IBM Software
Information Management Skills Development

DB2 SQL Query Tuning with BLU Acceleration II

Version/Revision# **1**

Module ID **10304**

Length **2 hours**

IBM

Table of Contents

1 Introduction 3

2 Suggested Reading 3

3 Lab Preparation 4

3.1 Start Virtual Machine and DB2 4

4 Optim Query Workload Tuner Lab 6

4.1 Preparing the Database 6

4.2 Tune a Workload 9

4.2.1 Launching Data Studio 9
4.2.2 Create a Optim Query Tuner Project 11
4.2.3 Run Workload Advisors 15

4.3 Access Path Explorer and Access Graph 20

4.3.1 Run Single Query Advisors 25

4.4 Optimization Profiles 29

4.4.1 Create an Optimization Profile 29
4.4.2 Create a New Join Sequence 30
4.4.3 Set Access Path to IXSCAN 33
4.4.4 Validate Optimization Profile 34
4.4.5 Compare the Access Plan change 34
4.4.6 Deploy Optimization Profile 35

5 SQL Performance Tuning – Clean Up 37

6 *Compare query processing for row-organized tables to column-organized tables lab* 38

6.1 Lab Preparation 38

6.1.1 Initialize Environment 38

6.2 What this exercise is about 38

6.2.1 Query processing for BLU and non-BLU access plans 38

1 Introduction

DB2's performance monitoring framework has been improved in DB2 10.5. Historically DB2's monitoring utilities and interfaces have been based on the monitoring elements presented by the Snapshot Monitoring stream. DB2's Health monitor, snapshot commands, administrative views and table functions all utilize the Snapshot Monitoring stream of monitoring elements.

DB2 V10 brings along with it a new and full complement of monitoring table functions and administrative views that enable you to take advantage of the monitoring framework. The monitoring framework is robust, light weight and features many identical or similar monitoring elements as the Snapshot monitoring.

This lab is intended to teach you the essentials to monitoring a DB2 database system. The scope of a "DB2 database system" extends capabilities through the Enterprise Edition, and up to and including version 10.5. This module is intended for someone who has some experience with DB2 and wishes to broaden their knowledge in the area of performance-related monitoring.

These labs are performed on a VMware image of a 64bit SUSE Linux machine with minimal resources declared. There are some commands used in the lab that are typically found on Linux/Unix systems, however basic Unix command-line tools can be downloaded and easily employed on Windows-based machines at no cost. Additionally, comparable commands are available in the Windows environment, like find in place of grep.

There are two main tips that will make the lab progress better.

1. Use the up arrow key to recall previous commands. These commands can be edited and re-executed as needed.
2. Follow all the instructions, don't skip steps. These labs have been created so that they can be rerun but to avoid that please follow all instructions.

2 Suggested Reading

Information Management Technical Library at developerWorks
Contains articles and tutorials and best practices
http://www.ibm.com/developerworks/views/db2/library.jsp
http://www.ibm.com/developerworks/data/bestpractices

An Expert's Guide to DB2 Technology
Great read and useful information.
http://blogs.ittoolbox.com/database/technology

DB2 10 Fundamentals Certification Study Guide
Learn the basics and get ready for certification

3 Lab Preparation

Before getting started with the lab exercises you will set up you're environment then work through a series of exercises focused on DB2 performance monitoring techniques.

3.1 Start Virtual Machine and DB2

You may skip this section if you have started your VM and db2 instance in a previous lab or exercise.

1. Choose the **DB2_BLU-AWSE_10.5....vmx** file when first opening the lab image in the **/DB2_BLU_PerfMonitoring...** folder and run it.

Figure 1 – Opening a VMware Image File

2. If this is the first time this virtual machine has been started, a series of licensing agreements will prompt you. To proceed, acknowledge by responding by selecting the "**`Yes, I Agree to the License Agreement`**" selection.

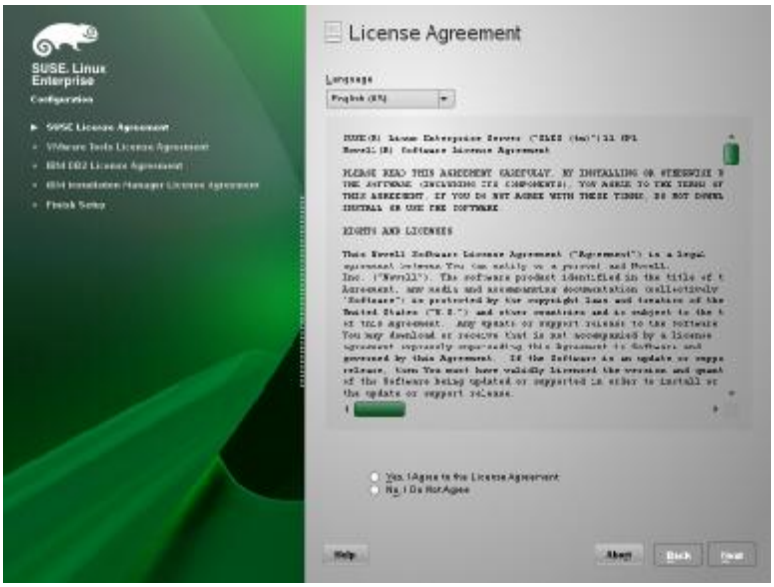


Figure 2 – License Agreements

3. Login in at the console prompt as **db2inst1**. You will be prompted for the password which is **password**. You may be prompted for the root **password** as well, if so, it is also password.

Figure 3 – Login credentials

4. Open a terminal window as follows:
   - Open a new terminal window by right-clicking on the **Desktop** and choosing the "**Open Terminal**" item:

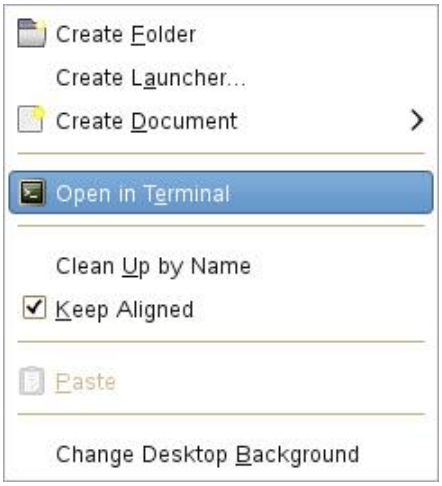


Figure 4 – Opening a Terminal

From the new terminal session start the Database Manager as follows:

- Enter the command "**db2start**" to start the database instance. Make sure the instance is started successfully. (**Note**: If you notice a message that says "The database manager is already active", that's "OK. Please continue to the next step.)
- In order to enable the aliases defined for this lab, copy the /home/db2inst1/labbashrc to /home/db2inst1/.bashrc and source that file.

```
cd ~
cp labbashrc .bashrc
. .bashrc
```

4 Optim Query Workload Tuner Lab

In this part of the lab, you will work with the Optim Query Tuner graphical tool with DB2 10. You will run a workload on the database server and use advisors to obtained recommendations indexes, statistics and views are suggested to improve the workload performance. The recommendations received will be applied to the database server and the workload rerun.

4.1 Preparing the Database

1. Open a new terminal window by right-clicking on the **Desktop** and choosing the "**Open Terminal**" item:

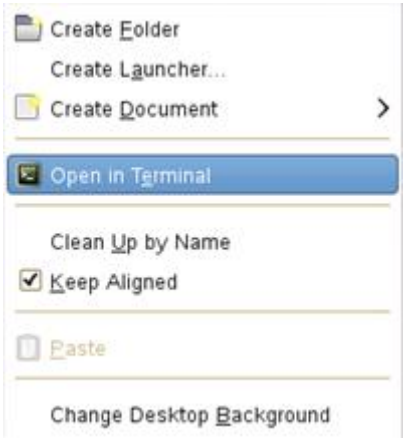


Figure 2 – Opening a Terminal

2. Use the terminal window directory to prepare the database with script oqwt01.sh located in /home/db2inst1/Documents /LabScripts/db2pt/querytuning/oqwt. Run the following command:

```
oqwtdir
./oqwt01.sh
```

After some time, the following should display, indicating that everything has been processed successfully:

```
------------------------------------------------------------------------------
Run EXPLAIN.DDL in the SAMPLE database
------------------------------------------------------------------------------

   Database Connection Information

 Database server        = DB2/LINUXX8664 10.5.3
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

DB20000I  The SQL command completed successfully.
------------------------------------------------------------------------------
Applying OQWT License to the SAMPLE database
------------------------------------------------------------------------------

   Database Connection Information

 Database server        = DB2/LINUXX8664 10.5.3
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE

DROP FUNCTION DB2OE.QT_LIC
DB20000I  The SQL command completed successfully.

CREATE  FUNCTION  DB2OE.QT_LIC()  RETURNS  VARCHAR(255)  LANGUAGE  SQL  CONTAINS  SQL  NO  EXTERNAL  ACTION
            DETERMINISTIC  RETURN VARCHAR('wVLwF/a4OBBLTyNX4bsUKw==')
DB20000I  The SQL command completed successfully.

GRANT EXECUTE ON FUNCTION DB2OE.QT_LIC TO PUBLIC WITH GRANT OPTION
DB20000I  The SQL command completed successfully.

DB20000I  The SQL command completed successfully.
------------------------------------------------------------------------------
Note: You have applied OQWT license to the SAMPLE database
```

4.2 Tune a Workload

4.2.1 Launching Data Studio

1. Open IBM Data Studio by double-clicking on the IBM Data Studio icon on the Desktop.

Figure 3 – Open IBM Data Studio

2. If this is the first time Data Studio is launched, it may take a moment. Click **OK** to accept the default workspace and check the option "**Use this as the default and do not ask again**".

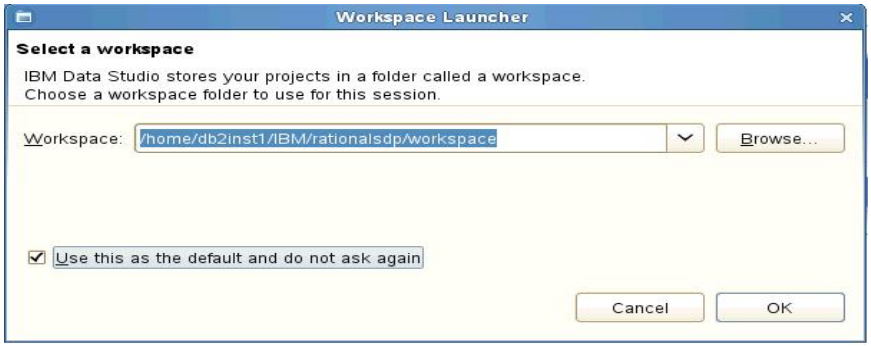


Figure 4 – Select a workspace

3. By default, the Database Administration perspective is opened. Your workspace should look like the following:

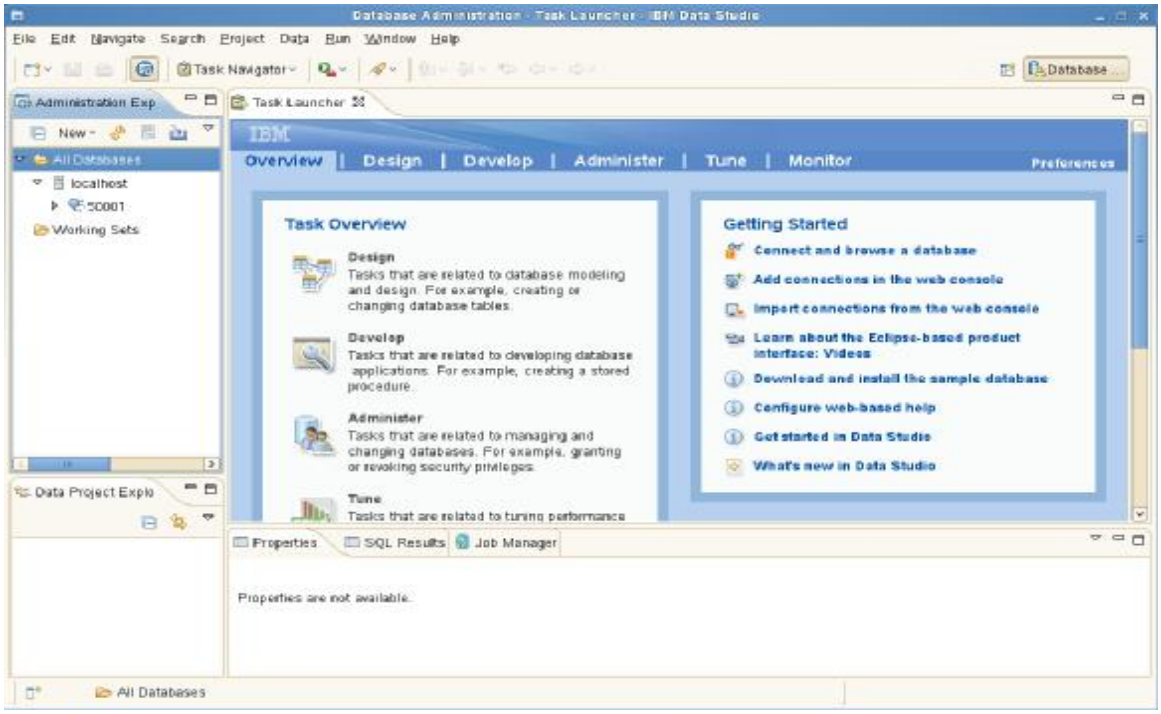


Figure 5 – Database Administration perspective

4. We want to change to the ***IBM Query Tuning*** perspective. To do this, click on the "**Open Perspective**" icon and click **Other**, as shown below.

Figure 6 – Open perspective

Click on IBM Query Tuning and click on the **OK** Button.

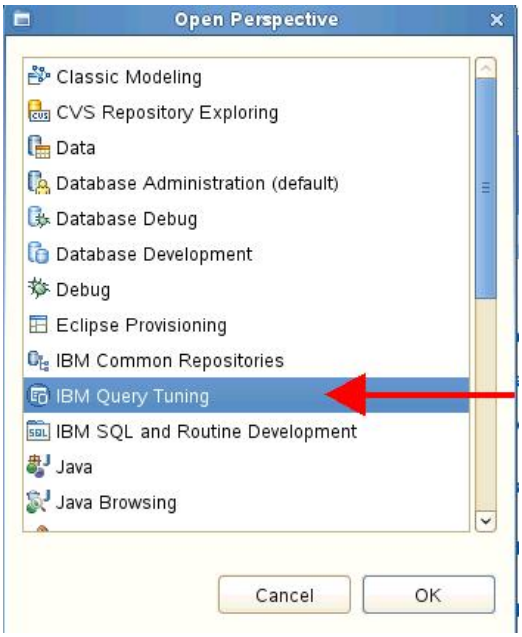


Figure 7 – IBM Query Tuning Perspective

4.2.2 Create a Optim Query Tuner Project

1. Move your mouse pointer to the **Project Explorer** section, right-click and select **New**, and then **Project…**

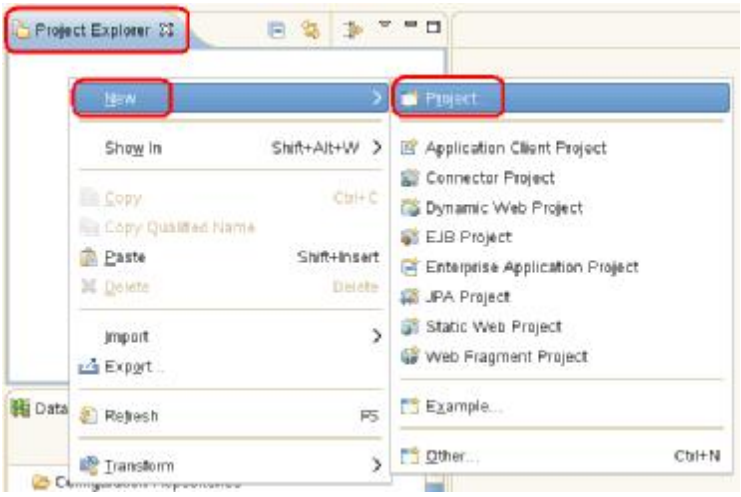


Figure 8 – Create New Project

2. Select **Query Tuner Project** from the wizard list and click **Next.**

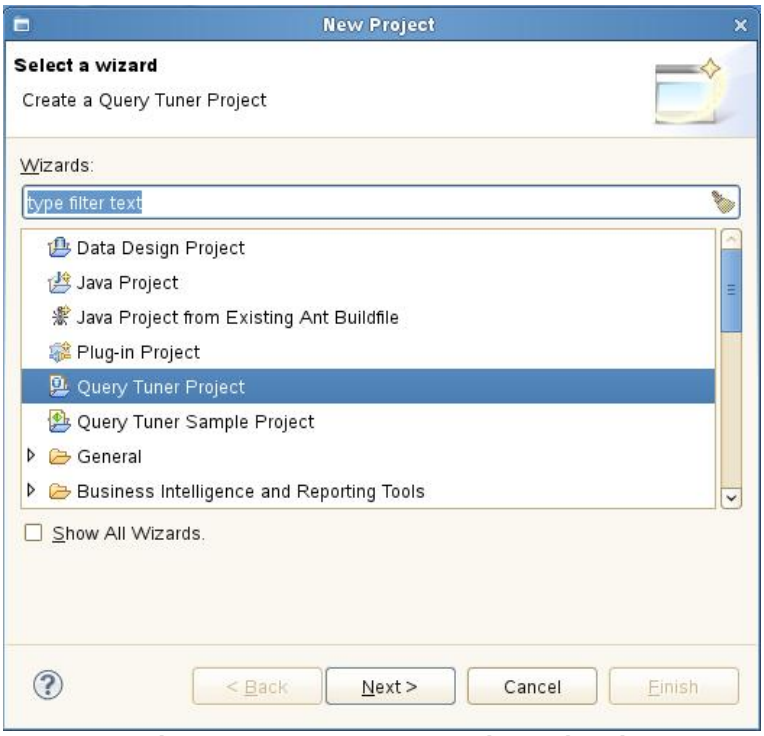


Figure 9 – Query Tuner Project Wizard

3. Leave the **Project Name** as suggested in the wizard and click **Next.**

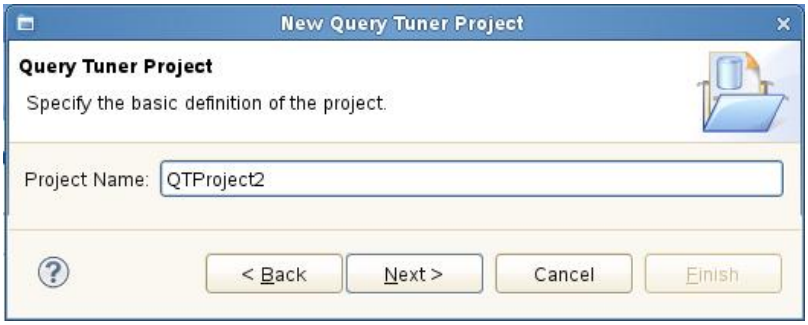


Figure 10 – Query Tuner Project Name

4. Select the **SAMPLE** database for the connection and click **Next.** If necessary create connection following steps 6 to 8 in Lab 3.3 of last module's workbook. Remember use SAMPLE as database name. Then click **Finish.**

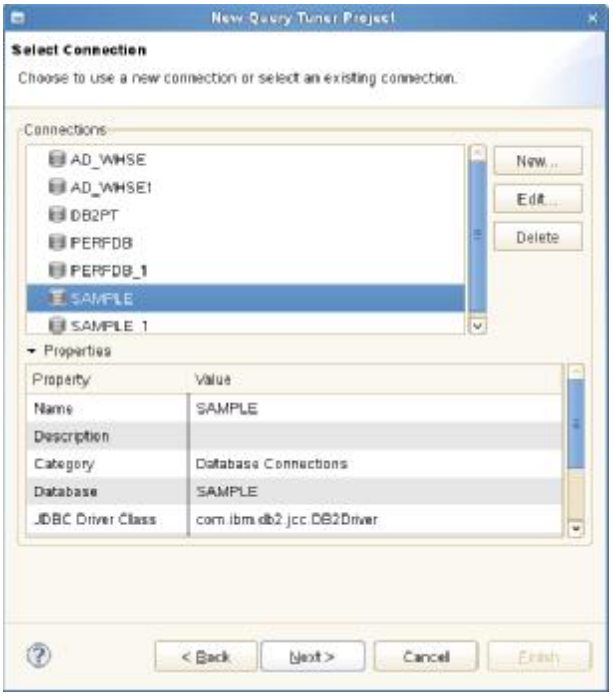


Figure 11 – Select Database connection

5. In this lab, we will use a workload file, with multiple SQL statements. First select **File** and click **Browse.**

Figure 12 – Capture SQL Sentences

6. Select the file **OQWT.sql** located in `/home/db2inst1/Documents/LabScripts/db2pt/querytunning/oqwt`, and click **OK.**

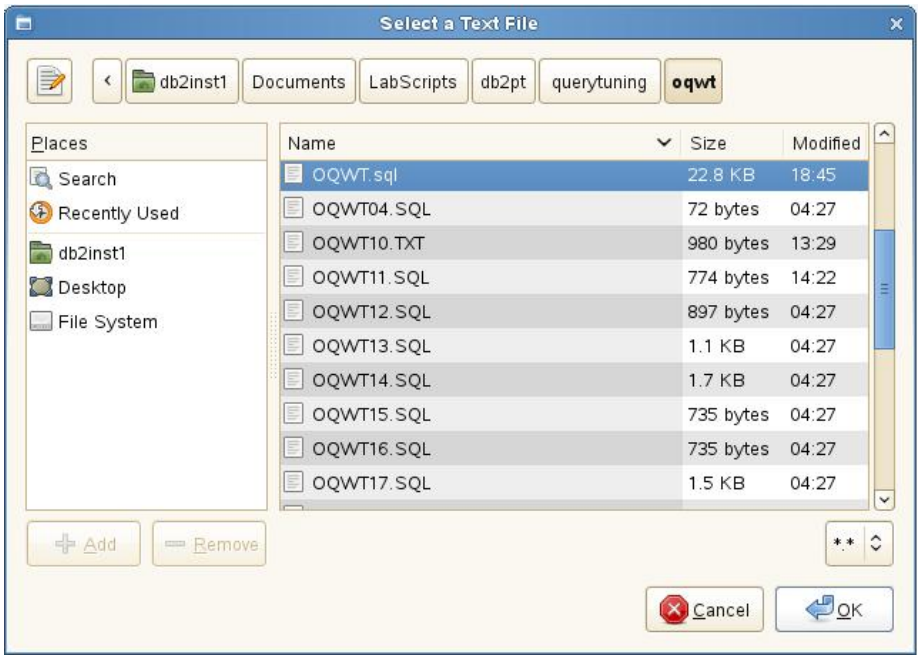


Figure 13 – Select query workload file

7. Use **";"** as statement delimiter, click **Capture**. Once the statement appears on screen, click **Save All to Workload…**

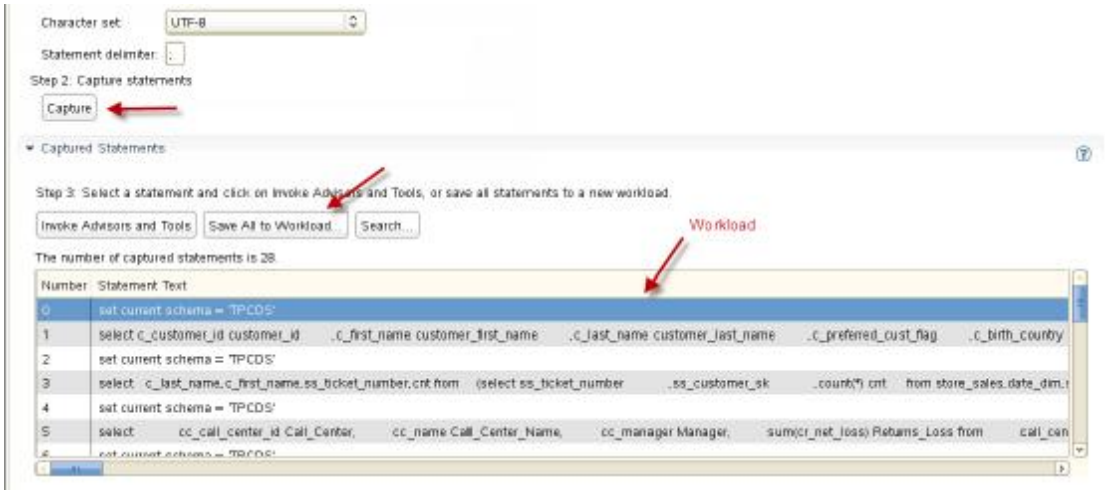


Figure 14 – Capture workload

8. Accept the default workload name and click **OK.**

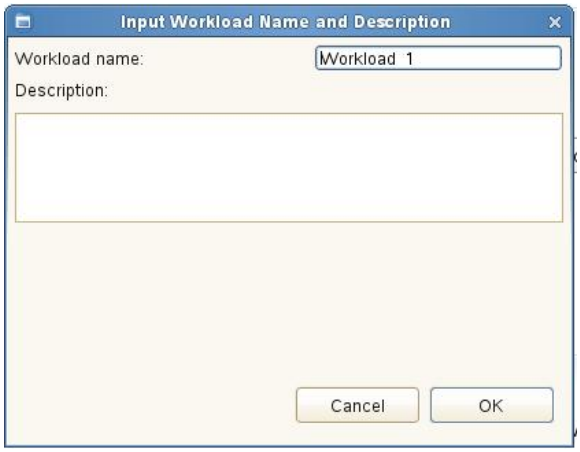


Figure 15 – Workload Name

4.2.3 Run Workload Advisors

1. Click **Invoke Advisors** to call workload advisors and tools.

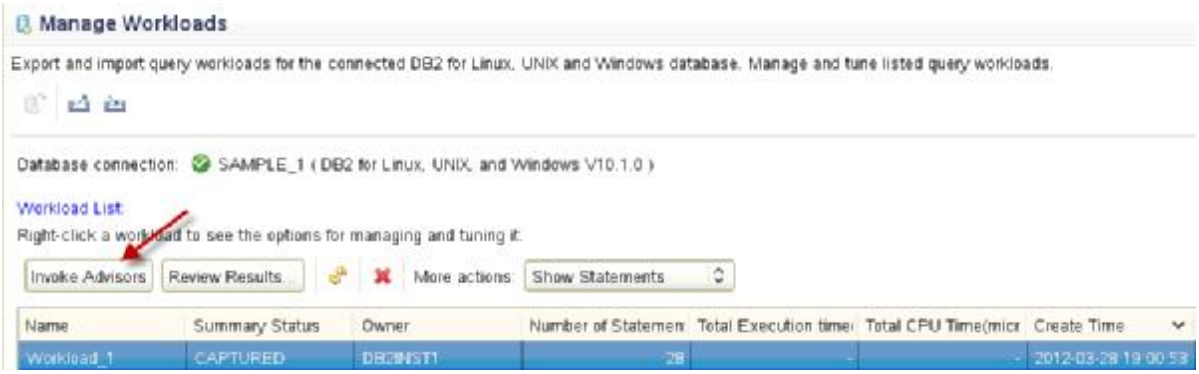


Figure 16 – Invoke Advisors

2. Check **Re-collect EXPLAIN information** before running workload advisors. Click **Select What To Run…**. Click **Select All**, and click **OK.**

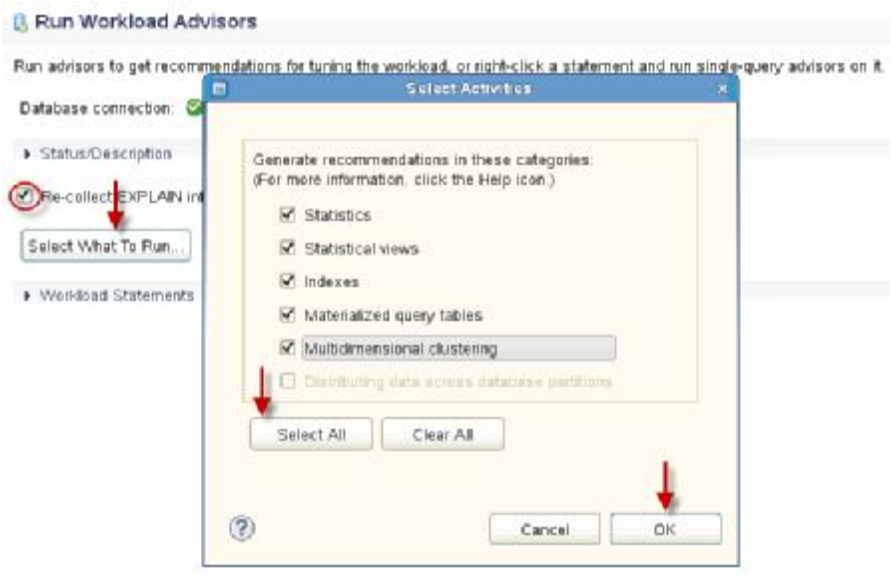


Figure 17 – Select categories for recommendation

3. Specify the **TPCDS** schema and click **Start EXPLAIN.**

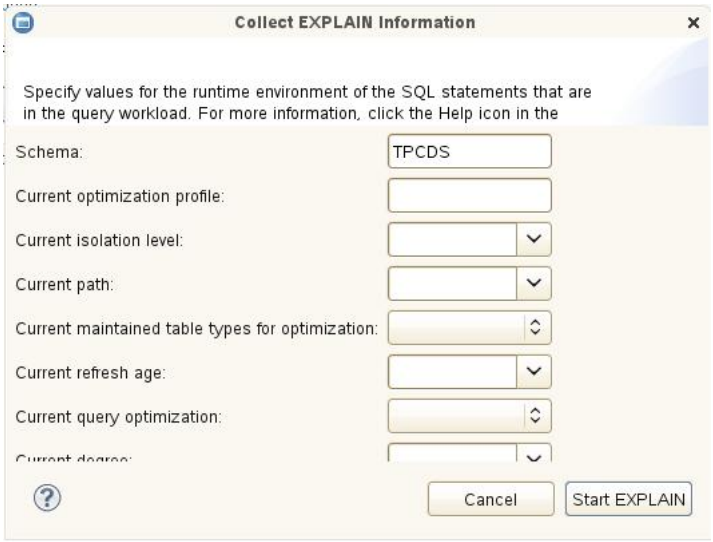


Figure 18 – Specify Explain Values

4. There may be Unexplained Statements such as “set current schema= ‘TPCDS’”, Click **Yes** to continue.

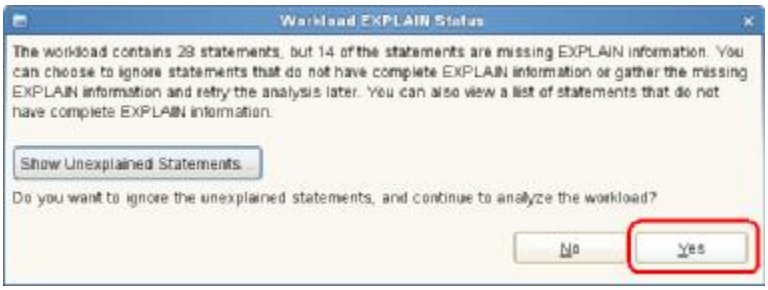


Figure 19 – Status

5. Workload advisors will start working on the workload. Please wait for this to finish.

Figure 20 – Running advisors

6. In case if an error window is displayed, press OK.

7. After this, the recommendations are presented. We'll review them in the next few steps.

Figure 21 – Recommendation Summary

8. Click the **Statistics** tab and click **View RUNSTATS**.

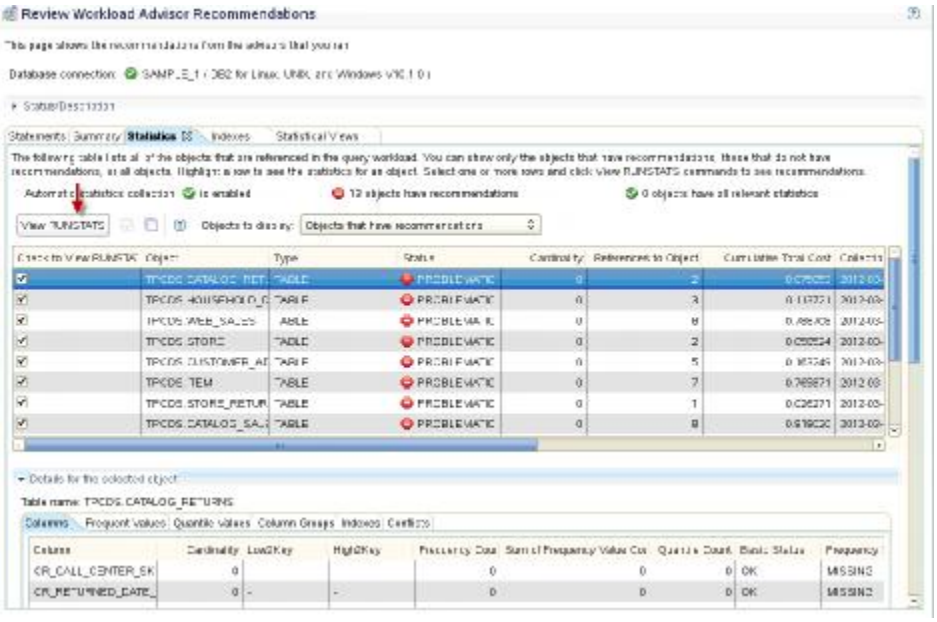


Figure 22 – Statistics Tab

9. Review the recommendations, in particular Merged RUNSTATS. Click **Next**.

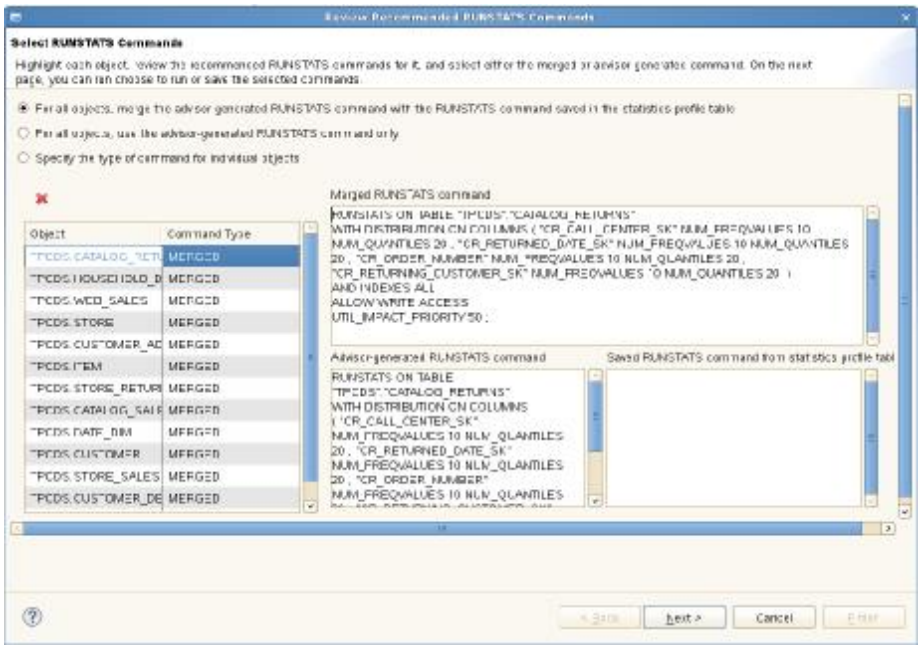


Figure 23 – Statistics Recommendations

10. Click **Finish** to execute the RUNSTATS COMMANDS on the database. Click **OK** in the confirmation window.

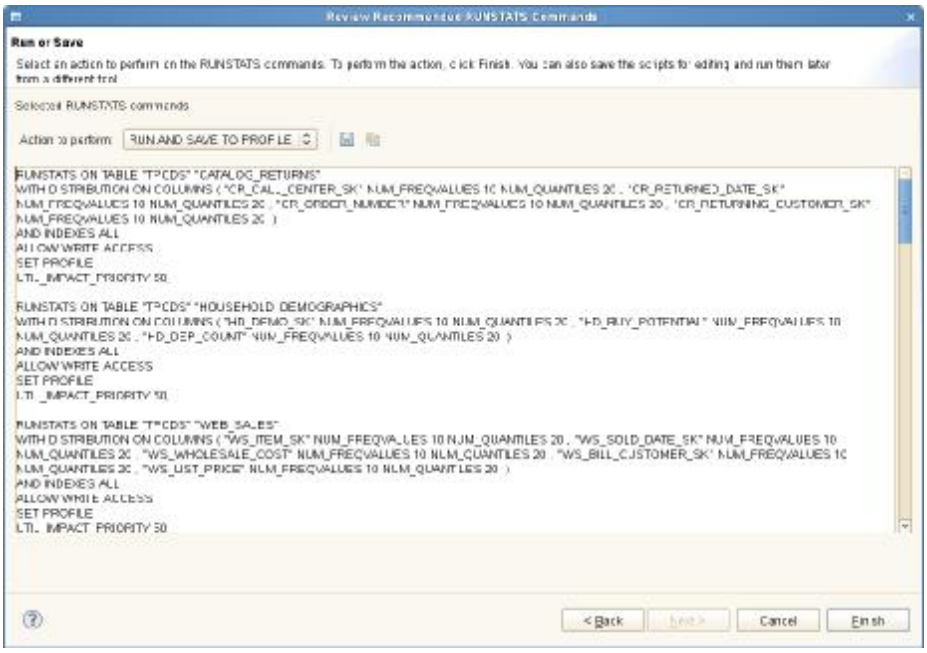


Figure 24 – RUNSTATS Commands

Note: In DB2 10, a number of improvements have been made to the RUNSTATS command to make statistics gathering faster. The command parameters have also been simplified.

11. Click the **Indexes** tab to explore recommendations for indexes.

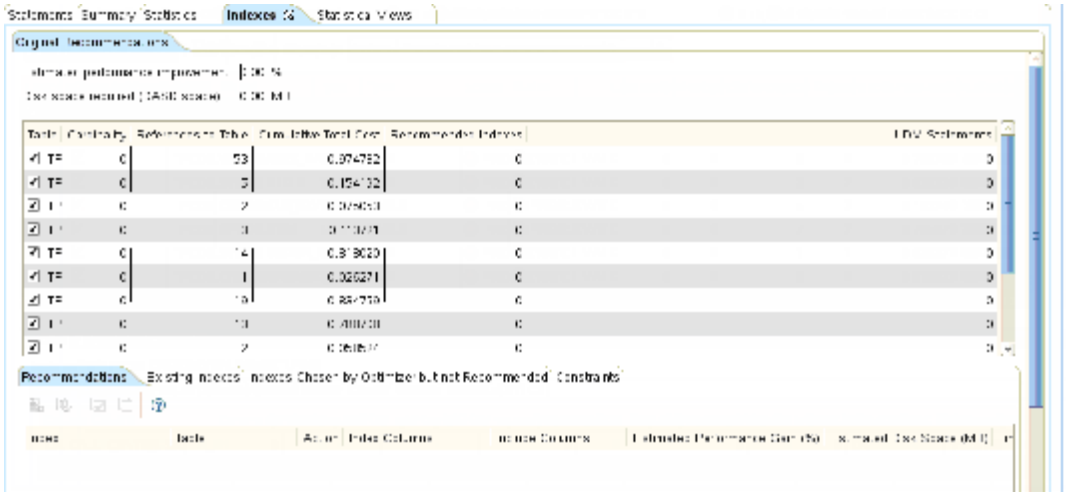


Figure 25 – Indexes Recommendations

12. Click **Statistical Views** tab to see the recommendations. The use of statistical view helps to improve the performance of queries and OQWT helps to determine these views based upon the workload. It is not an easy task to do by hand.

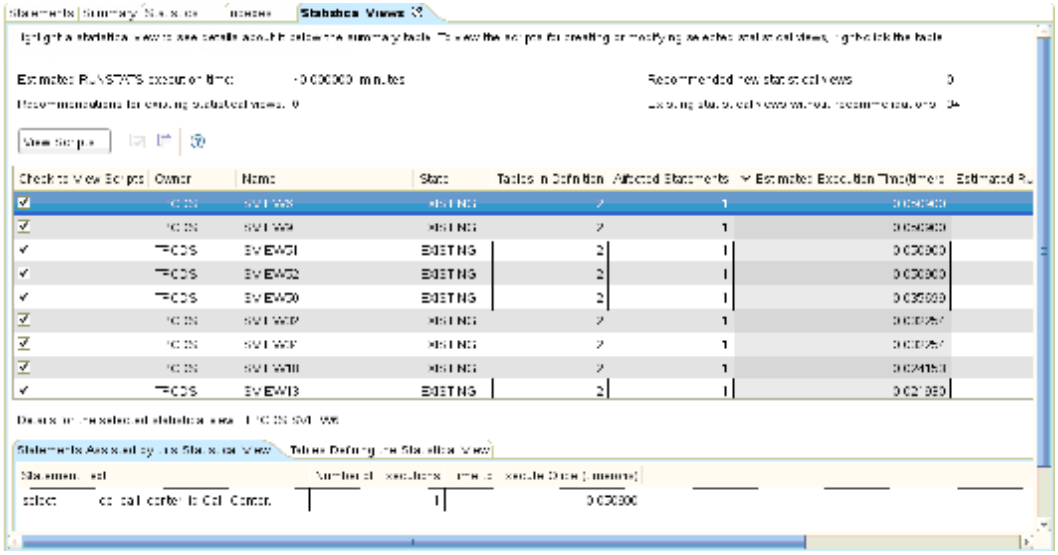


Figure 26 – Statistical Views Recommendations

13. Close project tab as shown, **Save** changes and **Exit**.

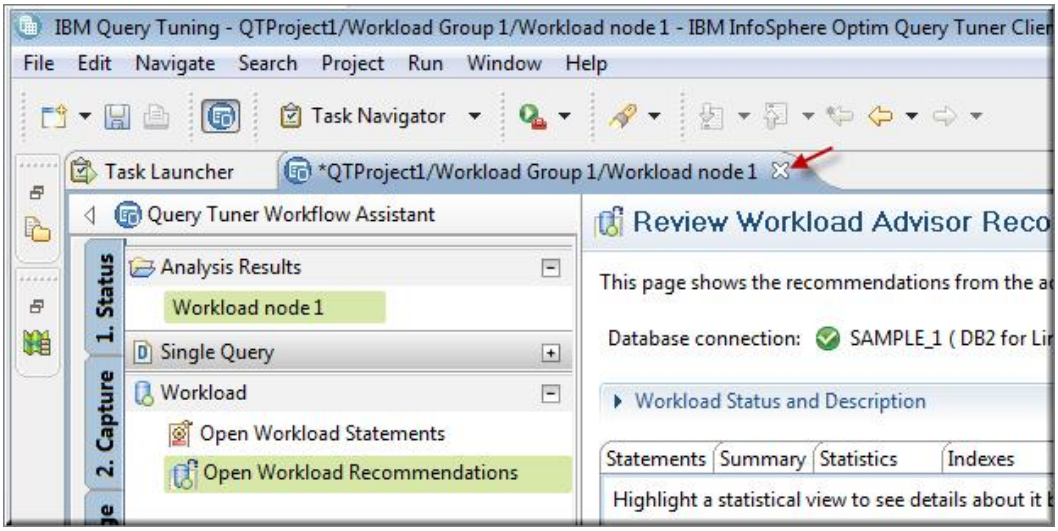


Figure 27 – Close Project Tab

4.3 Access Path Explorer and Access Graph

1. Open a new terminal window by right-clicking on the **Desktop** and choosing "**Open in Terminal.**":

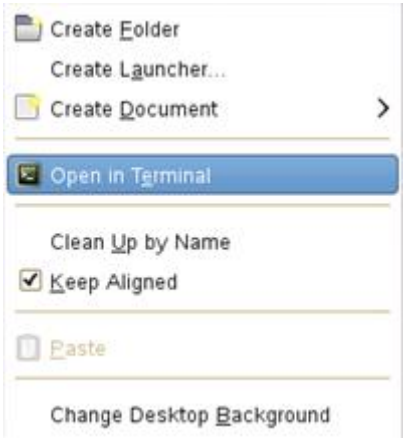


Figure 28 – Opening a Terminal

2. Run the oqwt03.sh script located in directory /home/db2inst1/Documents/LabScripts/db2pt/querytuning/oqwt to create two tables, wait for a message stating that it "Successfully Executed".

```
oqwtdir
./oqwt03.sh
```

3. Back in Data Studio, right-click on the QTProject1 and select **Create Query Group**. [We saved this project in the previous section.]

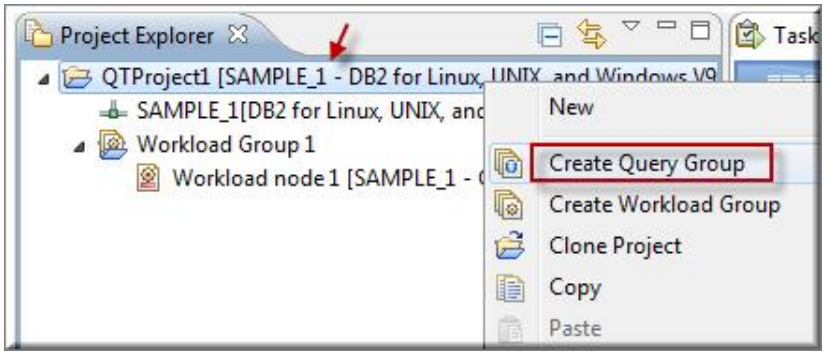


Figure 29 – Create Query Group

4. Accept the default name and click **OK**.

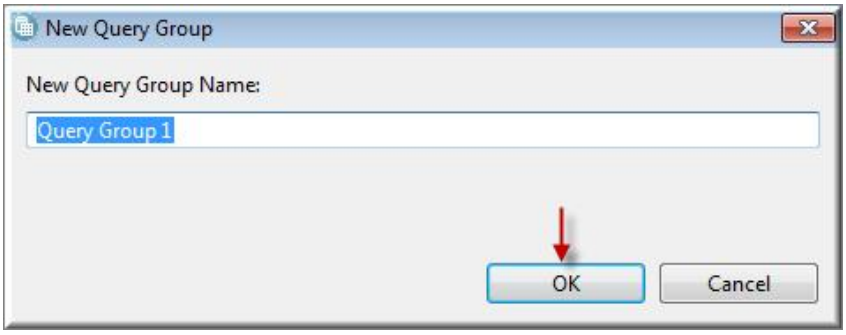


Figure 30 – Default Query Group name

5. Right click on Query Group X, where X is the generated group number, and select the option **Tune Query**.

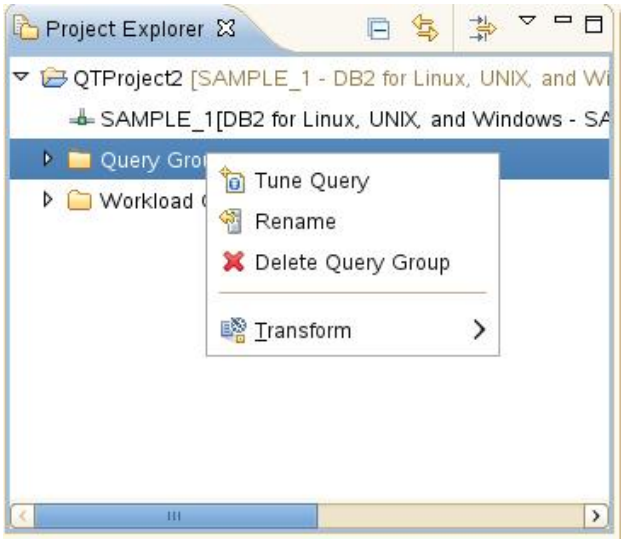


Figure 31 – Tune Query

6. As in the previous section we need a file with a SQL statement to optimize Click **File,** and then click **Browse**.

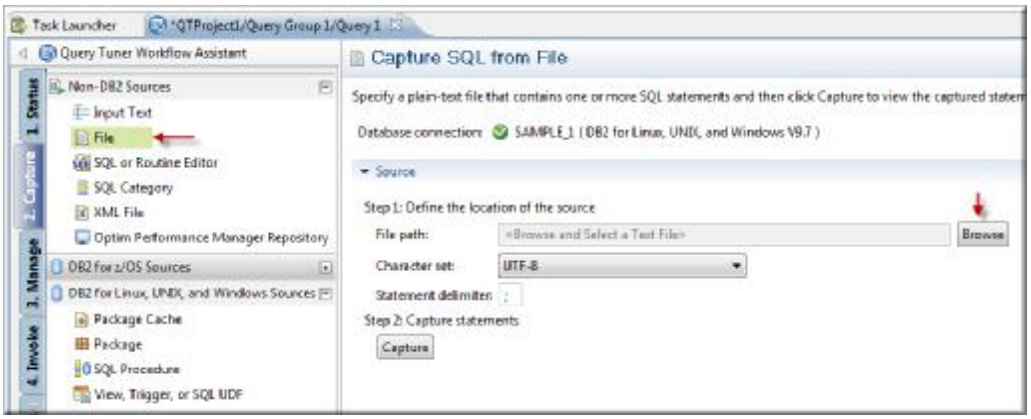


Figure 32 – Select file with SQL

7. Select the file **OQWT04.sql** located in `/home/db2inst1/Documents/LabScripts/db2pt/querytunning/oqwt`. Click **OK**, use **";"** as statement delimiter and click **Capture.** Once the statement appears on screen, click **Save All to Workload…**

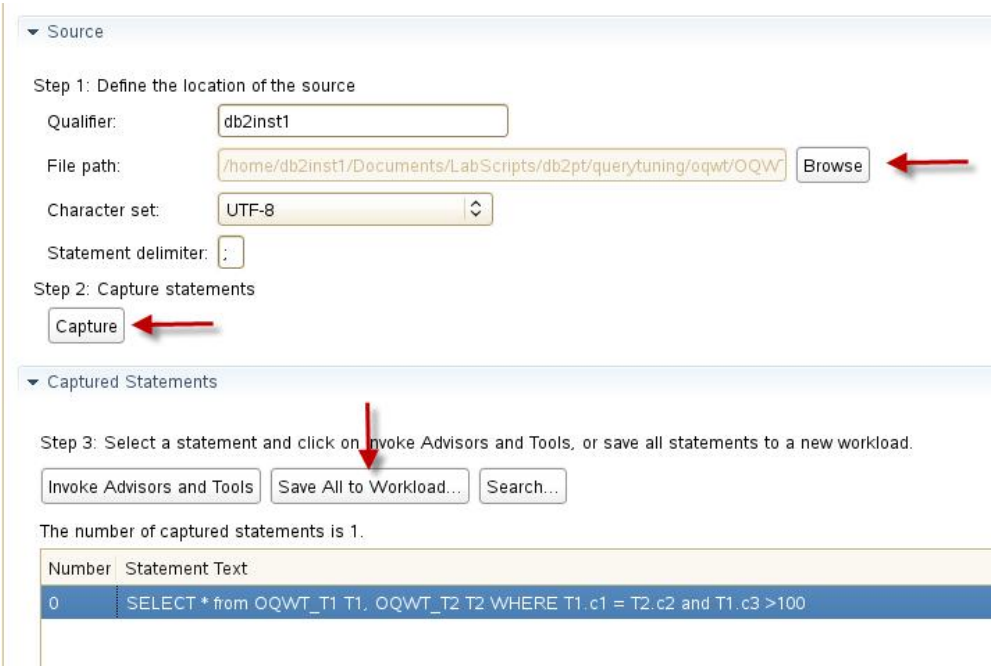


Figure 33 – Capture SQL from a File

8. Select workload name you just saved in the previous step, double click to open.

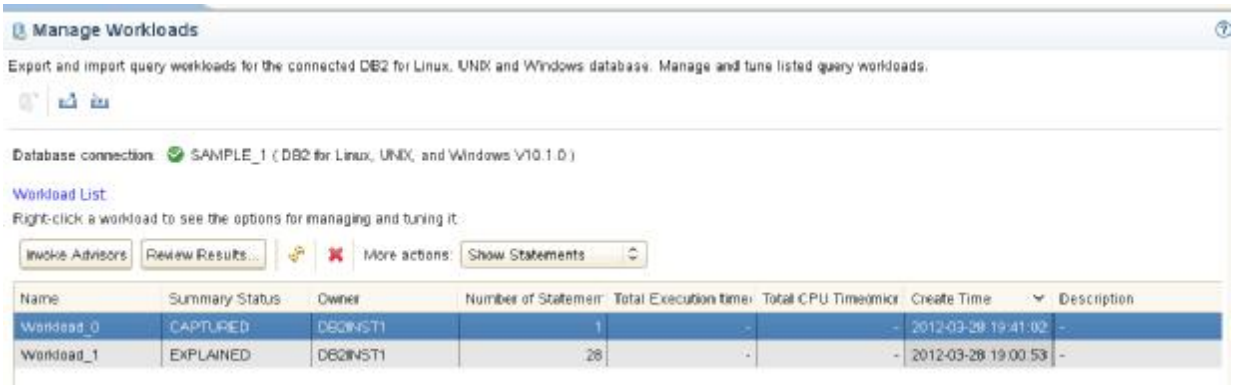


Figure 34 – Manage Workload

4.3.1 Run Single Query Advisors

1. Click the vertical tab **"2. Capture"** so that we can run advisors. Click **Invoke Advisors and Tools**.

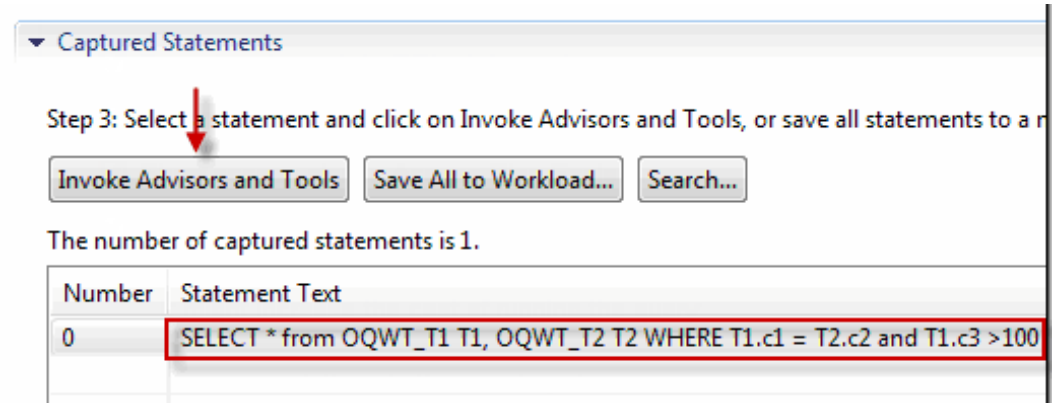


Figure 35 – Invoke Advisors

2. Click **Select What To Run…**

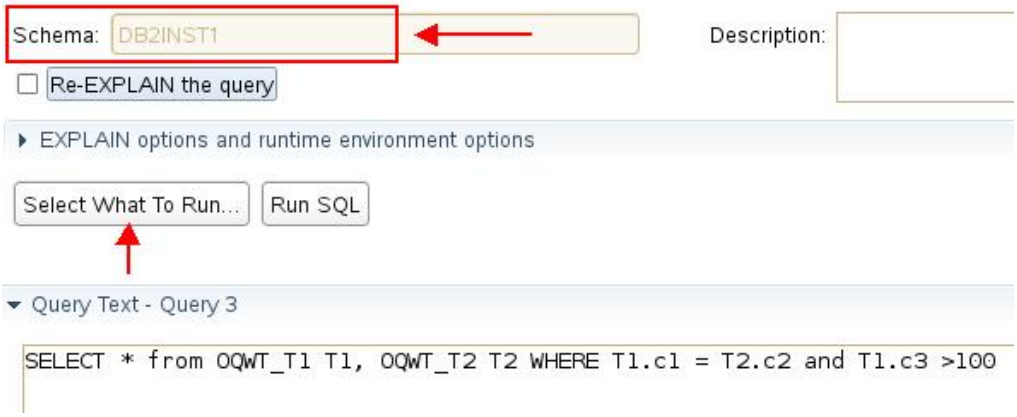


Figure 36 – Invoke Advisors

3. Click **Select All,** then **OK.**

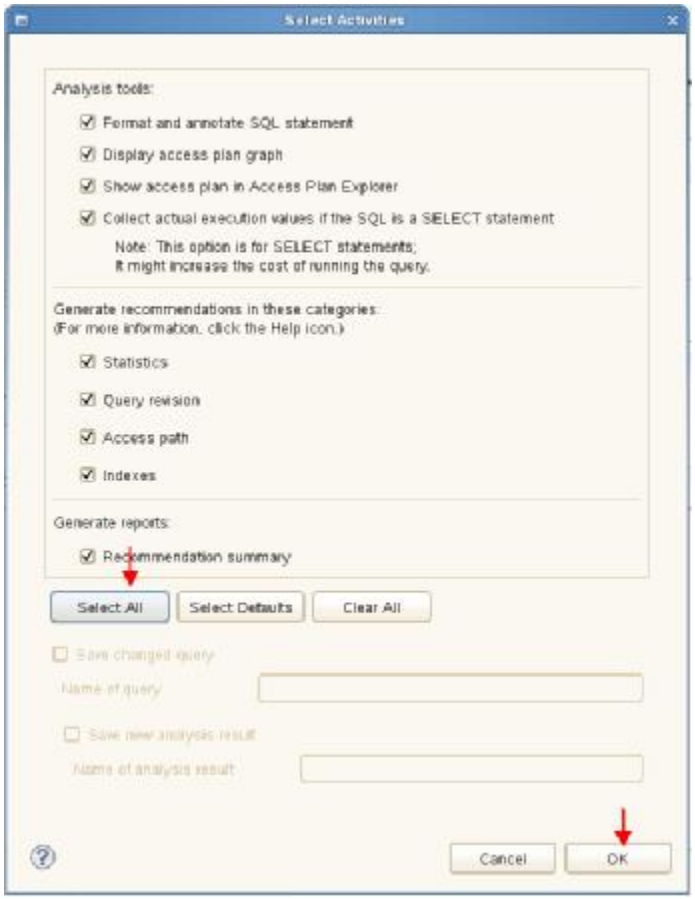


Figure 37 – Select Activities

4. The single query tune advisor will start to run. Please wait for this to finish.

Figure 38 –Tuning Query

5. Click **Open Access Plan Explorer.**

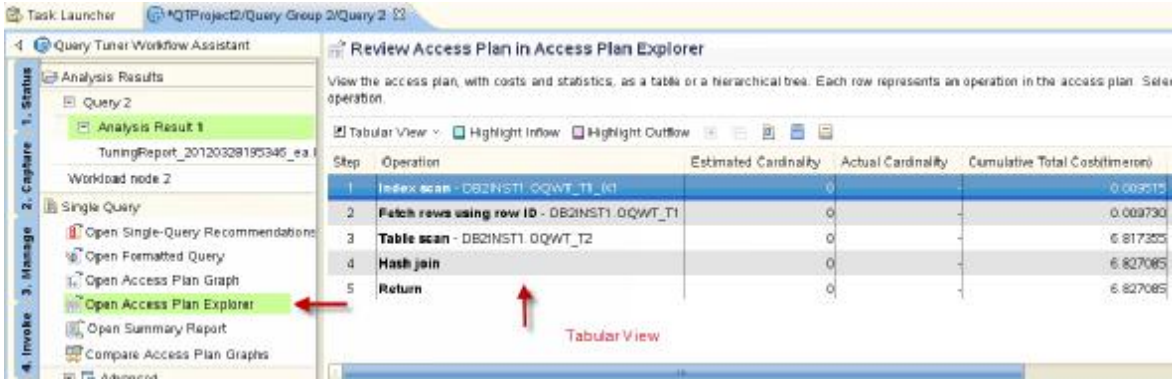


Figure 39 – Access Plan Tabular View

6. Click **Open Access Plan Graph.**

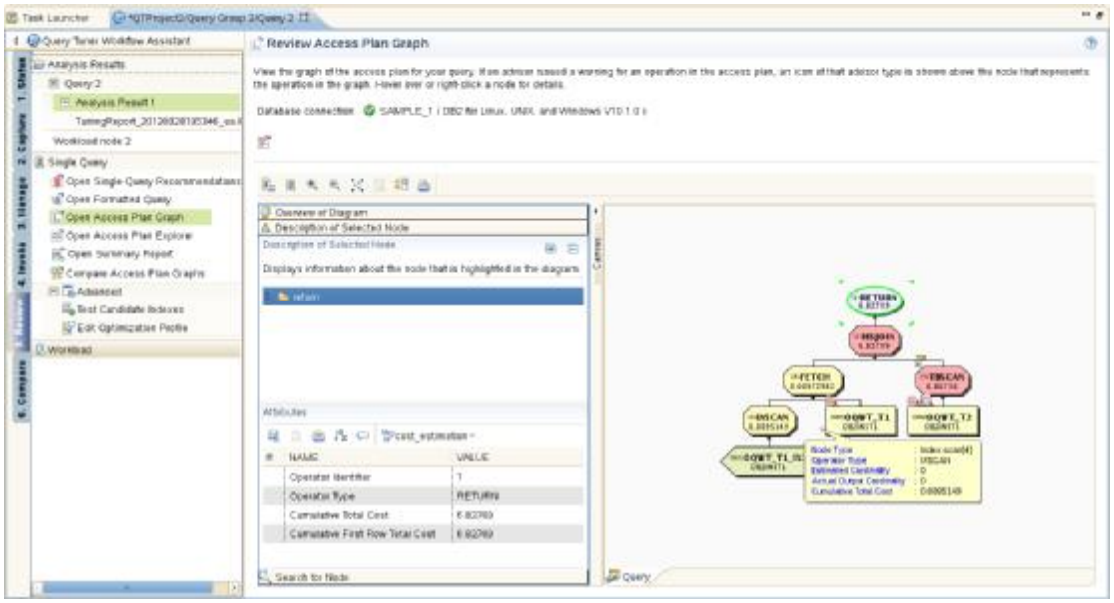


Figure 40 – Access Plan Graph View

7. Notice in the access plan graph, there is a HASH JOIN between a full table scan on the table OQWT_T2 and IXSCAN on the table OQWT_T1. We will take this as an example and show you how to create an optimization profile to influence the query access path.

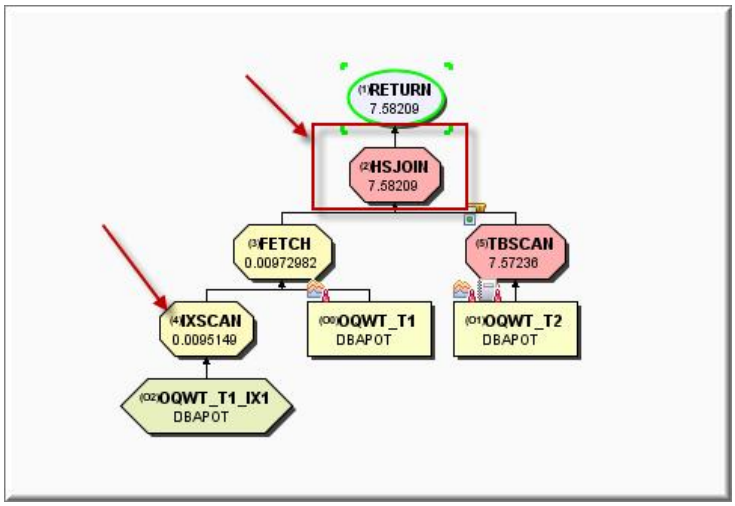


Figure 41 – Access Plan Graph Detail

4.4 Optimization Profiles

An optimization profile is a DB2 feature that allows a DBA to provide a "hint" for an SQL statement. Normally, the DB2 optimizer does not require "hints", but it may be necessary to influence the optimizer to do things in a particular fashion ONLY for a particular query.

To set up the optimization profile, we will use the access plan graph of the query we saw in the previous section.

We will create an optimization profile using the following requirements.

1. OQWT_T1 table to be accessed with IXSCAN and it should use index OQWT_T1_IX1
2. OQWT_T2 must be a leading table in the join sequence and the join method for OQWT_T1 must be nested-loop join and not the hash-join shown in the access plan graph

4.4.1 Create an Optimization Profile

1. Click vertical tab "**4. Invoke."** Click **Create Optimization Profile.**

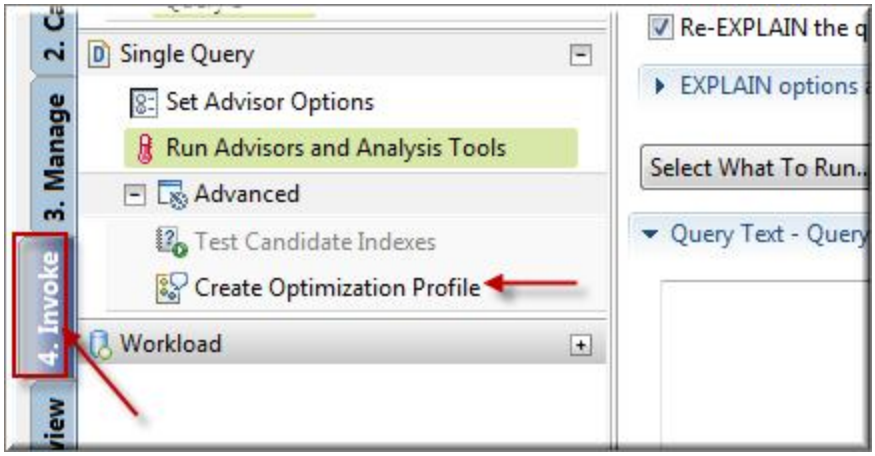


Figure 42 – Create Optimization Profile

2. Please notice that the Plan-Hint 1 is created. You will see two panes on the right-hand side. One of them is the Join Diagram, based on existing access plan. The other is the Editable Join Sequence Diagram. We will work from the second graph window.

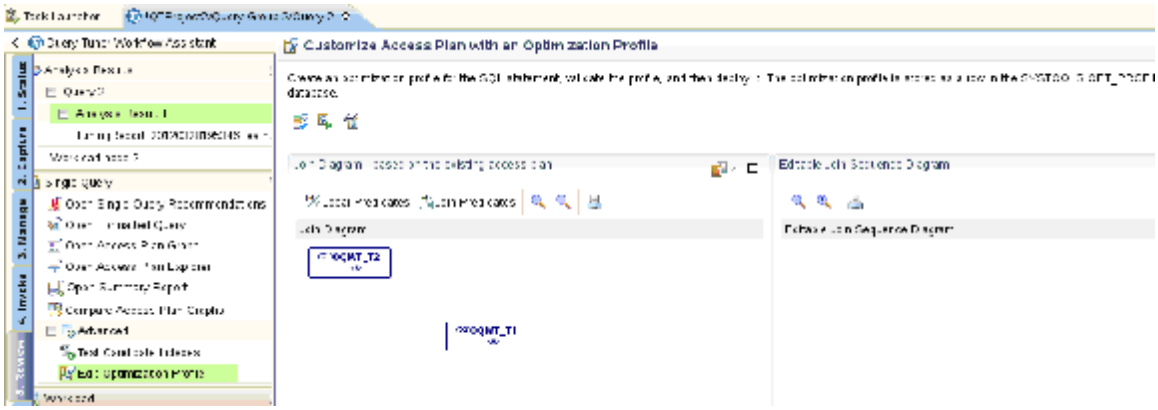


Figure 43 – Create Plan Hint

4.4.2 Create a New Join Sequence

1. Right-click anywhere in the Editable Join Sequence Diagram. Select **Add Join as Child node.**

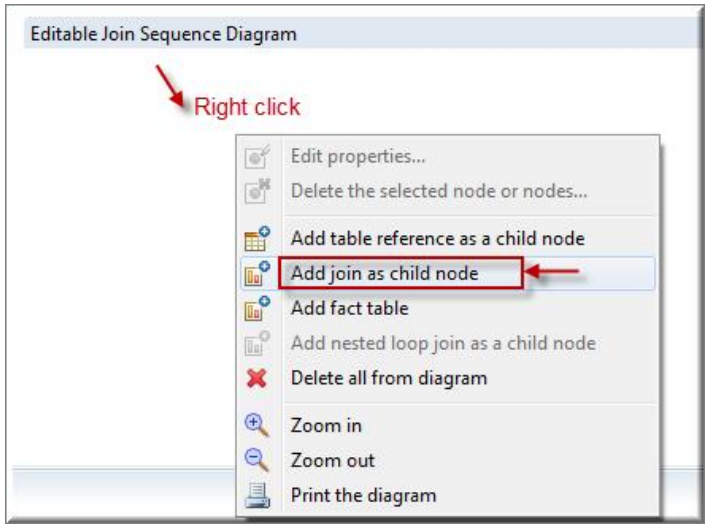


Figure 44 – Adding Join

2. Click the **Join Type** dropdown, select **NLJOIN** and click **OK**.

Figure 45 – Join Type

3. An NLJOIN box is created in the diagram window. Click to select it. Right-click and select **Add table reference as a child node.**

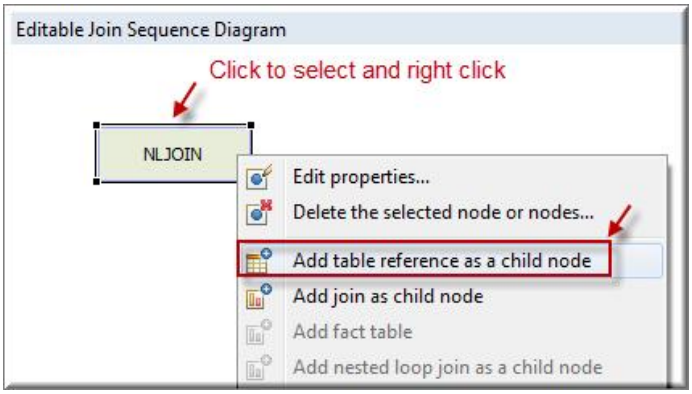


Figure 46 – Add table reference

4. Select Q2 (OQWT_T1) and click **OK.**

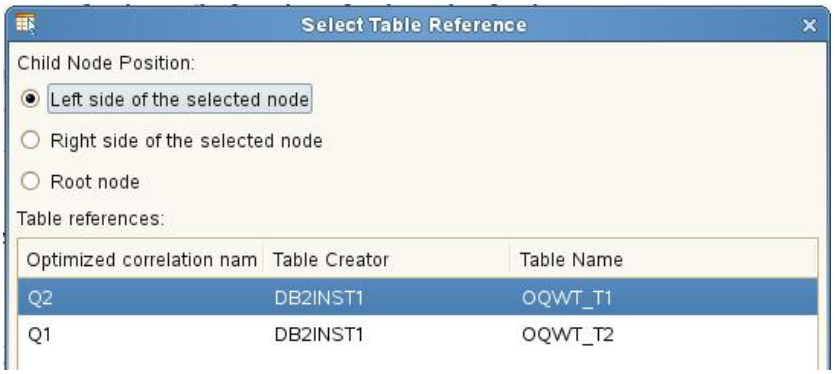


Figure 47 – Select Right Table Reference

5. Right-click on the NLJOIN box. Select **Add table reference as a child node.**

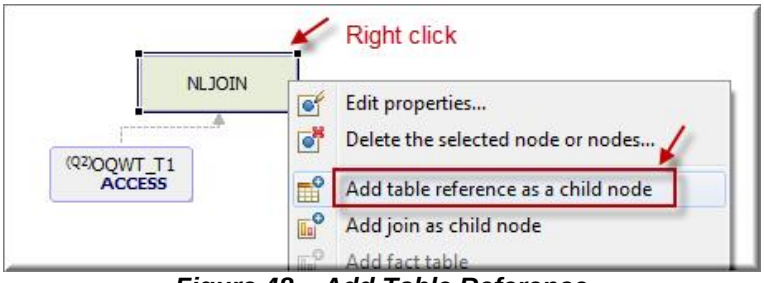


Figure 48 – Add Table Reference

6. Click Q1 line (OQWT_T2). Click **OK** in the dialog box.

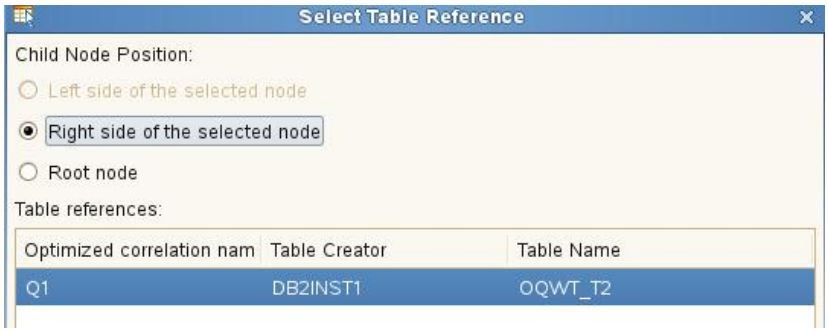


Figure 49 – Select left Table Reference

4.4.3 Set Access Path to IXSCAN

1. You will notice the nested loop join sequence created between two tables as shown. Double click the OQWT_T1 box.

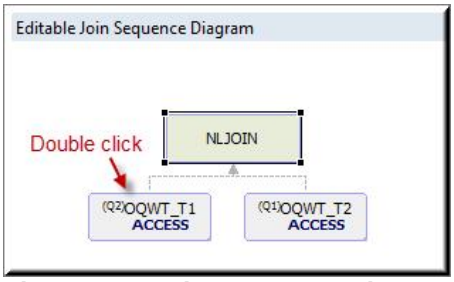


Figure 50 – Join Sequence Diagram

2. Click the **Value in Guideline** dropdown. Select **IXSCAN** and click **Select Index.** Check DB2INST1.OQWT_T1_IX1 and click **OK**.

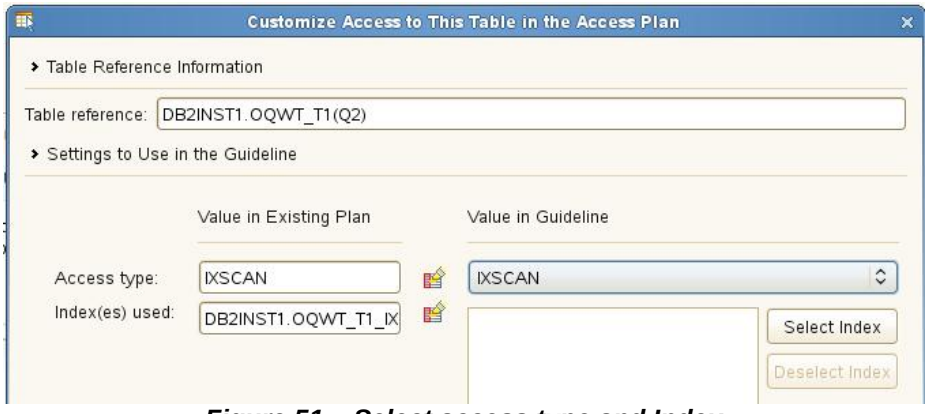


Figure 51 – Select access type and Index

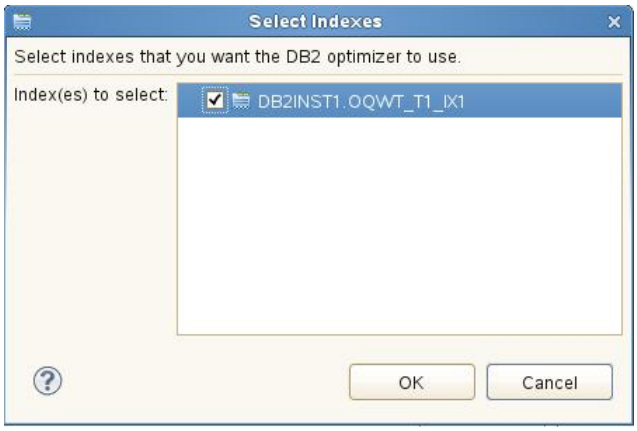


Figure 52 – Select DB2INST1.OQWT_T1_IX1Index

3. You will notice the following join graph between two tables.

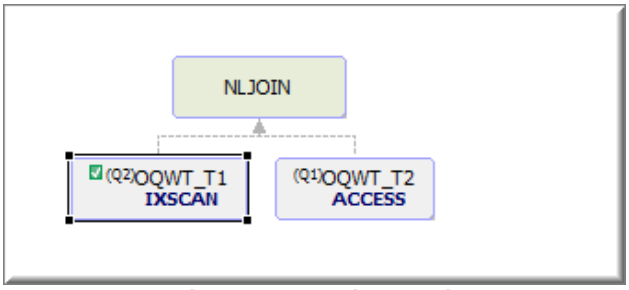


Figure 53 – Join Graph

4.4.4 Validate Optimization Profile

1. Click the **Validate Optimization Profile** icon, which is the left one and accept all default values. Click **Validate**. Please wait for the validation process to complete.

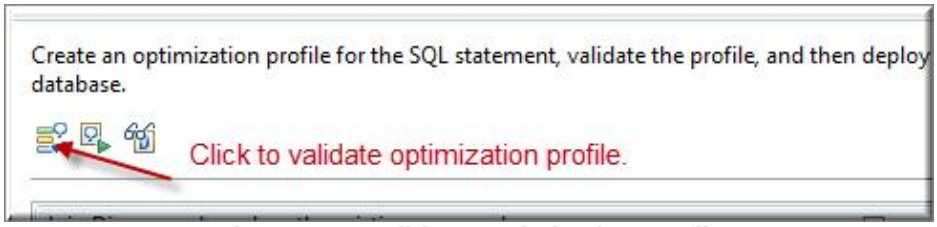


Figure 54 – Validate Optimization profile

4.4.5 Compare the Access Plan change

1. After validation, a new window opens with the validation results. Scroll all the way down. Notice that there are no errors or warnings.
2. Click the link (at the bottom of the window) to compare the new access path after applying the optimization profile.

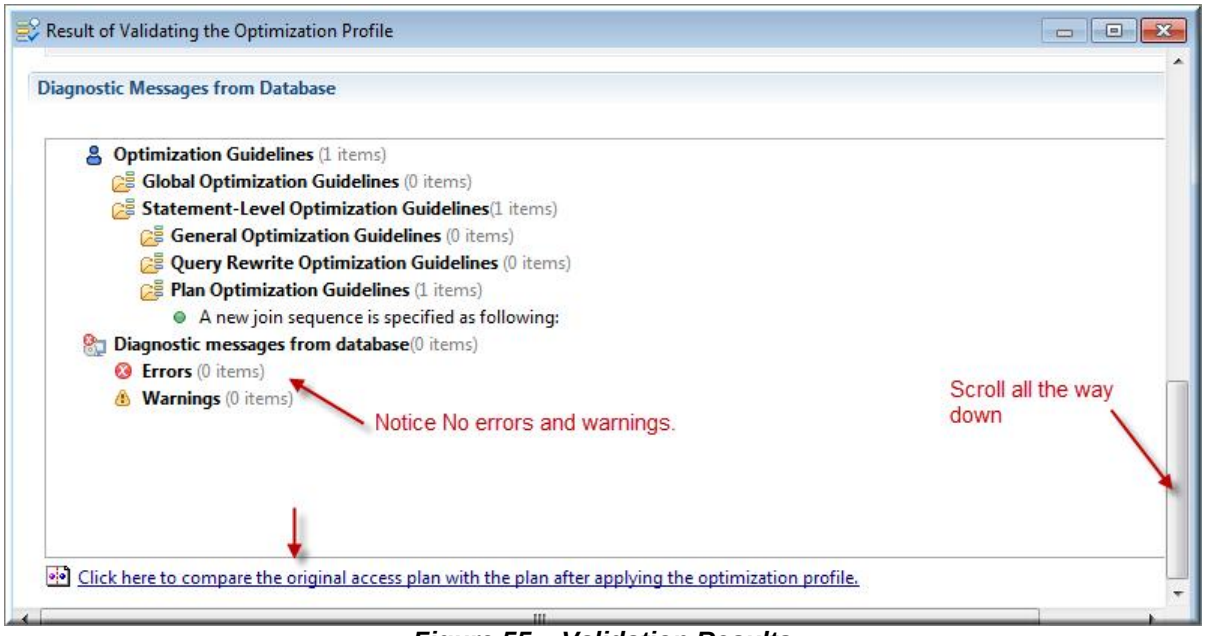


Figure 55 – Validation Results

3. A new window will open with two access path graphs. Compare them and notice that the new plan is using the nested loop join. Close the window.

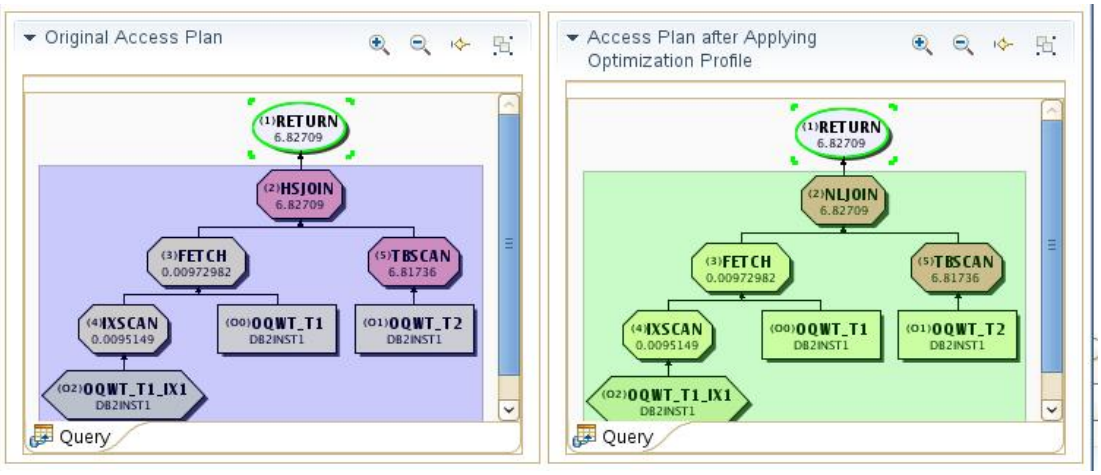


Figure 56 – Access Plan comparison

4.4.6 Deploy Optimization Profile

1. Click the second icon, which is Deploy Optimization Profile.

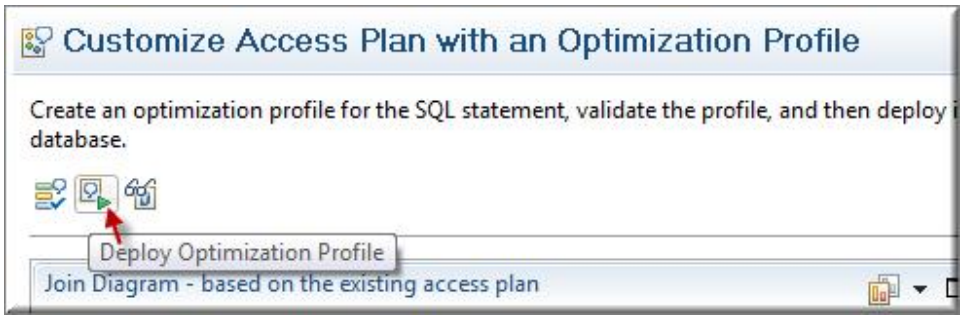


Figure 57 – Deploy Optimization Profile

2. Enter **OQT_PROFILE** as the profile name and click **Deploy**.

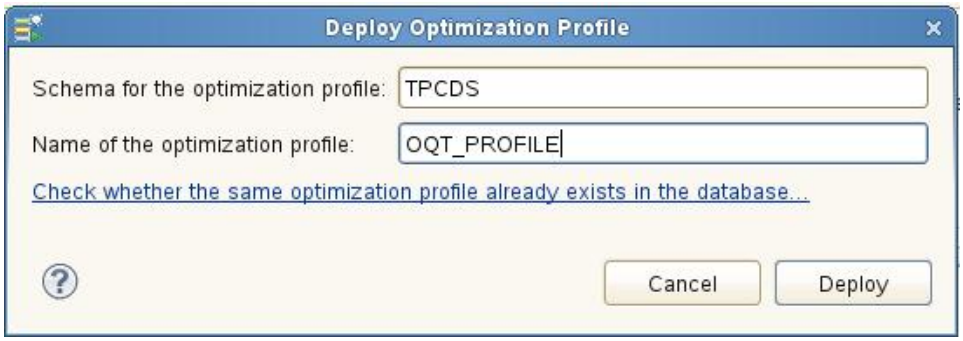


Figure 58 – Schema and Optimization Profile Name

3. A new window will open. Scroll to the bottom, browsing through the script generated. Click **OK** to deploy this optimization profile to the database.

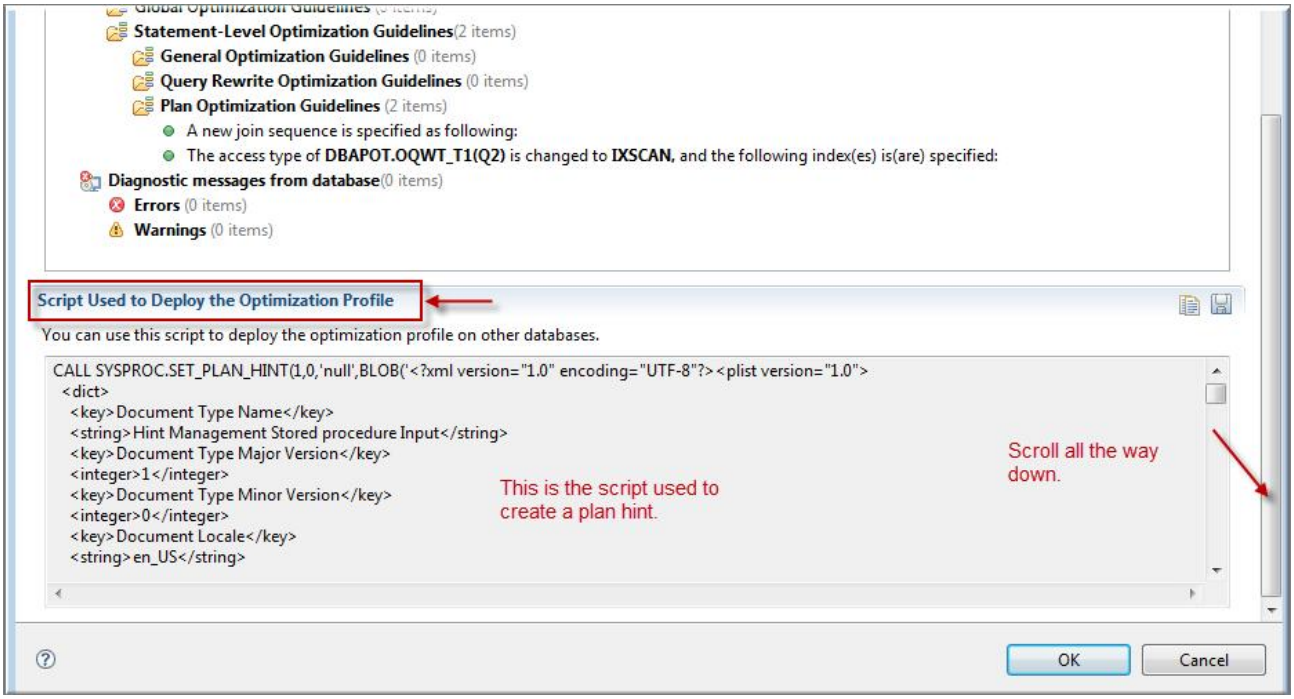


Figure 59 – Deploy

4. **Close** QTProject1 as shown below. Click **Yes** to save changes.

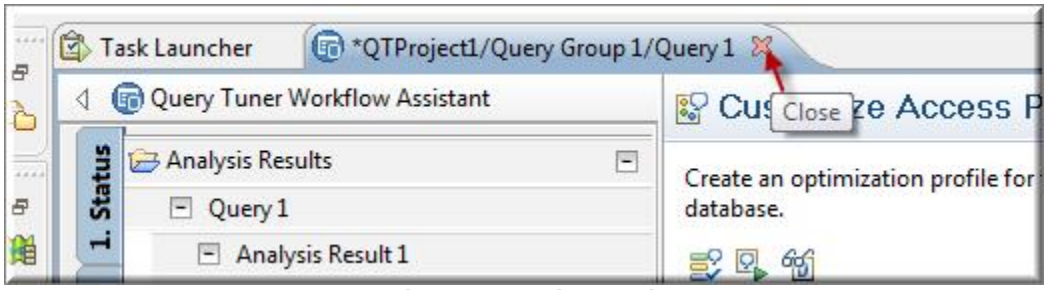


Figure 60 – Close Tab

5 SQL Performance Tuning – Clean Up

Run the following script to reset your lab environment.

```
qtdir
. cleanup.sh
```

6 *Compare query processing for row-organized tables to column-organized tables lab*

6.1 Lab Preparation

Before getting started with the lab exercises you will set up you're environment then work through a series of exercises focused on DB2 performance monitoring techniques.

6.1.1 Initialize Environment

You have been provided with a script, which creates a database, the needed tablespaces and the tables that you will work with in this portion of the lab. **These are mandatory steps and must be completed, as mandatory steps so often are.**

1. If you no longer have the terminal opened from previous exercise: By right clicking on the Gnome desktop (as in 3.1) **open a new terminal window**.
2. While logged in as user "**db2inst1**", change your working directory to the folder containing the scripts.

```
/home/db2inst1/Documents/LabScripts/db2pt/db2blu
```

6.2 What this exercise is about

I this part of the BLU Acceleration exercise, we compare various performance characteristics of SQL queries using the DB2 BLU Acceleration with the same queries based on row-organizedtables with standard indexes. We will look at the access plans using the **db2exfmt** explain tool to review the pre-execution estimated processing cost.

6.2.1 Query processing for BLU and non-BLU access plans

1. In first SQL query that we will analyze accesses a single table and produces a summarized result from a subset of the table data. We will use the **db2batch** application to execute the query and report basic elapsed time statistics.We will also include several SQL queries in the db2batch sequence that return important DB2 performance metrics that we will use tocompare the row-organized and columns-organized table processing.

   We will start by running the SQL query using the row-organized tables. The file *rowquery1.sql* contains the following statements:

```
set current explain mode yes;
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 20
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;
set current explain mode no;
```

```
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
```

We will define several NOT ENFORCED constraints on the column-organized tables to provide the optimizer information similar to the unique indexes that are defined for the row-organized tables.

We will collect new table statistics for the column-organized tables

The CURRENT EXPLAIN setting will generate the explain data that can be formatted using **db2exfmt** to show estimated processing costs. We will use the file *explain.dll* to create a new set of explain tables.

From your Linux VMware, login with the userid **db2inst1.** Start a new terminal session. In the terminal session, issued the following commands:

```
db2 force application all
db2 terminate
db2 activate db db2blu
db2 –tvf explain.ddl
db2 –tvf pkeys.ddl
```

The output should look similar to the following:

```
alter table colorg.acct add primary key (acct_id) not enforced
DB20000I  The SQL command completed successfully.

alter table colorg.branch add primary key (branch_id)  not enforced
DB20000I  The SQL command completed successfully.

alter table colorg.teller add primary key (teller_id)  not enforced
DB20000I  The SQL command completed successfully.

alter table colorg.history add constraint fkeyteller foreign key (teller_id) references
colorg.teller not enforced
DB20000I  The SQL command completed successfully.
```

```
alter table colorg.history add constraint fkeybranch foreign key (branch_id) references
colorg.branch not enforced
DB20000I  The SQL command completed successfully.
```

```
db2 –tvf runstats.cmd
```

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f rowquery1.sql –I complete –ISO CS | tee rowq1bat.txt
More rowq1bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 20
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                          BR_TRANS
--------- ---------------------------------- -----------
       10                     2045913300.00       27777
       11                     2068728300.00       27544
       12                     2554543800.00       31732
       13                     3685920900.00       40134
       14                     2787550200.00       33372

* 11 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.205942 seconds
* Execute Time is:       0.570747 seconds
* Fetch Time is:         0.005214 seconds
* Elapsed Time is:       0.781903 seconds (complete)
```

The example elapsed time is 0.781903.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
------------------------------------------
* Comment: " CHECK STATS"

------------------------------------------

* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- -----------------
                   0                 1835                  297                2132

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.106602 seconds
* Execute Time is:       0.031500 seconds
* Fetch Time is:         0.000160 seconds
* Elapsed Time is:       0.138262 seconds (complete)

------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                   0                  344                  297               272814
630

* 1 row(s) fetched, 1 row(s) output.
```

```
* Prepare Time is:       0.002589 seconds
* Execute Time is:       0.000295 seconds
* Fetch Time is:         0.000141 seconds
* Elapsed Time is:       0.003025 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(2132 in the sample)

TOTAL_WAIT_TIME:_____(650 in the sample)

POOL_READ_TIME:_____(297 in the sample)

Now loot at the access plan for the db2exfmt explain report.
In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o exrowq1.txt
more exrowq1.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:         588.169
   Query Degree:       1

            Rows
           RETURN
           (   1)
            Cost
             I/O
             |
             11
           GRPBY
           (   2)
           588.168
           255.345
             |
             11
           TBSCAN
           (   3)
           588.168
           255.345
             |
             11
           SORT
```

```
              (   4)
              588.167
              255.345
                |
                11
              pGRPBY
              (   5)
              588.165
              255.345
                |
              56493.4
              FETCH
              (   6)
              579.791
              255.345
             /---+----\
          56493.4      513576
          RIDSCN   TABLE: ROWORG
          (   7)       HISTORY
          116.992        Q1
          32.9881
            |
          56493.4
          SORT
          (   8)
          116.992
          32.9881
            |
          56493.4
          IXSCAN
          (   9)
          103.901
          32.9881
            |
          513576
      INDEX: ROWORG
          HISTIX1
            Q1
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(588 in the sample)

Total I/O cost:_____(255 in the sample)

The access plan uses one index to access the table ROWORG.HISTORY.

We will run a similar db2batch report using the same SQL query but the column-organized table CLORORG.HISTORY will be used instead of the row-organized table.

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f colquery1.sql –I complete –ISO CS | tee colq1bat.txt
more colq1bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM COLORG.HISTORY AS HISTORY
   WHERE HISTORY.BRANCH_ID between 10 and 20
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                       BR_TRANS
--------- ---------------------------------- -----------
       10                     2045913300.00       27777
       11                     2068728300.00       27544
       12                     2554543800.00       31732
       13                     3685920900.00       40134
       14                     2787550200.00       33372

* 11 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.173632 seconds
* Execute Time is:       0.169731 seconds
* Fetch Time is:         0.024093 seconds
* Elapsed Time is:       0.367456 seconds (complete)
```

The example elapsedtime is 0.367456 seconds compared to 0.781903 seconds for the row-organized table based query.

Lookt at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

The output should look similar to the following:

```
------------------------------------------------
* Comment: " CHECK STATS"

------------------------------------------------

* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- --------------------
                 130                  274                  201                  605

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.102977 seconds
* Execute Time is:       0.026895 seconds
* Fetch Time is:         0.000187 seconds
* Elapsed Time is:       0.130059 seconds (complete)

------------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                 410                  287                  305               171293
505

* 1 row(s) fetched, 1 row(s) output.
```

```
* Prepare Time is:       0.002879 seconds
* Execute Time is:       0.000368 seconds
* Fetch Time is:         0.000153 seconds
* Elapsed Time is:       0.003400 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(605 in the sample)

TOTAL_WAIT_TIME:_____(505 in the sample)

POOL_READ_TIME:_____(305 in the sample)

Now loot at the access plan for the db2exfmt explain report.
In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o excolq1.txt
more excolq1.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:           394.716
   Query Degree:         1

          Rows
         RETURN
         (   1)
          Cost
           I/O
           |
           11
         TBSCAN
         (   2)
         394.716
         44.169
           |
           11
         SORT
         (   3)
         394.715
         44.169
```

```
              |
              11
            CTQ
            (   4)
            394.714
            44.169
              |
              11
            GRPBY
            (   5)
            394.712
            44.169
              |
            56493.4
            TBSCAN
            (   6)
            393.884
            44.169
              |
            513576
      CO-TABLE: COLORG
            HISTORY
              Q1
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(394 in the sample)

Total I/O cost:_____(44 in the sample)

The access plan uses a table scan to access the table COLORG.HISTORY.

Note: The estimated I/O cost as well as the actual logical pages read for the query using the column-organized table are less than the query using the row-organized table.

The next SQL query that we will analyze accesses a single table and produces a summarized result from a subset of the table data. In this case the subset of data from the table ROWORG.HISTORY is not based on a column that is indexed. We will use the db2batch application to execute the query and report basic elapsed time and statistics. We ill

also include several SQL queries in the db2batch sequence that return important DB2 performance metrics that we will use to compare the row-organized and column-organized table processing.

We will start by running the SQL query using the row-organized tables. The file *rowquery2.sql* contains the following statements:

```
set current explain mode yes;
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.acct_ID between 5000 and 80000
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

set current explain mode no;
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
```

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f rowquery2.sql –I complete –ISO CS | tee rowq2bat.txt
more rowq2bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
* SQL Statement Number 2:
```

```
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM ROWORG.HISTORY AS HISTORY
   WHERE HISTORY.acct_ID between 5000 and 80000
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                       BR_TRANS
--------- ---------------------------------- -----------
        1                        3163900.00         176
        2                        3964500.00         217
        3                        3162100.00         174
        4                        1184000.00          71
        5                        2442000.00         136

* 100 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.047030 seconds
* Execute Time is:       0.303496 seconds
* Fetch Time is:         0.004525 seconds
* Elapsed Time is:       0.355051 seconds (complete)
```

The example elapsed time is 0.355051.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- ---------------
                   0                 1213                   84                1297

* 1 row(s) fetched, 1 row(s) output.
```

```
* Prepare Time is:       0.000129 seconds
* Execute Time is:       0.010047 seconds
* Fetch Time is:         0.000222 seconds
* Elapsed Time is:       0.010398 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                   0                   47                    0               67514
270

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000138 seconds
* Execute Time is:       0.000185 seconds
* Fetch Time is:         0.000074 seconds
* Elapsed Time is:       0.000397 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(1297 in the sample)

TOTAL_WAIT_TIME:_____(270 in the sample)

POOL_READ_TIME:_____(0 in the sample)

The sample results indicate that the query was able to use data in the buffer pool rather than reading the pages from disk.

Now look at the access plan for the db2exfmt explain report.

In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o exrowq2.txt
more exrowq2.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:         2407.47
   Query Degree:       1

      Rows
     RETURN
     (   1)
      Cost
       I/O
       |
       100
     GRPBY
     (   2)
     2407.47
      1118
       |
       100
     TBSCAN
     (   3)
     2407.46
      1118
       |
       100
     SORT
     (   4)
     2407.46
      1118
       |
     100.673
     pGRPBY
     (   5)
     2407.44
      1118
       |
     38521.7
     TBSCAN
     (   6)
     2401.71
      1118
       |
     513576
 TABLE: ROWORG
     HISTORY
       Q1
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(2407 in the sample)

Total I/O cost:_____(1118 in the sample)

The access plan uses a table scan to access the table ROWORG.HISTORY.

We will run a similar db2batch report using the same SQL query but the column-organized table CLORORG.HISTORY will be used instead of the row-organized table.

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f colquery2.sql –I complete –ISO CS | tee colq2bat.txt
more colq2.bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM COLORG.HISTORY AS HISTORY
   WHERE HISTORY.acct_ID between 5000 and 80000
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                      BR_TRANS
--------- ------------------------------ -----------
        1                     3163900.00         176
        2                     3964500.00         217
        3                     3162100.00         174
        4                     1184000.00          71
        5                     2442000.00         136

* 100 row(s) fetched, 5 row(s) output.
```

```
* Prepare Time is:       0.012101 seconds
* Execute Time is:       0.114467 seconds
* Fetch Time is:         0.004360 seconds
* Elapsed Time is:       0.130928 seconds (complete)
```

The example elapsedtime is 0.130928 seconds compared to 0.355051 seconds for the row-organized table based query.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

The output should look similar to the following:

```
------------------------------------------------
* Comment: " CHECK STATS"

------------------------------------------------

* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- --------------------
                 148                   91                   88                  327

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000167 seconds
* Execute Time is:       0.009893 seconds
* Fetch Time is:         0.000145 seconds
* Elapsed Time is:       0.010205 seconds (complete)

------------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
```

```
------------------- ------------------- ------------------- ------------------- -------
------------
                327                  12                  35               56207
192

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000133 seconds
* Execute Time is:       0.000171 seconds
* Fetch Time is:         0.000231 seconds
* Elapsed Time is:       0.000535 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(327 in the sample)

TOTAL_WAIT_TIME:_____(192 in the sample)

POOL_READ_TIME:_____(35 in the sample)

Now loot at the access plan for the db2exfmt explain report.
In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o excolq2.txt
more excolq2.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:         395.575
   Query Degree:       1

        Rows
       RETURN
       (   1)
        Cost
         I/O
         |
         100
       TBSCAN
       (   2)
       395.575
       49.6901
         |
         100
       SORT
       (   3)
       395.572
```

```
        49.6901
          |
          100
        CTQ
        (   4)
        395.555
        49.6901
          |
          100
        GRPBY
        (   5)
        395.549
        49.6901
          |
        38521.7
        TBSCAN
        (   6)
        394.983
        49.6901
          |
        513576
    CO-TABLE: COLORG
        HISTORY
          Q1
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(395 in the sample)

Total I/O cost:_____(49 in the sample)

The access plan uses a table scan to access the table COLORG.HISTORY.
Compare these estimated costs with the explain report based on the row-organized table.

The next SQL query that we will analyze accesses a single table and produces a summarized result from a subset of the table data. In this case the subset of data is based on two predicates that are supported by indexes on the table ROWORG.HISTORY. We will use the db2batch application to execute the query and report basic elapsed time and statistics. We ill also include several SQL queries in the db2batch sequence that return important DB2 performance metrics that we will use to compare the row-organized and column-organized table processing.

We will start by running the SQL query using the row-organized tables. The file *rowquery3.sql* contains the following statements:

```
set current explain mode yes;
```

```
SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM ROWORG.HISTORY AS HISTORY
   where branch_id between 10 and 20 and teller_id > 800
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

set current explain mode no;
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
```

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f rowquery3.sql –I complete –ISO CS | tee rowq3bat.txt
more rowq3bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM ROWORG.HISTORY AS HISTORY
   where branch_id between 10 and 20 and teller_id > 800
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;
```

```
BRANCH_ID BR_BALANCE                       BR_TRANS
--------- ---------------------------------- -----------
       10                      273951900.00        4521
       11                      684840000.00        7459
       12                      222541400.00        4126
       13                      478056000.00        6117
       14                      220693100.00        4266

* 11 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.048327 seconds
* Execute Time is:       0.547737 seconds
* Fetch Time is:         0.007361 seconds
* Elapsed Time is:       0.603425 seconds (complete)
```

The example elapsed time is 0.603425.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- ---------------
                   0                 1671                  272                1943

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.020876 seconds
* Execute Time is:       0.010026 seconds
* Fetch Time is:         0.000147 seconds
* Elapsed Time is:       0.031049 seconds (complete)

-----------------------------------------
```

```
* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                   0                   81                   52               166897
426

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.002257 seconds
* Execute Time is:       0.000268 seconds
* Fetch Time is:         0.000124 seconds
* Elapsed Time is:       0.002649 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(1943 in the sample)

TOTAL_WAIT_TIME:_____(426 in the sample)

POOL_READ_TIME:_____(52 in the sample)

The sample results indicate that the query was able to use data in the buffer pool rather than reading the pages from disk.

Now look at the access plan for the db2exfmt explain report.

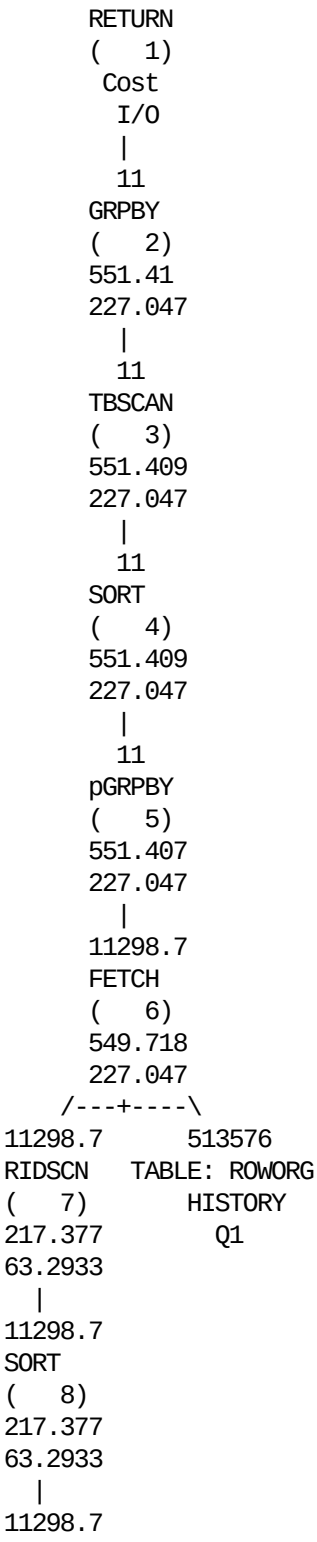
In the terminal session issue the commands:

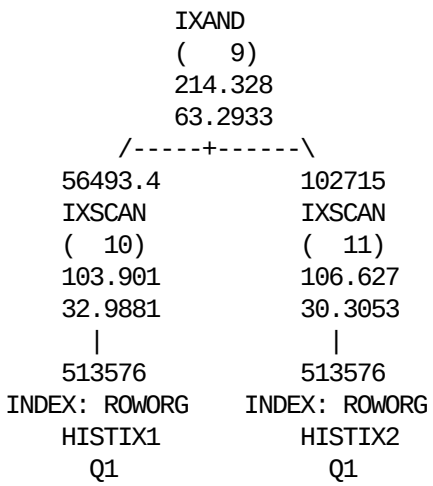
```
db2exfmt -1 –d db2blu –o exrowq3.txt
more exrowq3.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:         551.41
   Query Degree:       1

                    Rows
```





Review the db2exfmt report noting the following information:

Total estimated cost:_____(551 in the sample)

Total I/O cost:_____(227 in the sample)

The access plan uses the index anding IXAND operation to combine two indexe scan results to access the table ROWORG.HISTORY.

We will run a similar db2batch report using the same SQL query but the column-organized table CLORORG.HISTORY will be used instead of the row-organized table.

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f colquery2.sql –I complete –ISO CS | tee colq3bat.txt
more colq3bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
* SQL Statement Number 2:

SELECT HISTORY.BRANCH_ID, sum(HISTORY.balance) as br_balance, count(*) as
br_trans
   FROM COLORG.HISTORY AS HISTORY
```

```
   where branch_id between 10 and 20 and teller_id > 800
   GROUP BY HISTORY.BRANCH_ID
   ORDER BY HISTORY.BRANCH_ID ASC ;

BRANCH_ID BR_BALANCE                          BR_TRANS
--------- ----------------------------------- -----------
       10                      273951900.00        4521
       11                      684840000.00        7459
       12                      222541400.00        4126
       13                      478056000.00        6117
       14                      220693100.00        4266

* 11 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.040934 seconds
* Execute Time is:       0.049338 seconds
* Fetch Time is:         0.023248 seconds
* Elapsed Time is:       0.113520 seconds (complete)
```

The example elapsedtime is 0.113520 seconds compared to 0.603425 seconds for the row-organized table based query.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

The output should look similar to the following:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- --------------------
                 157                   98                   90                  345

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000121 seconds
* Execute Time is:       0.010118 seconds
* Fetch Time is:         0.000180 seconds
* Elapsed Time is:       0.010419 seconds (complete)
```

```
------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                 138                   41                    6                42409
113

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000136 seconds
* Execute Time is:       0.000177 seconds
* Fetch Time is:         0.000071 seconds
* Elapsed Time is:       0.000384 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(90 in the sample)

TOTAL_WAIT_TIME:_____(113 in the sample)

POOL_READ_TIME:_____(6 in the sample)

Now loot at the access plan for the db2exfmt explain report.
In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o excolq3.txt
more excolq3.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:         400.431
   Query Degree:       1

        Rows
       RETURN
       (   1)
        Cost
```

```
              I/O
              |
              11
            TBSCAN
            (   2)
            400.431
            49.6901
              |
              11
            SORT
            (   3)
            400.431
            49.6901
              |
              11
            CTQ
            (   4)
            400.429
            49.6901
              |
              11
            GRPBY
            (   5)
            400.428
            49.6901
              |
             11299
            TBSCAN
            (   6)
            400.262
            49.6901
              |
            513576
      CO-TABLE: COLORG
            HISTORY
              Q1
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(400 in the sample)

Total I/O cost:_____(49 in the sample)

The access plan uses a table scan to access the table COLORG.HISTORY.
Compare these estimated costs with the explain report based on the row-organized table.

Note: We have used thre SQL queries to compare performance of row-organized and column-organized tables. For the row-organized table, one query did not use an index, one used a single index and one used multiple indexes. In the sample results, the column-organized table was able to produce the same results faster with no traditional indexes.

The next SQL query that we will analyze joins two tables to produce a summarized result from a subset of the table data. In this case the subset of data is based on one predicate on the larger table that is supported by an indexe on the table ROWORG.HISTORY. We will use the db2batch application to execute the query and report basic elapsed time and statistics. We ill also include several SQL queries in the db2batch sequence that return important DB2 performance metrics that we will use to compare the row-organized and column-organized table processing.

We will start by running the SQL query using the row-organized tables. The file *rowquery4.sql* contains the following statements:

```
set current explain mode yes;

SELECT Teller.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
TELLER.TELLER_NAME , count(*) as transactions
  FROM ROWORG.HISTORY AS HISTORY, ROWORG.TELLER AS TELLER
   WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
    and teller.teller_id between 10 and 30
   GROUP BY TELLER.TELLER_ID, TELLER.TELLER_NAME
   order by 4 desc ;

set current explain mode no;
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
```

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f rowquery4.sql –I complete –ISO CS | tee rowq4bat.txt
```

```
more rowq4bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
* SQL Statement Number 2:

SELECT Teller.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
TELLER.TELLER_NAME , count(*) as transactions
  FROM ROWORG.HISTORY AS HISTORY, ROWORG.TELLER AS TELLER
   WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
    and teller.teller_id between 10 and 30
   GROUP BY TELLER.TELLER_ID, TELLER.TELLER_NAME
   order by 4 desc ;

TELLER_ID TOTAL_BALANCE                     TELLER_NAME
TRANSACTIONS
--------- --------------------------------- -------------------- -----------
-
       30                      273968000.00 <--20 BYTE STRING-->
2078
       16                      197057800.00 <--20 BYTE STRING-->
1706
       25                        6940400.00 <--20 BYTE STRING-->
426
       28                        6447500.00 <--20 BYTE STRING-->
403
       10                        5705900.00 <--20 BYTE STRING-->
385

* 21 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.095087 seconds
* Execute Time is:       0.176888 seconds
* Fetch Time is:         0.002835 seconds
* Elapsed Time is:       0.274810 seconds (complete)
```

The example elapsed time is 0.278410.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- --------------------
                   0                 1690                  185                 1875

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000125 seconds
* Execute Time is:       0.010717 seconds
* Fetch Time is:         0.000150 seconds
* Elapsed Time is:       0.010992 seconds (complete)

-------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                   0                   95                   51                38732
245

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000216 seconds
* Execute Time is:       0.000187 seconds
```

```
* Fetch Time is:          0.000085 seconds
* Elapsed Time is:        0.000488 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(1875 in the sample)

TOTAL_WAIT_TIME:_____(245 in the sample)

POOL_READ_TIME:_____(51 in the sample)

The sample results indicate that the query was able to use data in the buffer pool rather than reading the pages from disk.

Now look at the access plan for the db2exfmt explain report.

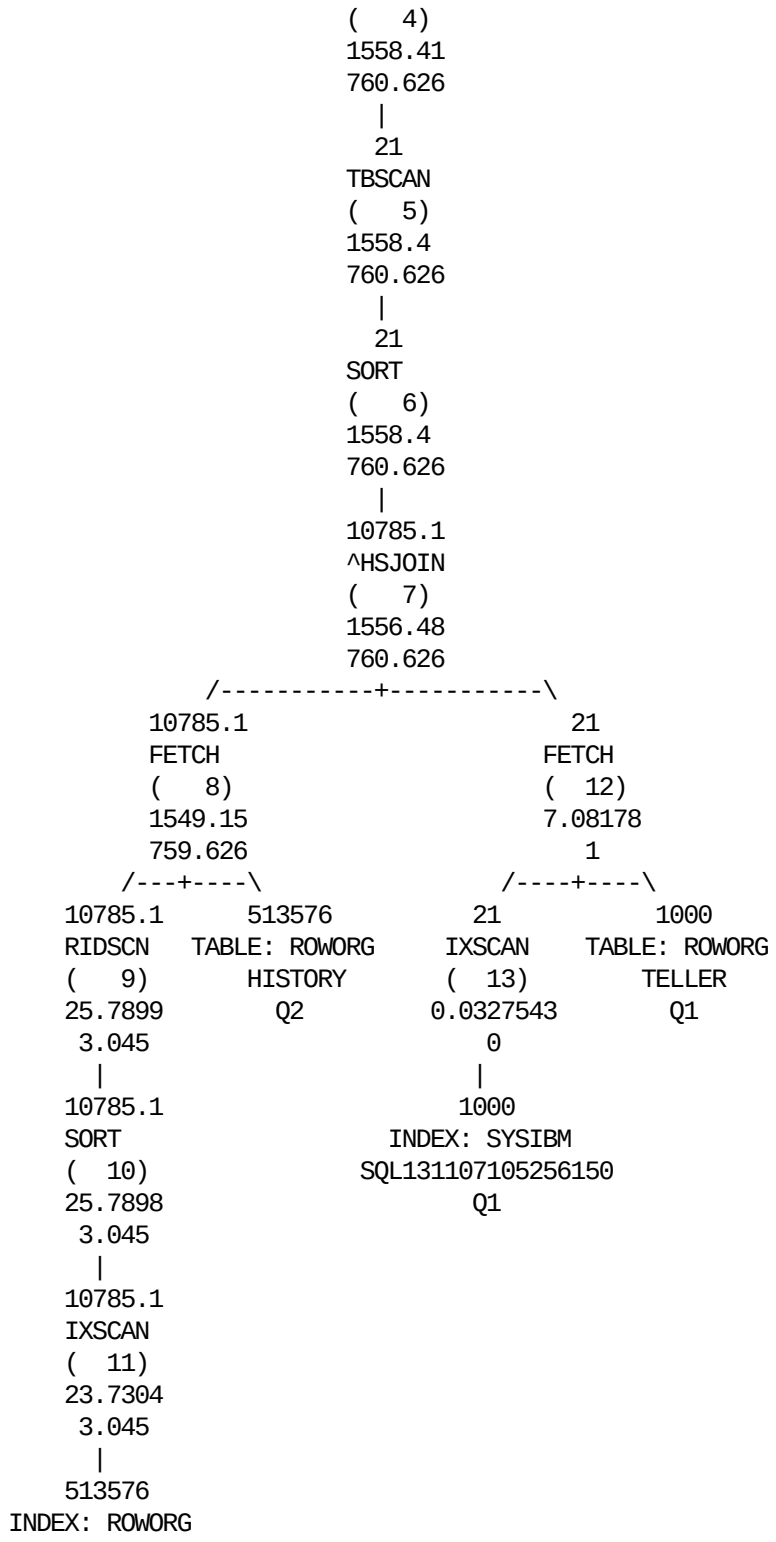
In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o exrowq4.txt
more exrowq4.txt.
```

The output should look similar to the following:

```
Access Plan:
-----------
    Total Cost:         1558.41
    Query Degree:       1

                          Rows
                         RETURN
                         (   1)
                          Cost
                           I/O
                           |
                           21
                         TBSCAN
                         (   2)
                         1558.41
                         760.626
                           |
                           21
                         SORT
                         (   3)
                         1558.41
                         760.626
                           |
                           21
                         GRPBY
```



```
        HISTIX2
          Q2
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(1558 in the sample)

Total I/O cost:_____(760 in the sample)

The access plan uses the indexes on both tables. The join processing uses a hash join method.

We will run a similar db2batch report using the same SQL query but the column-organized table CLORORG.HISTORY will be used instead of the row-organized table.

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f colquery4.sql –I complete –ISO CS | tee colq4bat.txt
more colq4bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
   ------------------------------------------

   * SQL Statement Number 2:

   SELECT Teller.TELLER_ID, sum(HISTORY.BALANCE) as total_balance,
   TELLER.TELLER_NAME , count(*) as transactions
     FROM COLORG.HISTORY AS HISTORY, COLORG.TELLER AS TELLER
      WHERE HISTORY.TELLER_ID = TELLER.TELLER_ID
       and teller.teller_id between 10 and 30
      GROUP BY TELLER.TELLER_ID, TELLER.TELLER_NAME
      order by 4 desc ;

   TELLER_ID TOTAL_BALANCE                    TELLER_NAME
   TRANSACTIONS
   --------- ---------------------------------- -------------------- -----------
   -
```

```
       30                      273968000.00 <--20 BYTE STRING-->
2078
       16                      197057800.00 <--20 BYTE STRING-->
1706
       25                        6940400.00 <--20 BYTE STRING-->
426
       28                        6447500.00 <--20 BYTE STRING-->
403
       10                        5705900.00 <--20 BYTE STRING-->
385

* 21 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.012669 seconds
* Execute Time is:       0.081109 seconds
* Fetch Time is:         0.016374 seconds
* Elapsed Time is:       0.110152 seconds (complete)
```

The example elapsedtime is 0.110152 seconds compared to 0.274810 seconds for the row-organized table based query.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

The output should look similar to the following:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- --------------
               160                  165                  144                469

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.018891 seconds
* Execute Time is:       0.045660 seconds
* Fetch Time is:         0.000256 seconds
* Elapsed Time is:       0.064807 seconds (complete)
```

```
------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                 238                   32                   47                87176
156

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000422 seconds
* Execute Time is:       0.000454 seconds
* Fetch Time is:         0.000187 seconds
* Elapsed Time is:       0.001063 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(469 in the sample)

TOTAL_WAIT_TIME:_____(238 in the sample)

POOL_READ_TIME:_____(47 in the sample)

In the sample report there are nearly as many index object pages with logical reads as column-organized object and data object page reads.

Column-organized tables do have the page map indexes which are used internally to locate the pages allocated for each column, an these references are counted as index page reads. The page map indexes do not appear in the access plans in explain reports.

Now loot at the access plan for the db2exfmt explain report.

In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o excolq4.txt
more excolq4.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
    Total Cost:         416.306
```

```
Query Degree:       1

              Rows
             RETURN
             (   1)
              Cost
               I/O
               |
             21.0153
             TBSCAN
             (   2)
             416.306
             46.6953
               |
             21.0153
             SORT
             (   3)
             416.306
             46.6953
               |
             21.0153
             CTQ
             (   4)
             416.303
             46.6953
               |
             21.0153
             ^HSJOIN
             (   5)
             416.302
             46.6953
          /----+----\
       1000         20.679
      TBSCAN        GRPBY
      (   6)        (   7)
      17.8417       398.455
      2.52632       44.169
        |             |
       1000         10792.9
CO-TABLE: COLORG    TBSCAN
      TELLER        (   8)
        Q1          398.296
                    44.169
                      |
                    513576
                CO-TABLE: COLORG
                    HISTORY
                      Q2
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(416 in the sample)

Total I/O cost:_____(46 in the sample)

The access plan uses a table scan to access the tables COLORG.HISTORY and COLORG.TELLER.
Compare these estimated costs with the explain report based on the row-organized table.

Locate the detail for GROUP BY (GRPBY) operation number 7 in the explain report. It will look similar to the following.

For example:

```
       7) GRPBY : (Group By)
                    Cumulative Total Cost:            398.455
                    Cumulative CPU Cost:              4.49065e+08
                    Cumulative I/O Cost:              44.169
                    Cumulative Re-Total Cost:  54.78
                    Cumulative Re-CPU Cost:    4.48934e+08
                    Cumulative Re-I/O Cost:    0
                    Cumulative First Row Cost: 7.25177
                    Estimated Bufferpool Buffers:     0

                    Arguments:
                    ---------
                    AGGMODE : (Aggregation Mode)
                           HASHED COMPLETE
                    GROUPBYC: (Group By columns)
                           TRUE
                    GROUPBYN: (Number of Group By columns)
                           1
                    GROUPBYR: (Group By requirement)
                           1: Q3.TELLER_ID
                    JN INPUT: (Join input leg)
                           INNER

                    Input Streams:
                    -------------
                           4) From Operator #8

                                  Estimated number of rows:  10792.9
                                  Number of columns:         2
                                  Subquery predicate ID:           Not Applicable

                                  Column Names:
                                  ------------
                                  +Q3.TELLER_ID+Q3.BALANCE
```

```
Output Streams:
--------------
       5) To Operator #5

              Estimated number of rows:  20.679
              Number of columns:         3
              Subquery predicate ID:            Not Applicable

              Column Names:
              ------------
              +Q4.$C2+Q4.$C1+Q4.TELLER_ID
```

The aggregation mode is 'Hashed Complete', which uses the sort heap memory to perform the grouping without the need to sort the input first. The small summarized result then becomes the inner table for the hash join operation.

The next SQL query that we will analyze joins three tables to produce a composite result from a subset of the table data. The query includes predicates on two of the tables, ROWORG.HISTORY and ROWORG.ACCT. the tables are joined using columns that are unique indexes of the ACCT and TELLER tables. The SQL query contains an ORDER BY clause, but no GROUP BY clause. We will use the db2batch application to execute the query and report basic elapsed time and statistics. We ill also include several SQL queries in the db2batch sequence that return important DB2 performance metrics that we will use to compare the row-organized and column-organized table processing.

We will start by running the SQL query using the row-organized tables. The file *rowquery5.sql* contains the following statements:

```
set current explain mode yes;

SELECT ACCT.ACCT_ID, ACCT.NAME, TELLER.TELLER_ID, TELLER.TELLER_NAME,
  TELLER.TELLER_CODE, ACCT.ADDRESS ,
  HISTORY.BRANCH_ID, HISTORY.BALANCE, HISTORY.PID, HISTORY.TEMP
   FROM roworg.ACCT AS ACCT, roworg.TELLER AS TELLER, roworg.HISTORY AS
HISTORY
   WHERE ACCT.ACCT_ID = HISTORY.ACCT_ID
   AND ACCT.ACCT_GRP BETWEEN 100 AND 200
   AND HISTORY.TELLER_ID = TELLER.TELLER_ID
   AND HISTORY.BRANCH_ID BETWEEN 40 AND 50
   ORDER BY HISTORY.PID ASC ;

set current explain mode no;
```

```
SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as
total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time,
total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;
```

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f rowquery5.sql –I complete –ISO CS | tee rowq5bat.txt
more rowq5bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
* SQL Statement Number 2:

SELECT ACCT.ACCT_ID, ACCT.NAME, TELLER.TELLER_ID, TELLER.TELLER_NAME,
  TELLER.TELLER_CODE, ACCT.ADDRESS ,
  HISTORY.BRANCH_ID, HISTORY.BALANCE, HISTORY.PID, HISTORY.TEMP
   FROM roworg.ACCT AS ACCT, roworg.TELLER AS TELLER, roworg.HISTORY AS HISTORY
   WHERE ACCT.ACCT_ID = HISTORY.ACCT_ID
   AND ACCT.ACCT_GRP BETWEEN 100 AND 200
   AND HISTORY.TELLER_ID = TELLER.TELLER_ID
   AND HISTORY.BRANCH_ID BETWEEN 40 AND 50
   ORDER BY HISTORY.PID ASC ;

ACCT_ID     NAME                 TELLER_ID TELLER_NAME          TELLER_CODE ADDRESS
BRANCH_ID BALANCE           PID         TEMP
----------- -------------------- --------- -------------------- ----------- ---------------
--------------- --------- ----------------- ----------- ------
       9517 <--20 BYTE STRING-->       893 <--20 BYTE STRING--> ab          <------ 30 BYTE
STRING ------>        45          26100.00           0 TP1ST
       9517 <--20 BYTE STRING-->       893 <--20 BYTE STRING--> ab          <------ 30 BYTE
STRING ------>        45          17700.00           0 TP1ST
```

```
      15393 <--20 BYTE STRING-->       263 <--20 BYTE STRING--> ab          <------ 30 BYTE
STRING ------>        40          27600.00           0 TP1ST
      15393 <--20 BYTE STRING-->       263 <--20 BYTE STRING--> ab          <------ 30 BYTE
STRING ------>        40          16700.00           0 TP1ST
      15393 <--20 BYTE STRING-->       263 <--20 BYTE STRING--> ab          <------ 30 BYTE
STRING ------>        40          24800.00           0 TP1ST

* 1874 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.063917 seconds
* Execute Time is:       0.334208 seconds
* Fetch Time is:         0.005953 seconds
* Elapsed Time is:       0.404078 seconds (complete)

---------------------------------------------
```

The example elapsed time is 0.404078.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

For example:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- ----------------
                   0                 1364                  193                1557

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000185 seconds
* Execute Time is:       0.010261 seconds
* Fetch Time is:         0.000148 seconds
* Elapsed Time is:       0.010594 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
```

```
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                   0                   64                   53               224943
152

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000128 seconds
* Execute Time is:       0.000170 seconds
* Fetch Time is:         0.000085 seconds
* Elapsed Time is:       0.000383 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(1557 in the sample)

TOTAL_WAIT_TIME:_____(152 in the sample)

POOL_READ_TIME:_____(53 in the sample)

The sample results indicates that the query was able to use data in the buffer pool rather than reading the pages from disk.

Now look at the access plan for the db2exfmt explain report.

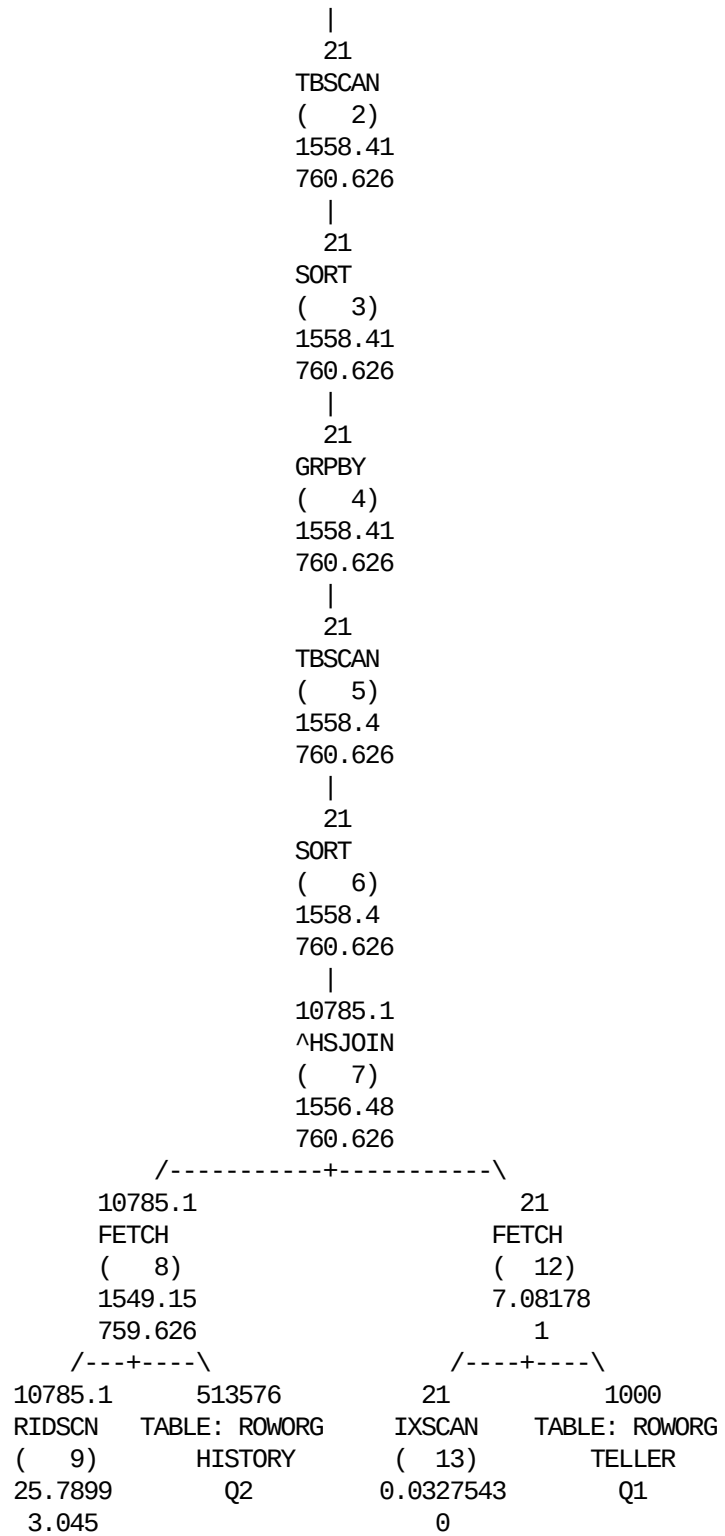
In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o exrowq5.txt
more exrowq5.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:         1558.41
   Query Degree:       1

                      Rows
                     RETURN
                     (   1)
                      Cost
                       I/O
```



```
        |                          |
    10785.1                      1000
    SORT                   INDEX: SYSIBM
    (  10)                SQL1311071052561 50
    25.7898                      Q1
     3.045
       |
    10785.1
    IXSCAN
    (  11)
    23.7304
     3.045
       |
    513576
INDEX: ROWORG
    HISTIX2
      Q2
```

Review the db2exfmt report noting the following information:

Total estimated cost:_____(2135 in the sample)

Total I/O cost:_____(832 in the sample)

The access plan uses the indexes on the HISTORY and TELLER tables, but perforams a table scan on the ACCT table, since the predicate on the ACCT_GRP column is not supported by an index. The join processing utilizes two hash join methods.

We will run a similar db2batch report using the same SQL query but the column-organized table COLORG.HISTORY will be used instead of the row-organized table.

In the Linux terminal session, issue the command:

```
db2batch –d db2blu –f colquery5.sql –I complete –ISO CS | tee colq5bat.txt
more colq5bat.txt
```

Review the db2batch report noting the following information

Elapsed time for the query (Statement2): _____

For example:

```
---------------------------------------------

* SQL Statement Number 2:
```

```
SELECT ACCT.ACCT_ID, ACCT.NAME, TELLER.TELLER_ID, TELLER.TELLER_NAME,
  TELLER.TELLER_CODE, ACCT.ADDRESS ,
  HISTORY.BRANCH_ID, HISTORY.BALANCE, HISTORY.PID, HISTORY.TEMP
   FROM colorg.ACCT AS ACCT, colorg.TELLER AS TELLER, colorg.HISTORY AS
HISTORY
   WHERE ACCT.ACCT_ID = HISTORY.ACCT_ID
   AND ACCT.ACCT_GRP BETWEEN 100 AND 200
   AND HISTORY.TELLER_ID = TELLER.TELLER_ID
   AND HISTORY.BRANCH_ID BETWEEN 40 AND 50
   ORDER BY HISTORY.PID ASC ;

ACCT_ID     NAME                   TELLER_ID TELLER_NAME          TELLER_CODE
ADDRESS                          BRANCH_ID BALANCE           PID         TEMP
----------- -------------------- --------- -------------------- -----------
------------------------------ --------- ---------------- ----------- -----
-
     305649 <--20 BYTE STRING-->       147 <--20 BYTE STRING--> ab
<------ 30 BYTE STRING ------>        41          10100.00           0 TP1ST
     144705 <--20 BYTE STRING-->       899 <--20 BYTE STRING--> ab
<------ 30 BYTE STRING ------>        48          10000.00           0 TP1ST
     609072 <--20 BYTE STRING-->       398 <--20 BYTE STRING--> ab
<------ 30 BYTE STRING ------>        50          10000.00           0 TP1ST
     122255 <--20 BYTE STRING-->       623 <--20 BYTE STRING--> ab
<------ 30 BYTE STRING ------>        43          10100.00           0 TP1ST
     184091 <--20 BYTE STRING-->       972 <--20 BYTE STRING--> ab
<------ 30 BYTE STRING ------>        42          10000.00           0 TP1ST

* 1874 row(s) fetched, 5 row(s) output.

* Prepare Time is:       0.013802 seconds
* Execute Time is:       0.376213 seconds
* Fetch Time is:         0.008739 seconds
* Elapsed Time is:       0.398754 seconds (complete)
```

The example elapsedtime is 0.398754 seconds compared to 0.404078 seconds for the row-organized table based query.

Look at the two queries that return performance metrics using the table function MON_GET_CONNECTION.

The output should look similar to the following:

```
* SQL Statement Number 4:

SELECT pool_col_l_reads, pool_data_l_reads, pool_index_l_reads ,
  ( pool_col_l_reads + pool_data_l_reads + pool_index_l_reads ) as total_l_reads
   from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

POOL_COL_L_READS     POOL_DATA_L_READS    POOL_INDEX_L_READS   TOTAL_L_READS
-------------------- -------------------- -------------------- ----------------
                 538                  178                  169                885

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000126 seconds
* Execute Time is:       0.010261 seconds
* Fetch Time is:         0.000146 seconds
* Elapsed Time is:       0.010533 seconds (complete)

---------------------------------------------

* SQL Statement Number 5:

SELECT total_col_time, total_compile_time , pool_read_time, total_cpu_time, total_wait_time
 from table(mon_get_connection(null,-1)) as con
  where application_name = 'db2batch'
 ;

TOTAL_COL_TIME       TOTAL_COMPILE_TIME   POOL_READ_TIME       TOTAL_CPU_TIME
TOTAL_WAIT_TIME
-------------------- -------------------- -------------------- -------------------- -------
-------------
                 939                   14                   77               227041
472

* 1 row(s) fetched, 1 row(s) output.

* Prepare Time is:       0.000133 seconds
* Execute Time is:       0.000169 seconds
* Fetch Time is:         0.000080 seconds
* Elapsed Time is:       0.000382 seconds (complete)
```

Note the following statistics from your copy of the report:

TOTAL_L_READS:_____(885 in the sample)

TOTAL_WAIT_TIME:_____(472 in the sample)

POOL_READ_TIME:_____(77 in the sample)

In the sample report the increase in column-oriented object page references relates to the access to the ACCT table, which is relatively large. This increases the pool read time and total wait time.

Now loot at the access plan for the db2exfmt explain report.

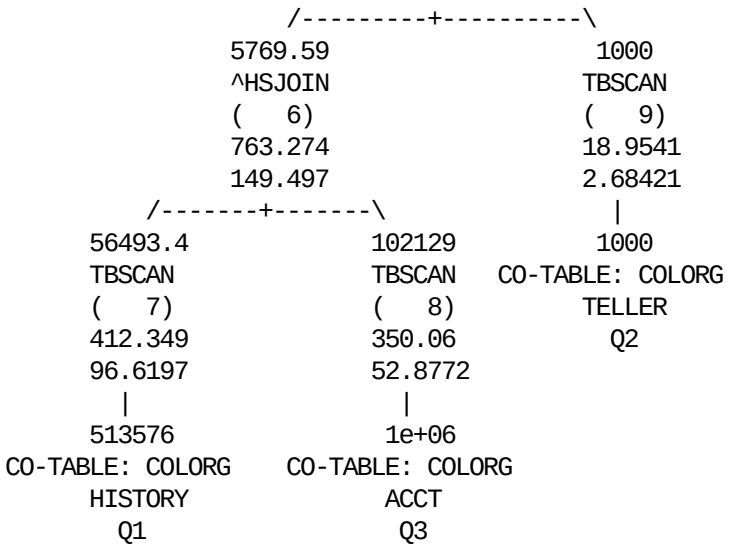
In the terminal session issue the commands:

```
db2exfmt -1 –d db2blu –o excolq5.txt
more excolq5.txt
```

The output should look similar to the following:

```
Access Plan:
-----------
   Total Cost:         784.282
   Query Degree:       1

                      Rows
                     RETURN
                     (   1)
                      Cost
                       I/O
                       |
                     5863.4
                     TBSCAN
                     (   2)
                     784.282
                     152.181
                       |
                     5863.4
                     SORT
                     (   3)
                     784.103
                     152.181
                       |
                     5863.4
                     CTQ
                     (   4)
                     782.339
                     152.181
                       |
                     5863.4
                     ^HSJOIN
                     (   5)
                     782.261
                     152.181
```



Review the db2exfmt report noting the following information:

Total estimated cost:_____(784 in the sample)

Total I/O cost:_____(152 in the sample)

These estimated costs are leass than the costs estimated to access the row-organized tables.

Note: Your results may have been different, but in each case the column-organized tables used the DB2 BLU Acceleration support to return results with good performance and reduced system resources compared to a set of standard row-organized tables.

We coould have performed some additional tuning with the row-organized tables, looking for better indexes or table organizations, but the column-organized tables may offer a simple solution that performans well and does not require the additional performance analysis.

Use the command file named db2bluend to reset the DB2 Registry variable DB2_WORKLOAD and to drop the test database DB2BLU.

In the terminal session, issue the command:

```
./db2bluend
```

Close all remaining terminal sessions now.

For each remaining terminal sessions:

```
db2 terminate
exit
```

This concludes the second part of BLU Acceleration exercise.

IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada