

# *DB2<sup>®</sup> SQL Query Tuning with BLU Acceleration*

Version/Revision# 1  
Module ID 10303  
Length 1.5 hours



## Table of Contents

<b>1</b>	<b>Objectives .....</b>	<b>3</b>
<b>2</b>	<b>Suggested reading .....</b>	<b>3</b>
<b>3</b>	<b>DB2 Design Advisor Lab.....</b>	<b>3</b>
3.1	Start DB2.....	4
3.2	Create Database and Tables .....	5
3.3	Examine and Execute Workload .....	6
3.4	DB2 Design Advisor.....	17
3.5	Examine and Execute Workload after recommendations .....	19
<b>4</b>	<b>Monitoring CPU and I/O Bound Queries.....</b>	<b>20</b>
4.1	Collect Activity metrics .....	20
4.2	Monitor I/O bound query .....	21
4.3	Monitor CPU bound query.....	23
<b>5</b>	<b>Administrative Views Lab .....</b>	<b>25</b>
5.1	Monitor switches and start workload .....	25
5.2	Monitor database .....	25
5.3	Execute workload .....	27
5.4	DB2 Design Advisor.....	27
5.5	Execute workload after recommendations.....	27

## 1 Objectives

DB2 10 contains numerous SQL performance enhancements that continue to make the DB2 data server an industry - leading data server solution that is suitable for any size of organization.

A number of performance improvements have been included in DB2 10 to improve the speed of many queries. These improvements are automatic; there are no configuration settings or changes to the SQL statements required.

DB2 10 has up to 3x faster query performance than its predecessor, with new functionality and enhanced features like:

- Queries over star schemas
- Queries with joins and sorts
- Queries with aggregation
- Hash joins

In this lab, you will learn:

- § How to identify problematic SQL statements in your application
- § How to monitor SQL statement resource usage
- § How to view SQL query execution plans
- § Which database features help to improve the SQL statement performance
- § How to use the DB2 Design Advisor to gather SQL performance tuning recommendations
- § How to influence access plan selections to optimize query execution

## 2 Suggested reading

### Information Management Technical Library at developerWorks

Best Practices: Tuning and Monitoring Database System Performance

<http://www.ibm.com/developerworks/data/bestpractices/systemperformance/>

### Optim Query Tuner for DB2 product overview

Useful information for SQL based optimization

[http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=%2Fcom.ibm.datatools.qrytune.nav.doc%2Ftopics%2Fhelpindex\\_qt.html](http://publib.boulder.ibm.com/infocenter/idm/v2r2/index.jsp?topic=%2Fcom.ibm.datatools.qrytune.nav.doc%2Ftopics%2Fhelpindex_qt.html)

### IBM Data Studio version 3.1 Information Center

<http://publib.boulder.ibm.com/infocenter/dstudio/v3r1/index.jsp>

## 3 DB2 Design Advisor Lab

In this part of the lab, you will work with the DB2 Design Advisor utility in DB2 10. You will run a workload on the database server and use the DB2 Design Advisor to decide what indexes or Materialized Query Tables (MQTs) are required to improve workload performance. After implementing the recommendations, the workload will be rerun. At the end of each run the workload timings as well as the workload query access plans will be captured and compared.

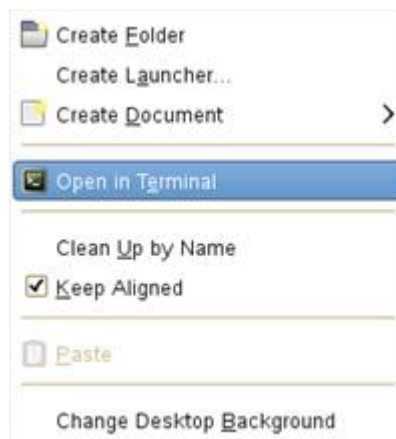
### 3.1 Start DB2

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

1. Login to the VMware virtual machine using the following information:

User: **db2inst1**  
Password: **password**

2. Open a terminal window as by right-clicking on the Desktop area and choose the “**Open Terminal**” item.



**Figure 1 – Opening a Terminal**

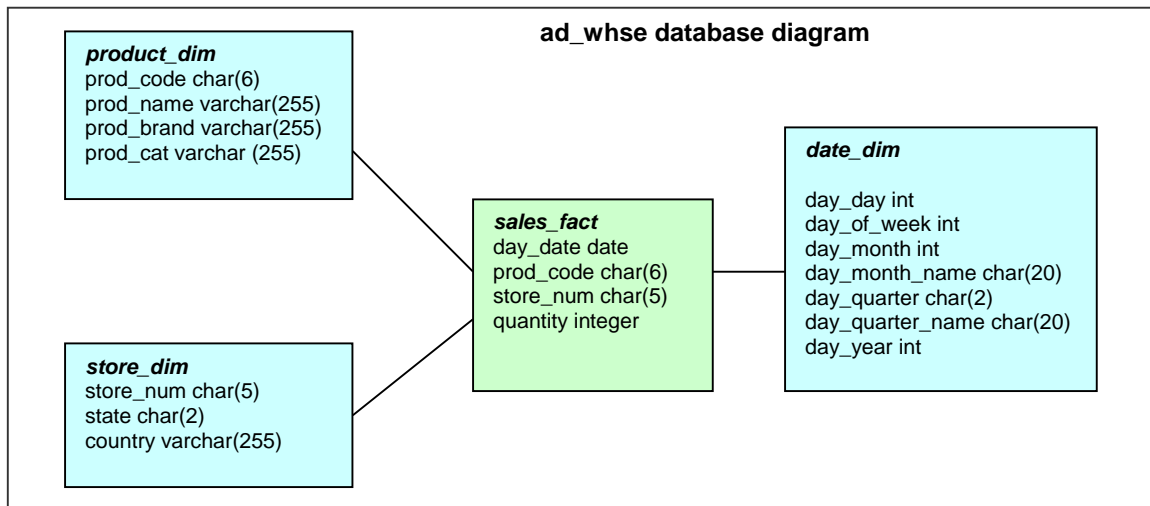
3. Start up DB2 Server by typing “**db2start**” in the terminal window.

4. In order to enable the aliases defined for this lab and all labs that follow, copy the /home/db2inst1/labbashrc to /home/db2inst1/.bashrc and source that file.

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

## 3.2 Create Database and Tables

We have provided you with a script, which creates a database and the tables that you will use in this portion of the lab. The diagram of the database utilized for this exercise is shown below.



**Figure 2 – Database ad\_whse diagram**

1. While logged in as user "db2inst1", change your working directory to the folder containing the work scripts.

```
qtdir  
Or  
cd ~/Documents/LabScripts/db2pt/querytuning
```

2. Before executing, take a look at the script, and note how the database, and tables are created.

```
cat crDbAndTabs.sql
```

3. Execute the script as follows:

```
db2 terminate  
db2 -tvf crDbAndTabs.sql
```

**i** **Note:** Make sure that each SQL statement, except the first (database drop may fail because ad\_whse had not been created yet), has finished successfully. The following message should be returned after the successful completion of the SQL statement:

```
DB20000I The SQL command completed successfully.
```

4. With the database created run the following two scripts to populate the database tables and update the system catalogs with the data statistics.

```
db2 -tvf load_data.sql
db2 -tvf run_stats.sql
```

**i** **Note:** In DB2 10, a number of improvements have been made to the RUNSTATS command to make statistics gathering faster. One of these improvements is that you no longer have to fully qualify object names by specifying a schema name. If you do not specify a schema name, the default schema is used. You can view this displaying the file run\_stats.sql

5. In this exercise the DB2 Design Advisor and the Visual Explain utility are used. These require the system Explain tables. Create the Explain tables by running the following commands:

Note that midir is a shortcut for directory ~/db2inst1/sqllib/misc.

```
midir
db2 -tvf EXPLAIN.DDL
```

### 3.3 Examine and Execute Workload

With the environment setup, now examine the workload containing two problematic SQL statements.

1. Display the file batch\_query.sql to view the two SQL statements. You may need to resize your terminal window to see the first query.

```
qtdir
cat batch_query.sql
```

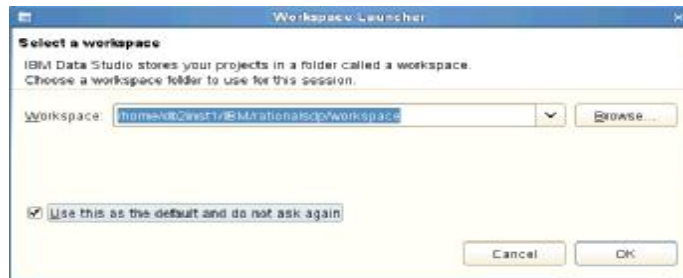
Leave this window open.

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



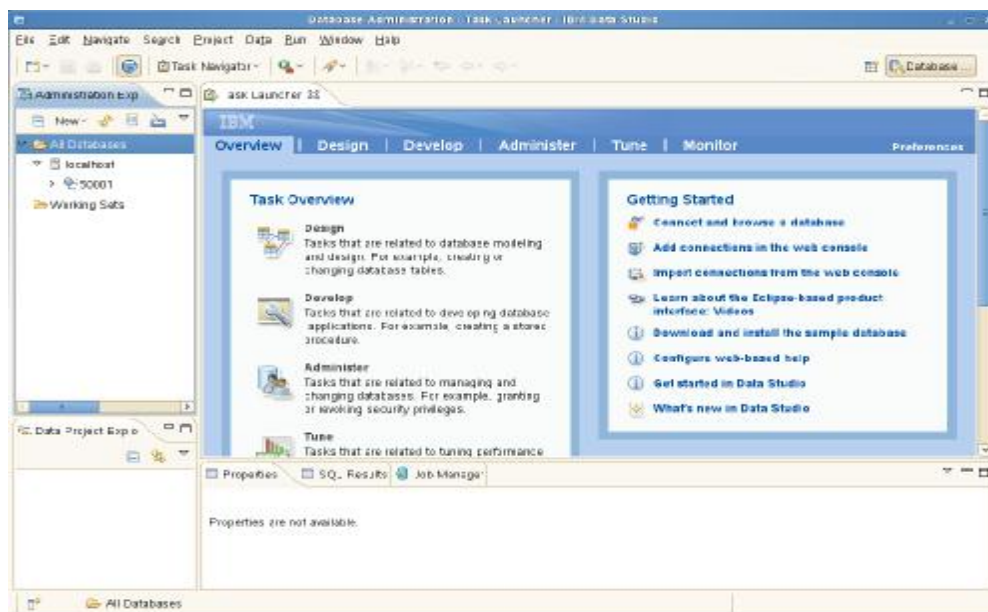
**Figure 3 – Open IBM Data Studio**

3. 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**

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



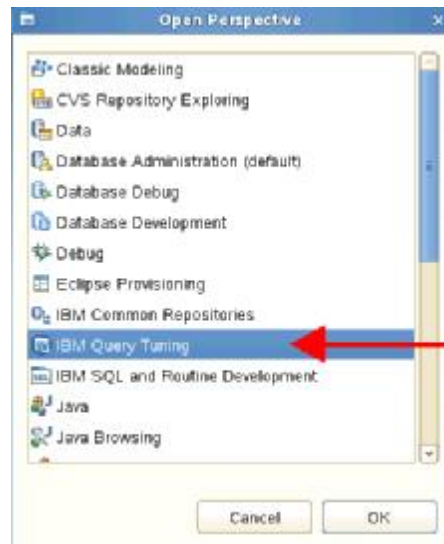
**Figure 5 – Database Administration perspective**

5. 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**

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

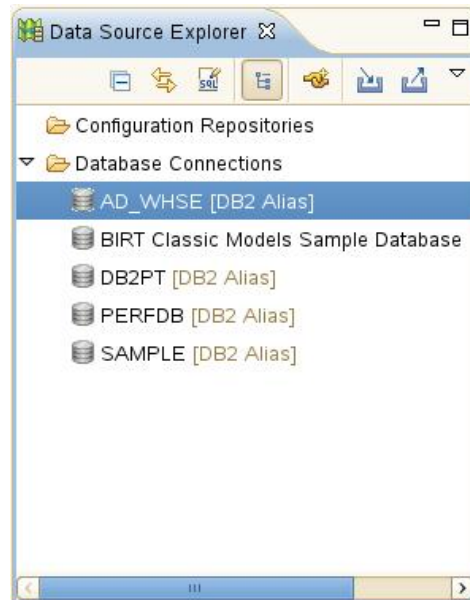


**Figure 7 – Choose IBM Query Tuning perspective**

**i** **Note:** Perspective defines the initial set and layout of the components in your screen. Each perspective provides a set of functionality aimed at accomplishing a specific type of tasks or works with specific types of resources.

7. In Data Source Explorer, expand the folder Database Connections until you see AD\_WHSE.





**Figure 8 – Choose AD\_WHSE Database**

8. Still in Data Source Explorer, right click AD\_WHSE and select **Connect**.
9. In the pop-up dialog, fill in the connection details as follows and Click **OK**:

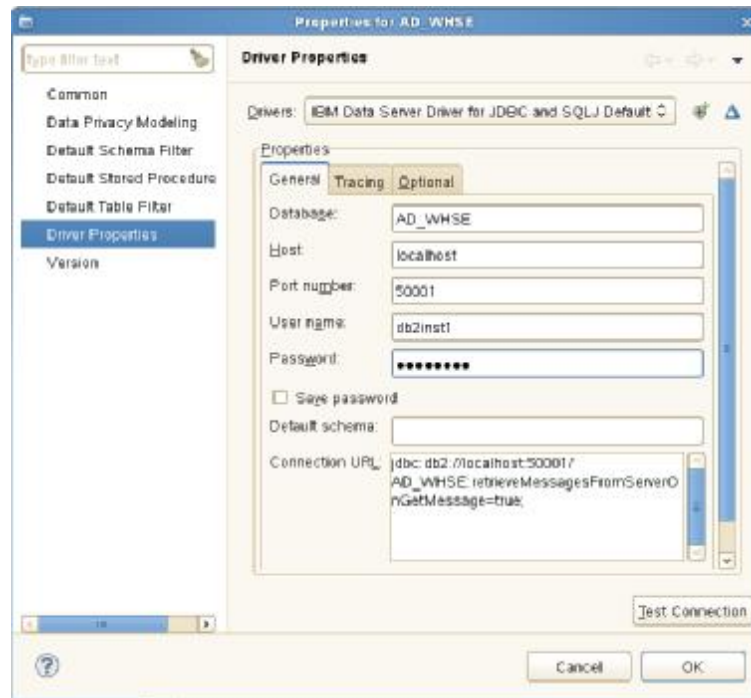
Database: **AD\_WHSE**

Host: **localhost**



Port number: **50001**

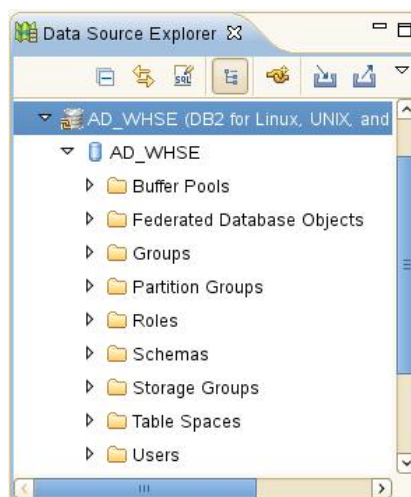
User name: **db2inst1**

Password: **password**



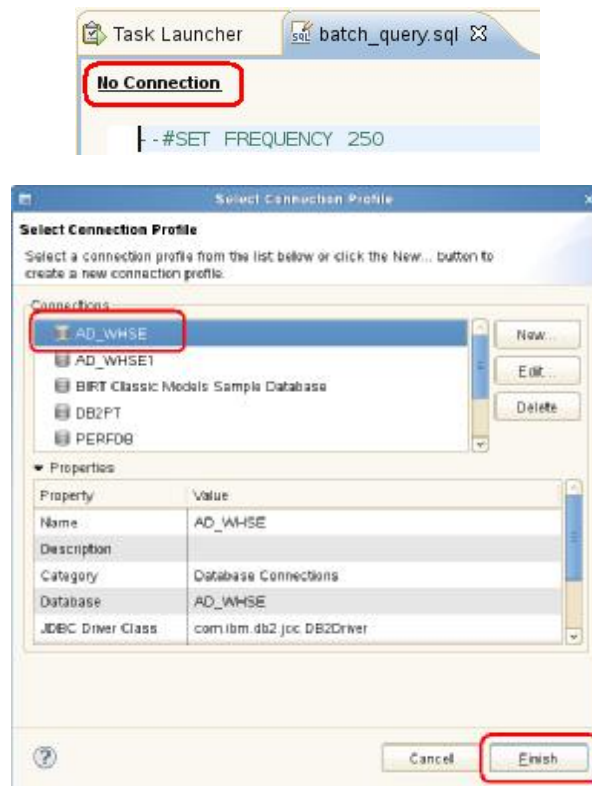
**Figure 9 – Connection details window**

10. After a connection has been successfully established, you should see this  icon beside the **AD\_WHSE** database in the Data Source Explorer, which indicates a connected database. Disconnected databases are listed with . Expanding the **AD\_WHSE** database folder will list supported database object types, as seen in the following screen capture:




**Figure 10 – Supported database object types**

11. At the top menu bar in Data Studio, select File -> Open File, then open /home/db2inst1/Documents/LabScripts/db2pt/querytuning/**batch\_query.sql**, which is the file you just examined in Step 1 of this section.
12. You may need to specify the database connection against which the script is going to be executed. To do that, click on **"No Connection"**, select AD\_WHSE, and click Finish.



**Figure 11 – Open batch\_query.sql**

13. Right click anywhere on the query and choose **'Open Visual Explain'**, click **'Next'** to explore the options, then choose **"Finish"** to generate the explain plan and diagram. You may want to double-click the tab  **Access Plan Diagram** to maximize the view.

**i Note:** In DB2 10, the Control Center and related components have been discontinued. These have been replaced by a new set of GUI tools: IBM Data Studio and IBM InfoSphere Optim tools.

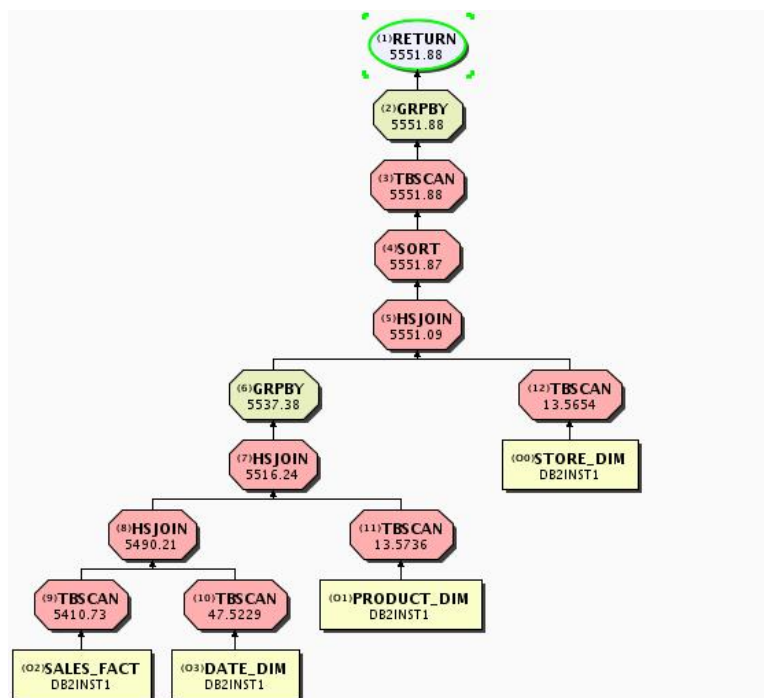
You'll see the current query access plan that includes table scan operations. Drill down into the objects for additional details. The access plan is provided below for your reference. Notice the total cost is approximately 5,000 timerons. The cost you see in your environment could be different.

Cost, in the context of access plans, is the estimated total resource usage necessary to execute the access plan for a statement (or the elements of a statement).

Cost is derived from a combination of CPU cost (in number of instructions), I/O (in number of seeks and page transfers), and communications cost.

The unit of cost is the timeron. A timeron does not directly equate to any actual elapsed time, but gives a rough relative estimate of the resources (cost) required by the database manager to execute different plans for the same query.

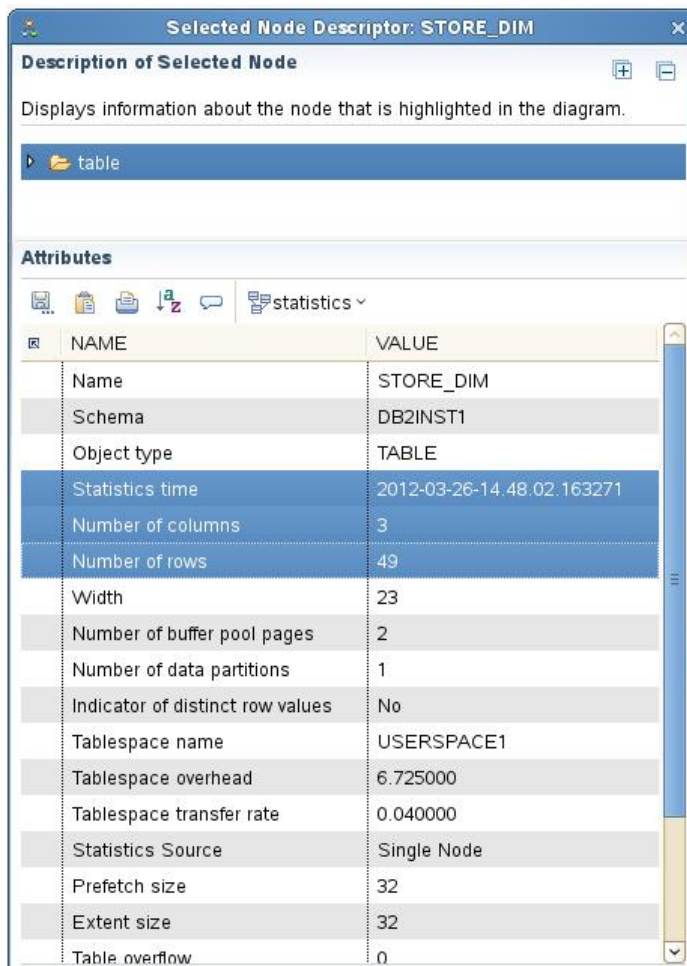
The cost shown in each operator node of an access plan graph is the cumulative cost, from the start of access plan execution up to and including the execution of that particular operator. It does not reflect other factors, such as the workload on the system or the cost of returning rows of data to the user.



**Figure 12 – Query Access Plan**

Digging deeper into Statement 2, the join is on four tables, SALES\_FACT, STORE\_DIM, PRODUCT\_DIM and DATE\_DIM. Table scans (TBSCAN) are being performed on each table. Three of these are dimension tables and are small in size.

Right click on the '**DB2INST1.STORE\_DIM**' node and select '**Show Description**'. The Statistics time indicates when the last time statistics were updated in the system catalog for the table you are viewing. Make sure the time is current. Scroll through the statistics look for information like the Number of columns, the Number of rows, etc. The STORE\_DIM table has just 49 rows, making indexes unhelpful. You can do the same analysis on the other dimension tables too.



**Selected Node Descriptor: STORE\_DIM**

**Description of Selected Node**

Displays information about the node that is highlighted in the diagram.

table

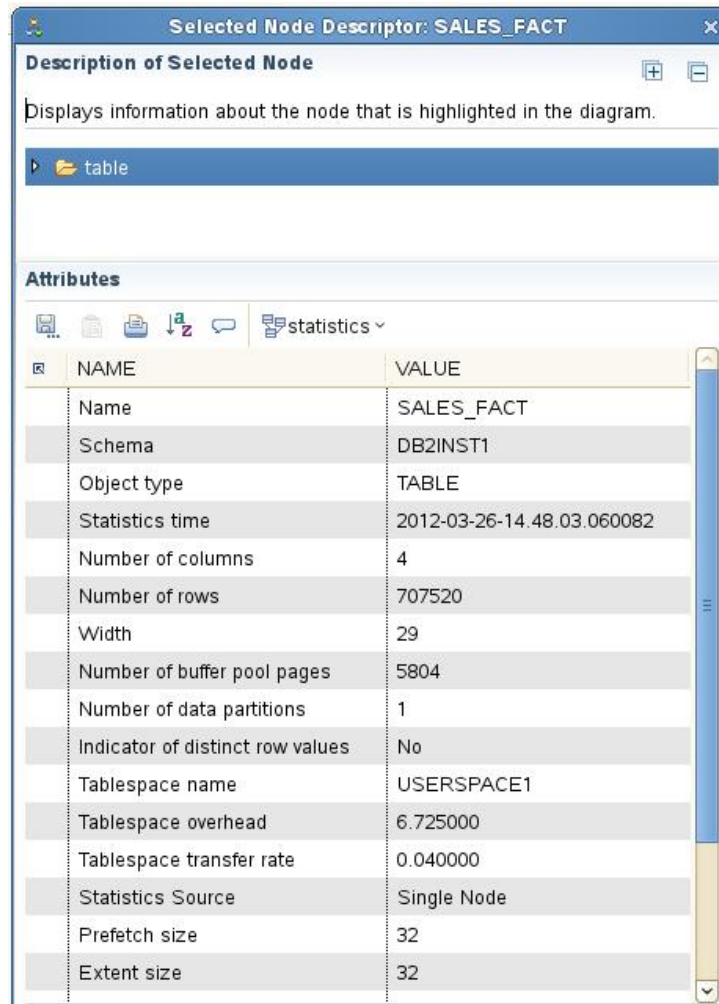
**Attributes**

statistics

NAME	VALUE
Name	STORE_DIM
Schema	DB2INST1
Object type	TABLE
Statistics time	2012-03-26-14.48.02.163271
Number of columns	3
Number of rows	49
Width	23
Number of buffer pool pages	2
Number of data partitions	1
Indicator of distinct row values	No
Tablespace name	USERSPACE1
Tablespace overhead	6.725000
Tablespace transfer rate	0.040000
Statistics Source	Single Node
Prefetch size	32
Extent size	32
Table overflow	0

**Figure 13 – DB2INST1.STORE\_DIM Description**

Right click the '**DB2INST1.SALES\_FACT**' node. Here you can look for 'Number of rows' and see that there are 707520 rows. You may think indexes will help, but look again. This warehouse statement has to read all the data in the table; an index won't help in this case either.



**Selected Node Descriptor: SALES\_FACT**

**Description of Selected Node**

Displays information about the node that is highlighted in the diagram.

▶ table

**Attributes**

statistics ▾

NAME	VALUE
Name	SALES_FACT
Schema	DB2INST1
Object type	TABLE
Statistics time	2012-03-26-14.48.03.060082
Number of columns	4
Number of rows	707520
Width	29
Number of buffer pool pages	5804
Number of data partitions	1
Indicator of distinct row values	No
Tablespace name	USERSPACE1
Tablespace overhead	6.725000
Tablespace transfer rate	0.040000
Statistics Source	Single Node
Prefetch size	32
Extent size	32

**Figure 14 – DB2INST1.SALES\_FACT Description**

In this scenario, DB2 optimizer first performs a Hash Join between SALES\_FACT and STORE\_DIM tables on column STORE\_NUM.

Right click on the HSJOIN(8) node joining the SALES\_FACT and DATE\_DIM tables and select '**Show description**'.

The 'Cumulative IO cost' information is the estimated cumulative I/O cost (in data page I/Os) of executing the chosen access plan up to and including this operator. The 'Cumulative CPU Cost' is the estimated cumulative CPU cost (in instructions) of executing the chosen access plan up to and including this operator. The 'Cumulative Total Cost' is the estimated cumulative cost (in timerons) of fetching the first row for the access plan up to and including this operator and includes any initial overhead required.

The Cumulative properties section provides information on the tables in the join, the selected columns and the join columns.

NAME	VALUE
Operator Identifier	8
Operator Type	HSJOIN
Cumulative Total Cost	5490.21
Cumulative CPU Cost	1.39028e+09
Cumulative IO cost	5811
Cumulative Re-execution Total Cost	5490.21
Cumulative Re-execution CPU Cost	1.39028e+09
Cumulative Re-execution IO cost	5811
Cumulative First Row Total Cost	5490.21

**Figure 15 – HSJOIN Description**

Hash join executes by requiring one or more predicates in the form `table1.columnX = table2.columnY`. In DB2 10, the column types may be different.

First, the designated inner table, `DATE_DIM` is scanned and rows are copied into memory buffers that are drawn from the sort heap. The memory buffers are divided into sections, based on a hash value that is computed on the columns of the join predicates. If the size of the inner table exceeds the available sort heap space, buffers from selected sections are written to temporary tables.

When the inner table has been processed, the second (or outer) table, `SALES_FACT` is scanned and its rows are matched with rows from the inner table by first comparing the hash value that was computed for the columns of the join predicates. If the hash value for the outer row column matches the hash value for the inner row column, the actual join predicate column values are compared.

Outer table rows that correspond to portions of the table that are not written to a temporary table are matched immediately with inner table rows in memory. If the corresponding portion of the inner table was written to a temporary table, the outer row is also written to a temporary table. Finally, matching pairs of table portions from temporary tables are read, the hash values of their rows are matched, and the join predicates are checked.

**i Note:** Default value is subject to change by the DB2 Configuration Advisor after initial database creation.

For the full performance benefit of hash joins, the value of the `sortheap` database configuration parameter and the `sheapthres` database manager configuration parameter may have to be changed. Alternatively, you can configure sort heaps to be controlled by STMM by setting the `sheapthres` to 0 and the `sortheap` to automatic.

**i** **Note:** `sheapthres` default value is subject to change by Configuration Advisor after initial database creation.

Hash join performance is best if you can avoid hash loops and overflow to disk. To tune hash join performance, estimate the maximum amount of memory that is available for `sheapthres` and then tune the `sortheap` parameter. Increase its setting until you avoid as many hash loops and disk overflows as possible, but do not reach the limit that is specified by the `sheapthres` parameter.

You can monitor the `TOTAL_HASH_JOINS`, `POST_SRTTHRESHOLD_HASH_JOINS`, `TOTAL_HASH_LOOPS`, `HASH_JOIN_OVERFLOW`s and `HASH_JOIN_SMALL_OVERFLOW`s elements using the SNAPAPPL administrative view.

DB2 introduces new columns to SNAPAPPL Administrative view like `DBPARTITIONNUM`, `TOTAL_OLAP_FUNCS`, `OLAP_FUNC_OVERFLOW`s and `MEMBER`. Consult SNAPAPPL administrative view and `SNAP_GET_APPL` table function section from DB 10 Information Center for details.

Use the `db2batch` utility to find out how long it takes to execute this workload. The `db2batch` tool can read SQL statements from either a flat file or from standard input, dynamically prepare and execute the statements, and return a result set. It also enables you to control the number of rows that are returned to `db2batch` and the number of rows that are displayed. You can specify the level of performance-related information that is returned, including elapsed time, processor time, buffer pool usage, locking, and other statistics collected from the database monitor. If you are timing a set of SQL statements, `db2batch` also summarizes the performance results and provides both arithmetic and geometric means.

DB2 10 introduces Command parameter **WFO** which specifies that the `db2batch` operation should wait for the outcome of an operation. For Cursor Stability and higher scans, `db2batch` will wait for the commit or rollback when encountering data in the process of being updated or deleted. Rows in the process of being inserted are not skipped.

14. Run the following command in the terminal window:

```
db2batch -d ad_whse -f batch_query.sql -r result.out,summary.out -v on -g off
```

**i** **Note:** There is no space between `result.out` and `summary.out`.

This command will run the two workload statements 25 times and capture the results in the files `result.out` and `timings` in the file `summary.out`.

15. View the contents of the `.out` files by running the commands:

```
cat summary.out
cat result.out
```



The contents of the summary.out file are provided below. Consider this result can be a little different.

* Summary Table:						
Type	Number	Repetitions	Total Time (s)	Min Time (s)	Max Time (s)	Arithmetic Mean
Geometric Mean Row(s)	Row(s)	Fetches	Row(s) Output			
Statement 0.756770	1	25	19.777257	0.382162	1.551512	0.791090
Statement 1.905023	4	4				
	2	25	49.666674	1.087114	3.072242	1.986667
	132	132				
* Total Entries: 2						
* Total Time: 69.443931 seconds						
* Minimum Time: 19.777257 seconds						
* Maximum Time: 49.666674 seconds						
* Arithmetic Mean Time: 34.721965 seconds						
* Geometric Mean Time: 31.341196 seconds						

In this scenario it took about 49 seconds to execute statement 2 and about 69 seconds to execute the workload. The performance of this workload can be improved. In the next section, we'll show you how by using the DB2 Design Advisor. You can leave all windows and applications open, as we'll use them in the next section.

### 3.4 DB2 Design Advisor

DB2 Design Advisor identifies all of the objects that are needed to improve the performance of your workload. When the DB2 Design Advisor analyzes a specific workload, it considers many factors, such as the type of statements in the workload, the frequency a particular statement occurs, and the characteristics of your database, to generate recommendations that minimize the total cost of running the workload.

Given a set of SQL statements in a workload, the Design Advisor will generate recommendations for:

- New indexes
- New clustering indexes
- New Materialized Query Tables (MQTs)
- Conversion to multidimensional clustering (MDC) tables
- The redistribution of tables in a partitioned database

**i Note:** The wizard interface for the design advisor is discontinued in DB2 10, but the design advisor is still available using the `db2advis` command.

1. To gather the recommendations of the DB2 Design Advisor, run the following command:

```
db2advis -d ad_whse -i batch_query.sql -m IMCP -o advise.out
```

In this workload the first statement will be run 250 times and the second statement will run 200 times. The statements are available in the file `batch_query.sql`.

The recommendations of the DB2 Design Advisor are captured in the file `advise.out`.

**i** Note: Since the Query Patroller is no longer supported in Version 10, the `-qp` option parameter for the `db2advise` command has been discontinued.

2. To view the recommendations run:

```
cat advise.out
```

For our test workload, the DB2 Design Advisor provided the following recommendations. Typically you would want to edit this file and change the system generated names to something more meaningful. The recommendations you get on your environment could be different.

```
-- LIST OF RECOMMENDED INDEXES
-- =====
-- index[1],      0.021MB
CREATE INDEX "DB2INST1"."IDX1203232129310" ON "DB2INST1"."DATE_DIM"
("DAY_DATE" ASC, "DAY_QUARTER_NAME" ASC) ALLOW REVERSE
SCANS COLLECT SAMPLED DETAILED STATISTICS;
COMMIT WORK ;
-- index[2],      0.013MB
CREATE INDEX "DB2INST1"."IDX1203232128250" ON "DB2INST1"."DATE_DIM"
("DAY_YEAR" ASC, "DAY_DATE" ASC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;
COMMIT WORK ;
-- index[3],      0.013MB
CREATE INDEX "DB2INST1"."IDX1203232129060" ON "DB2INST1"."PRODUCT_DIM"
("PROD_CAT" ASC, "PROD_BRAND" ASC, "PROD_CODE" ASC)
ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;
COMMIT WORK ;
```

**i** Note: Index Management re-defined in DB2 10:

- Jump Scan
  - Need fewer indexes
- Smart Index Pre-fetching
  - Faster index access & fewer index reorgs
- Smart Data Pre-fetching
  - Faster index scans & fewer index reorgs
- Predicate Evaluation Avoidance
  - Faster index scans

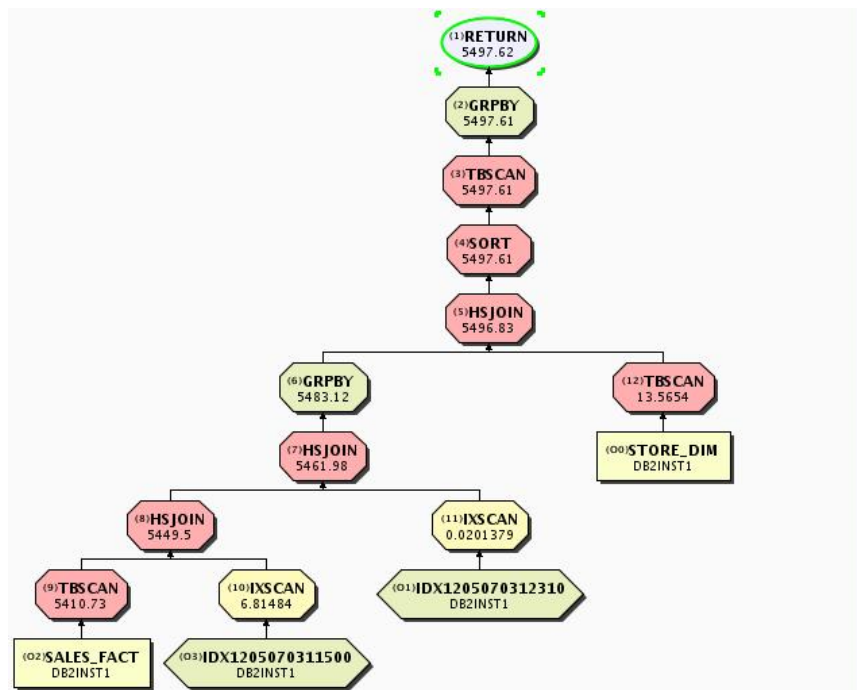
3. To implement the recommendations run the following commands:

```
db2 connect to ad_whse
db2 -tvf advise.out
```

### 3.5 Examine and Execute Workload after recommendations

Now that the recommendations have been implemented, we should check if they improved workload performance. This is done by generating the query access plans and running the workload again. Visual Explain will be used for this purpose.

1. You should still have **batch\_query.sql** open in the editor inside your **Data Studio** workspace. Right click anywhere within the SQL editor, select Open Visual Explain, keep all default values, click Finish to view the updated access plan graph. Maximize the Access Plan Diagram view if you want.
2. The access plan for the second SQL statement is provided below for your reference. In the example, the query execution cost has changed and now the execution of the query has been improved.



**Figure 16 – New Access Plan**

3. To re-determine the time it takes to execute this workload run the following command from the command window:

```
db2batch -d ad_whse -f batch_query.sql -r result_new.out,summary_new.out -v
on -g off
```

This will run the two workload statements 25 times and capture the results in a result\_new.out and timings in summary\_new.out file.

4. To compare the old and new timings, display the .out files:

```
cat summary.out
cat summary_new.out
```

Sample contents of the summary\_new.out file are provided below, actual numbers may vary.

* Summary Table:						
Type	Number	Repetitions	Total Time (s)	Min Time (s)	Max Time (s)	Arithmetic Mean
Geometric Mean	Row(s)	Row(s)	Output			
Statement	1	25	14.132961	0.389929	0.804882	0.565318
0.553693	4	4				
Stat	2	25	35.373943	0.885524	2.556759	1.414958
1.362658	132	132				
* Total Entries:		2				
* Total Time:		49.506904	seconds			
* Minimum Time:		14.132961	seconds			
* Maximum Time:		35.373943	seconds			
* Arithmetic Mean Time:		24.753452	seconds			
* Geometric Mean Time:		22.359306	seconds			

As you can see the first statement did not benefit from the MQT or the indexes. But the second statement execution time has substantially reduced due to the MQT. The net result is that the total workload time has been reduced due to the DB2 Design Advisor recommendations.

**You can close the IBM Data Studio but leave the command window open for the next set of exercises.**

## 4 Monitoring CPU and I/O Bound Queries

DB2 10 provides relational monitoring interfaces that are more efficient and have a lower impact on the system than previous system monitor and snapshot interfaces. These interfaces can be accessed directly by SQL and provide enhanced reporting and monitoring of the database system, data objects, and the package cache to help you quickly identify issues that might be causing problems.

In this lab, we'll use the `MON_GET_PKG_CACHE_STMT` table function interface to identify the problematic SQL statements. `MON_GET_PKG_CACHE_STMT` reports information for both static and dynamic SQL statements, unlike the dynamic SQL snapshot which only reports information for dynamic statements. We'll use the DB2 Design Advisor to suggest indexes to increase the performance of these SQL statements. These will be applied followed by an execution time comparison.

## 4.1 Collect Activity metrics

First you will execute a DB2 script file to create the new event monitor for locking. Open up a terminal window and change your current directory to `'/home/db2inst1/Documents/LabScripts/db2pt/querytuning'`:

- i **Note:** In DB2 10 new event monitors features exist:
  - All event monitors now support the WRITE TO TABLE target
  - Existing event monitors that write to tables can be altered to capture additional logical data groups

1. Restart the database and check if the database configuration parameter `mon_act_metrics` which provides metrics about activities being performed on the database is turned on. It is on by default, with a setting of `BASE`. To confirm execute:

```
db2stop force
db2start
qtdir
db2 connect to ad_whse
db2 get db cfg | grep -i mon
```

**i** **Note:** Make sure that `MON_ACT_METRICS=BASE`. On databases created prior to V9.7 and then upgraded to V9.7 or higher, the `mon_act_metrics` parameter is set to `NONE` by default. In version 10 the `BASE` setting of `MON_ACT_METRICS` includes metrics reported through `MON_GET_PKG_CACHE_STMT` and `MON_GET_PKG_CACHE_STMT_DETAILS`.

2. View Bad query SQL statement with the following command:

```
cat badquery.sql
```

## 4.2 Monitor I/O bound query

1. Use the `db2batch` utility to execute the bad query statement by issuing the following command:

```
db2batch -d ad_whse -f badquery.sql -r beforeresults.out,beforesummary.out
```

This will run the bad query and capture the results in a `beforesummary.out` and timings in `beforesummary.out`.

View the SQL statement execution timings by running.

```
cat beforesummary.out
```

2. Let's investigate by collecting the top ten dynamic & static SQL statements ordered by the average CPU time by using the `MON_GET_PKG_CACHE_STMT` table function. It returns a point-in-time view of SQL statements in the database package cache. This allows us to examine the aggregated metrics for a particular SQL statement, allowing us to quickly determine the reasons for poor query performance.

**i** **Note:** In DB2 10, `MON_GET_PKG_CACHE_STMT` returns additional columns that report metrics about I/O server efficiency, processing time for authentication, statistics generation, statement execution, high water mark input values, and extended latch waits.

It also allows us to compare the behavior of an individual cached section, relative to the other statements, to assist in identifying the most expensive section or statements.

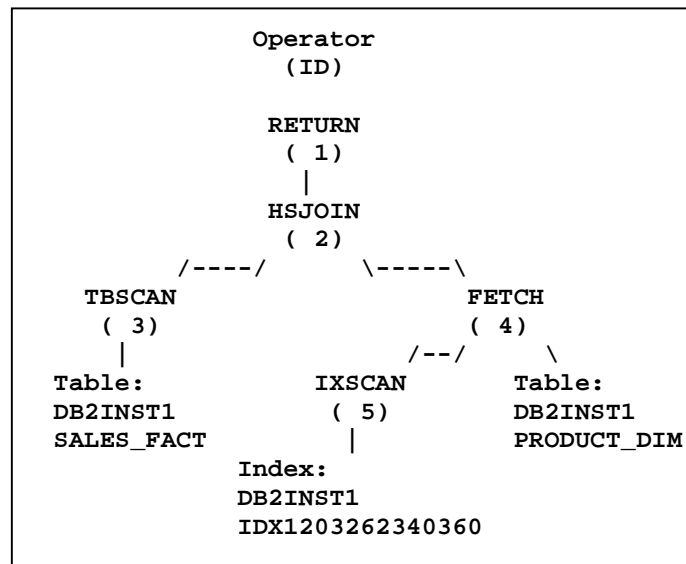
```
db2 -td! -vf mon.sql
```

We can see that from the output that the bad query spent most of its activity time in a waiting state. The `TOTAL_ACT_WAIT_TIME` is approximately eighty percent of the `TOTAL_ACT_TIME`. We can also see that The `LOG_DISK_WAIT`, `LOCK_WAIT` and query `PREP_TIME` are almost negligible. This gives us a hint that this query maybe I/O bound. We can also see that the `ROWS_READ` to `ROWS_RETURNED` ratio is high. This tells us that this query is not optimized and is missing an index. The operating system monitoring tools like **iostat** can also show if the database workload is I/O bound.

STATEMENT	LOCK_WAIT_TIME	LOG_DISK_WAIT_TIME	PREP_TIME	AVG_CPU_TIME ROWS_READ	TOTAL_ACT_TIME ROWS_RETURNED	TOTAL_ACT_WAIT_TIME
select b.prod_name, b.prod_brand from sales_fact a	0	0	66	166736 707575	797 85913	725
SQL0445W Value "select b.prod_name, b.prod_brand from sales_fact a, product_d" has been truncated.SQLSTATE=01004						
SELECT varchar(container_name, 70) as container_nam	0	0	79	120908 0	122 4	0
SELECT CURRENT QUERY OPTIMIZATION FROM SYSIBM.SYSD	0	0	61	89 0	0 1	0
select current CLIENT_APPLNAME, current CLIENT_ACC	0	0	1	28 0	0 1	0
4 record(s) selected with 1 warning messages printed.						

3. Let's look at the query plan of this bad query to confirm that it maybe missing an index. You will use DB2 Explain to do so. Run this command to get the query plan:

```
db2expln -d ad_whse -f badquery.txt -g -t
```



We can see that the query is doing a table scan on the table SALES\_FACT which has 707520 rows. An index on this table should improve performance by reducing the I/O.

4. Now let us use the DB2 Design Advisor to get recommendations to improve this query.

```
db2advis -d ad_whse -i badquery.sql -o indexrec.sql
```

```
-- LIST OF RECOMMENDED INDEXES
-- =====
-- index[1],      3.927MB
CREATE INDEX "DB2INST1"."IDX1203232156340" ON "DB2INST1"."SALES_FACT"
```

```
("QUANTITY" ASC, "PROD_CODE" DESC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;  
COMMIT WORK ;
```

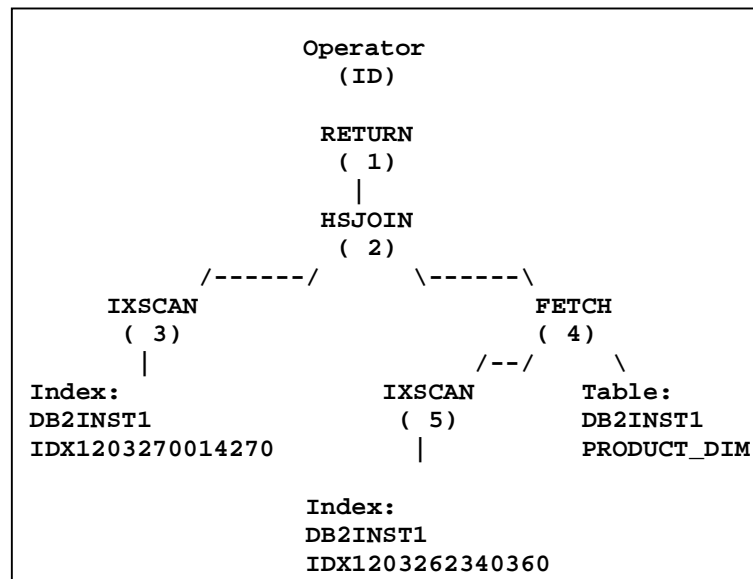
- Let us create the recommended index for this query and run the statistics.

```
db2 -tvf indexrec.sql  
db2 -tvf run_stats.sql
```

- Let us now see the query plan of this bad query to confirm that it has improved by using this new index. Run this command to get the query plan.

```
db2expln -d ad_whse -f badquery.txt -g -t
```

We can see the plan has improved by doing an Index scan instead of a table scan and the estimated cost of the query is lower.



### 4.3 Monitor CPU bound query

We will now run a bad query which is CPU intensive.

- We need to create a test table for this exercise. This command will create a table with more than a million rows.

```
db2 -td! -vf initscript.sql
```

- View and execute the bad query2 statement using the db2batch utility.

```
cat badquery2.sql  
db2batch -d ad_whse -f badquery2.sql -r oldresults.out,oldsummary.out
```

This will run the bad query2 and capture the results in oldresult.out and timings in oldsummary.out.

3. View the SQL statement execution timings by running.

```
cat oldsummary.out
```

4. Let us again get the top ten dynamic & static SQL statements ordered by the average CPU time using the MON\_GET\_PKG\_CACHE\_STMT table function. It returns a point-in-time view of SQL statements in the database package cache. This allows us to examine the aggregated metrics for a particular SQL statement, allowing us to quickly determine the reasons for poor query performance.

```
db2 -td! -vf mon.sql
```

We can see that from the output of this interface that this bad query spent most of the time doing CPU processing and very little wait time. The `TOTAL_ACT_WAIT_TIME` is pretty low. We can also see that the `LOG_DISK_WAIT`, `LOCK_WAIT` are almost negligible. This gives us a hint that this query maybe CPU bound. We can also see that the `ROWS_READ` to `ROWS_RETURNED` ratio is high. This tells us that this query is not optimized and is missing an index. This is causing the query to process more rows in memory making it CPU intensive.

5. Let us now see the query plan of this bad query2 to confirm that it may be missing an index. Run this command to get the query plan.

```
db2expln -d ad_whse -f badquery2.txt -g -t
```

We can see that the query is doing a table scan on the table `MANYROWS`. An index on this table will improve performance by reducing the number of rows accessed.

6. Now let us use the design advisor to provide recommendations to improve this query.

```
db2advis -d ad_whse -i badquery2.sql -l -1 -o indexrec2.sql
```

**i** **Note:** The first `-l` is a lowercase L; the second `-1` is a negative one.

```
-- LIST OF RECOMMENDED INDEXES
-- =====
-- index[1],    23.040MB
CREATE INDEX "DB2INST1"."IDX909250528440000" ON "DB2INST1"."MANYROWS"
("COL1" ASC, "COL2" DESC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;
COMMIT WORK
```

7. Let us create the recommended index for this query and run the statistics.

```
db2 -tvf indexrec2.sql
db2 runstats on table db2inst1.manyrows
```

8. Let us now see the query plan of this bad query2 to confirm that it has improved by using this new index. Run this command to get the query plan:

```
db2expln -d ad_whse -f badquery2.txt -g -t
```

We can see the plan has improved by doing an index scan instead of a table scan and the estimated cost of the query is much lower.



9. Execute the bad query2 statement again using the db2batch utility.

```
db2batch -d ad_whse -f badquery2.sql -r newresults.out,newsummary.out
```

This will run the bad query2 and capture the results in a newresults.out and timings in newsummary.out.

10. Compare the OLD and NEW execution times and observe the improvement in query performance.

```
cat oldsummary.out  
cat newsummary.out
```

## 5 Administrative Views Lab

Using the database 'ad\_whse', we'll run administrative views to monitor the SQL statements currently running on the database server, capture the longest running SQL statements, and check the buffer pool hit ratio. After identifying the problematic SQL statements we'll use the DB2 Design Advisor to suggest indexes, MQTs or MDCs to increase the performance of these SQL statements. These will be applied followed by an execution time comparison.

### 5.1 Monitor switches and start workload

1. As you'll be monitoring the database server resource usage, turn on the system monitor switches. For these settings to take effect restart the database server as well. Do this by running the script:

```
qtdir  
mon_switch_on
```

2. The workload for this exercise consists of five SQL statements that are continuously run against the database server. To start the workload, in the terminal windows type the following commands:

```
start_batch
```

3. To examine the five SQL statements type the following command:

```
less all_query.sql
```

**i** **Note:** You can use the arrow keys to scroll up and down in the script and type the letter "q" to exit anytime.

### 5.2 Monitor database

1. With the workload running in the background, connect to the database and monitor the database server activity. Use the following administrative views for this purpose.

```
db2 connect to ad_whse
```

2. To list all the dynamic & static SQL statements ordered by the average CPU time you could use the MON\_GET\_PKG\_CACHE\_STMT table function, as we have in previous exercises. However, by enabling both frameworks you will incur overhead from both monitoring frameworks, so careful consideration is required. As an alternative you will

use the `SYSIBMADM.SNAPDYN_SQL` administrative view and `SNAP_GET_DYN_SQL` table function, which are available in all supported versions of DB2, to return the top 5 dynamic SQL statements with regards to total CPU consumption.

```
db2 -tvf snapdyn_sql.sql
```

**i Note: In version 10, `SNAP_GET_DYN_SQL_V95` table function has been deprecated and replaced by the `SNAP_GET_DYN_SQL`**

3. To display a list of statements with information about the time required to prepare the statement, we use the `QUERY_PREP_COST` administrative view. When selecting from the view, an order by clause can be used to identify queries with the highest prep cost. We can examine this view to see how frequently a query is run as well as the average execution time for each of these queries. If the time it takes to compile and optimize a query is almost as long as it takes for the query to execute, you might want to change the optimization class that you are using. Lowering the optimization class might make the query complete optimization more rapidly, returning a result sooner. However, if a query takes a significant amount of time to prepare but is executed thousands of times (without being prepared again) then the optimization class might not be an issue.

```
db2 -tvf prep_cost.sql
```

4. To list the top dynamic SQL statements sorted by number of executions, average execution time, number of sorts, or sorts per statement, you use the `TOP_DYNAMIC_SQL` administrative view. These are the queries that represent some of the biggest resource consumers and should be analyzed to ensure they are well tuned.

```
db2 -tvf top_dyn.sql
```

5. The `LONG_RUNNING_SQL` administrative view returns the longest running SQL statements in the currently connected database and the current status of the query. Using agent ID as the unique identifier, you can investigate the application containing the SQL statement. If executing a long time and waiting on a lock, you might want to dig deeper using the `LOCKWAITS` or `LOCKS_HELD` administrative views. If “waiting on User”, this means that the database server is not doing anything and waiting for the application (like issue the next fetch or submit the next SQL statement).

```
db2 -tvf long_run.sql
```

6. The `BP_HITRATIO` administrative view returns bufferpool hit ratios, including total hit ratio, data hit ratio, XDA hit ratio and index hit ratio, for all bufferpools and all database partitions in the currently connected database. The ratio of physical reads to logical reads gives the hit ratio for the bufferpool. The lower the hit ratio, the more the data is being read from disk rather than the cached buffer pool which can be a more costly operation.

```
db2 -tvf buf_hit.sql
```

Once you've identified the slow performing SQL statements use the DB2 Design Advisor for recommendations on improving the workload performance.

7. First stop the workload by shutting down the database server. This will throw some error messages that can be safely ignored. Run.

```
db2stop force
```

8. Restart the database server for the next step.

```
db2start
```

**i** Note: in Version 10, some system catalog views and administrative SQL routines changed to include the database member information.

### 5.3 Execute workload

1. Capture the workload current execution timings by running the workload using the following command:

```
db2batch -d ad_whse -f all_query.sql -r result_OLDER.out,summary_OLDER.out
```

2. View the workload current execution timings by running

```
cat summary_OLDER.out
```

### 5.4 DB2 Design Advisor

1. Now invoke the DB2 Design Advisor for this workload using the command:

```
db2advis -d ad_whse -i all_query.sql -l -1 -m IMCP -o all_advise.out
```

**i** Note: The first **-l** is a lowercase L; the second **-1** is a negative one.

2. The recommendations of the DB2 Design Advisor are captured in the file all\_advise.out. View the recommendations using:

```
cat all_advise.out
```

3. As part of the recommendations new MQTs have been recommended and the earlier one will be reused. Implement the recommendations by running:

```
db2 connect to ad_whse  
db2 -tvf all_advise.out
```

### 5.5 Execute workload after recommendations

1. With the recommendations in place, rerun the workload and capture the new execution timings. Run the command:

```
db2batch -d ad_whse -f all_query.sql -r result_NEWER.out,summary_NEWER.out
```

2. Display the old and new timings by running:

```
cat summary_OLDER.out  
cat summary_NEWER.out
```

Compare the old and new execution times and observe the improvement in query performance.



---

© Copyright IBM Corporation 2014  
All Rights Reserved.

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

IBM, the IBM logo, ibm.com and Tivoli are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [ibm.com/legal/copytrade.shtml](http://ibm.com/legal/copytrade.shtml)

Other company, product and service names may be trademarks or service marks of others.

References in this publication to IBM products and services do not imply that IBM intends to make them available in all countries in which IBM operates.

No part of this document may be reproduced or transmitted in any form without written permission from IBM Corporation.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

IBM products are warranted according to the terms and conditions of the agreements (e.g. IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided.