

## How a Performance Database Can Help Save The Day

### Speaker

Mark Wilkinson, DBA, Amica Insurance  
(mwilkinson@amica.com)

Mary Petras, DB2 Tools Technical Support Team, IBM  
(marypetr@us.ibm.com)

Session Code: F13

May 14, 2010 8:30 – 9:30 PM

Platform: DB2 for z/OS

Come to this session to experience the value of building and maintaining a performance database (PDB). You will learn the critical steps you need to take for creating, loading and running reports from a PDB. The key benefit comes from the data you collect and from being proactive, rather than reactive. Learn to resolve issues before they become bottlenecks, You might notice performance spikes and correlate them to a new version of DB2 or a new deployed application. You can then zero in on the performance issues to identify the cause. A PDB can be the basis for capacity planning. Several examples of mining the PDB will be presented.

## Agenda

- General monitoring guidelines and basics
- General DB2 trace information
- Building a repository for performance data
  - Performance Database (PDB) or a Performance Warehouse (PWH)\*?
  - Why? Which? How? Where? What?
- Value of a PDB and a PWH as seen from Amica Insurance
- Summary

\*IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

First we will cover some general monitoring guidelines and basics especially in the DB2 trace area. This is needed to level-set with the audience.

The main topic is how to build a repository for performance data in DB2 – there are two choices available using IBM's monitoring product, IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS: a Performance Database (PDB) or a Performance Warehouse (PWH). We cover both of these options and the advantages and disadvantages for each.

Amica Insurance will then highlight their usage of each of these repositories and how they used the underlying performance data to find areas for improvement in their production and development systems.

## A Monitoring Approach: How a Monitor Can Help

- Few people have time to sit in front of a monitor and watch performance
  - Be proactive rather than reactive – use monitors to notify you when there are issues – take advantage of monitoring capabilities:
    - Capture exceptions
    - Send alerts when necessary
    - Lots of information within health graphs
- Historical Data
  - Report and analyze DB2 accounting and statistics trace data
  - Set up a PDB or a PWH to collect historical data for trending and analysis
  - Load accounting and statistics trace data into a repository

Performance monitors are important tools but with fewer DBAs taking charge of DB2 subsystems, there is even less time to sit in front of a monitor and watch what is happening. Nowadays, exceptions can provide DB2 DBAs and systems programmers important alerts when things go wrong; health graphs can also provide a view into the subsystem and highlight strange anomalies which then can be examined to find causes – fix these before they become a real bottleneck or problem.

Performance information about your DB2 subsystem is already collected. The idea is to take this important data and learn from it – using a performance repository can provide interesting and informative information about your transactions and their behavior as well as subsystem facts. Use this information for finding performance problems, trending purposes, and analysis of past behaviors.

Take the SMF data and load it into a performance repository for future analysis. This approach can provide you with valuable information that would otherwise be lost.

## Monitoring Basics – Types of DB2 Traces

- Trace Classes
  - Accounting
  - Statistics
  - Audit
  - Monitor
  - Performance
  - Global
- Multiple trace classes per trace type
- IFCID (Instrumentation Facility Component Identifier)
  - Basic unit of reporting
  - Specific to a particular function in DB2
  - IFCID details are documented in DSNxxx.SDSNIVPD(DSNWMSG)
- Destinations
  - SMF – Daily monitoring - default for accounting and statistics traces
  - GTF – High volume – default for performance trace
  - OPx – Buffers used by Online Monitors
  - Can be overridden by DEST parameter

This slide just level-sets different DB2 trace classes, destinations, and IFCIDs.

## Monitoring Basics – DB2 Trace Record Specifics

- Accounting and Statistics records
  - Relatively inexpensive to collect
  - Contains a wealth of information for analysis
  - Valuable information for problem determination
- Statistics Trace Records written as SMF 100
- Accounting Trace Records written as SMF 101
- Performance Trace Records written as SMF 102
  - Performance & Audit Trace Records can be expensive
  - Depends on the specific trace class and IFCID collected
- Accounting trace data is externalized when an event ends
- Statistics trace data is externalized based on user-defined time value

More basics as a point of reference.

## Monitoring Basics – DB2 Trace Records

- How to start, modify, stop traces
  - START TRACE(ABC) CLASS(1,2,3) PLAN(CUSTPLAN) DEST(SMF)
  - MODIFY TRACE(ABC) CLASS(1,2,3,7,8)
  - STOP TRACE(ABC)
- Recommended traces to start
  - Accounting Class 1, 2, 3, (7), (8), (10) (for detailed package tracing)
  - Statistics Class 1, 3, 4, 5, 6
  - DSNZPARMs: SMFACCT, SMFSTAT, STATIME, SYNCVAL
    - STATIME default is 5, set SYNCVAL to 0
    - Get statistics recorded every 5, 10, 15, etc. after the hour

This slide discusses some of the basics for starting, stopping and modifying traces.

The DB2 Administration Guide recommendation for what traces to start in DB2.

## Monitoring Basics – Filtering

- Filter trace data to minimize the amount of data and to reduce CPU overhead
  - Prior to DB2 9 - use PLAN, AUTHID, IFCID, or LOCATION
  - DB2 9 allows new filtering keywords with wildcarding capability that can be used in the -START TRACE for INCLUDE or EXCLUDE:
    - USERID or XUSERID: Client user ID
    - WRKSTN or XWRKSTN: Client workstation name
    - APPNAME or XAPPNAME: Client application name
    - PKGLOC or XPKGLOC: Package LOCATION name
    - PKGCOL or XPKGCOL: Package COLLECTION name
    - PKGPROG or XPKGPROG: PACKAGE name
    - CONNID or XCONNID: Connection ID
    - CORRID or XCORRID: Correlation ID
    - ROLE or XROLE: End user's database ROLE

This slide documents some of the new DB2 9 filtering keywords you may use to potentially lower the overhead of trace collection.

## DB2 Trace Typical CPU Overhead

- DB2 accounting trace
  - Class 2: 1 to 10% CPU overhead (higher percentage for fetch intensive applications)
  - Class 3: less than 1% CPU overhead (could be higher if latch contentions are higher)
  - Class 7 and 8 (Package level accounting): less than 5% CPU overhead
  - Class 10 (Package Detail) : higher CPU overhead and SMF volume
- DB2 monitor trace: similar to accounting
- DB2 statistics trace : negligible

The typical CPU overhead is mentioned here based on information in the DB2 manuals.

Remember mileage will vary – each user’s experience is different and can be workload dependent.



## DB2 Trace Records (extract from DSNWMSG5)

TYPE	CLASS	DATA COLLECTED	IFCIDS ACTIVATED	
STATISTICS	1	STATISTICAL DATA	1-2,105,106,202	
	2	INSTALLATION-DEFINED STATISTICS RECORD	152	
	3	DEADLOCK AND LOCK TIMEOUT INFORMATION, CONNECT OR DISCONNECT FROM A GROUP BUFFER POOL, LONG-RUNNING URS	172,196,250,261,262,313,258	
	4	DB2 EXCEPTION CONDITIONS	191,192,193,194,195,203,204,205, 206,207,208,209,210,235,236, (238),267-268	
	5	DATA SHARING GLOBAL STATISTICS	230	
	8	BUFFER POOL DATA SET STATISTICS	199	
	ACCOUNTING	1	ACCOUNTING DATA	3,106,239
		2	IN DB2 TIME	232
3		WAIT TIME FOR I/O,LOCKS, LATCHES, DRAINS, AND CLAIMS	6-7,8-9,32-33,44-45,(51-52,56-57) 117-118,127-128,170,171,174-175, 213-214,215-216,226-227,242-243	
4		INSTALLATION-DEFINED ACCOUNTING RECORD	151	
5		TIME SPENT PROCESSING IFI REQUESTS	187	
7		PACKAGE LEVEL ACCOUNTING IN-DB2 TIME	232,240	
8		PACKAGE LEVEL ACCOUNTING WAIT TIME IN DB2	6-7,8-9,32-33,44-45,(51-52,56-57) 117-118,127-128,170,171,174-175, 213-214,215-216,226-227,241-243	
AUDIT		1	AUTHORIZATION FAILURES	140
	2	EXPLICIT GRANT AND REVOKE	141	
	3	CREATE, DROP, AND ALTER OPERATIONS AGAINST AGAINST AUDIT TABLES	142	
	4	FIRST CHANGE OF AUDITED OBJECT	143	
	5	FIRST READ OF AUDITED OBJECT	144	
	6	SQL STATEMENT AT BIND	145	
	7	CHANGE IN AUTHORIZATION FOR AUDITED OBJECT	55,83,87,169,312	
	8	UTILITY ACCESS TO ANY OBJECT	23,24,25	
	9	INSTALLATION-DEFINED AUDIT RECORD	146	

Here is a sample extracted from the member DSNWMSG5 which defines the different classes and type of data collected.

## Monitoring Basics – PDB

- Long range history (daily, weekly, monthly, ...)
  - Store and accumulate over several days, weeks, years
  - Creation and updates of tables done manually
  - Load process can be defined and scheduled using scheduling software
  - Batch reporting SAVE/CONVERT and/or FILE function provides trace data in DB2 loadable format
  - Load the data using the DB2 LOAD utility
  - DDL and DML for Performance DB tables provided in sample library
- Write your own SQL against these tables for analysis and trending
- Summarize data on weekly and monthly basis
- Data can be grouped on an interval basis

This slide lists many of the advantages of a Performance Database.

## Monitoring Basics – PWH

- Long range history (daily, weekly, monthly, ...)
  - Installation option PERFORMANCEWAREHOUSE=YES
  - Automatic creation and updates of tables by PE Server
    - DDL and DML for Performance Warehouse not provided
  - Define and schedule load process from PE Client GUI
    - Batch SAVE/FILE function provides trace data in DB2 loadable format
    - DB2 load utility used to load data
- Analyze data
  - Use pre-defined ROT and SQL queries or write your own
  - Multiple reporting capabilities
  - Trend analysis available
- Supports ONLY statistics and accounting trace data

This slide lists many of the advantages of a Performance Warehouse.

## Performance Warehouse or Performance Database?

Both are a repository for performance data but there are differences:

	PDB	PWH
<b>CREATE</b>	MANUAL PROCESS	AUTOMATICALLY BY PE SERVER
<b>LOAD</b>	MANUAL PROCESS	SCHEDULED PROCESS
<b>ACCESS</b>	VIA SQL	VIA PE CLIENT PWH TOOL
<b>SW REQ'S</b>	N/A	DB2 CONNECT
<b>TYPE OF DATA</b>	ACCOUNTING, STATISTICS, AUDIT, SQL, etc.	ACCOUNTING & STATISTICS ONLY

This slide contains a summary of the differences between a PDB and a PWH as seen from the eyes of Tivoli Omegamon Performance Expert/Monitor for DB2 on z/OS!

## Trace Output

- Gather DB2 trace information from daily SMF data
- Collect Report Data (CRD) can also help
  - Use CRD to generate appropriate DB2 trace (IFCID) data in a sequential dataset
  - Run batch reports using this dataset as input
  - Requires a PWH be installed for the DB2 subsystem in order to use CRD from PE Client for that DB2
  - If you run CRD, you can only run accounting and statistics reports from the PWH. If you collected Audit trace records, run a batch job to get the Audit report!
- A batch CRD is available – we provide a sample batch job
  - FPEZCRD located in prefix.RKO2SAMP
  - Just collects trace data – you may need to add a report step

Other options available to you within Tivoli Omegamon Performance Expert/Monitor for DB2 on z/OS.

Other monitors on the market may have similar functions.

## Creating a Performance Database

- Consists of several tables logically grouped by trace types
- Before you can load these tables, the data must be made available in a non-aggregated format, compatible with the DB2 LOAD utility
- The FILE subcommand of the DB2 PE report facility generates a data set containing data in this non-aggregated format.
- The SAVE subcommand generates aggregated data and must be converted to a non-aggregated format compatible with the DB2 LOAD utility.
- The accounting and statistics Save-File utilities (see sample jobs DGOPJACO and DGOPJSCO in the prefix.SFPESAMP library) can be used for that purpose

This slide summarizes how to create a PDB.

## Performance Database – Accounting Tables

- Accounting tables are related in a 1:n relationship.
- General table is the parent table - all other tables are dependent
- For each row in the General table, there may be none, one, or many rows in the other tables.
- For accounting FILE tables, the column `TIMESTAMP` represents the relationship and can be used for joining data.
- For accounting SAVE tables, use columns `LOCAL_LOCATION` to `CLIENT_TRANSACTION` for joining related data in different tables

This slide summarizes the accounting tables and its relationships.

## Performance Database – Accounting Tables Detail

<b>General data</b>	One row per DB2 accounting record
<b>Group buffer pool</b>	One row per group buffer pool used
<b>Package data</b>	One row per package and DBRM executed
<b>DDF data</b>	One row per remote location participating in distributed activity
<b>Buffer pool data</b>	One row per buffer pool used
<b>RLF data</b>	One row per resource limit type encountered

This slide summarizes the type of data in each accounting table structure.



## Performance Database – Accounting Source RKO2SAMP

TYPE OF DATA	TABLE NAME	MEMBER for CREATE	DESCRIPTION	MEMBER for LOAD
General data	DB2PMFACCT_GENERAL	DGOACFGE	DGOABFGE	DGOALFGE
Group buffer pool	DB2PMFACCT_GBUFFER	DGOACFGP	DGOABFGP	DGOALFGP
Package data	DB2PMFACCT_PROGRAM	DGOACFPK	DGOABFPK	DGOALFPK
DDF data	DB2PMFACCT_DDF	DGOACDFD	DGOABDFD	DGOALDFD
Buffer pool data	DB2PMFACCT_BUFFER	DGOACFBU	DGOABFBU	DGOALFBU

This slide summarizes the different members in the sample library RKO2SAMP for the specific accounting table structure.

## Performance Database – Statistics Tables

- Statistics tables are related in a 1:n relationship.
- General table is the parent table - all other tables are dependent
- For each row in the General table, there may be none, one, or many rows in the other tables
- Use these columns for joining related data in different tables:
  - LOCAL\_LOCATION
  - GROUP\_NAME
  - SUBSYSTEM\_ID
  - MEMBER\_NAME
  - INTERVAL\_TSTAMP
  - BEGIN\_REC\_TSTAMP

This slide summarizes the statistics tables and its relationships.

## Performance Database: Statistics Tables Detail FILE Data

<b>General data</b>	One row for each statistics delta record, containing data from IFCID 0001 and 0002 *
<b>Group buffer pool data</b>	One row per group buffer pool active at the start of the corresponding delta record
<b>DDF data</b>	For each delta record, one row per remote location participating in distributed activity using the system-directed access method and one for all remote locations that used the application-directed access method
<b>Buffer pool data</b>	One row per buffer pool active at the start of the corresponding delta record
<b>Buffer pool data set</b>	One row for each open data set that has an I/O event rate at least one event per second during the reporting interval

\* = A delta record is a set of counters describing the DB2 activity between two consecutive DB2 statistics records pairs.

This slide summarizes the type of data in each statistics table structure for FILE data.

## Performance Database: Statistics Tables Detail SAVE Data

<b>General data</b>	One row for each statistics interval record, containing data from IFCID 0001 and 0002 *
<b>Group buffer pool data</b>	One row per group buffer pool active at the start of the corresponding interval record
<b>DDF data</b>	For each interval record, one row per remote location participating in distributed activity using the system-directed access method and one for all remote locations that used the application-directed access method
<b>Buffer pool data</b>	One row per buffer pool active at the start of the corresponding interval record
<b>Buffer pool data set</b>	One row for each open data set that has an I/O event rate at least one event per second during the reporting interval

\* = A Statistics interval record is a set of counters describing the DB2 activity within the interval specified by the user.

This slide summarizes the type of data in each statistics table structure for SAVE data.

## Performance Database: Statistics Source in RKO2SAMP

TYPE OF DATA	TABLE NAME	MEMBER for CREATE	DESCRIPTION	MEMBER for LOAD
General data	DB2PM_STAT_GENERAL	DGOSCGEN	DGOSBGEN	DGOSLGEN
Group buffer pool data	DB2PM_STAT_GBUFFER	DGOSCGBP	DGOSBGBP	DGOSLGBP
Buffer pool data	DB2PM_STAT_BUFFER	DGOSCBUF	DGOSBBUF	DGOSLBUF
DDF data	DB2PM_STAT_DDF	DGOSCDDF	DGOSBDDF	DGOSLDDF
Buffer pool dataset	DB2PM_STAT_DATASET	DGOSCSET	DGOSBSET	DGOSLSET

This slide summarizes the different members in the sample library RKO2SAMP for the specific statistics table structure.

## Sample DB2PM Control Statements for a SAVE Data Set

```
ACCOUNTING
      REDUCE
      FROM (,00:00) TO (,23:59)
      INTERVAL (60)
      BOUNDARY (60)
SAVE
  DATATYPE(GENERAL,DDF,PACKAGE)
  DDNAME (ACSAVDD1)
EXEC
```

Here is a sample set of control statements for a DB2PM accounting report.

Here we reduce the data on an hourly interval and save ONLY the general, DDF, and package data in a sequential file (with ddname of ACSAVDD1) to be loaded into a performance repository.

## Sample DB2PM Control Statements for a FILE Data Set

```
ACCOUNTING
  FILE
      DATATYPE(GENERAL)
      DDNAME (ACFILDD1)
  FILE
      DATATYPE(DDF)
      DDNAME (ACFILDD2)
  FILE
      DATATYPE(PACKAGE)
      DDNAME (ACFILDD3)
EXEC
```

Here is another sample set of control statements for a DB2PM accounting report.

Here we save the raw data - once again we are interested ONLY in general, DDF, and package data.

The three FILE parameters allow us to segregate the different data into 3 separate files.

This way we can load these tables in parallel and possibly save elapsed time during the load.

## Amica Mutual Insurance Company



- ✓ **Headquarters** One Hundred Amica Way, Lincoln, Rhode Island
- ✓ **Established** 1907—Amica is the oldest mutual insurer of automobiles in the United States.
- ✓ **Company Profile** Amica is a direct writer of personal lines insurance with Automobile, Homeowners, Personal Umbrella Liability, and Marine coverages. The company is well known in the industry for its financial strength and for providing exceptional customer service to policyholders.
- ✓ **Ratings**
  - A++ (Superior) from A.M. Best Company as of May 7, 2009
  - Repeatedly ranked #1 for customer service as a national auto insurer by a leading consumer publication.
- ✓ **Financial Strength** Assets at year-end 2008—\$3.6 billion
  - Premiums earned in 2008—\$1.33 billion
  - Surplus at year-end 2008—\$1.9 billion
  - Policies in force at year-end 2008—1.2 million



## Environments

- ✓ **Mainframes:** System z10 Business Class 2098-T04 and 2098-Q04.  
z/OS v1.10 operating system
- ✓ **DB2 z/OS** – Two production subsystems and two testing subsystems
  - Production CICS – 4 TB
  - Production PeopleSoft (HR, Fin) – 250 GB
  - Testing CICS, Core warehouse – 2 TB
  - Testing PeopleSoft – 900 GB
- ✓ **DB2 V9.x LUW** – IBM Content Manger, Data Warehouse – 352 GB,  
Billing system, etc.
- ✓ **Performance Database** – 16 months of Accounting and Statistics  
records for the two production subsystems. Located on the Testing CICS  
subsystem ~15 GB

AMICA has two z/OS machines, one for testing and one for production.

The production machine has two DB2 subsystems

- One for our production CICS applications
- And one for our production Peoplesoft application.

The testing machine also has two subsystems

- One for our (testing CICS applications and our Core Data warehouse)
- And one for our testing Peoplesoft application.

The performance database resides on our testing CICS subsystem and it holds 16 months of Accounting and Statistics records for the two production subsystems.

The size of our performance database is approximately 15 GB.

## Which to use – Performance Warehouse or Performance Database

### Performance Database

#### Pros

- ✓ Stores historical data
- ✓ Can hold reports sets for Accounting, Statistic, Audit, Locking, Record Trace, Exception and System Parameters
- ✓ You create the database objects, therefore it's customizable
- ✓ DDL is located in sample library
- ✓ You load the data. We use a mainframe scheduling product.
- ✓ Data can be aggregated or non-aggregated
- ✓ Can automate deletion of old data

#### Cons

- ✓ You have to write your own queries (but you can 'adapt queries' from PWH)
- ✓ You have to really understand the relationships between the tables
- ✓ You have to really understand the column definitions \*
- ✓ You have to create the load jobs
- ✓ Only historical data

\*Sample create table, load control and table column descriptions are located in 'The Reporting User's Guide' Chap. 19, pg 241 SC18-7979-00

The PRO's for the performance database may be that

It stores historical data

It can hold report sets for Accounting, Statistics, Audit, Locking, Record traces, Exceptions and system parameters

The DDL is provided and can be customized ( for example, we partitioned the tables for ease of maintenance)

We use our mainframe scheduling product to load and delete the data

The data can be aggregated or non-aggregated (the acct file tables hold the individual accounting records and the acct summary tables hold the summarized accounting records)

The CONS for the performance database may be

You have to write your own queries (but you can adapt queries from the performance warehouse)

You have to understand the relationships between the tables and the column definitions

You have to create the load jobs

It ONLY contains historical data

(we load the tables from the SMF records from the previous day therefore we cant look at today's transactions)

## Which to use – Performance Warehouse or Performance Database

### Performance Warehouse

#### Pros

- ✓ Can capture current data via traces
- ✓ Can store historical data by reading SMF datasets
- ✓ Only holds report sets for Accounting and Statistics
- ✓ Can easily share queries via GUI
- ✓ A number of canned queries and rule of thumb queries provided
- ✓ Useful for debugging currently active problems (using trace & report)

#### Cons

- ✓ Database is automatically created by Omegamon Server, and not customizable
- ✓ Data is only loaded via Omegamon sever, can be scheduled.
- ✓ Two-step manual process to delete old data
- ✓ Separate PWH on each DB2 subsystem
- ✓ Can only select against a single schema (it's hard coded - DB2PM)

### Amica's Choice:

We primarily use PDB for historical analysis and occasionally use PWH for problem analysis as it's happening.

The PRO's for the performance warehouse may be that

It can capture current data via traces

It can store historical data by reading SMF datasets

It only holds report sets for Accounting or Statistics records

It can easily share queries via GUI (you are able to see the queries from other users of the tool)

A number of canned queries and “Rules of Thumb” queries are provided

Useful for debugging currently active problems (using trace and report options)

The CONS for the performance warehouse may be

The database is created by Omegamon and it is not customizable

Data is ONLY loaded by the Omegamon server but it can be scheduled

Deleting data is a two step manual process

A separate Performance Warehouse is needed on each subsystem

You can only select against a single schema (It's hard coded – DB2PM)

At AMICA, we primarily use the PDB for historical analysis and we occasionally use the PWH for problem analysis as it's happening.

## Amica's Performance Database - Statistics

**PDB Statistics data is collected from Amica's SMF records**

### ✓ Statistics Data

- CICS and Peoplesoft production subsystems
- Collected daily
- 16-month retention period (rotating partitions)
- Summarized five minute period (zparm setting)
- Captured using the Statistics Report with the "FILE" command, then sorted to different load files and loaded directly into the Statistics tables

```
GLOBAL
  FROM (05/14/09,00:00:00.00)
  TO   (05/14/09,24:00:00.00)
STATISTICS
FILE
  INCLUDE(SUBSYSTEMID(DSN, PSFP))
NOEXCEPTION
DDNAME(ACFIL01)
EXEC
```

Basically, the AMICA PDB is set up as follows:

The information is collected from the SMF records.

We are collecting data for our two production applications.

The statistics tables contain daily information that is summarized within 5 minute intervals over the course of the entire day.

The 5 minute interval is determined by a DB2 ZPARM setting.

We are keeping 16 months of data and we roll off the old data a month at a time.

We changed the DDL to partition the tables by the interval time. This gave us the ability to remove the old data by rotating the partitions to roll off the old month and add the new month. We do this through an automated monthly job.

The DB2PM cards are show above.

The statistics records are captured with the "FILE" command and the records are sorted and loaded directly into the PDB Statistics tables.

As you can see in the cards, we are only including our production subsystems – DSN and PSFP and we are collecting data for the whole day.

This is run at 6am daily and the database is available for the remainder of the day.

Note: The times within the performance database are stored in Greenwich Mean Time

## Amica's Performance Database - Accounting

PDB Accounting data is collected from Amica's SMF records

### ✓ Accounting Data

- CICS production subsystem, at this time
- Collected daily, Monday through Friday, core business hours
- Summarize on the Connection Type, Correlation Name and Interval time, 15-minute intervals
- 16-month retention period (rotating partitions)
- Captured using the "REDUCE, REPORT" commands and the "SAVE" file utility
- "CONVERT" step converts the saved file (VSAM) into a sequential dataset. Then sorted to different load files and loaded directly into the Accounting tables.

```

GLOBAL
  TIMEZONE(+04:00)
  FROM (05/14/09,08:00:00.00)    TO(05/14/09,17:15:00.00)
  INCLUDE (SUBSYSTEMID(DSN))    PRESORTED(NO)
ACCOUNTING
  REDUCE
    INTERVAL(15) BOUNDARY(60)
  REPORT
    DDNAME(ACRPTDD) LAYOUT(SHORT)
    NOEXCEPTION    SCOPE(MEMBER)
    ORDER(CONNECT-CORRNAME-INTERVAL)
  SAVE
    DDNAME(ACSAVDD)
EXEC
  
```

```
// EXEC PGM=DGOPMICO,
// PARM=CONVERT
```

The accounting data is collected from the SMF records as well.

The accounting summary tables contain accounting information that is summarized over a 15 minute interval.

We are capturing data for our CICS production subsystem – DSN.

Our accounting summary tables also contain 16 months of data and these are partitioned for ease of maintenance.

We are summarizing on Connection Type, Correlation Name and Interval Time.

As you can see in the DB2PM cards above we are capturing the accounting records from our production subsystem (DSN) during our core business hours 8 am to 5:15 pm.

We are summarizing the records on a 15 minute interval and we are ordering the records by the three columns CONNECTION , CORRELATION NAME, and INTERVAL TIME.

We are also starting the summary interval on the top of the hour with the BOUNDARY(60) clause.

This process creates a VSAM file. The CONVERT step is needed to change this VSAM file into a sequential dataset, which can then be sorted and loaded into the

## Example 1 - zIIP Engine Overflow

- ✓ **Problem** – System Administration group notice the zIIP engine usage had jumped radically, from ~2% to 17-18% usage, overnight.
- ✓ **1<sup>st</sup> Analysis** – Initially we used Accounting Long reports. This proved too cumbersome. Reports had too much detail and took an hour-plus to run.
- ✓ **2<sup>nd</sup> Analysis** – Using QMF to query the Performance Database (PDB), we were able to quickly generate reports and graphs zeroing in on only plans that had high zIIP engine usage. We identified multiple plans that had one module in common. The module had recently been changed.
- ✓ **PDB SQL:**

```
SELECT SUM(CLASS2_IIP_CPU),PLAN_NAME,DATE(INTERVAL_TIME)
FROM DB2PWH.DB2PMSACCT_GENERAL
WHERE
  SUBSYSTEM_ID ='DSN'
  AND INTERVAL_TIME > '2009-05-04-09.00.00.000000'
  AND INTERVAL_TIME < '2009-05-28-15.30.00.000000'
GROUP BY PLAN_NAME,DATE(INTERVAL_TIME)
ORDER BY PLAN_NAME,DATE(INTERVAL_TIME)
```

Now I would like to present some ways that we used our performance database to solve problems.

We received a call one day from our systems administration group that our ZIIP engine usage had jumped from approximately 2% to 18% overnight. At first we started running accounting long reports. This proved to be cumbersome. The reports had too much detail and they took over an hour to run.

We decided to use the performance database and we found a column for the ZIIP engine CPU. It was the CLASS2\_IIP\_CPU field.

We easily wrote a query looking for plans with a high ZIIP engine usage. We discovered that multiple plans were returned and each of them had a common program. We also learned this program had been changed the previous day.

I have included the query that we ran. As you can see, we summed up the zIIP engine usage and we grouped the results by plan and date.

This gave us the ZIIP usage grouped by plan for almost every day in May. At his point, we could then easily identify the plans where the ZIIP usage had spiked.

## Example 1 - zIIP Engine Overflow

- ✓ **The Explain:** Using Optimization Service Center (OSC) to explain the package SQL, we quickly identified one costly SQL statement, with a cost of 300 su. OSC showed the query was using a non-matching index scan on the second column of the index.
- ✓ **Investigation:** Using the DB2 Administration tool we were able to see that there were three indexes defined on the table, all with the same first two columns, one of which had just the first two columns. We dropped that index and created one with just the column being scanned. This dropped the cost to about eight su.
- ✓ **What Was Happening:** The query was using parallelism which caused the high number of calls and get pages

Once the program was identified, we used the IBM Optimization Service Center to explain the SQL. We quickly found one costly SQL statement with a cost of 300 service units (su).

This query was doing a non-matching index scan on the second column of the index. Although, 300 su is usually not a bad cost for a query, we later discovered this program was being executed over 14,000 times per hour.

When we investigated the table, we found that it had 3 indexes, two of which were redundant. We dropped the redundant index and created a new index on the column being scanned above. This dropped the cost of the query to about 8 su.

As a result of the non-matching index scan, the query was spawning a high number of parallel tasks and it was retrieving a high number of getpages.

## Example 1 - zIIP Engine Overflow

✓ **By the Numbers:** The execution time for the calls (according to IBM DB2 Query Monitor) averaged about .005 second; not a bad time but this module was executing about 425,000 calls per hour. After the index change it dropped to .0003 second and 76,000 calls to DB2, there by reducing the get pages from almost 50 million to 508 thousand .

### Module statistics before and after (from IBM DB2 Query Monitor):

Program	Occurrences	Calls	Elapsed	Avg Elapsed	CPU	Avg CPU	GetPages
BUY1	14,814	425,704	03:23.8	0.008935	38:27.5	0.00542	49,678,590
BUY2	16,016	76,900	01:21.7	0.001062	00:21.2	0.00027	508,425

We used the IBM Query Monitor tool to review the results with our new index.

In the chart above, the first line (program BUY1) is the initial problem and the second line (program BUY2) is the results with the new index.

As you can see, the number of parallel tasks (the CALLS column) dropped from 425,000 to approximately 76,000 and the getpages dropped from 49,000,000 to 508,000. As a result of this, the average elapsed times and the average CPU times were also significantly lower.

This same information could have been returned from the performance database instead of IBM's Query Monitor but we would have had to wait for the following day for the information to be loaded from the SMF records.



## Example 2 – RID Pool Failures

✓ **Adapt.** You can 'adapt queries' from the Performance Warehouse (PWH) and use them on your PDB. The following RID pool failure query was copied from the PWH queries and modified to evaluate our production subsystem:

```
SELECT SUM(RID_POOL_FAIL_MXLT) AS RIDPOOL_FAILURES, PLAN_NAME,
SUBSYSTEM_ID, MAINPACK
FROM DB2PWH.DB2PMSACCT_GENERAL
WHERE DATE(INTERVAL_TIME) BETWEEN '2009-05-01' AND '2009-05-31' AND
SUBSYSTEM_ID = 'DSN'
GROUP BY PLAN_NAME, SUBSYSTEM_ID, MAINPACK
HAVING SUM(RID_POOL_FAIL_MXLT) > 0
ORDER BY 1 DESC
```

✓ **Results** The number of RIDPOOL failures for the month of May is as follows:

Rid Pool Failures	Plan Name	Main Pack
3530	AR21	IRC
207	AR13	LWK
1	CT	CT

The second example refers to RIDPOOL failures. You can adapt queries from the performance warehouse and use them on your performance database. The following RID pool failure query was copied from the PWH queries and modified to evaluate our production subsystem

As you can see from the SQL, we are looking for the sum of RID pool failures and we are grouping the sum by the plan, package and subsystem.

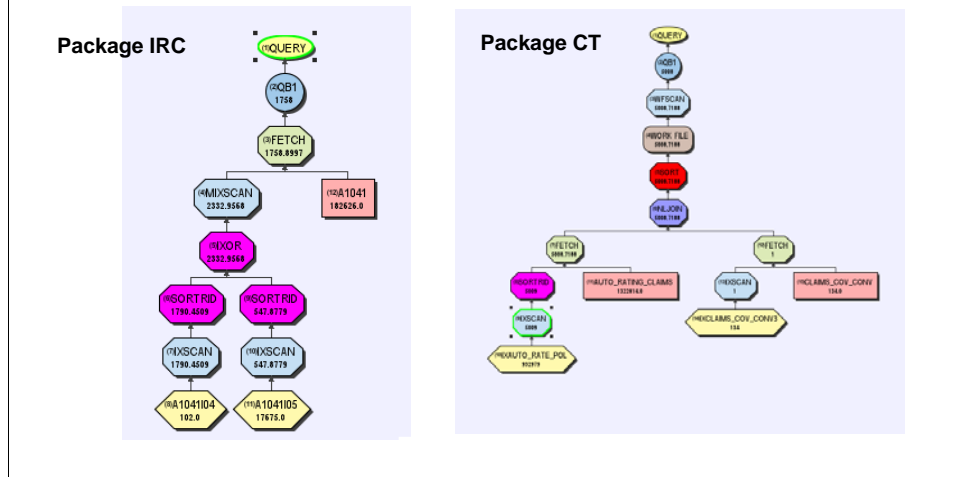
We are looking for the number of failures for the whole month of May. The results of the queries are listed in the chart above.

Package IRC and LWK run in our CICS applications and package CT ran as a batch job.

We were surprised that we had so many failures in our CICS application and we never realized it.

## Example 2 – RID Pool Failures

✓ **Analysis:** Using Optimization Service Center (OSC), we found that packages IRC and LWK were doing index OR'ing on two indexes. Package CT was sorting RIDs from an Index Scan. We increased the size of the RIDPOOL to prevent RIDPOOL failures. The initial RIDPOOL size was approximately 27 Meg. We increased it to 54 Meg and eliminated the RIDPOOL failures.



Using the IBM Optimization Service Center, we discovered that packages IRC and LWK were doing index OR'ing on two indexes.

Package CT was sorting RID's from an index scan.

We increased the size of the RIDPOOL to prevent RIDPOOL failures. The initial size was approximately 27 Meg.

We increased the size of the RIDPOOL to 54 meg and eliminated the failures.

The performance warehouse has quite a few queries that can be run to find performance problems. This was just one example. The queries can be run from within the performance warehouse or modified slightly to run against the performance database. I used the performance database in this presentation because that is where we keep our historical information.

## Example 3 – Expensive Packages

The following query shows the class 7 elapsed time by package, and the output is sorted slowest to fastest.

```

SELECT PCK_COLLECTION_ID, PCK_ID,
       AVG(CLASS7_ELAPSED/PCK_ALLOCS_CLASS7) AS AVG_CLASS7_ELAPSED,
       SUM(PCK_ALLOCATIONS) AS NO_TIMES_PACKAGE_EXECUTED
FROM DB2PWH.DB2PMSACCT_PROGRAM
WHERE CONNECT_TYPE = 'CICS'
AND INTERVAL_TIME BETWEEN '2009-08-24-08.00.00.000000'
                       AND '2009-08-28-16.00.00.000000'
AND PLAN_NAME IN ('AR10', 'AR24', 'AR08', 'AR13', 'AR05')
GROUP BY PCK_COLLECTION_ID,
         PCK_ID
ORDER BY 3 DESC
FETCH FIRST 20 ROWS ONLY

```

The third example was written to look for our most expensive packages. The following query shows the average class 7 elapsed time by package and the output is sorted slowest to fastest. The query is also looking at our production CICS application plans for a week in August.

As you can see in the query on the second line, we are calculating the average of the CLASS7\_ELAPSED time DIVIDED BY the number of times the package was called in this interval (the PCK\_ALLOCS\_CLASS7). Remember, we are using summary tables and the information has been pre-summarized in a 15 minute interval specified in the DB2PM job. If you want to find the average times per package, you have to divide the time (which has been summed within the interval) by the number of times the package was called within the interval.

For Example, when calculating averages, the CLASS 7 times in the accounting summary package table (DB2PMSACCT\_PROGRAM) must use the CLASS7 divisor PCK\_ALLOCS\_CLASS7. This is shown in the SQL statement above. All “class7” times have columns that are prefixed with “CLASS7” and each must use this divisor when calculating averages.

There are also CLASS8 times in this table as well and the CLASS8 times have a separate divisor. The CLASS8 divisor is the PCK\_ALLOCS\_CLASS8 column. The class8 times have column names that are prefixed with “CLASS8.”

When calculating averages for all other columns that contain times or counters in this

### Example 3 – Expensive Packages

PCK_COLLECTION_ID	PCK_ID	AVG_CLASS 7_ELAPSED	NO_TIMES_PACKAGE_EXECUTED
BIABIAS	BX31EQU	0.69388	80312
AR13	LWK	0.50716	60
AR13	LYB	0.46990	149
AR13	LZO	0.43487	254
AR13	DWU	0.24926	19
AR13	LWR	0.23208	2
AR13	LMY	0.17670	3
AR10	IFB	0.16259	8114
AR13	RSK	0.14181	155
AR24	HIS	0.12111	1414

After evaluating the SQL in the worst packages, we discovered tablespace scans and inefficient indexes, IX-ORing and IX-ANDing with thousands of rows being merged and non-matching index scans.

The results of the query are shown above.

As you can see in the chart, the slowest packages and the number of times they are executed are on top.

After evaluating the SQL within these packages, we found tablespace scans, inefficient indexes, IXOR'ing and IXAND'ing with huge indexes being merged and non-matching index scans.

The number of times the package is executed is useful to determine whether or not time should be spent tuning these queries.

In our example, our worst package was executed over 80,000 times in a 5 day period and in this case, we needed to create an index for this SQL statement to run efficiently.

This query was useful in showing us the slowest packages in our CICS applications. We were also surprised that every package in this list had an SQL statement that needed to be rewritten or it identified indexes that needed to be created or changed.

## Example 4 – Comparison Over Time

- ✓ The query on the next foil will compare the Class 7 elapsed times for a business week in August 2008 with a business week in August 2009.
- ✓ Percentage Change is calculated showing the difference in the elapsed time.
- ✓ This could be useful when comparing programming changes to an application or migrating to a new version of DB2.
- ✓ The query could be modified to look at different class times or to check runtimes on specific packages as changes are made.

The query on this foil will compare the CLASS7\_ELAPSED time for a week in August 2008 to a week in August 2009.

Percentage Changed is calculated showing the difference in the elapsed times.

This is useful when comparing changes to an application or migrating to a new version of DB2

## Example 4 – Comparison Over Time

```

SELECT PCK_COLLECTION_ID , PCK_ID
      , MAX(AVG_CLASS7_ELAPSED_2008) AS AVG_CLASS7_ELAPSED_2008
      , MAX(AVG_CLASS7_ELAPSED_2009) AS AVG_CLASS7_ELAPSED_2009
      , INT(MAX(AVG_CLASS7_ELAPSED_2009)/MAX(AVG_CLASS7_ELAPSED_2008) * 100) AS
PERCENT_CHANGED
FROM (SELECT PCK_COLLECTION_ID , PCK_ID
      ,(CASE
        WHEN YEAR(INTERVAL_TIME) = 2009
        THEN AVG(CLASS7_ELAPSED/PCK_ALLOCS_CLASS7)
        ELSE 0
      END) AS AVG_CLASS7_ELAPSED_2009
      ,(CASE
        WHEN YEAR(INTERVAL_TIME) = 2008
        THEN AVG(CLASS7_ELAPSED/PCK_ALLOCS_CLASS7)
        ELSE 0
      END) AS AVG_CLASS7_ELAPSED_2008
FROM DB2PWH.DB2PMSACCT_PROGRAM
WHERE CONNECT_TYPE = 'CICS'
      AND (INTERVAL_TIME BETWEEN '2008-08-24-08.00.00.000000' AND '2008-08-28-16.00.00.000000'
      OR INTERVAL_TIME BETWEEN '2009-08-24-08.00.00.000000' AND '2009-08-28-16.00.00.000000')
      AND PLAN_NAME IN ('AR10', 'AR24', 'AR08', 'AR13', 'AR05')
GROUP BY PCK_COLLECTION_ID, PCK_ID , YEAR(INTERVAL_TIME) ) AS TEMP
HAVING
((MAX(AVG_CLASS7_ELAPSED_2009)/MAX(AVG_CLASS7_ELAPSED_2008) * 100) > 0) -- greater than zero shows
all packages, greater than 100 shows slower packages, between 0 and 100 shows faster packages.
AND (MAX(AVG_CLASS7_ELAPSED_2008) ) > 0 -- to eliminate divisions by zero
ORDER BY 5 DESC
FETCH FIRST 12 ROWS ONLY

```

This query is set up to compare two timeframes through one pass of the data.

The query could be changed to look at different class times for an application. Examples of different class times could be CLASS7\_CPU, CLASS8\_SYNC\_IO, or CLASS8\_LOCK\_LATCH. You could also change the query to compare runtimes on specific packages as changes are made to the program or the SQL.

This example compares a week this year to a week last year but you can easily change the query to compare week to week, day to day, etc.

The “having clause” near the bottom of the page is dividing the 2009 time by the 2008 time to calculate a percentage changed.

Comparing this value to >0 will return all rows

Comparing this value to > 100 will return packages that are slower in 2009

Comparing this value between 0 and 100 will show packages that are faster in 2009

We are ordering the results by the percent changed descending so the biggest discrepancies show up on top.

Division by zero needs to be checked for packages that exist in 2009 but did not in 2008.

**Example 4 – Comparison Over Time**

PCK COLLECTION ID	PCK_ID	AVG CLASS 7 ELAPSED 2009	AVG CLASS 7 ELAPSED 2008	PERCENT CHANGED
EDB	ERI	0.03163	0.00517	611
AR13	LWH	0.01072	0.00198	541
AR13	RLU	0.00533	0.00136	391
AR13	LMB	0.01108	0.00315	351
AR24	PYU	0.00422	0.00121	348
EDB	EHY	0.00385	0.00112	343
AR03	GTS	0.00489	0.00151	323
EDB	EIS	0.00343	0.00107	320
EDB	EHB	0.00545	0.00173	315
EDB	ERD	0.01107	0.00363	304
AR13	LMY	0.15451	0.0513	301
ACFN	BJG	0.03838	0.01304	294

As you can see, these are the packages that are slower in 2009 and we are sorting with the slowest on top.

We used this query when we migrated to DB2 version 9. We were able to compare packages on version 9 with the same packages a year back on version 8.

## Example 5 – Buffer Pool Tuning

- ✓ **Information gathering:** The two queries on the next slides were run against the PDB to gather the bufferpool hit ratios and DB2 elapsed time for our CICS production applications.
- ✓ **Frequency:** The buffer pool hit ratio from PDB was collected weekly and plotted in Microsoft Excel. We were able to gather this information and compare the bufferpool usage and DB2 elapsed times from week to week.
- ✓ **Buffer Pool Analysis:** We used the IBM Bufferpool Analyzer (BPA) tool to move objects to different buffer pools and we used the BPA tool to change the bufferpool sizes as well.
- ✓ **Effectiveness:** The Performance database gave us the flexibility to make changes to the buffer pools and then compare the effect of these changes from week to week.

My last example was used to monitor bufferpool changes.

The two queries on the next slides were run against the PDB to gather the bufferpool hit ratios and the DB2 elapsed time for our CICS production applications. The bufferpool hit ratios were collected weekly and plotted in Microsoft Excel. We were able to gather this information and compare the bufferpool usage and DB2 elapsed times from week to week.

We used the IBM Bufferpool Analysis tool to change bufferpool sizes and characteristics and we moved objects to different pools as well.

The PDB gave us the flexibility to make changes to the bufferpools and then compare the affect of these changes from week to week.



## Example 5 – Buffer Pool Tuning

```

SELECT CASE
  WHEN CHAR(A.BP_ID) = '0' THEN 'BP0'
  WHEN CHAR(A.BP_ID) = '1' THEN 'BP1'
  WHEN CHAR(A.BP_ID) = '2' THEN 'BP2'
  WHEN CHAR(A.BP_ID) = '3' THEN 'BP3'
  WHEN CHAR(A.BP_ID) = '4' THEN 'BP4'
  WHEN CHAR(A.BP_ID) = '5' THEN 'BP5'
  WHEN CHAR(A.BP_ID) = '6' THEN 'BP6'
  WHEN CHAR(A.BP_ID) = '9' THEN 'BP9'
  WHEN CHAR(A.BP_ID) = '80' THEN 'BP32K'
  WHEN CHAR(A.BP_ID) = '81' THEN 'BP32K1'
  WHEN CHAR(A.BP_ID) = '87' THEN 'BP32K7'
  WHEN CHAR(A.BP_ID) = '121' THEN 'BP16K1'
  WHEN CHAR(A.BP_ID) = '125' THEN 'BP16K5'
  ELSE 'UNKNOWN'
END,
A.THE_DATE, A.SUBSYSTEM_ID, A.THE_HOUR, AVG (CASE WHEN A.GET_PAGE > 0
  THEN (INT (( (A.GET_PAGE - (A.SYNC_READ_IO + A.SEQ_PREFETCH_PAGE +
  A.LIST_PREFETCH_PAGE + A.DYN_PREFETCH_PAGE )) / A.GET_PAGE ) * 100 ) END) AS HIT_RATIO
FROM
(SELECT BP_ID, DATE(BEGIN_REC_TSTAMP) AS THE_DATE, SUBSYSTEM_ID,
  HOUR (BEGIN_REC_TSTAMP) AS THE_HOUR, HPOOL_READ_SYNC, HPOOL_READ_ASYNC,
  HPOOL_WRITE_SYNC, HPOOL_WRITE_ASYNC, GET_PAGE, SYNC_READ_IO,
  SEQ_PREFETCH_PAGE, LIST_PREFETCH_PAGE, DYN_PREFETCH_PAGE
FROM DB2PWH.DB2PM_STAT_BUFFER
WHERE DATE (BEGIN_REC_TSTAMP) BETWEEN '2007-01-15' AND '2007-01-19'
  AND HOUR(BEGIN_REC_TSTAMP) BETWEEN 08 AND 16 AND SUBSYSTEM_ID
GROUP BY A.BP_ID, A.THE_DATE, A.SUBSYSTEM_ID, A.THE_HOUR
ORDER BY A.BP_ID, A.THE_DATE

```

This query calculates the bufferpool hit ratios using the bufferpool statistics table (DB2PM\_STAT\_BUFFER). This information was gathered for an entire week (Monday thru Friday). We are also grouping the data by bufferpool id, subsystem, date and hour.

## Example 5 – Buffer Pool Tuning

```
SELECT PLAN_NAME,  
       AVG(CLASS2_ELAPSED) AS CLASS2_ELAPSED  
FROM   DB2PWH.DB2PMSACCT_GENERAL  
WHERE  INTERVAL_TIME BETWEEN  
       '2007-01-15-08.00.00.000000' AND '2007-01-19-16.15.00.000000'  
       AND PLAN_NAME IN ('AR08', 'AR10', 'AR13', 'AR24', 'LYBIAS00')  
GROUP BY PLAN_NAME  
ORDER BY PLAN_NAME
```

This query was used to calculate the CLASS2\_ELAPSED time for our production CICS application plans that we were monitoring.

## Example 5 – Buffer Pool Tuning

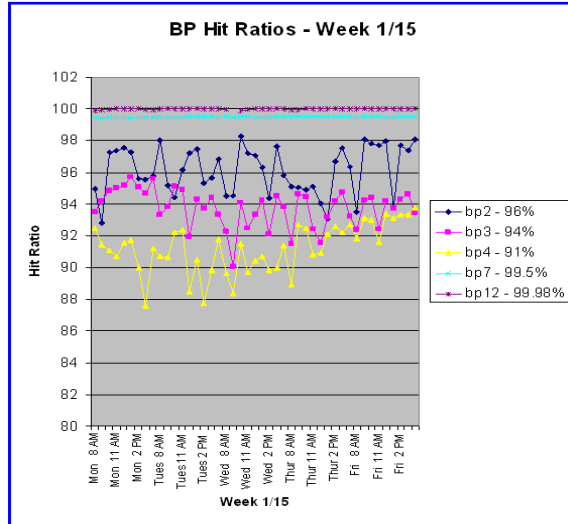
Week of 01/15/2009

Bufferpool Size	Avg. Hit Ratios	Plan	Tot Class 2 Elap	Avg. prior to BPA Changes		
BP2	26,500	96%	AR08	0.078908	Mon.	0.030169
BP3	108,180	94%		0.027469	Wed.	
BP4	83,000	91%				
BP7	22,030	99.50%	AR10	0.017257	Mon.	0.019336
BP12	9,000	99.98%		0.017661	Wed.	
			AR13	0.024965	Mon.	0.028605
				0.027186	Wed.	
			AR24	0.044425	Mon.	0.047128
				0.038953	Wed.	
			LYBIAS00	0.038621	Mon.	0.042792
				0.030108	Wed.	

This chart shows an example of how we summarized the information for a given week. We recorded the bufferpools that were being changed and the new size of each bufferpool. The plans were listed with the Average Class2 Elapsed times that were recorded on Mondays and Wednesdays.

The last column showed the times before any changes were made. We would record this information from week to week to compare how the changes were affecting our applications.

## Example 5 – Buffer Pool Tuning



This graph shows how we plotted the bufferpool hit ratios from the results of the bufferpool hit ratio query. Plotting the hit ratios in Excel helped us to determine if the changes were beneficial from week to week.

## Summary

- A PDB gives you the opportunity to know all there is to know about your DB2 subsystems
- Often too much data to store
- Consider reducing the data on an interval basis as a single record
- Refer to these tables regularly looking for areas of improvement
- Also useful when your system is acting strange and you need to determine why
- Load the tables daily with a summarization of the activity from the previous day to reduce the amount of data kept
- Keep the data for several months or years for historical analysis
- Use the provided examples to help you solve potential performance problems

-The PDB or PWH gives you the opportunity to know all there is to know about your DB2 subsystems or DB2 applications

-Often, there is too much data to store (especially if you capture the individual accounting records in the accounting file tables)

-Consider reducing the data on an interval basis as a single record (the accounting summary tables)

-Refer to these tables regularly looking for areas of improvement

-They can also be useful when your system is acting strange and you need to determine why

-Keep the data for several months or years for historical analysis

-Use the provided examples to help you solve potential performance problems.

## Resources and References

- PDB and PWH information
  - Chapter 19, "The Performance Database and the Performance Warehouse", DB2 Performance Monitor for z/OS Reporting User's Guide, SC18-7979-04
  - Redbook - IBM DB2 Performance Expert for z/OS Version 2, SG24-6867 – Chapter 5 on PDB and PWH
- General reporting information including details about the FILE and SAVE subcommands
  - IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS, Report Reference, SC19-2504

Mary and I have included some resources and references.

PDB and PWH information can be found in

the DB2 Performance Monitor for z/OS Reporting User's Guide

the Redbook – IBM DB2 Performance Expert for z/OS Version 2

ALSO

General reporting information including details about the FILE and SAVE subcommands can be found in the

IBM Tivoli Omegamon XE for DB2 Performance Expert on z/OS

And IBM Tivoli Omegamon XE for DB2 Performance Monitor on Z/os Report Reference

**Mary Petras – [marypetr@us.ibm.com](mailto:marypetr@us.ibm.com)  
Mark Wilkinson – [mwilkinson@amica.com](mailto:mwilkinson@amica.com)**

Many thanks for attending our session – Session F13 - How a Performance Database Can Help Save The Day

Here are our speaker bio's:

**Mark Wilkinson, Amica Mutual Insurance Co., [mwilkinson@amica.com](mailto:mwilkinson@amica.com)**

Mark Wilkinson has been working with DB2 for the past 16 years. He is currently a Database

Administrator for the Rhode Island based Amica Mutual Insurance Company, the oldest mutual

insurer of automobiles in the nation. Mark has been working in this position for 10 years. He is the

DBA that is responsible for managing the Amica data warehouse on the Z/OS and LUW platforms.

His responsibilities include tuning on both platforms. Mark trouble shoots and maintains databases on

DB2 Z/OS and LUW SQL. He uses the Amica DB2 performance database and the Amica DB2

performance warehouse day to day to perform his duties.

**Mary Petras, IBM Corporation, [marypetr@us.ibm.com](mailto:marypetr@us.ibm.com)**

Mary Petras is a Consulting Product Design Professional at the IBM Silicon Valley Lab and works on

the DB2 Tools Technical Specialist team supporting DB2 for z/OS Tools. Her areas of expertise

include DB2 data sharing, DB2 utilities, and performance and tuning. Mary co-authored the redbooks: