease of use.

Microsoft SQL Server is disadvantaged by their SQL dialect which appears to be the most limited, especially with the use of mathematical and statistical functions. It may be because of this qualitative hurdle that it is the worst performing database system out of the three. Although Microsoft SQL Server's SQL dialect may be a burden for financial analysts, its management is by far the most intuitive and friendliest to use. And on that note is Oracle's downfall. Oracle is just an inconvenience to use. I am in no way a professional or even close to a certified Oracle user, but I see myself as a savvy IT individual and found Oracle very frustrating.

However, if you're comfortable with oracle then oracle may be the best database performance wise as these experiments suggest. I personally couldn't get over the inconvenience hump with Oracle.

SQL Server has the best interface of the three and their performance and monitoring package [i.e. profiler] is very useful. Depending on the needs of the database system, it may make up for the limited SQL dialect. That's what I've seen first hand so far.

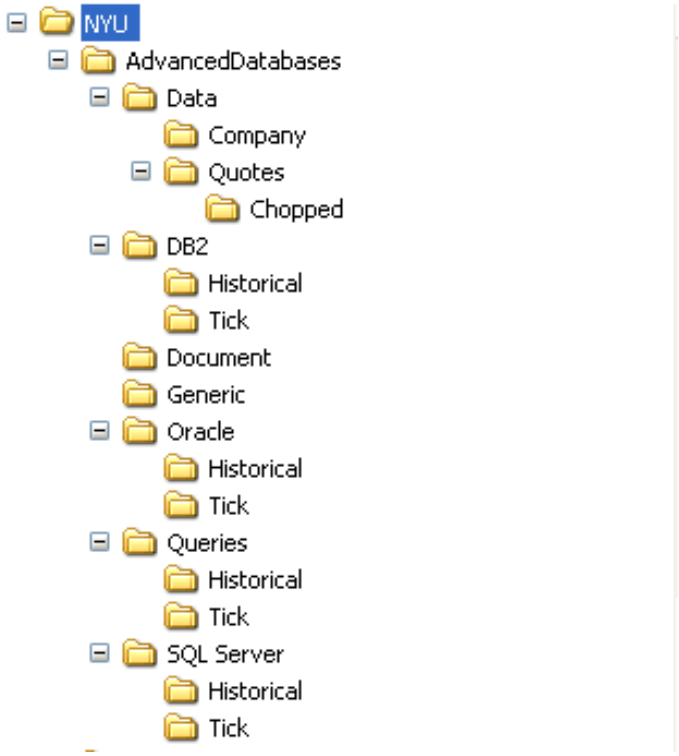
For the financial analysis Preft application and research, I'll continue to use SQL Server because it is convenient to integrate with my development environment [.NET]. But if I figure out how to tie DB2 conveniently into .NET then I would strongly consider changing database systems, or at least supporting both.

A cross comparison of these systems can't be made fairly because there is a large gap that needs to be filled before there can be an accurate comparison of performance. Right now, it's a qualitative issue and IBM DB2 mixes its SQL power and flexibility with a convenient user interface. The performance or quantitative measures come secondary to the quality of the SQL that can be expressed.

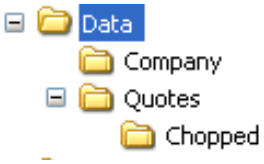
SQL Server needs OLAP functionality built into the SQL dialect so that statistical functions such as truncate, prev, rank, and partition can be built into SQL queries. It may very well be my lack of understanding of SQL Server's SQL dialect but as far as I know, they separate their analysis package from their SQL dialect. Assuming this SQL distinction is correct among these database systems, I can't see financial institutions using SQL Server seriously for modeling needs. For moving averages, ranking and other statistical needs, SQL Server's native language simply can't compare to IBM DB2 or Oracle.

At the end each system has their advantages. For my financial analysis needs IBM DB2 offers the best mix of everything.

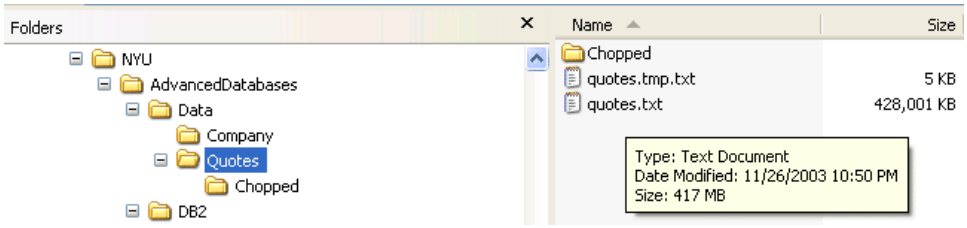
9. About / Readme. Feel free to take a look around the companion DVD and use the available data for more intimate financial research or added pleasure. Everything in this document and much [**much**] more is available [on the companion DVD]. If you have any questions or comments, please feel free to email me at: nyu@radovici.com



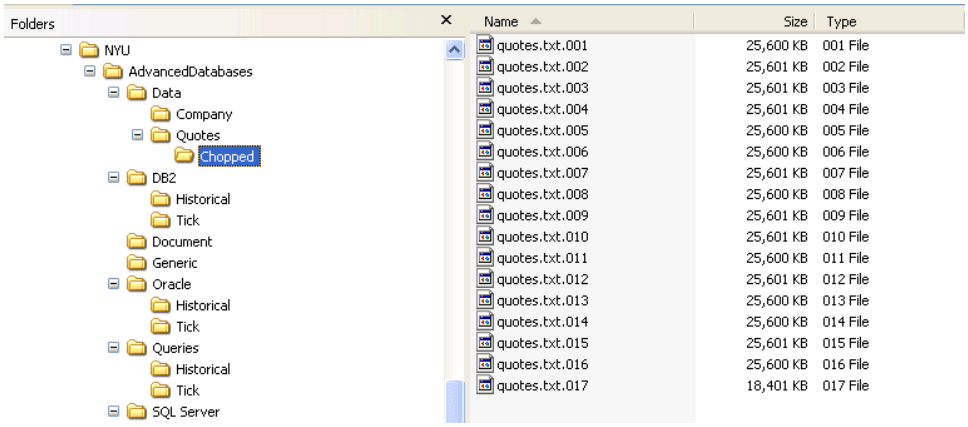
Content and Directory Structure [available on the companion DVD]



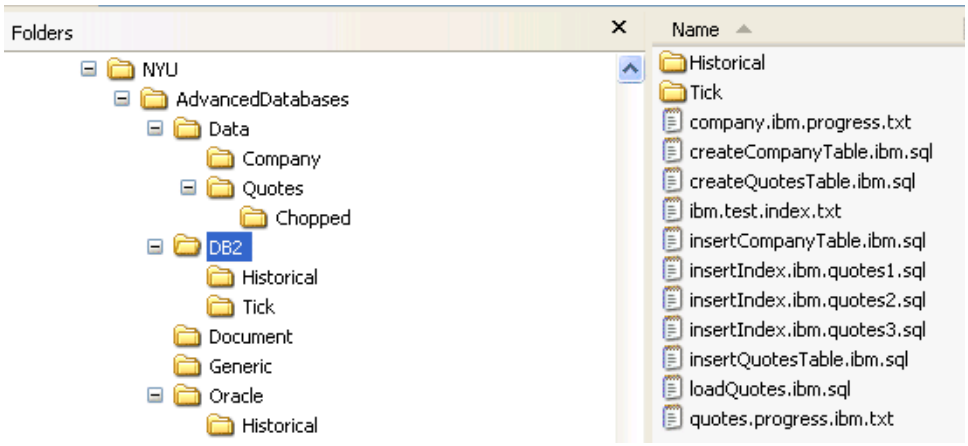
Data



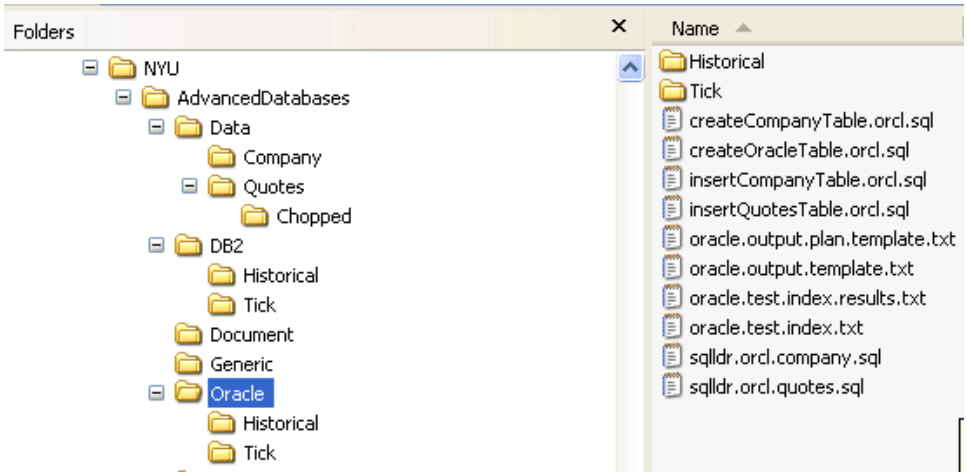
Quotes data.



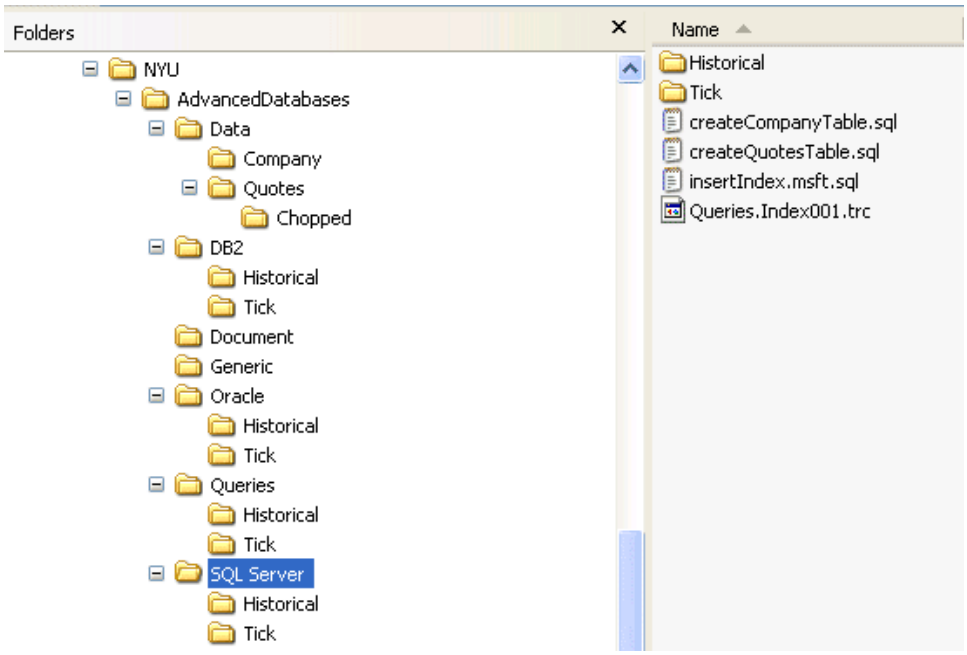
Chopped data to simply usage and distribution.



IBM DB2 Content



Oracle Content



Microsoft SQL Server Content

10. References

Kaippallimalil J. Jacob and Dennis Shasha. FinTime. A financial time series benchmark
http://www.cs.nyu.edu/cs/faculty/shasha/fintime.d/bench.html

Pareena Parikh, Evaluation of DB2 Query Optimizer and SQL99 Implementation
http://www.cs.nyu.edu/cs/faculty/shasha/fintime.d/DB2project.htm

Oracle FinTime Study. Final Report
http://www.cs.nyu.edu/cs/faculty/shasha/fintime.d/Oracleproject.htm

"SQL Server 2000 versus Oracle 9i"
http://www.mssqlcity.com/Articles/Compare/sql_server_vs_oracle.htm

"SQL Function Reference: Oracle vs. SQL Server"
http://www.webucator.com/resources/sql/reference.html

"Correlation, Analysis Services"
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/olapdmad/agmdxfunctions_2gq6.asp

"Oracle9i Limitation"
http://www.iselfschooling.com/mc4articles/OracleLimitation.htm

"Use Oracle's Explain Plan to Tune Your Queries"
http://www.evolt.org/article/Use_Oracle_s_Explain_Plan_to_Tune_Your_Queries/17/2986/

"How does one trace (and explain) SQL statements from SQL*Plus?" [Oracle]
http://www.orafaq.com/faqplus.htm#TRACE

"Creating an Access Plan Graph" [DB2]
http://www.seas.ucla.edu/db2/db2help/index.htm#tve02

Witten, Ian H. and Eibe Frank. (1999) **Data Mining Practical Machine Learning Tools and Techniques.** 146

Murphy, John. (1999) **Technical Analysis of the Financial Markets.**