

2-6 October, 2006

Hilton Vienna

Vienna, Austria

IDUG® 2006

Europe

**B04**

*DB2 Legacy Migrations: Experiences of  
The Industry*

Susan Lawson and Dan Luksetich  
YLA

Oct 3, 2006 • 09:00 – 10:00

Platform: z/OS

GoFurther



Susan Lawson and Dan Luksetich a combination of more than 36 years of hands on DB2 experience in many areas. The recommendations for these areas researched came from knowing what works and what does not in these typical environments.

Susan and Dan specialize in legacy migrations, performance auditing, education, architecture and database design for high performance. They have worked on some of the largest and most complex database implementations in the world. This presentation is a product of several of those experiences.

## Abstract

**In order to make data into information we must have it in a flexible, easy-to-use data store such as DB2 z/OS. This presentation will take a look at several different approaches to migrating legacy data to DB2. We will look at what has worked well and what does not work at all. There have been some very creative approaches to achieving performance while not completing converting the application at the same time. There have also been some disasters when expectations were not correctly set and the future was not considered during the conversion effort.**



### 5 Client Examples:

- DB2 and Java – complete rewrite of application
- DB2 with legacy application programming interface
- DB2 with an I/O layer approach to legacy
- DB2 CICS migrating to DB2 Websphere
- A few other smaller conversion efforts.

Objective 1: Look at 5 major types of conversion efforts

Objective2: Discuss how some conversions have been successful

Objective3: Discuss common pitfalls of conversions

Objective4: Learn how to set expectations

Objective5: Look at some cool ideas for helping performance

### **Disclaimer PLEASE READ THE FOLLOWING NOTICE**

---

- **The information contained in this presentation is based on techniques, algorithms, and documentation published by the several authors and companies, and in addition is the result of research. It is therefore subject to change at any time without notice or warning.**
- **The information contained in this presentation has not been submitted to any formal tests or review and is distributed on an “As is” basis without any warranty, either expressed or implied.**
- **The use of this information or the implementation of any of these techniques is a client responsibility and depends on the client’s ability to evaluate and integrate them into the client’s operational environment.**
- **While each item may have been reviewed for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.**
- **Clients attempting to adapt these techniques to their own environments do so at their own risks.**
- **Foils, handouts, and additional materials distributed as part of this presentation or seminar should be reviewed in their entirety.**



DB2 is a registered trademark of IBM Corporation.

No animals were harmed during testing.

## Legacy Migrations

---

- **Issues with Legacy Migrations**
  - **Generic Development**
  - **Blind Development**
  - **“Smart” Transaction Processing**
- **Validation and Proof of Concept**
- **Balancing Performance, Availability, and Compatibility**
  - **“Fat” versus “Skinny” Cursors**
  - **Programmatic versus SQL Joins**
  - **Getting Just the Data You Need**
  - **Using Indicator Columns for Performance**
  - **Availability Example: “Instant” Data Recovery**
  - **Generic Attach**
- **Five Legacy Migration Case Studies**



Legacy migration refers to converting existing data to a new database or converting an existing application to a new language. When we convert from a legacy data store or other Database Management System (DBMS) to DB2 z/OS, there are some specific considerations. We must handle DB2 z/OS differently if we want superior performance and availability. There are creative solutions to consider and common mistakes to avoid.

Turning data into information requires getting it in a flexible, easy-to-use data store such as DB2 z/OS. This presentation looks at several different approaches to migrating legacy data to DB2. It explores:

- What has worked well and what hasn't
- How to improve performance without completely converting the application
- The disastrous results of failing to properly plan and set expectations
- How to set expectations for successful conversions
- Five major types of conversion efforts
- Common pitfalls of conversions
- Techniques for helping performance in database and application migrations.

## Generic Development

---

- **Can be a legacy conversion or new application**
- **Legacy Conversion**
  - Old data stores moved into DB2 tables
  - Single statements coded against each table
- **A new application can have a similar design**
  - Developers want ultimate flexibility
  - Object oriented design
- **Here we have a quantity of SQL Issue!**
  - 100's of Millions per Day!
  - The cost adds up



In a legacy to DB2 migration with no application impact, non-intelligent black-box I/O layers present the greatest challenge to performance. This is because these I/O layers are typically written to read each DB2 table separately. This leads to a large number of SQL statements issued, and high CPU costs.

## Blind Development

---

- **SQL Coded for Any DBMS on Any Platform**
  - **Generic SQL**
  - **Procedural Stored Procedures**
    - **Do Not Translate Well to DB2 for z/OS**
      - **V9 Helps!!! Thank You!**
- **Great Flexibility**
- **Great Leverage**
- **Does Not Take Advantage of DBMS That SQL is Directed Towards**



Blind development is similar to generic development, but takes things a step further in that the database interface layer is written in such a way that it is transportable across many diverse DBMS software. This pushes more data intensive logic away from the database, and into the application layer. Great for leveraging platforms and software, and for being flexible. The price paid for this type of implementation is performance.

## SMART Transaction Processing

---

- Most of our experience
- Need to accommodate legacy and *new* applications!
- Generic SQL used UNLESS
  - Performance is required
    - Complex SQL utilized
    - SQL joins
  - Minimization of change is required
- When to take the generic approach versus SMART?
  - SMART (SQL Made to Accurately Represent Transactions)
  - Start with generic and wait until it 'Hits a Wall'
    - Really bad idea
    - Can destroy a project
  - Analyze up front
    - Data validation
    - Proof of concept



Generic block box implementation is easy to do, and doesn't require a significant amount of up front analysis. However, customers are expecting the same level of performance for an application that essentially behaves the same as it always has. A generic implementation generally cannot achieve this level of performance.

If performance is essential, then an advanced analysis phase is required to build a smarter interface. This SMART interface, built upon the up front analysis and testing, will use generic processing when performance is not impacted, but specific SQL and application coding for performance when critical.

## Data Validation

---

- **Build a “Validation” Database**
  - **Quick and dirty conversion from the Legacy Data Store**
    - **All data groups become tables**
    - **All character data columns**
- **Run data analysis queries**
  - **Run extensive queries to analyze all the data**
  - **Report on invalid data**
- **Validation database can be used for a proof of concept**
  - **Sample application built**
    - **Simple COBOL programs – little or no business rules**
  - **Test queries created**
- **Validation query reports become an “Input” to the target database**
  - **Physical design**
  - **Validation database queries**
  - **Proof of Concept report**
  - **Legacy application interface requirements**
  - **Future processing requirements**
  - **Enterprise data models**



Many times the legacy data is in a proprietary compressed format, it was difficult to evaluate the data. A sample of each can be converted into a simple DB2 database that matched the legacy data format. All data is stored as character display, and an extensive series of queries are executed to analyze the condition of the data. This information helps reveal undocumented codes, invalid data, and the frequency of relationships in the data.

Database queries that produce significant quantities of information about data quality, relationships, and format provide as a direct input to physical design.

## Proof of Concept (POC)

- **A Proof of Concept is critical when you want to balance**
  - Performance for legacy access
  - Performance for future Access
  - High availability
  - Compatibility with legacy processes
- **You need to build a sample application**
  - Simple test programs
    - COBOL
    - REXX
  - Simple query simulations
    - Query Scripts
    - Generated Data
    - REXX
- **Concept is to simulate current and future access needs**
  - Address the problems before development
- **POCs can take months to years!**
  - Start ASAP!



This data validation phase is then followed by a POC to identify and simulate application functionality. This includes process simulations using both SQL scripts and COBOL programs. Various tests are conducted to measure performance, application compatibility, and data integrity. Although much of this code is thrown away after the POC is completed, the information gathered was used to set physical database design and application API program logic.

Measurements taken from SQL scripts and POC program execution are used to obtain initial CPU capacity estimates and application elapsed time estimates. All these measurements are presented to management so they are well aware of the performance impact of moving to DB2 and can plan accordingly and set realistic expectations.

A list of issues is developed from the POC as input to the physical database model and application API development.

## Fat Versus Skinny Cursors

- **Recent Test – Sequential Processing**
  - **Multiple cursors in a program**
  - **Large batch process**
  - **Fat Cursors**
    - **Each cursor has a range predicate**
    - **Supports batch scans**
    - **Can mean double the cursors**
      - **One each for keyed access (Online)**
      - **One each for batch**
  - **Skinny Cursors**
    - **Supports keyed access only**
    - **Can support online or batch**
- **Performance**
  - **6 Cursor test**
  - **Skinny was 3X more expensive than Fat**
  - **Management has a choice**



For performance reasons we may elect to use fat cursors for sequential table access. Fat cursors will be opened at the beginning of the sequential read process for an entire set of data, read continuously during the process, and then close at the end. A generic implementation would only drive the sequential process off of one cursor, and then reuse the generic keyed queries to all needed related tables.

Advantages to fat cursors:

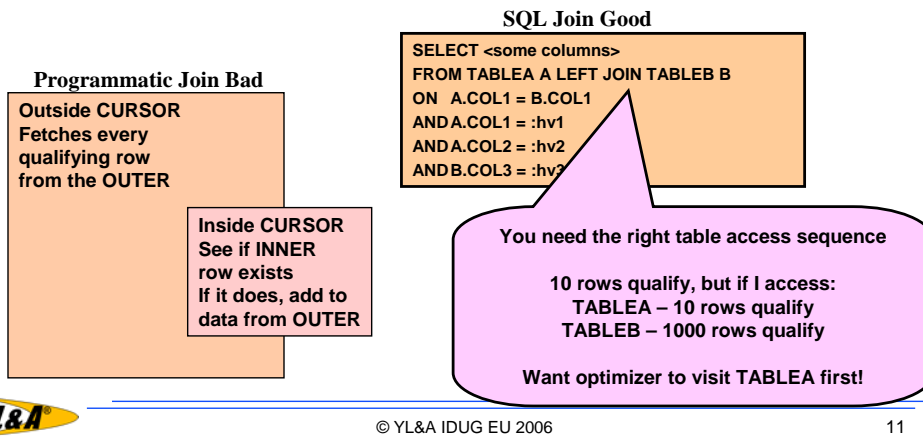
Better performance. Only one open and close per statement, and DB2 sequential prefetch as a predetermined access method. Using fat cursors will significantly reduce the number of SQL statements issued, and that will result in less CPU consumed.

Advantages to skinny cursors:

Less program code to maintain. We can use the same SQL statements for direct readers as well as sequential readers. Given the variations of SQL needed for logical recovery, joins versus single table access, and fat versus skinny cursors there could be a lot of SQL statements and more program code to maintain so skinny cursors would reduce the number of statements in half.

## Program Join Versus SQL Join

- **Generic Code – Programmatic Join**
  - Less coding, more flexible
  - 30% more expensive for two table join (in one benchmark)
    - As long as your SQL join gets the correct access path
      - Programmatic join forces the table join sequence
      - Wrong table join sequence can be more expensive than the programmatic join



In a legacy to DB2 migration with no application impact, non-intelligent black-box I/O layers present the greatest challenge to performance. This is because these I/O layers are typically written to read each DB2 table separately. This leads to a large number of SQL statements issued, and high CPU costs.

The major entities can be accessed individually, or by joins. If we use joins we will be issuing fewer SQL statements. However, the program logic will have to contain control break logic to parse through the result. Also, for joins there would be columns returned repetitively. Careful testing will demonstrate the difference in performance, and this document will be updated with that information.

## Getting Just the Columns You Need

```
SELECT *  
FROM DEPARTMENT
```



```
SELECT DEPTNO, DEPTNAME  
FROM DEPARTMENT
```

Select only the columns needed.  
Each column costs CPU and  
may negate index-only access.

```
SELECT DEPTNO, DEPTNAME  
FROM DEPARTMENT  
WHERE DEPTNO = 'D01'
```



```
SELECT DEPTNAME  
FROM DEPARTMENT  
WHERE DEPTNO = 'D01'
```

Don't reselect a column  
already in the WHERE clause  
as an equals predicate.

Comparison of 10 vs 40 columns selected:  
50% more CPU/Elapsed to select 40 columns



In generic development single queries return all columns for the table accessed, but what does the application actually use?

By determining which transactions need which data, customer SQL statements can be written for the most frequently executing queries to get only the data needed. For high volume systems the savings can be dramatic.

## Optional Data - Adding Indicator Columns

- Indicator columns are used for relational existence without having to look in additional tables
- May require a synchronization tool/utility to run checks
  - If maintained in triggers, probably not necessary
- Use the indicators to reduce SQL and I/O → only get the data needed
  - Unless the data exist and must be returned majority of the time, then do the outer joins
- The existence of optional data may be enough to satisfy the query
- Could write simple query to return data and indicators to screen
  - If the optional data is needed, use a second query

Acct_No	Name	OptAddr	OptBill	Optxx
1234	Smith	Y	Y	N
1235	Jones	N	N	N
1236	Ray	N	Y	N
...	...			...

Opt Columns are for physical existence checking only and do not need to be logically modeled



In many implementations a parent table has many child tables with optional data. The generic interface processing will read each table individually to gather all of the data together. This results in access to each optional table, even if it contains no data. Random index probes add a lot of getpages and I/O, and performance suffers.

In many cases the developers decide to denormalize to reduce the quantity of tables accessed. This may reduce I/O and elapsed time (must be careful of number of columns processed), but at what price?

When we come across situations like this in which denormalization seems to be the course of action, we use indicator columns rather than pure denormalization. Each indicator column determines the presence of data in a child table. A “milder” form of denormalizing.

## Optional Data – Use of Indicator Columns

```

SELECT columns
FROM ACCOUNT A LEFT JOIN BILLING O1
ON A.ACCT_NO = O1.ACCT_NO
AND A.OPT_BILL = 'Y'
LEFT JOIN ADDRESS O2
ON A.ACCT_NO = O2.ACCT_NO
AND A.OPT_ADDR = 'Y'
.....
WHERE A.ACCT_NO = 1
    
```

DB2 will only join to optional tables when ON clause is true. Avoids unnecessary joins when optional data does not exist.

DB2 will evaluate the during join predicate before going to the inner table.

Keep in mind the during join predicate does not filter. Still need an after join predicate.

Acct_N	Name	Opt_Addr	Opt_Bill	Opt_xxx
o				
1234	Smith	Y	Y	N
1235	Jones	N	N	N
1236	Brown	N	Y	N
1237	George	Y	N	Y
1238	Black	Y	Y	N
1239	Ray	Y	N	Y
...	...			...

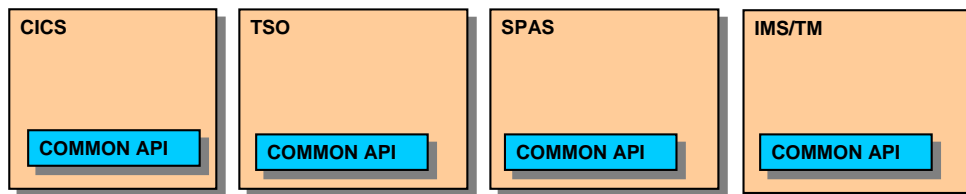


DB2 “during join” processing can be used to avoid access to the child tables in a single SQL statement. Here the indicator column is testing as part of the join condition. DB2 will only join when the join condition is true. In this way access to the child tables only happens when there is data present. This eliminates the index probes, and can dramatically improve response times.

Of course, there is additional responsibility of keeping the indicator columns in sync, which is true of any denormalization.

## Generic Attach

- Legacy Migrations are Huge
- Common Modules Becoming More Common
  - Reusable API's
  - Legacy and New Clients Need Access
- Today We Need to Maintain Multiple Copies of the Load Modules
  - RRSAF, CICS, TSO, CAF, SP, with/without Connection Established
  - DSNELI, DSNCLI, DSNALI, DSNRLI,.....ugh!
- Generic can be Used In Multiple Operational Environments
  - Use DSNHLI Generic Attach (Don't Specify an Environment at Precompile)
  - Rename Each Attach Module in Its Own Load Library Concatenated Before SDSNLOAD
    - Need Dynamic CALL Option for Program Compile



YL&A

© YL&A IDUG EU 2006

15

To use DB2 programs have to be processed by the DB2 pre-compiler. The pre-compiler translates DB2 SQL calls into calls to particular subprograms. The subprogram, otherwise known as the attach module) being called is controlled by a pre-compiler option ATTACH. IBM provides different attach modules for each execution environment:

DSNELI for TSO

DSNALI for Call Attach

DSNCLI for CICS

DSNRLI for RRS Attach

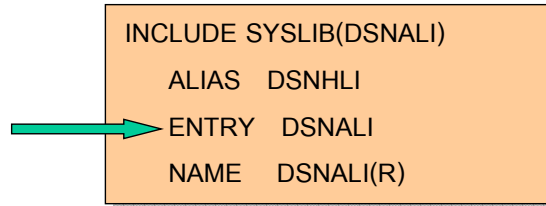
For example, if ATTACH(TSO) is specified as a pre-compiler option then all SQL calls will be translated by the pre-compiler into subprogram calls to program DSNELI. When the programmer link edits the program, the DSNELI module is included in the load module, and all calls to it are static.

When no ATTACH pre-compiler option is specified the pre-compiler will place calls to a generic module named DSNHLI into the program. By default, the DSNHLI module is a replica of the DSNELI TSO attach module. However, all of the attach modules have a DSNHLI entry point, so all of the generic calls will work for any of the attach modules statically linked into the target load module.

## Generic Attach

- **Documented in the IBM Stored Procedure Redbook**
  - Documentation in the application programming guide is outdated
    - Don't use it
  - Documentation in the redbook is mostly accurate
    - Missing entry point in attach module link-edit (binder) examples

```
INCLUDE SYSLIB(DSNALI)
      ALIAS DSNHLI
      ENTRY DSNALI
      NAME DSNALI(R)
```



- **We also coded for connecting**
  - Caller can, but not necessarily be already connected
  - Our module attempts a call attach
    - Call will fail if already connected with appropriate reason code
    - Our module handles the connection failure and continues



It is possible to not include the appropriate attach module at link edit time, and instead call it dynamically. This technique is useful when a particular program that accesses DB2 tables needs to be called from within several execution environments. The concept is that if the appropriate attach module is available within the load library accessed within each execution environment, then the generic call to DSNHLI will pick it up. The trick is to name each specific attach module DSNHLI in a load library specific to each environment. This generic attach technique is described in the IBM redbook SG24-7083-00; DB2 for z/OS Stored Procedures: Through the Call and Beyond, and basically requires the following steps:

1. Do not specify the ATTACH option for the DB2 pre-compiler
2. Allow for dynamic calls from the user program. This can be done by including the "PROCESS DYNAM" keywords at the beginning of a COBOL program, or via specifying the DYNAM compiler option.
3. Do not include a DB2 attach module during link edit.
4. Link edit the attach module for the appropriate execution environment into its own special load library, and rename it DSNHLI.
5. Concatenate the special load library ahead of the SDSNLOAD library at execution time.

If these things are done properly and specifically to the various execution environments then the generic attach calls will execute the appropriate environment attach module.

This is a benefit in that only one user program load module and library is then required for multiple execution environments.

## Five Legacy Migrations Stories

---

- **DB2 and Java Complete Rewrite**
- **DB2 Application with “Hooks” To Legacy**
- **DB2 Application I/O Layer Access From Legacy Applications**
- **DB2 CICS Migrating to WebSphere**
- **Other DBMS Migrations to DB2**

## DB2 and Java Complete Rewrite – Legacy Conversion

---

- Data was stored in a proprietary highly compressed format
- Complex business rules were used to translate the data as stored in the files, and into a human readable display format
  - Custom conversion modules written in both BAL (Basic Assembler) and COBOL programming languages to interface with the legacy API and create DB2 'load ready' datasets
  - Needed to maintain flexibility of the design, and adaptability of the conversion programs
  - Custom hand written gave us in-house expertise in quickly changing the conversion programs based upon refinements of business rules and changes to database tables
- Custom coded programs allow for a fast conversion of the data
- A 400GB source database can be converted into a 5TB DB2 database in less than one day
- Making changes to tables, changing the conversion process to match, and rerunning conversion during development
  - Allowed further refinements to be made to the conversion process enabling a faster conversion time
- Database design evolved over time
  - Even though the conversion programs were written early in the process
- Database changes did impact the conversion programs
  - By having real data, practicing the conversion, and have full volume copies of the database for performance testing was extremely valuable



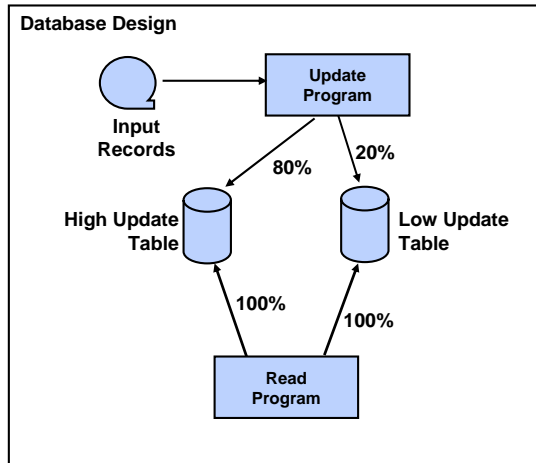
## DB2 and Java Complete Rewrite – Very Large Database

- Legacy record format was broken into 20 DB2 (3<sup>rd</sup> normal form) tables
- Challenge #1 - very large tables, and continuing operations
  - UNION in VIEW SQL was utilized for several of the very large tables
  - Avoided NPSI's and REORGs due to the requirements for high availability
    - Full availability of tables was required
      - With NPSI's, availability strategy needed
    - Had to code availability features into the applications
- Challenge #2 - number of tables dealing with data changes
  - Legacy application used a record replacement technology using match/merge logic
    - Replacing all of a 5TB DB2 database every night is not possible
    - Spent two years in preparing for the new database and change
    - The data was analyzed, and the process of updating data was analyzed
    - Built several proof of concept (POC) databases, and COBOL programs to simulate how the application would work in various situations
- Analysis of data and testing of options
  - Provided a design to minimize change in the database
  - Ultimately saved more than 60% of operating expense for an extremely large system
    - Two examples to follow
      - Splitting Tables based on data updated
      - Using CRCs to minimize change



## Finding New Designs to Meet Performance Objectives

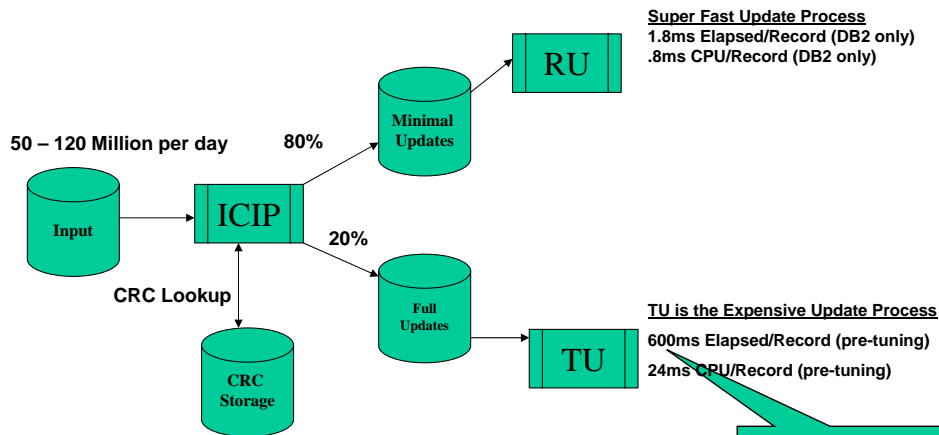
- 90% of Activity is Updates
  - 80% of updates just to a few columns
- 10% of Activity is Reads
- Splitting
  - High Update Table
  - Low Update Table
- Clustering
  - High Update Table
    - Update Sequence
  - Low Update Table
    - Read Sequence



- **Result**
  - 80% of updates are really fast

## Know Data and Process to Find Performance Solutions

### The Savings are Obvious!



SMART processing  
has reduced this by  
50%

**Proof of concept, proper planning, and allowing the time for upfront analysis is critical**

## DB2 and Java Complete Rewrite - Application

- Legacy application was written in BAL and COBOL
  - New application was written in Java
- Java gave leverage in server platforms and operating systems
  - Java was portable across z/OS, zLinux, Windows, and AIX
    - With little or no coding changes
  - Write/unit test programs on PC's
    - Before porting to mainframe for integration testing
  - Research was done to get Java to interact properly with the DB2 database
    - Use of different DB2 data types
      - CHAR vs. VARCHAR
- SMALLINT vs. CHAR(1) code fields
- NULL vs. NOT NULL WITH DEFAULT
  - These all work differently for Java than for other languages
- Cost of Java
  - Java is twice the cost of COBOL in terms of CPU on the mainframe
  - zAAP engines helps to mitigate some of this cost
    - Portability of Java gives us the opportunity to offload some or all of the cycles to cheaper platforms

	CHAR with RTRIM	VARCHAR w/o RTRIM
Avg CPU	15.75 min	16.29 min
Avg DB2 CPU	7.27 min	8.12 min
Avg Elapsed	22.29 min	23.01 min



## DB2 and Java Complete Rewrite - Connectivity

---

- One issue with the portability of the Java programs
  - Dealing with how DB2 behaved differently
    - When using remote connections versus local connections
- Moving the application to another server
  - Now you are remotely connecting to DB2, even if it is on zLinux
  - Have to consider network bandwidth and availability, DRDA settings, and DDF settings, such as the CMTSTAT installation parameter and its affect on memory utilization and accounting records
- Other issues included the loss of such efficiencies as sequential detection and index lookaside
  - Memory for these is destroyed with no RELEASE(DEALLOCATE)
- Be ready to make sacrifices, and measure them, when you move the application such that it connects remotely
  - However, maybe you can get some buy back with zIIPs

**The bottom line...**  
**If you design the application**  
**with respect to the database (and platform)**  
**you are going to get the best performance.**

## Case #2 - DB2 with Application “Hooks” to Legacy

---

- Four very large BDAM systems
- Records were stored in a proprietary highly compressed format
- Based upon the size and complexity of each source system, and subsequent DB2 database complexities, two conversion strategies were created
  1. Simply utilize the efficient legacy API to extract data into a display format, and then convert that data in a single pass into a set of DB2 load files
  2. Employ a hand coded set of metafiles as input to a REXX and COBOL custom process that generated conversion programs
    - The REXX code generation process utilized artifacts created during the modeling process as input, as well as the DB2 catalog, and legacy copy books
      - Allowed the generation process to adapt to any changes.
      - Handled most of the simple conversion business rules.
      - For more complex rules, custom COBOL libraries were created, and source code was merged with the generated code.
      - Enabled flexibility of the database design during development, and a reusable process for the most complicated files.
- Having the development of the conversion in house enabled the programmers to have intimate knowledge of the conversion programs
  - Allowed database to be highly adaptable.
- Once the database tables reached a point of stability the conversion generation process was abandoned
  - Generated programs then maintained by hand



## DB2 with Application “Hooks” to Legacy - Database

- Databases contained highly sensitive/mission critical data for the enterprise
- Started the process with a 2 year proof of concept.
  - POC was performed in two phases
  - Legacy data was in proprietary compressed format and difficult to evaluate
  - A sample of each file was converted into a very simple DB2 database that matched the format of the legacy data.
  - All data was stored as character display, and then an extensive series of queries were executed to analyze the condition of the data.
  - Information helped to identify undocumented codes, invalid data, and determine the frequency of relationships in the data.
- Data evaluation was followed by a POC to identify and simulate application
  - Process simulations using both SQL scripts and COBOL programs
  - Various tests were conducted, and measurements for performance, application compatibility, and data integrity were taken.
    - Much of this code written was thrown away after the POC was completed
      - Information gathered was used for both physical database design and application API program logic
      - Measurements taken from SQL scripts and POC program execution
        - To get initial CPU capacity estimates, as well as application elapsed time estimates
      - All of these measurements were to set expectations about the performance impact of moving to DB2



## DB2 with Application “Hooks” to Legacy - Application

- Not allowed to change any application programs
- A data access API already existed, and the API was to be rewritten
  - This impacted DB2 physical database design decisions
  - Although it was desirable to create a normalized database design for the future, we had to blend that design with the fact that we had to accommodate a “read everything” legacy approach
  - Use of modest denormalization, indicator columns, and taking the time to study the legacy API, allowed for us to meet the future design direction, and also meet performance objectives
    - One of the key things to note here is that we carefully studied the behavior of the legacy API
      - Were able to minimize DB2 access in some situations, and dramatically improve performance (4X performance improvement).
- During migrations in which legacy applications will not be changing it is sometimes beneficial to come to an understanding that the migration will take place in phases
  - The first phase will do its best to accommodate the past, and prepare for the future
  - A later phase may involve another conversion to a more normalized design when the legacy access is abandoned and we need to position for the future
- Planning for this and expecting initial “less than desirable performance” during the initial phase will set reasonable expectations!

### Case #3 - DB2 Application I/O Layer Access from Legacy

---

- Conversion from convert to DB2 from VSAM without any changes to the application
  - Do not want to upset the user community
  - However, you have to be willing to take a performance hit
    - Or design for it
- There are advantages to going to DB2
  - Higher level of availability
  - Enhanced backup and recovery
  - Options for built-in data integrity
  - Flexibility of SQL
- With these advantages of course comes a longer code path, and reduced performance.
  - If the move is from a highly denormalized record-based system to a normalized DB2 database, then there will be an I/O impact as well
    - Spend the time to do a significant amount of data analysis, such as in case #2, then you will have plenty of information in hand to make database design and application API decisions

### Case #3 - DB2 Application I/O Layer Access from Legacy

- New database design based upon a 3rd normal form was developed based upon the new model
  - Required sophisticated logic to translate from the legacy record format to the DB2 database
  - Impacted both the conversion process, and the API
  - The decision was made to use an edit transform and load (ETL) tool to create the conversion programs.
    - ETL tools can be extremely useful for moving transaction data to warehouses and such, they are not always the best choice for a one time conversion process
      - The complex conversion rules, and B-level conversion code generated by the tool resulted in an extremely poor performing conversion process.
      - In addition, this complexity made the conversion code so confusing that any database changes were discouraged, so workarounds were created
- Moving to a normalized database with no detailed analysis of the application layer lead to a very inefficient API
- The “read everything” philosophy to a fully normalized database was very cumbersome
- The decisions made at that point were to denormalize the database
- This impacted the conversion, future applications, and the wisdom of the migration in the first place
  - The path of case #2 is far better even though it may take longer



## Case #4 - DB2 CICS Migrating to WebSphere

- Convert a CICS based legacy application on an older Amdahl machine that was using DB2 to a new Websphere based application running on an IBM z800 machine

**Goal:**

To have the conversion be as easy and non-disruptive as possible

**Challenge:**

How to convert a production legacy application from CICS to Websphere was the biggest challenge without a “big switch” change over

**Solution:**

DB2 z/OS data sharing in the parallel sysplex environment

- The parallel sysplex can run DB2 on different hardware platforms
  - While still being able to share data among the DB2 subsystems
  - By using DB2 data sharing and linking the machines/DB2s together via the coupling facility technology
  - This way the same database can be shared by applications running on two different machines
- Can convert and migrate transactions in a controlled manner

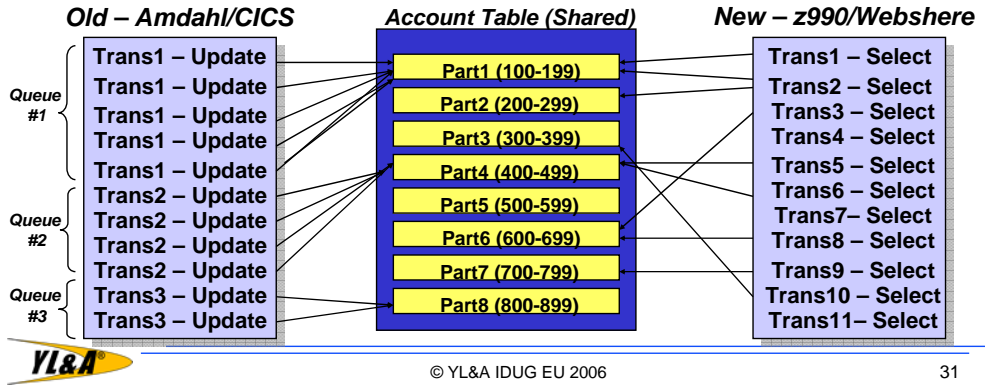


## Case #4 - DB2 CICS Migrating to WebSphere (cont..)

- Database was designed to receive orders from queues
  - Orders corresponded to certain partitions that were key limited the same as the queues
  - Queries, inserts and updates would all be involved
- Convert a set of related transactions together from Cobol/CICS on the original subsystem on the older Amdahl to Java/WebSphere on the new IBM z/800 machine.
  - By using DB2 data sharing only one copy of the database was needed
    - Both machines could be in the same data sharing group and allowed to share the same data via the coupling facility
  - Transactions could then be moved from the legacy interface to the new interface in a controlled fashion
    - While the performance is being monitored
- Allowed for a smooth and controlled transition from old to new without an outage
  - Transactions were studied carefully first
  - A plan was developed as to which type of transactions would be moved/converted first
    - A plan to test performance was also employed
- Very creative solution to bring this application forward to use new technology
  - By leveraging the flexibility of data sharing

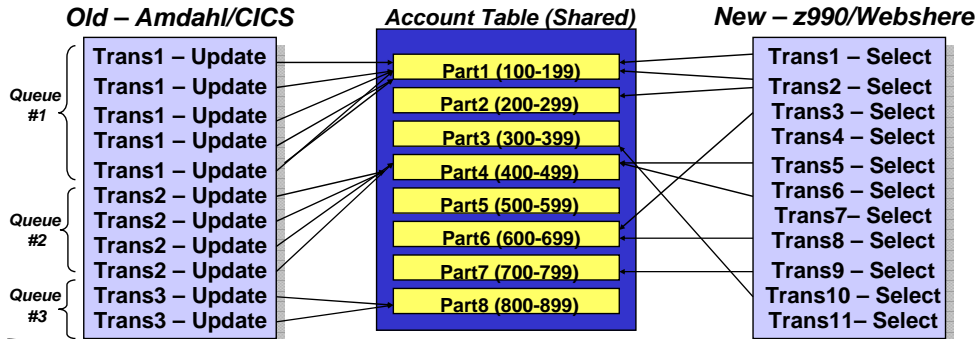
## Case #4 – Interesting Challenge

- Customer migration plan:
  - Existing Amdahl with CICS transactions/applications
  - New z990 with Websphere
  - Want to have a ‘seamless’ migration from the old system to the new
    - Data Sharing: Two members (old/new)
    - Movement: Transaction at a time(grouped by function)
- Problem:
  - Transaction response slow at certain times (explanation on next slide...)



## Case # 4 - Solution

- Read/write interest was not maintained
  - Pseudo-close occurred
    - Data set reopened – page registration!
      - All pages being read by ‘New’ have to be registered in the GBP
- Resolution
  - For now....have a dummy program to keep interest open from ‘New’



## Case #4 Other Database Servers Migration to DB2

What may work well on other platforms may not work well in z/OS environment!!!

- Often 1:1 conversions experience performance expectations not just lower than expected, but work was simply not able to work
- When recentralizing data onto the z/OS platform or simply migrating an application to the z/OS platform
  - Need consideration for the uniqueness of the DB2 z/OS environment
  - Take advantage of it and be aware of where differences are
- What if the DBMS being migrated from offers functions that don't exist on DB2 for z/OS?
  - In these particular situations all is not lost
  - DB2 for z/OS is extremely flexible when it comes to designing your own user-defined functions (UDF's)
    - Have been able to solve all of these incompatibilities with UDF's by coding our own SQL and external UDF's
      - It takes a little more programming effort, but easier than a redesign of the application when that is even possible.
- SQL, data types (e.g. VARCHARs), row level locking and other items also work very differently on DB2 z/OS
  - Don't assume everything works the same!
  - Best performance will be achieved by converting/migrating with an understanding of the uniqueness and capabilities of the z/OS platform.



## Case #4 Other Database Servers – Stored Procedures

- **Stored procedures work differently on other platforms**
  - On database servers such as Sybase, Microsoft SQL Server and Oracle a common application design is a stored procedure for every SQL statement
  - Example: 200 tables and 4 stored procedures for each table
    - SELECT, INSERT, DELETE and UPDATE = a total of 800 stored procedures
    - Black box I/O module design that performs horribly on the DB2 z/OS platform
- **Stored procedure execution on z/OS**
  - DB2 has to go cross-memory to call the stored procedure in another address space
    - Subsequently the stored procedure goes cross-memory to execute SQL statements from another allied address space
    - Operating system has to manage the stored procedure address spaces
      - Several z/OS system settings to consider - (WLM policies, number of TCBs, etc.).
    - With Java there are issues with starting virtual machines within the address spaces and Java programs consume 2x the CPU as other sp languages
- **I/O module designs need to be evaluated carefully**
  - Place the SQL into the application to make use of the SQL language
- **Programmers who are used to coding in Sybase or Oracle also tend to write highly inefficient DB2 z/OS stored procedures**
  - Sybase and Oracle procedures run within the database server
  - Some tend to use the procedures extensively as simple extensions to their applications
  - Using only single SQL statements in a DB2 stored procedures can be a significant performance detriment
    - If you are calling stored procedures from other stored procedures then your performance detriment is amplified



## Conclusion

- Any type conversion/migration requires extensive testing on various designs
  - Needs to be performed to ensure
    - That the data structures are as optimal as possible for the application
    - And integrity and future use of the database is not sacrificed
- The methodologies used for conversion should do not box you in and make future database changes for performance impossible
- Expectations need to be set that there will be some performance degradation during most legacy conversions
  - Patching the problems without thought as to how the data is going to be used in future can make the migration a waste of time and resources
  - There are clever solutions to many problems, and you need to reserve the time to explore them

Set  
Expectations!

Develop  
POC!

Know Your  
Data!

Test!

Test Again!

---

**B04**

***DB2 Legacy Migrations: Experiences of  
The Industry***

**Susan Lawson  
and Dan Luksetich**

**YLA**

**Susan\_lawson@ylassoc.com**

**Dan Luksetich@ylassoc.com**

