# BL03 - Mastering SQL Performance using Visual Explain Advanced Workshop

International Technical Support Organization

February 2001

# Contents

# Lab 1. Advanced Keyed Access analysis

## Introduction

This lab shows you how Visual Explain presents the Advanced Keyed Access Methods - Parallel Keyed Access and Index-From-Index Access method.

## Objectives

This lab teaches you how to:

1. Interpret Visual Explain graphs and attributes for Advanced Keyed Access Methods
2. Compare Visual Explain with debug messages detailed in job logs

## Lab Prerequisites

Before you begin this lab, make sure you are familiar with, and have, the following requirements:

- IBM iSeries server with V4R5 with latest cumulative fix package and the latest Database Fix Package
- Operations Navigator V4R5M0 with latest Service Pack
- Operations Navigator and Visual Explain interface

You should also have completed the Basic Course of this lab.

## Lab Environment

Query runtime depends on several factors, for example, the number of other jobs running on the system and their impact on CPU, DASD, and memory utilization.

**Disclaimer**

This workshop describes query implementations observed in a laboratory environment. Depending on the hardware configuration of your machine and its current workload, you may see a different implementation plan for your query. While creating an access plan for a query, the Query Optimizer takes into account many factors, which are machine specific.

The results contained in this publication have not been submitted to any formal IBM test and are distributed *as is.*

## Time required

The time required to efficiently complete this lab is 20 minutes.

## Task 1: Setting up the environment

In this exercise, you set up an environment required to enable the use of keyed access methods.

__ 1. If the SQL Script Center is not already active, launch it from the Database component of Operations Navigator. Use the worksheet for the user ID and password.

__ 2. Verify the existence of ITEM_IDX1 and ITEM_IDX2 in your library. If they do not exist, open the x:\VisualExplain\SQLScripts\Advanced\BuildIndexes.SQL script file, and run all the statements. Verify that no errors occurred.

__ 3. Open the x\VisualExplain\SQLScripts\Advanced\Advanced Lab1.SQL script file, which will be used for this lab.

__ 4. Run the statement to enable parallel processing, and verify that the statement completed successfully:

```
CL: CHGQRYA DEGREE(*OPTIMIZE);
```

## Task 2: Parallel Keyed access method

In this exercise, you see how parallelism can aid keyed access methods and how Visual Explain details these methods.

__ 1. Update the SELECT statement in the Run SQL Script window so that the statement selects all columns from the ITEM_FACT table where *month* is 3, 5 or 7 and *partkey* is between 99 and 1234.

```
SELECT *
FROM item_fact
WHERE month IN (3, 5, 7)
AND partkey BETWEEN 99 AND 1234
OPTIMIZE FOR ALL ROWS;
```

__ 2. Select the **Run and Explain** icon from the toolbar to execute the statement.

__ 3. Verify that the graph you received resembles the example shown in Figure 1.


*Figure 1. Parallel keyed access*

__ 4. Analyze the query implementation by reviewing the Visual Explain window. Use the *flyover* or *attributes and values panel* to interpret the graph, and complete Table 1.

*Table 1. Parallel key access questions*

| Question | Answer |
|---|---|
| What access methods were used? | |
| Which indexes were optimized (also include the reason code for used or not used)? | |
| What index was used to access the table? | |
| If an index was *not* used, what was the reason for not using it? | |
| What parallel degree was requested? | |

__ 5. Close the Visual Explain window. A window appears asking if you wish to save this data as a performace monitor. Click the **No** button.

## Task 3: Index-From-Index access method

This exercise shows you how an Index-From-Index access method is explained.

__ 1. Run the statement to disable parallel processing and verify that the statement completed sucessfully:

```
CL: CHGQRYA DEGREE(*NONE);
```

__ 2. Update the SELECT statement in the Run SQL Script window so that the local selection for *year* is `2000`, *month* is `10` and *returnflag* is `R`.

```
SELECT  a.year,a.month,a.returnflag,a.quantity,a.revenue_wo_tax
FROM    item_fact a
WHERE   a.year = 2000
AND     a.month = 10
AND     a.returnflag = 'R'
GROUP BY a.year,a.month,a.returnflag,a.quantity,a.revenue_wo_tax
OPTIMIZE FOR ALL ROWS;
```

__ 3. Click the **Visual Explain Only** icon to analyze the statement.

__ 4. Verify that the graph you received resembles the example shown in Figure 2.


*Figure 2. Index-From-Index*

__ 5. Analyze the query optimization by reviewing the Visual Explain window and complete Table 2 below:

*Table 2. Index-From-Index questions*

| Question | Answer |
|---|---|
| What access methods were used? | |
| Was a Temporary Index created? | |
| Why was the Temporary Index created? | |
| Was the index created from an existing index or from a table? | |
| Was the Temporary Index created from a SPARSE index? | |
| What were the key columns used for the Temporary Index creation? | |

__ 6. Close the Visual Explain window. A window appears asking if you wish to save this data as a performace monitor. Click the **No** button.

## Task 4: Comparing Visual Explain and job log messages for analysis (Optional)

In this exercise, you compare the results of Visual Explain with detailed debug messages captured into a job log for the Index-From-index access method.

__ 1. Activate the **Include Debug Messages** option, and launch the Job Log viewer.

__ 2. Update the statement so that re-optimization occurs(by changing SELECT for sELECT) and then Run the statement.

__ 3. Refresh the Job Log viewer.

Figure 3 shows the job log and a copy of the Visual Explain graph with the orientation set to “top”.

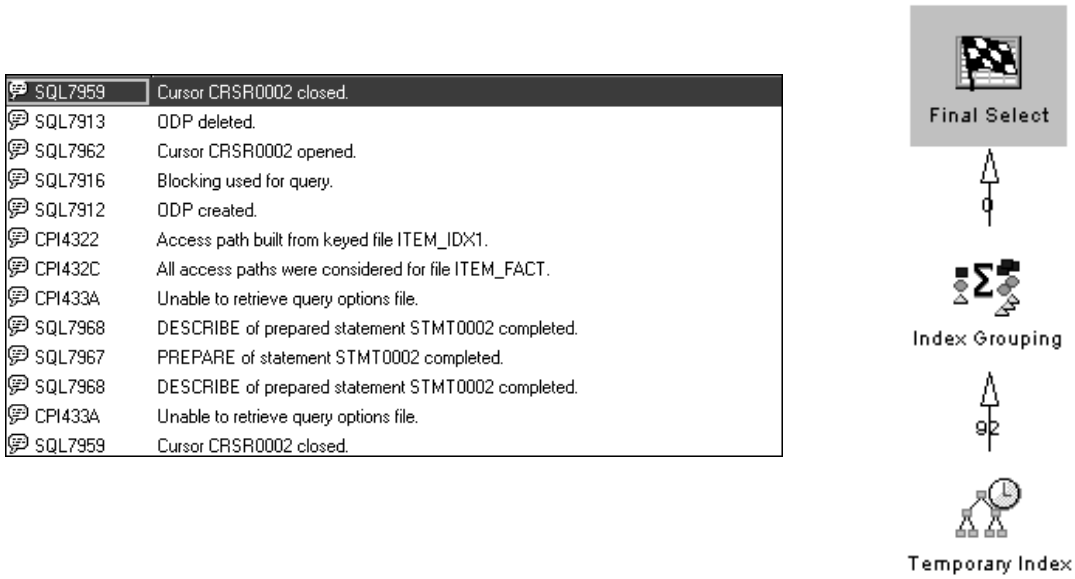



*Figure 3. Index-From-Index: Job log and Visual Explain graph*

__ 4. Analyze the job log.

__ 5. Close the Job Log viewer and the Run SQL Script window.

This completes this lab.

# Lab 2. Analyzing join methods

### Introduction

This lab shows you how Visual Explain is used to analyze join methods.

### Objectives

This lab teaches you how to:

1. Interpret Visual Explain graphs and attributes for join methods
2. Compare Visual Explain results with job log messages

### Lab environment

Ensure that all the indexes created during the previous labs exist.

### Time required

The time required to efficiently complete this lab is 30 minutes.

## Task 1: Setting up the environment

This exercise prepares the query environment for the join method by utilizing the QAQQINI options file created in your library to activate the following preferences:

- Parallelism
- Debug

__ 1. Open the x:\Visual Explain\SQLScripts\Advanced\Advanced Lab2.SQL

__ 2. Activate the options by changing the Query Options Library for your job's query attributes using the `CHGQRYA` CL command. Verify that the command was successful:

```
CL: CHGQRYA QRYOPTLIB(veteamxx);
```

## Task 2: Nested Loop Join method

This exercise shows you how Visual Explain represents a Nested Loop Join.

__ 1. Update the SELECT statement in the Run SQL Script window so that the local selection for *year* is `2000`, *month* is `10` and *returnflag* is `R`:

```
SELECT    a.year,a.month,a.returnflag,b.part,b.mfgr,a.quantity
        , a.revenue_wo_tax
FROM      item_fact a , part_dim b
WHERE     a.partkey = b.partkey
AND       a.year = 2000
AND       a.month = 10
AND       a.returnflag = 'R';
```

__ 2. Run and explain the statement to analyze the statement

__ 3. Verify that the graph you received resembles the example in Figure 4.

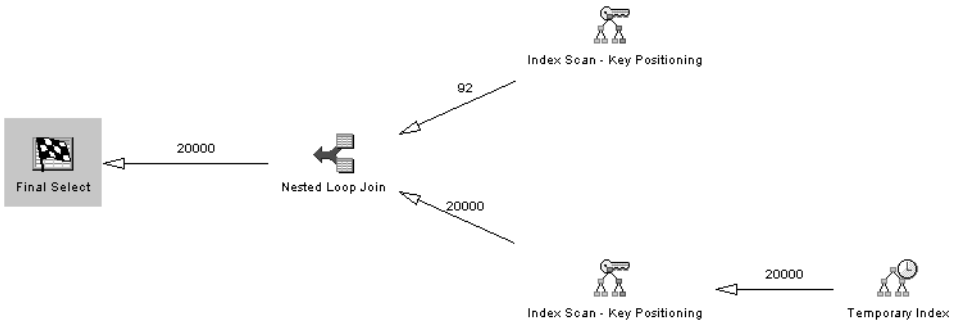



*Figure 4. Nested Loop Join*

__ 4. Analyze the query optimization by reviewing the Visual Explain window and complete Table 3.

*Table 3. Nested Loop Join*

| Question | Answer |
|---|---|
| What was the access method for ITEM_FACT? | |
| What was the access method for PART_DIM? | |
| What was the join method? | |
| What was the join order (positions)? | |
| What was the join type? | |
| What index was used to access ITEM_FACT? | |
| What index was used to access PART_DIM? | |
| If a temporary access path was created, what key fields should be used to create a permanent index? | |
| Why was a temporary access path created? Use the *flyover* function to find the reason code, and then use associated documentation to find the meaning. | |

__ 5. Close the Visual Explain window. A window appears asking if you wish to save this data as a performace monitor. Click the **No** button.

## Task 3: Visual Explain and Hash Join method

This exercise shows you how Visual Explain represents a Hash Join method.

__ 1. Move the cursor in the Run SQL Script window to the next SELECT statement, and then click the Run and Explain icon.

```
SELECT    a.year,a.month,a.returnflag,b.part,b.mfgr,a.quantity
```

```
      , a.revenue_wo_tax
FROM     item_fact a , part_dim b
WHERE    a.partkey = b.partkey
AND      a.year = 2000
AND      a.month = 10
AND      a.returnflag = 'R'
OPTIMIZE FOR ALL ROWS;
```

__ 2. Verify that the graph you received resembles the example in Figure 5.

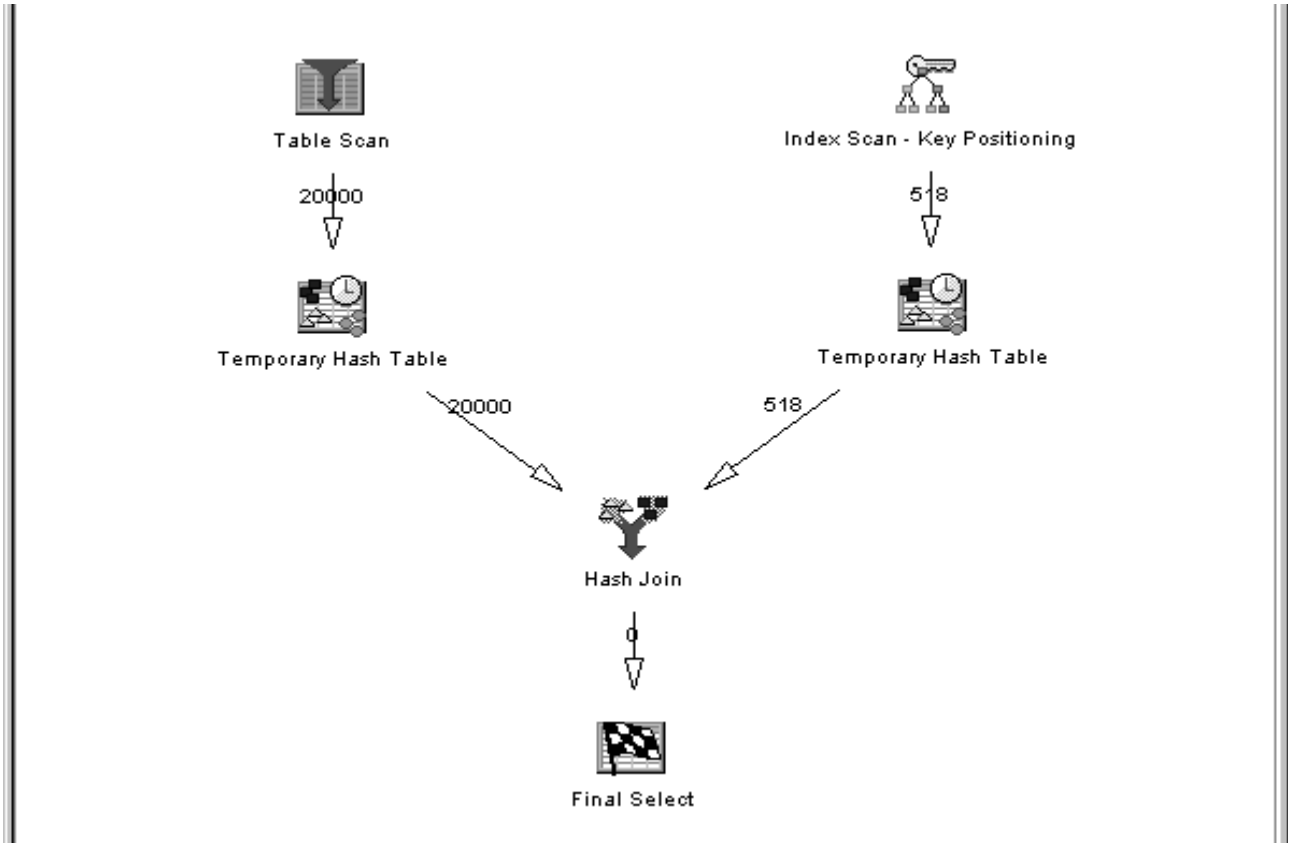



*Figure 5. Hash Join*

__ 3. Analyze the query optimization by reviewing the Visual Explain window and complete the following table.

| Question | Answer |
|---|---|
| What was the access method for ITEM_FACT | |
| What was the access method for PART_DIM? | |
| What was the join method? | |
| What index was used to access ITEM_FACT? | |

__ 4. Close the message window.

## Task 4: Nested Loop 3-way join

This exercise shows you multiple keyed access methods while performing a 3-way join.

__ 1. Move the cursor to the next SELECT statement and then update the statement in the Run SQL Script window so that the local selection for *partkey* is 13213:

```
SELECT    i.orderdate, p.part, s.supplier
FROM      item_fact i , part_dim p , supp_dim s
WHERE     i.partkey = p.partkey
AND       i.suppkey = s.suppkey
AND       i.partkey = 13213 ;
```

__ 2. Use the Run and Explain icon to execute the statement.

__ 3. Verify that the graph you received resembles the example in Figure 6.

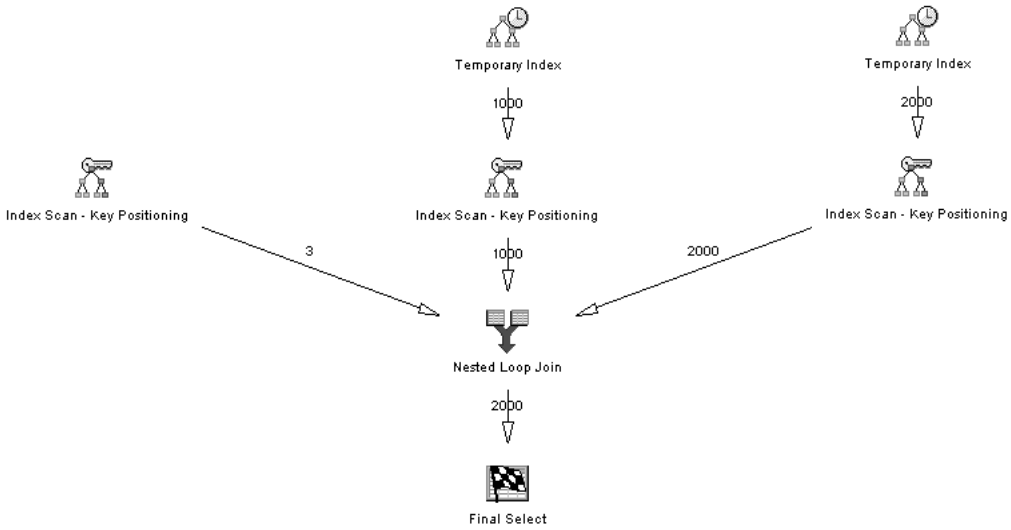



*Figure 6. Three-way Nested Loop Join*

__ 4. Analyze the query implementation by reviewing the Visual Explain window, and complete Table 4.

*Table 4. Three-way Nested Loop Join questions*

| Question | Answer |
|---|---|
| What was the join method? | |
| What was the join type? | |
| What was the join order (positions)? | |
| What was the estimated number of rows to be selected from ITEM_FACT? | |
| What was the estimated number of rows to be joined from SUPP_DIM? | |

| Question | Answer |
|---|---|
| What was the estimated number of rows to be joined from PART_DIM? | |

__ 5. Close the Results viewer.

## Task 5: Comparing Visual Explain with job log messages for analysis (Optional)

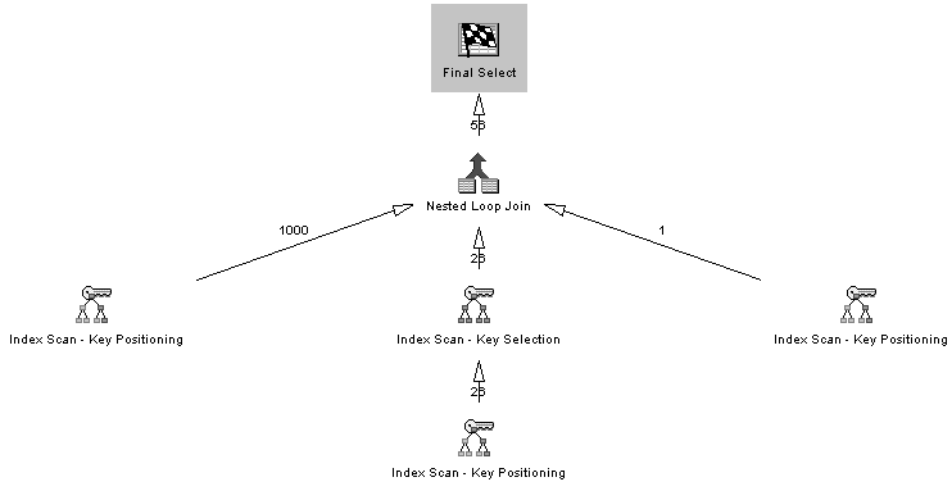
In this exercise, compare the results of Visual Explain with detailed debug messages captured into a job log for a 3-way Nested Loop Join.

__ 1. Activate the **Include Debug Messages** option, and launch the Job Log viewer from the Run SQL Scripts window.

__ 2. Move the cursor to the next SQL statement, create the two indexes and then run the next SELECT statement. Note that we are running the same SQL Statement that was used on the previous task but we have changed SELECT for sELECT to force a re-optimization).

```
CREATE INDEX SUPP_IDX1 ON SUPP_DIM (SUPPKEY);
CREATE INDEX PART_IDX1 ON PART_DIM (PARTKEY);
sELECT    i.orderdate, p.part, s.supplier
FROM      item_fact i , part_dim p , supp_dim s
WHERE     i.partkey = p.partkey
AND       i.suppkey = s.suppkey
AND       i.partkey = 13213 ;
```

__ 3. Refresh the Job Log viewer. Compare the job log with the graph as shown in Figure 7.




| Message ID | Message |
|---|---|
| CPI4326 | File SUPP_DIM processed in join position 3. |
| CPI4326 | File ITEM_FACT processed in join position 2. |
| CPI4326 | File PART_DIM processed in join position 1. |
| CPI432C | All access paths were considered for file ITEM_FACT. |
| CPI433D | Query options used to build the OS/400 query access plan. |
| CPI4339 | Query options retrieved file QAQQINI in library VETEAM30. |
| CPI4339 | Query options retrieved file QAQQINI in library VETEAM30. |

*Figure 7. Three-way Nested Loop Join job log entries and the equivalent Visual Explain graph*

__ 4. Analyze the job log.

__ 5. Close the Job Log viewer and the Visual Explain and Run SQL Script windows.

This completes this lab.

# Lab 3. Grouping and Ordering analysis

### Introduction

Grouping and Ordering are very powerful features that are frequently used in SQL. They are a major topic to consider in performance analysis.

This lab shows you how Visual Explain presents methods involving Grouping and Ordering.

### Objectives

After completing this lab, you will be able to:

1. Distinguish the different Grouping and Ordering methods and how Visual Explain shows them.
2. Be familiar with the attributes that affect the behavior of queries involving Grouping and Ordering.

### Lab environment

Ensure that all the indexes created in the previous labs exist.(SUPP_IDX1 and PART_IDX1)

### Time required

The time required to efficiently complete this lab project is 60 minutes.

## Task 1: Setting up the environment

In this exercise, you set up the environment required to analyze Grouping and Ordering methods.

__ 1. Launch the Run SQL Scripts window from the Database component of Operations Navigator if it is not already active.

__ 2. Open the x:\VisualExplain\SQLScripts\Advanced\Advanced Lab3a.SQL script file.

__ 3. Ensure tha SUPP_IDX1 and PART_IDX1 exist. If they do not exist run the following SQL Statements:

```
CREATE INDEX SUPP_IDX1 ON SUPP_DIM(SUPPKEY);
CREATE INDEX PART_IDX1 ON PART_DIM(PARTKEY);
```

__ 4. Run the statement that enables parallel processing and verify that the statement completed successfully:

```
CL: CHGQRYA DEGREE(*OPTIMIZE);
```

## Task 2: Index Group By

This exercise show you how Visual Explain represents an Index Group By method.

__ 1. Update the SELECT statement in the Run SQL Script window so that the local selection for *year* is `2000`, *month* is `10` and *returnflag* is `R`.

```
SELECT    a.year ,a.month ,a.returnflag ,SUM(a.quantity)
```

```
        ,SUM(a.revenue_wo_tax)
FROM     item_fact a
WHERE    a.year = 2000
AND      a.month = 10
AND      a.returnflag = 'R'
GROUP BY  a.year ,a.month ,a.returnflag
OPTIMIZE FOR ALL ROWS;
```

__ 2. Click the **Run and Explain** icon.

__ 3. Verify that the graph you received is like that shown in Figure 8.


*Figure 8. Index Group By*

__ 4. Analyze the query optimization by reviewing the Visual Explain window, then complete Table 5.

*Table 5. Index Group By*

| Question | Answer |
|---|---|
| What was the estimated number of rows selected? | |
| What method was used for Grouping? | |
| What index was used for Grouping? | |
| What was the average number of rows in each group? | |

__ 5. Close the Visual Explain window. A window appears asking if you wish to save this data as a performace monitor. Click the **No** button.

## Task 3: Hash Group By method

This exercise shows you how Visual Explain presents the analysis of a Hash Group By method.

__ 1. Move the cursor in the Run SQL Script window to the next SELECT statement and update the statement so that the local selection for MFGR is `Manufacturer#1`:

```
SELECT      mfgr , brand , part , partkey
FROM        part_dim
WHERE       mfgr = 'Manufacturer#1'
GROUP BY    mfgr , brand , part , partkey;
```

__ 2. Run and explain the statement.

__ 3. Verify that the graph you received resembles the example shown in Figure 9.

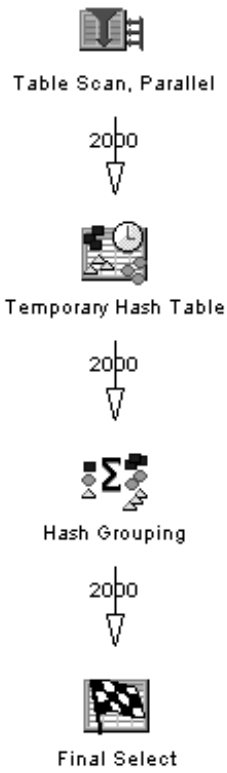



*Figure 9. Hash Group By*

__ 4. Analyze the query optimization by reviewing the Visual Explain window and complete Table 6.

*Table 6. Hash Group By*

| Question | Answer |
|---|---|
| What was the Grouping method? | |
| How many rows were in the hash table? | |
| What was the average number of rows in each group of the hash table? | |

__ 5. Close the Visual Explain window. A window appears asking if you wish to save this data as a performace monitor. Click the **No** button.

## Task 4: Hash Grouping with Nested Loop Join

This exercise shows you the explanation of a join that also has group and order by clauses.

__ 1. Disable parallelism by setting the *degree* parameter to `*NONE`.

```
CL: CHGQRYA DEGREE(*NONE);
```

__ 2. Update the SELECT statement in the Run SQL Script window so that the local selection for *year* is `2000`, *month* is `10` and *returnflag* is `R`.

```
SELECT     a.year , a.month , a.returnflag , b.part , b.mfgr
         , SUM(a.quantity) AS TOTAL_QUANTITY
         , SUM(a.revenue_wo_tax) AS TOTAL_REVENUE
FROM       item_fact a , part_dim b
WHERE      a.partkey = b.partkey
AND        a.year = 2000
AND        a.month = 10
AND        a.returnflag = 'R'
GROUP BY   a.year , a.month , a.returnflag , b.part , b.mfgr
```

__ 3. Run and explain the statement.

__ 4. Verify that the graph you received resembles the example shown in Figure 10.

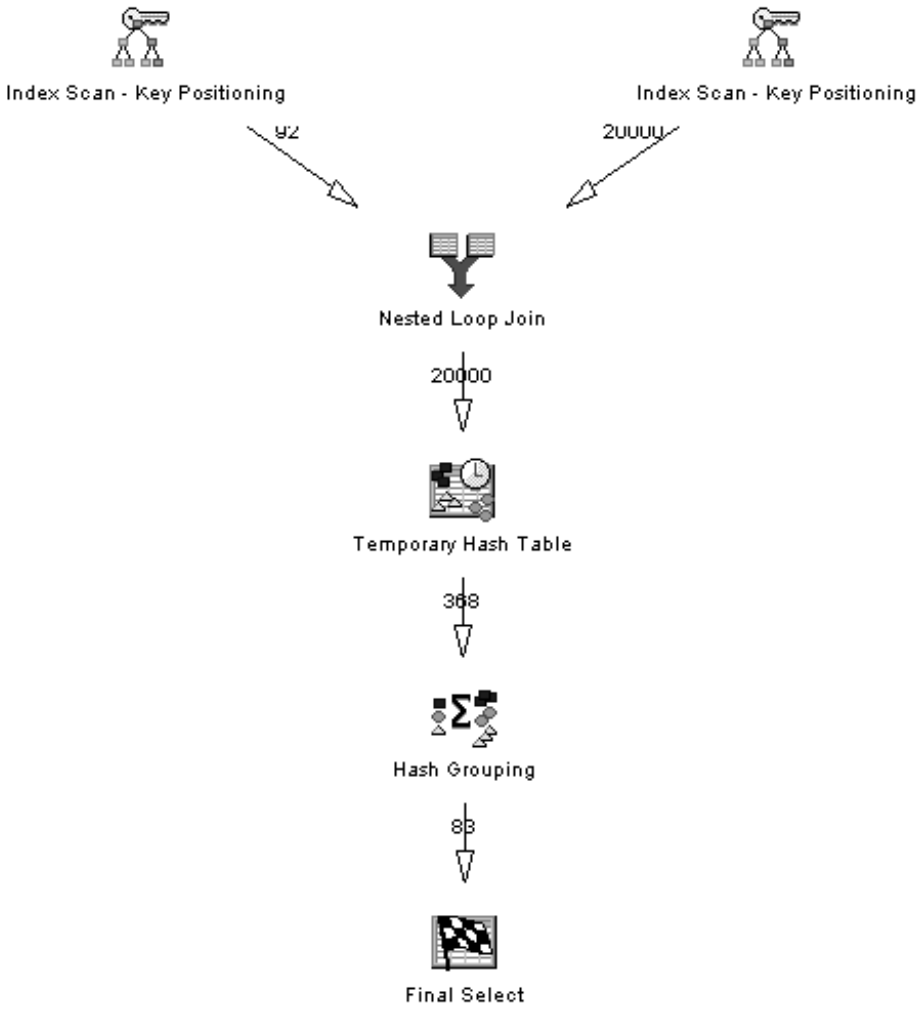



*Figure 10. Hash Group By with NLJ*

__ 5. Analyze the query optimization by reviewing the Visual Explain window. Then, complete Table 7.

*Table 7. Hash Group By with NLJ*

| Question | Answer |
|---|---|
| What was the join method? | |
| What was the join order (positions)? | |
| What index was used to access ITEM_FACT? | |
| What was the Grouping method? | |
| How many rows were in the hash table? | |

__ 6. Close the Visual Explain window. A window appears asking if you wish to save this data as a performace monitor. Click the **No** button.

## Task 5: Explanation of Index Ordering method

This exercise shows you how Visual Explain presents the analysis of an Ordering method using an index.

__ 1. Open the x:\VisualExplain\SQLScripts\Advanced\Advanced Lab3b.SQL script file.

__ 2. Move the cursor to the SELECT statement in the Run SQL Script window, and click the **Run and Explain** icon.

```
SELECT    partkey , part , mfgr , brand
FROM      part_dim
ORDER BY  partkey;
```

__ 3. Verify that the graph you received resembles the example shown in Figure 11.

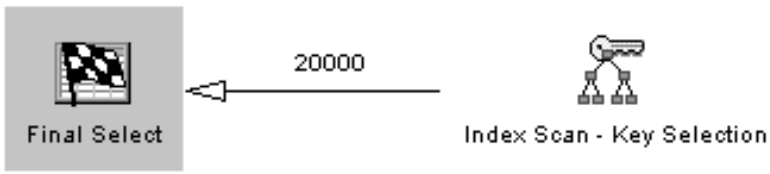



*Figure 11. Index Ordering*

__ 4. Analyze the query optimization by reviewing the Visual Explain window and complete Table 8.

*Table 8. Index Ordering*

| Question | Answer |
|---|---|
| What method was used for Ordering? | |
| What index was used for Ordering? | |
| What was the reason code for using the index? | |

__ 5. Close the Visual Explain window and the Results viewer.

## Task 6: Ordering by a Sort method

This exercise shows you how Visual Explain represents Ordering by using a Sort method.

__ 1. Move the cursor to the next SELECT statement in the Run SQL Script window, and click the **Run and Explain** icon.

```
SELECT    partkey , part , mfgr , brand
FROM      part_dim
ORDER BY  mfgr;
```

__ 2. Verify that the graph you receive resembles the example shown in Figure 12.


*Figure 12. Order by a Sort*

__ 3. Analyze the query optimization by reviewing the Visual Explain window. Then, complete Table 9.

*Table 9. Order by a Sort*

| Question | Answer |
|---|---|
| What access method was used for PART_DIM? | |
| What were the reason code(s) for not using an index to select the records from PART_DIM? | |
| What method was used for Ordering? | |

__ 4. Close the Visual Explain window. A window appears asking if you wish to save this data as a performace monitor. Click the **No** button.

## Task 7: Comparison of Visual Explain and job log information (Optional)

In this exercise, you compare the results from Visual Explain with the debug messages captured in a job log.

__ 1. Move the cursor to the next SELECT statement in the Run SQL Script window.

```
SELECT a.year, a.month, a.returnflag, b.part, b.mfgr,
sum(a.quantity) as TOTAL_QUANTITY,
sum(a.revenue_wo_tax) AS TOTAL_REVENUE
```

```
FROM item_fact a, part_dim b
WHERE a.partkey = b.partkey
AND a.year = 2000
AND a.month = 10
AND a.returnflag = 'R'
GROUP BY   a.year, a.month, a.returnflag, b.part, b.mfgr
ORDER BY  TOTAL_QUANTITY DESC, b.part , b.mfgr
```

__ 2. Activate the Include Debug Messages option and launch the Job Log viewer.

__ 3. Run and Explain the statement.

__ 4. Close the Results viewer, and refresh the Job Log viewer. Analyze the graph and the job log as shown in Figure 13.

| Message ID | Message |
|---|---|
| CPI4325 | Temporary result file built for query. |
| CPI4326 | File PART_DIM processed in join position 2. |
| CPI4326 | File ITEM_FACT processed in join position 1. |
| CPI432C | All access paths were considered for file PART_DIM. |
| CPI432C | All access paths were considered for file ITEM_FACT. |

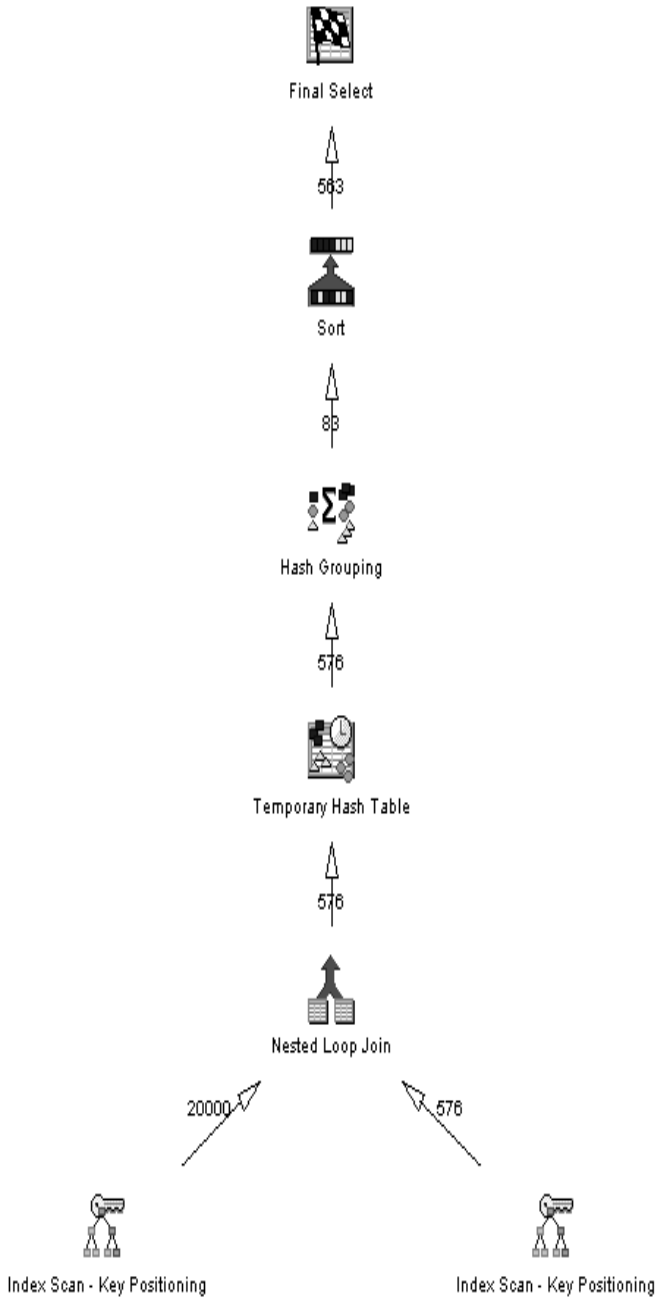



*Figure 13. Comparison of job log and Visual Explain graph*

__ 5. Analyze the job log.

__ 6. Close the Results and the Job Log viewer.

This completes this lab.

# Lab 4. Analyzing UNIONs

## Introduction

SQL operations involving UNION and UNION ALL sentences are becoming common. Sometimes, these operations require a careful review to assure that they are performing optimally.

## Objectives

This lab teaches you to:

1. Distinguish UNION and UNION ALL methods and see how Visual Explain shows them.
2. Be familiar with the SQL Performance Monitor data collection feature and learn how to use Visual Explain in combination with SQL Performance Monitor.

## Time required

The time required to efficiently complete this lab project is 35 minutes.

## Task 1: Setting up the environment

This exercise shows you how to set up the environment required to run the SQL Performance Monitor.

__ 1. If the SQL Script Center is not already active, launch it from the Database component of Operations Navigator.

__ 2. Open the x:\VisualExplain\SQL Scripts\Advanced\Advanced Lab4.SQL script file.

__ 3. Identify the fully qualified job name for your job and complete the job data details shown in the following table:

| | |
|---|---|
| Job number | |
| User name | |
| Job name | |

__ 4. Switch to the main Operations Navigator window. Start the SQL Performance Monitor by right-clicking the **SQL Performance Monitors** icon and selecting **New->Detailed**. The New Detailed SQL Performance Monitor dialog box appears.

On the General tab, enter `VETEAMxx UNION & UNION ALL` for the Name. Enter your team name for the library, and click the **Monitored Jobs** tab. The system refreshes the list of jobs currently active on the system. Click the **Number** column, which sorts the jobs into number order. Scan down the list until you find your job number. Select your job by selecting the job name and clicking the **Select** button. Click **OK** to start the SQL Performance Monitor. The monitor status changed to *Started* in the right panel of the Operations Navigator window.

## Task 2: Collecting the performance data

In this exercise, you use the SQL Performance Monitor that has been started to collect data for analysis with Visual Explain.

__ 1. Update the SELECT statement so that the local selection for *country* is COLOMBIA and then run the statement (do not run and explain).

```
SELECT
          c.custkey as "Involved Party"
         ,1 as "Involved Party Type Id"
         ,c.customer as "Involved Party Name"
         ,c.address as "Address"
         ,c.phone as "Phone"
FROM
          cust_dim c
WHERE
          country = 'COLOMBIA'
UNION
SELECT
          s.suppkey as "Involved Party"
         ,2 as "Involved Party Type Id"
         ,s.supplier as "Involved Party Name"
         ,s.address as "Address"
         ,s.phone as "Phone"
FROM
          supp_dim s
WHERE
          country = 'COLOMBIA'
OPTIMIZE FOR ALL ROWS;
```

__ 2. Update the SELECT statement so that UNION becomes UNION ALL. Then, run the statement.

```
SELECT
          c.custkey as "Involved Party"
         ,1 as "Involved Party Type Id"
         ,c.customer as "Involved Party Name"
         ,c.address as "Address"
         ,c.phone as "Phone"
FROM
          cust_dim c
WHERE
          country = 'COLOMBIA'
UNION ALL
SELECT
          s.suppkey as "Involved Party"
         ,2 as "Involved Party Type Id"
         ,s.supplier as "Involved Party Name"
         ,s.address as "Address"
         ,s.phone as "Phone"
FROM
          supp_dim s
WHERE
          country = 'COLOMBIA'
OPTIMIZE FOR ALL ROWS;
```

__ 3. Switch to the main Operations Navigator.Right-click the **Performance Monitor** object, and select **End**. This ends the detailed monitor collection.

## Task 3: Explanation of UNION statement

In this exercise, you analyze the data that has been collected in the detailed monitor.

__ 1. Right-click the monitor object that has been used to collect the performance data, and select **List Explainable Statements**. The Explainable Statements For SQL Performance Monitor panel appears as shown in Figure 14 on page xxv.

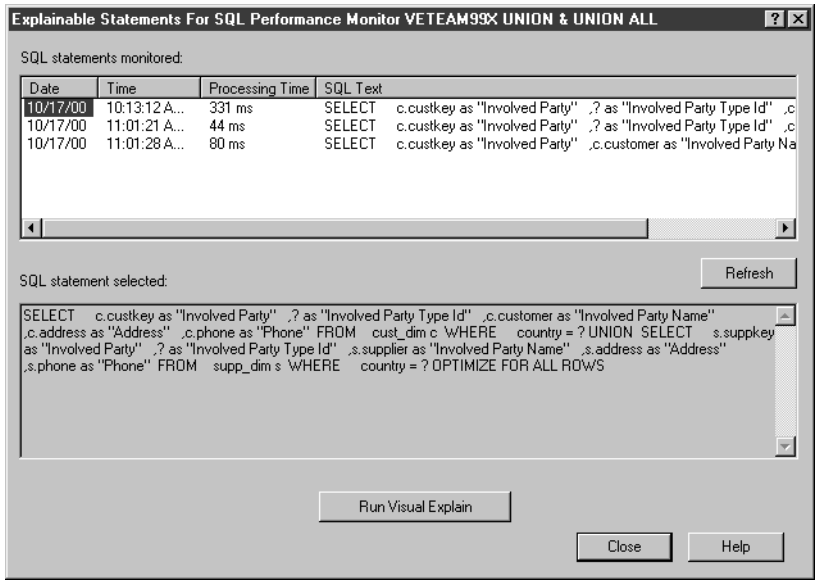



*Figure 14. Explainable Statements*

__ 2. Select the first SQL Statement, and then click **Run Visual Explain.**

__ 3. Verify that the graph you received looks like the example shown in Figure 15.

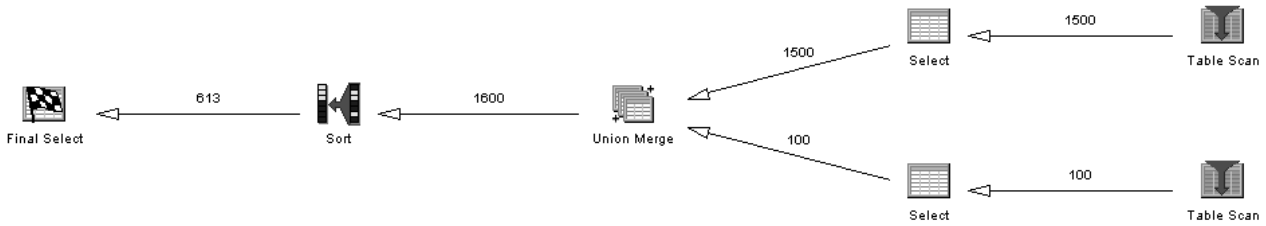



*Figure 15. UNION graph*

__ 4. Analyze the query optimization by reviewing the Visual Explain window and complete Table 10.

*Table 10. UNION questions*

| Question | Answer |
|---|---|
| What was the access method for CUST_DIM? | |
| What was the access method for SUPP_DIM? | |

| Question | Answer |
|---|---|
| What was the estimated number of rows from the UNION merge? | |
| What was the reason code for the Sort and what does the code mean? | |
| What was the estimated number of rows selected from the Sort? | |

__ 5. Close the Visual Explain window.

__ 6. Select the second SQL Statement from the Explainable Statements For SQL Performance Monitor panel, and then click **Run Visual Explain**.

__ 7. Verify that the graph that appears resembles the example shown in Figure 16.

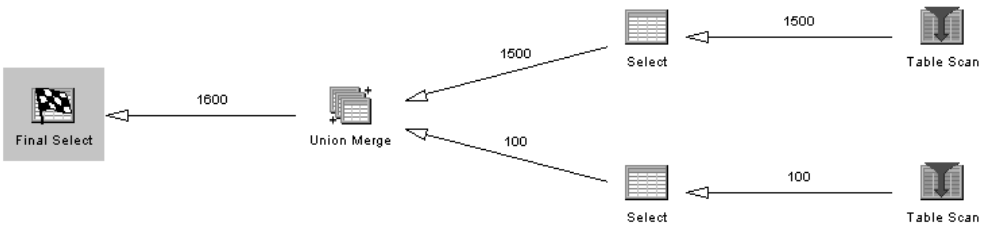



*Figure 16. UNION ALL graph*

__ 8. Analyze the query optimization by reviewing the Visual Explain window. Then, complete Table 11.

*Table 11. UNION ALL questions*

| Question | Answer |
|---|---|
| What was the estimated number of rows selected from SUPP_DIM? | |
| What was the estimated number of rows selected from CUST_DIM? | |
| What was the estimated number of rows from the UNION merge? | |
| Is there a difference in the total time to run? Why? | |

__ 9. Close the Visual Explain window.

__ 10. Close the SQL Script window.

**Note**: Queries contaning an UNION operator can also be analyzed with Run and Explain. In this lab we used the SQL Performance monitor just to ilustrate another way of doing it.

This lab is complete.

# Lab 5. Complex Query Analysis

## Introduction

In this lab, you analyze data from an SQL Performance Monitor that has been imported from another system using the Database Monitor. You also analyze the statement using debug messages in the job log.

## Objectives

This lab teaches you to:

1. Import Database Monitor detailed data into an SQL Performance Monitor
2. Analyze data from imported monitor data using Visual Explain
3. Compare the results from Visual Explain with Debug messages

## Time required

The time required to efficiently complete this lab project is 60 minutes.

## Task 1: Analysis of complex statements using Monitor Data

In this lab, you import the Database Monitor data into an SQL Performance Monitor and analyze the statements.

__ 1. Switch to the Operations Navigator window and right-click **SQL Performance Monitors**. Select **Import ...**. The Import SQL Performance Monitor Files dialogue panel appears as shown in Figure 17.

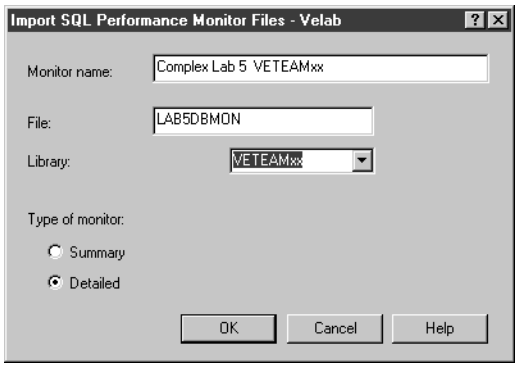



*Figure 17. Import SQL Performance Monitor Files*

__ 2. Enter the details shown in Figure 17 using your team name, and then click **OK**.

__ 3. Right-click the monitor object that was created by the import operation and select **List Explainable Statements**. From the list of explainable statements, select the first entry, and then click **Run Visual Explain** as shown in Figure 18 on page xxx.

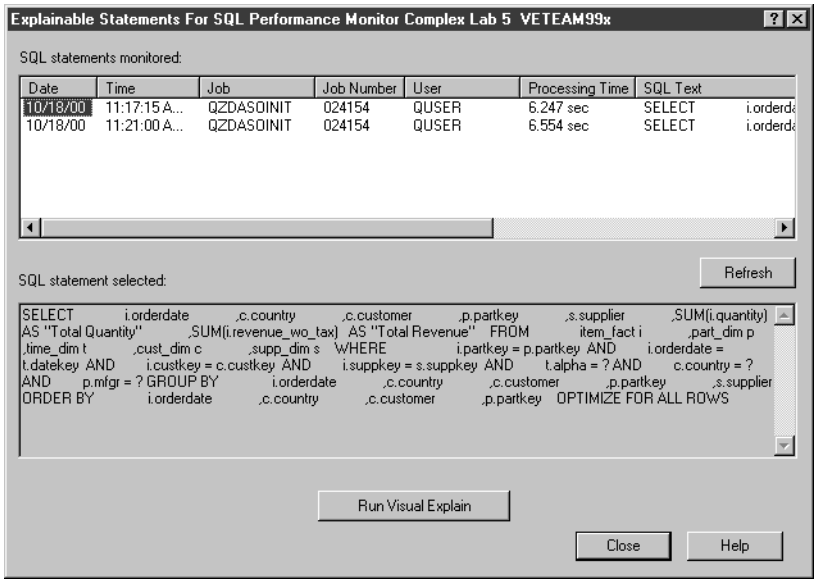



*Figure 18. List Explainable Statements*

__ 4. Verify that the graph you see resembles the example in Figure 19.

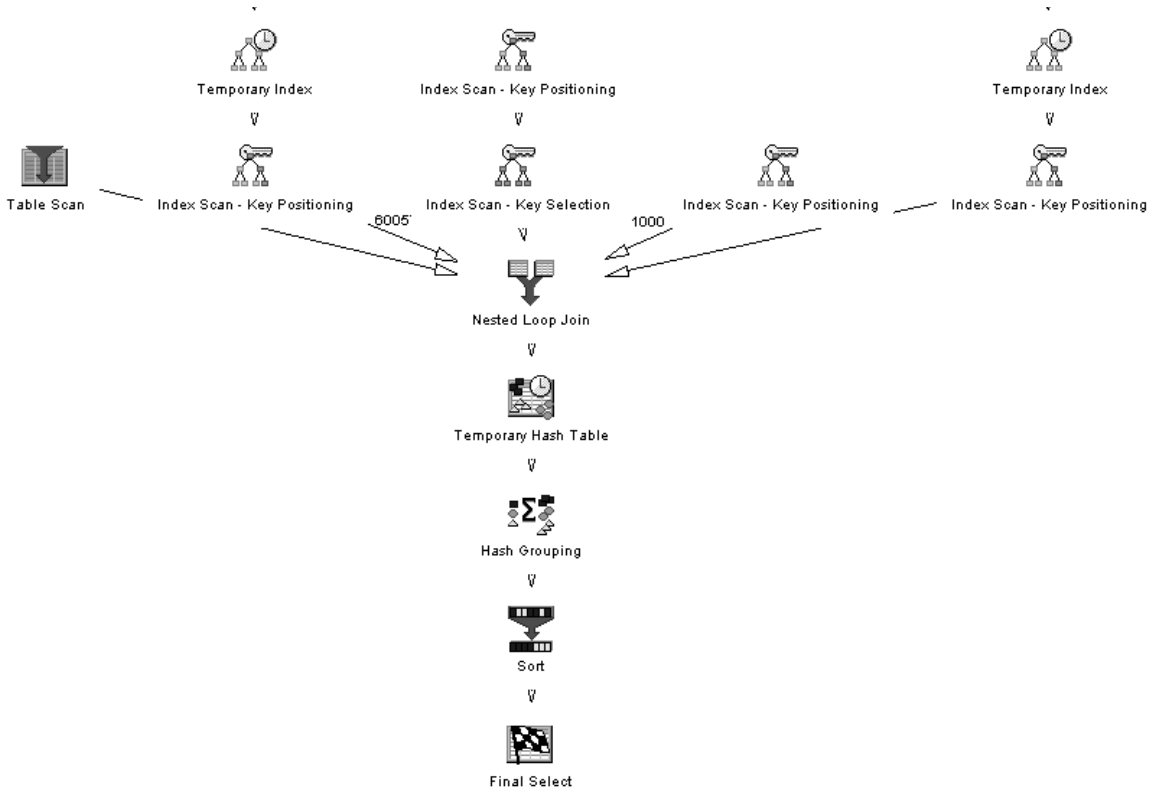



*Figure 19. Complex Query*

__ 5. Analyze the query optimization by reviewing the Visual Explain window and complete the tables that follow. Each table requires you to select the

corresponding icon in the graph You will be able to use the flyover and the Attributes and Values panel to answer the questions.

__ 6. Click the **Final Select** icon, and answer the question in Table 12.

*Table 12. Final Select*

| Question | Answer |
|---|---|
| What was the optimization time? | |
| What was the statement type? | |
| What were the host variable values? | |
| What was the Ordering implementation? | |
| What was the Grouping implementation? | |
| What was the join implementation? | |
| Was the qaqqini options file active for the statement? | |
| What was the parallel degree setting? | |

You will also find details of whether the statement was successful and the statement itself besides many other useful pieces of information.

__ 7. Click the **Hash Grouping** icon, and answer the questions in Table 13

*Table 13. Hash Grouping*

| Question | Answer |
|---|---|
| What was the Grouping implementation? | |
| What was the estimated number of groups? | |
| What was the estimated number of joined rows? | |
| What was the average number of rows in each group? | |
| What were the sum columns? | |
| How many rows were in the hash table created? | |

__ 8. Click the **Nested Loop Join** icon, and answer the questions in Table 14.

*Table 14. Nested Loop Join*

| Question | Answer |
|---|---|
| What was the number of estimated joined rows? | |
| How many tables were joined? | |

__ 9. List the tables being joined in their join order and the primary access methods used on each of those tables:

| Table Name | Primary Access Method |
|---|---|
| | |

| Table Name | Primary Access Method |
|---|---|
| | |
| | |
| | |
| | |

__ 10. Investigate the Data Access methods used to select and join the tables together. Click the **Table Scan** icon for TIME_DIM, and answer the questions in Table 15.

*Table 15. Table Scan: TIME_DIM*

| Question | Answer |
|---|---|
| What was the estimated processing time? | |
| What is the total number of rows in the table? | |
| What was the estimated number of rows to be selected? | |
| What was the join position for this table? | |
| Was data space selection used? | |
| Was Skip Sequential selection performed? | |
| Was the creation of an index advised? | |

__ 11. Click the **Temporary Index** icon for ITEM_FACT, and answer the questions in Table 16.

*Table 16. Temporary Index: ITEM_FACT*

| Question | Answer |
|---|---|
| How many entries did the index contain? | |
| Was the index created reusable? | |
| Was the index created a sparse index? | |
| What was the type of index created? | |
| What were the Key columns for the index build? | |

__ 12. Click the **Index Scan - Key Positioning** icon for ITEM_FACT, and answer the questions in Table 17.

*Table 17. Index Scan - Key Positioning: ITEM_FACT*

| Question | Answer |
|---|---|
| What is the join position? | |
| Was Index Only Access used? | |
| Did the index fit in memory? | |
| What was the reason code for the Index Scan? | |

| Question | Answer |
|---|---|
| What columns were used for Index Key Positioning? | |

__ 13. Click the **Index Scan - Key Positioning** icon for PART_DIM and answer the questions in Table 18.

*Table 18. Index Scan - Key Positioning: PART_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | |
| What was the total number of rows in the table? | |
| What was the estimated number of rows selected? | |
| What was the estimated number of joined rows? | |
| What was the join position? | |
| Was Index Only Access used? | |
| Did the index fit in memory? | |
| How many entries did the index have? | |
| Was the index a constraint? | |
| Did the optimizer time out? | |
| Which indexes were considered during optimization? | |
| What was the number of primary key columns? | |
| What columns were used for key positioning? | |

__ 14. Click the **Index Scan - Key Selection** icon for PART_DIM, and answer the questions in Table 19.

*Table 19. Index Scan - Key Selection: PART_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | |
| Was derived selection performed? | |
| What columns were used for derived selection? | |
| What columns were used for key selection? | |

__ 15. Click the **Index Scan - Key Positioning** icon for SUPP_DIM, and answer the questions in Table 20.

*Table 20. Index Scan - Key Positioning: SUPP_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | |
| What was the estimated number of joined rows? | |

| Question | Answer |
|---|---|
| What was the join position? | |
| Was Index Only Access used? | |
| Did the index fit in memory? | |
| How many entries did the index have? | |
| Was the index a constraint? | |

__ 16. Click the **Temporary Index** icon for CUST_DIM, and answer the questions in Table 21.

*Table 21. Temporary Index - CUST_DIM*

| Question | Answer |
|---|---|
| What was the name of the Temporary Index created? | |
| How many entries did the index contain? | |
| Was the index created reusable? | |
| Was the index created a sparse index? | |
| What was the type of index created? | |
| Was the index created from an existing index? | |
| What was the reason for the index scan? | |
| What were the Key columns for the index build? | |

__ 17. Click the **Index Scan - Key Positioning** icon for CUST_DIM, and answer the questions in Table 22.

*Table 22. Index Scan - Key Positioning - CUST_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | |
| What was the number of columns used for key positioning? | |
| Was Index Only Access used? | |
| Did the index fit in memory? | |
| Was derived selection performed? | |
| What column was used for derived selection | |
| What columns were used for Index Key Positioning? | |
| What was the join position for this table? | |

__ 18. Close the Visual Explain window and the Explainable Statements window.

## Task 2: Analysis of complex statement using a job log (Optional)

In this task, you run the statement that you analyzed in Task 1 and collect debug messages for comparison with the Visual Explain analysis.

__ 1. If the SQL Script Center is not already active, launch it from the Database component of Operations Navigator.

__ 2. Open the x:\VisualExplain\SQLScripts\Advanced\Advanced Lab 5.SQL script file.

__ 3. Run the statement that will disable parallel processing, and verify that the statement completed successfully:

```
CL: CHGQRYA DEGRE(*NONE);
```

__ 4. Activate the Include Debug Messages option, and then launch the Job Log viewer.

__ 5. Run the statement.

__ 6. Close the Results viewer, and refresh the Job Log viewer.

Figure 20 shows you an extract of the job log and the graph that was produced for the statement. The graph was rotated (*Orient top*) so that the results are at the top of the graph for an easier comparison with the job log.

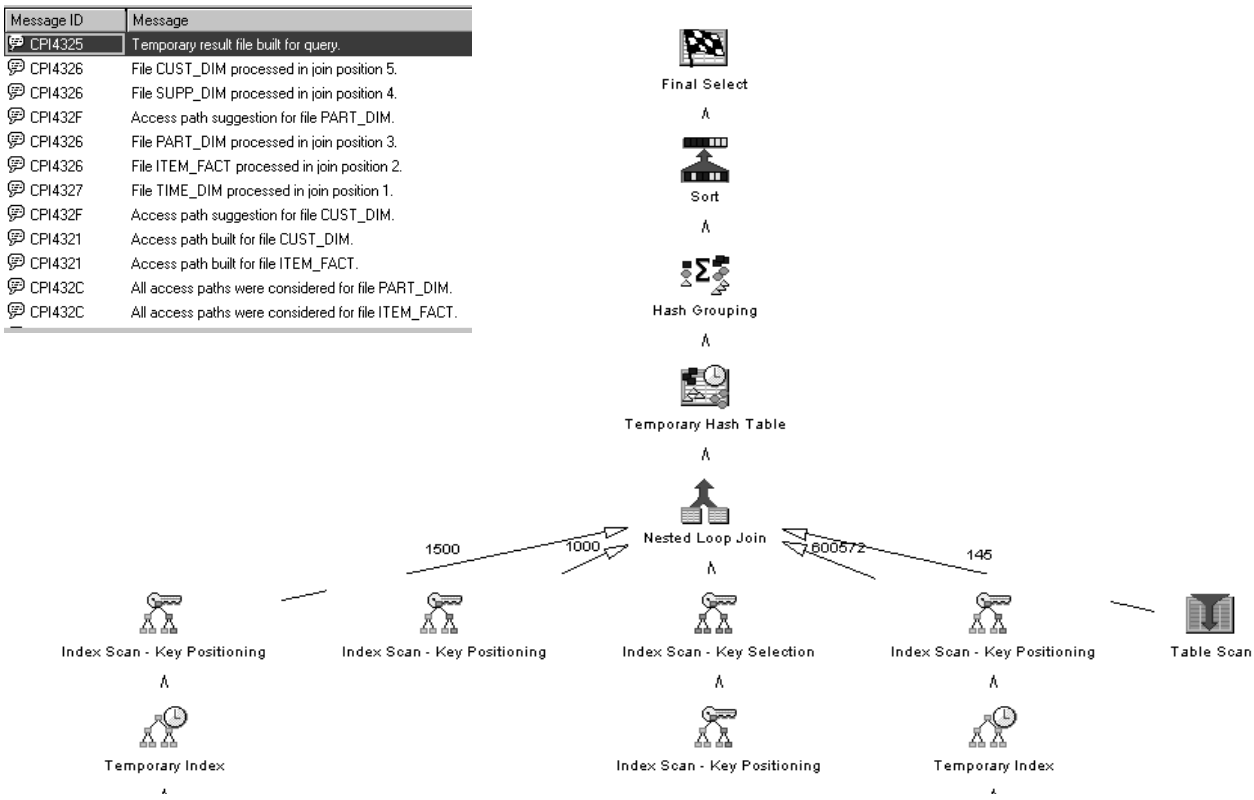



*Figure 20. Comparison of job log and Visual Explain graph*

__ 7. Analyze the optimization using *only* the job log details and complete Table 23.

*Table 23. Analysis of job log information*

| Question | Answer |
|---|---|
| What was the Ordering implementation? | |
| What was the Grouping implementation? | |
| What was the join implementation? | |
| What was the estimated number of groups? | |
| What was the estimated number of joined rows? | |
| What was the average number of rows in each group? | |
| What were the Grouping columns used for the hash table? | |
| What columns were used for the hash table? | |
| How many tables were joined? | |
| What was the join order? | |
| What access methods were used for:<br>ITEM_FACT?<br>PART_DIM?<br>TIME_DIM?<br>CUST_DIM?<br>SUPP_DIM? | |
| Is it possible to determine the number of rows joined from each table? | |
| Were any Temporary Access Paths built? | |
| Did the Index Advisor recommend any indexes to be built? | |

You can see that the job log does not provide the detailed analysis that Visual Explain can.

__ 8. Close the Job Log viewer and the SQL Script Center.

This lab is complete.

## Complex Query SQL statement

```
SELECT
          i.orderdate
         ,c.country
         ,c.customer
         ,p.partkey
         ,s.supplier
         ,SUM(i.quantity)         AS "Total Quantity"
         ,SUM(i.revenue_wo_tax)   AS "Total Revenue"
FROM
          item_fact i
         ,part_dim p
         ,time_dim t
         ,cust_dim c
         ,supp_dim s
WHERE
          i.partkey = p.partkey
AND       i.orderdate = t.datekey
AND       i.custkey = c.custkey
AND       i.suppkey = s.suppkey
AND       t.alpha = '2000-10-30'
AND       c.country = 'JAPAN'
AND       p.mfgr = 'Manufacturer#3'
GROUP BY
          i.orderdate
         ,c.country
         ,c.customer
         ,p.partkey
         ,s.supplier
ORDER BY
          i.orderdate
         ,c.country
         ,c.customer
         ,p.partkey
OPTIMIZE FOR ALL ROWS;
```

# Lab 1 answers: Advanced Keyed Access analysis

### Introduction

The solutions given for the labs are based on our results using V4R5M0 on an iSeries server Model 270 in laboratory conditions. This system has two processors, 2 GB of main storage, 12 disk drives and SMP installed. Some answers may differ when Visual Explain is run in your own iSeries environment.

**Note:** You may be working with smaller tables and for this reason the answer to some questions may be different than the ones published in this document.

## Task 1: Setting up the environment

## Task 2: Parallel Keyed access method

*Table 24. Parallel key access questions*

| Question | Answer |
|---|---|
| What access methods were used? | Index Scan - Key Row Positioning - Parallel<br>Index Scan - Key Row Selection - Parallel |
| Which indexes were optimized (also include the reason code for used or not used)? | ITEM_IDX2 - 0<br>ITEM_IDX1 - 17 |
| What index was used to access the table? | ITEM_IDX2 |
| If an index was *not* used, what was the reason for not using it? | ITEM_IDX1 - 17 - left most key did not match any columns used in the query selection |
| What parallel degree was requested? | 10 tasks |

## Task 3: Index-From-Index access method

*Table 25. Index-From-Index questions*

| Question | Answer |
|---|---|
| What access methods were used? | Index Scan - Key Row Positioning<br>Index Scan - Key Row Selection |
| Was a Temporary Index created? | YES |
| Why was the Temporary Index created? | I2 - Ordering/Grouping |
| Was the index created from an existing index or from a table? | YES |
| Was the Temporary Index created from a SPARSE index? | YES, INDEX |
| What were the key columns used for the Temporary Index creation? | 1.*MAP<br>2.*MAP<br>3.RETURNFLAG<br>4.QUANTITY<br>5.REVENUE_WO_TAX |

__ 9. Close the Visual Explain window.

## Task 4: Comparing Visual Explain and job log messages for analysis

# Lab 2 answers: Analyzing join methods

## Introduction

The solutions given for the labs are based on our results using V4R5M0 on an iSeries server Model 270 in laboratory conditions. This system has two processors, 2 GB of main storage, 12 disk drives and SMP installed. Some answers may differ when Visual Explain is run in your own iSeries environment.

**Note:** You may be working with smaller tables and for this reason the answer to some questions may be different than the ones published in this document.

## Task 1: Setting up the environment

## Task 2: Nested Loop Join method

*Table 26. Nested Loop Join*

| Question | Answer |
|---|---|
| What was the access method for ITEM_FACT? | Index Scan - Key Positioning |
| What was the access method for PART_DIM? | Index Scan - Key Positioning |
| What was the join method? | Nested Loop with selection |
| What was the join order (positions)? | ITEM_FACT (dial 1) to PART_DIM (dial 2) |
| What was the join type? | Inner Join |
| What index was used to access ITEM_FACT? | ITEM_IDX1 |
| What index was used to access PART_DIM? | *TEMPX0001 |
| If a temporary access path was created, what key fields should be used to create a permanent index? | PARTKEY Ascending |
| Why was a temporary access path created? Use the *flyover* function to find the reason code, and then use associated documentation to find the meaning. | I4<br>Nested Loop Join |

## Task 3: Visual Explain and Hash Join method

## Task 4: Nested Loop 3-way join

*Table 27. Three-way Nested Loop Join questions*

| Question | Answer |
|---|---|
| What was the join method? | Nested Loop Join |
| What was the join type? | Inner Join |
| What was the join order (positions)? | ITEM_FACT (dial 1) to SUPP_DIM (dial 2) to PART_DIM (dial 3) |
| What was the estimated number of rows to be selected from ITEM_FACT? | 26 |
| What was the estimated number of rows to be joined from SUPP_DIM? | 104 |
| What was the estimated number of rows to be joined from PART_DIM? | 416 |

__ 10.Close the Results viewer.

## Task 5: Comparing Visual Explain with job log messages for analysis

# Lab 3 answers: Grouping and Ordering analysis

## Introduction

The solutions given for the labs are based on our results using V4R5M0 on an iSeries server Model 270 in laboratory conditions. This system has two processors, 2 GB of main storage, 12 disk drives and SMP installed. Some answers may differ when Visual Explain is run in your own iSeries environment.

**Note:** You may be working with smaller tables and for this reason the answer to some questions may be different than the ones published in this document.

## Task 1: Setting up the environment

## Task 2: Index Group By

*Table 28. Index Group By*

| Question | Answer |
|---|---|
| What was the estimated number of rows selected? | 576 |
| What method was used for Grouping? | Index |
| What index was used for Grouping? | ITEM_IDX1 |
| What was the average number of rows in each group? | 8704 |

## Task 3: Hash Group By method

*Table 29. Hash Group By*

| Question | Answer |
|---|---|
| What was the Grouping method? | HASH |
| How many rows were in the hash table? | 833 |
| What was the average number of rows in each group of the hash table? | 2 |

## Task 4: Hash Grouping with Nested Loop Join

*Table 30. Hash Group By with NLJ*

| Question | Answer |
|---|---|
| What was the join method? | Nested Loop |

| Question | Answer |
|---|---|
| What was the join order (positions)? | ITEM_FACT (dial 1) to PART_DIM (dial 2) |
| What index was used to access ITEM_FACT? | ITEM_IDX1 |
| What was the Grouping method? | HASH |
| How many rows were in the hash table? | 83 |

## Task 5: Explanation of Index Ordering method

*Table 31. Index Ordering*

| Question | Answer |
|---|---|
| What method was used for Ordering? | Index |
| What index was used for Ordering? | PART_IDX1 |
| What was the reason code for using the index? | I2 - Ordering/Grouping |

## Task 6: Ordering by a Sort method

*Table 32. Order by a Sort*

| Question | Answer |
|---|---|
| What access method was used for PART_DIM? | Table Scan |
| What were the reason code(s) for not using an index to select the records from PART_DIM? | PART_IDX1 - 5 Key columns of index did not match columns specified for Ordering or Grouping<br>PART_IDX2 - 5 |
| What method was used for Ordering? | Sort |

# Lab 4 answers: Analyzing UNIONs

## Introduction

The solutions given for the labs are based on our results using V4R5M0 on an iSeries server Model 270 in laboratory conditions. This system has two processors, 2 GB of main storage, 12 disk drives and SMP installed. Some answers may differ when Visual Explain is run in your own iSeries environment.

**Note:** You may be working with smaller tables and for this reason the answer to some questions may be different than the ones published in this document.

## Task 1: Setting up the environment

## Task 2: Collecting the performance data

## Task 3: Explanation of UNION statement

*Table 33. UNION questions*

| Question | Answer |
|---|---|
| What was the access method for CUST_DIM? | Table Scan |
| What was the access method for SUPP_DIM? | Table Scan |
| What was the estimated number of rows from the UNION merge? | 1600 |
| What was the reason code for the Sort and what does the code mean? | F5<br>UNION specified for the query |
| What was the estimated number of rows selected from the Sort? | 613 |

*Table 34. UNION ALL questions*

| Question | Answer |
|---|---|
| What was the estimated number of rows selected from SUPP_DIM? | 100 |
| What was the estimated number of rows selected from CUST_DIM? | 1500 |
| What was the estimated number of rows from the UNION merge? | 1600 |
| Is there a difference in the total time to run? Why? | YES<br>Time required to perform Sort |

# Lab 5 answers: Complex Query Analysis

## Introduction

The solutions given for the labs are based on our results using V4R5M0 on an iSeries server Model 270 in laboratory conditions. This system has two processors, 2 GB of main storage, 12 disk drives and SMP installed. Some answers may differ when Visual Explain is run in your own iSeries environment.

**Note:** You may be working with smaller tables and for this reason the answer to some questions may be different than the ones published in this document.

## Task 1: Analysis of complex statements using Monitor Data

*Table 35. Final Select*

| Question | Answer |
|---|---|
| What was the optimization time? | 46 milliseconds |
| What was the statement type? | DYNAMIC |
| What were the host variable values? | 2000-10-30<br>JAPAN<br>Manufacturer#3 |
| What was the Ordering implementation? | Sort |
| What was the Grouping implementation? | HASH |
| What was the join implementation? | NESTED LOOP |
| Was the qaqqini options file active for the statement? | NO |
| What was the parallel degree setting? | NONE |

*Table 36. Hash Grouping*

| Question | Answer |
|---|---|
| What was the Grouping implementation? | HASH |
| What was the estimated number of groups? | 83333 |
| What was the estimated number of joined rows? | 232 |
| What was the average number of rows in each group? | 12 |
| How many rows were in the hash table created? | 83333 |

*Table 37. Nested Loop Join*

| Question | Answer |
|---|---|
| What was the number of estimated joined rows? | 232 |
| How many tables were joined? | 5 |

| Table Name | Primary Access Method |
|---|---|
| TIME_DIM | Table Scan |
| ITEM_FACT | Key Positioning |
| PART_DIM | Key Positioning |
| SUPP_DIM | Key Positioning |
| CUST_DIM | Key Positioning |

*Table 38. Table Scan: TIME_DIM*

| Question | Answer |
|---|---|
| What was the estimated processing time? | 1 |
| What is the total number of rows in the table? | 1450 |
| What was the estimated number of rows to be selected? | 145 |
| What was the join position for this table? | 1 |
| Was data space selection used? | Y |
| Was Skip Sequential selection performed? | N |
| Was the creation of an index advised? | N |

*Table 39. Temporary Index: ITEM_FACT*

| Question | Answer |
|---|---|
| How many entries did the index contain? | 600572 |
| Was the index created reusable? | N |
| Was the index created a sparse index? | N |
| What was the type of index created? | BINARY RADIX |
| What were the Key columns for the index build? | ORDERDATE ACEND |

*Table 40. Index Scan - Key Positioning: ITEM_FACT*

| Question | Answer |
|---|---|
| What was the join position? | |
| Was Index Only Access used? | N |
| Did the index fit in memory? | N |
| What was the reason code for the Index Scan? | I4<br>NESTED LOOP JOIN |
| What columns were used for Index Key Positioning? | ORDERDATE |

*Table 41. Index Scan - Key Positioning: PART_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | PART_IDX2 |
| What was the total number of rows in the table? | 20000 |
| What was the estimated number of rows selected? | 2000 |
| What was the estimated number of joined rows? | 58 |
| What was the join position? | 3 |
| Was Index Only Access used? | Y |
| Did the index fit in memory? | Y |
| How many entries did the index have? | 20000 |
| Was the index a constraint? | N |
| Did the optimizer time out? | N |
| Which indexes were considered during optimization? | PART_IDX3 -PART_IDX2 - PART_IDX1 - |
| What was the number of primary key columns? | 1 |
| What columns were used for key positioning? | PARTKEY |

*Table 42. Index Scan - Key Selection: PART_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | PART_IDX2 |
| Was derived selection performed? | Y |
| What columns were used for derived selection? | MFGR |
| What columns were used for key selection? | MFGR |

*Table 43. Index Scan - Key Positioning: SUPP_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | SUPP_IDX1 |
| What was the estimated number of joined rows? | 58 |
| What was the join position? | 4 |
| Was Index Only Access used? | N |
| Did the index fit in memory? | Y |
| How many entries did the index have? | 1000 |
| Was the index a constraint? | N |

*Table 44. DIM*

*Table 45. Temporary Index - CUST_DIM*

| Question | Answer |
|---|---|
| What was the name of the Temporary Index created? | *TEMPX0002 |
| How many entries did the index contain? | 622 |
| Was the index created reusable? | N |
| Was the index created a sparse index? | Y |
| What was the type of index created? | BINARY RADIX |
| Was the index created from an existing index? | N |
| What was the reason for the index scan? | I4<br>Nested Loop Join |
| What were the Key columns for the index build? | CUSTKEY ASCEND |

*Table 46. Index Scan - Key Positioning - CUST_DIM*

| Question | Answer |
|---|---|
| What was the name of the index used? | *TEMPX0002 |
| What was the number of columns used for key positioning? | 1 |
| Was Index Only Access used? | N |
| Did the index fit in memory? | Y |
| Was derived selection performed? | Y |
| What column was used for derived selection | COUNTRY |
| What columns were used for Index Key Positioning? | CUSTKEY |
| What was the join position for this table? | 5 |

## Task 2: Analysis of complex statement using a job log

*Table 47. Analysis of job log information*

| Question | Answer |
|---|---|
| What was the Ordering implementation? | Sort<br>CPI4325 |
| What was the Grouping implementation? | HASH<br>No messages in job log indicate INDEX GROUPING |
| What was the join implementation? | NESTED LOOP |
| What was the estimated number of groups? | Unable to determine from job log |

| Question | Answer |
|---|---|
| What was the estimated number of joined rows? | Unable to determine from job log |
| What was the average number of rows in each group? | Unable to determine from job log |
| What were the Grouping columns used for the hash table? | Unable to determine from job log |
| What columns were used for the hash table? | Unable to determine from job log |
| How many tables were joined? | 5 |
| What was the join order? | TIME_DIM (dial 1)<br>ITEM_FACT (dial 2)<br>PART_DIM (dial 3)<br>SUPP_DIM (dial 4)<br>CUST_DIM (dial 5) |
| What access methods were used for:<br>ITEM_FACT?<br>PART_DIM?<br>TIME_DIM?<br>CUST_DIM?<br>SUPP_DIM? | <br>Index Scan - KRP via Temp Index<br>Index Scan - KRP / Key Sel<br>Table Scan<br>Index Scan - KRP via Temp Index<br>Index Scan - KRP |
| Is it possible to determine the number of rows joined from each table? | NO |
| Were any Temporary Access Paths built? | YES<br>ITEM_FACT -600572 entries-ORDERDATE<br>CUST_DIM-622 entries-CUSTKEY (Sparse) |
| Did the Index Advisor recommend any indexes to be built? | YES<br>CUST_DIM - Key COUNTRY<br>PART_DIM - Key MFGR |