DB2 10: For Developers Only
for z/OS

Sponsored by:

http://www.compuware.com

Craig S. Mullins
Mullins Consulting, Inc.
http://www.craigsmullins.com
http://www.mullinsconsultinginc.com

© 2013 Mullins Consulting, Inc.
Author

This presentation was prepared by:

Craig S. Mullins
President & Principal Consultant

Mullins Consulting, Inc.
15 Coventry Ct
Sugar Land, TX 77479
Tel: 281-494-6153
Fax: 281.491.0637
E-mail: craig@craigmullins.com

This document is protected under the copyright laws of the United States and other countries as an unpublished work. Any use or disclosure in whole or in part of this information without the express written permission of Mullins Consulting, Inc. is prohibited.

© 2013 Craig S. Mullins, Mullins Consulting, Inc. All rights reserved.
Abstract

This presentation highlights the DB2 10 for z/OS enhancements that directly impact DB2 application developers. Every release of DB2 is chock full of new features and functionality and that can make it hard to focus on those things that are most helpful for programmers.

So instead of scanning volumes of manuals, you can utilize this presentation, which distills the DB2 10 information down to cover what should be most important to programmer/analysts.

If you are a programmer wanting to learn more about DB2 10, or a DBA looking for the programmer’s perspective on DB2 10, this presentation will have something interesting and informative to offer you.
Agenda

- Overview, Migration Issues and Concerns
  - Deprecated Features
- BIND, REBIND and Access Paths
  - Safe Query Optimization
  - Improved access path hints
  - Access to Currently Committed Data
  - Packages Only / No DBRM$s in plans
- Temporal Data Support
  - Table and SQL changes
  - Walk through examples
- Additional SQL Changes
  - Indicator Variables
  - TIMESTAMP precision and time zones
  - Moving sums and averages
  - Inline and Non-inline SQL scalar functions
  - SQL table functions
  - Extended implicit casting
- Other “Stuff”
• Some sites are still running DB2 Version 8
  – No longer supported by IBM

• IBM DB2 10 provides skip level migration
  – DB2 V8 → DB2 9 → DB2 10
  – Migrating directly to DB2 10 from V8 can be a smart move.

• Of course, you can also migrate to 10 from 9
  – DB2 9 → DB2 10

• DB2 11, recently announced, will not offer skip-level migration

  …and if you are still on V7 or earlier, you are WAY overdue to update!
DB2 10 for z/OS

• *Overall* highlights of this release include:
  – Significant out-of-the-box reduction in CPU utilization
  – From 5 to 10 times more concurrent users on a single DB2 subsystem
  – Greater concurrency for data management, data definition, and data access, including additional online changes for data definitions, utilities, and subsystems
  – Improved security with better granularity for administrative privileges, data masking, and audit capabilities
  – Temporal (or versioned) data to understand system and business times at the database level
  – Hashed indexes
  – pureXML™ and SQL enhancements to simplify portability from other database solutions
  – Productivity improved for database administrators, application programmers, and systems administrators
  – Additional performance and availability improvements

• ...and there are a lot of optimizer enhancements and performance improvements that you won’t get without a REBIND.
Deprecated Features (9 - 10)

• Private Protocol Distribution
  – No surprise since IBM has been indicating this was on its way out for a number of releases now. Not too difficult to convert to DRDA (some help w/DSNTP2DP).

• ACQUIRE(ALLOCATE) is no longer supported for BIND | REBIND PLAN
  – Rebind with ACQUIRE(USE)

• REBIND plans & packages that haven’t been rebound since V5 or before.

• EXPLAIN tables prior to V8 format are no longer supported (→UNICODE)
  – Information APAR PK85068.

• DB2 MQ XML functions and the DB2 XML Extender no longer supported
  – Use pureXML.

• DB2 Management Clients no longer supported
  (DB2 Administration Server, Control Center, and Development Center)
  – Data Studio is IBM’s new management client

• BookManager no longer supported
  – Instead use the PDF versions and/or the Info Center online.
The DB2 precompiler produces a DBRM and a modified source program with commented SQL statements.

The DB2 BIND command turns a DBRM into an executable format.

The DB2 Catalog stores information about the plan and package.

DB2 Directory stores the actual plan and package.

Plan

Bind Plan

DBRM

Bind Package

Modified Source

Compiled Source

Linkage Editor

Load Module

Compiler

DB2 Catalog

DB2 Directory

Source Program

DCLGEN

Precompiler

Module

Program Preparation - V10
No Longer Supported: Plans w/ DBRMss

• As of DB2 10, plans containing DBRMs are no longer supported
  – MEMBER parameter is no longer supported for BIND PLAN
  – You will have to convert the DBRMs to packages.

• REBIND PLAN(X) COLLID(*)
  – "*" is the default value
    • It creates DSN_DEFAULT_COLLID_X (where X is the plan name)
  – This enables customer to convert plans with DBRMs bound directly into them into packages under a collection ID (DSN_DEFAULT_COLLID_X) automatically.
Don’t forget that if you are skipping from V8 to 10 then you need to consider the features deprecated for DB2 9, too!

- AIV Extender, Text Extender, and Net Search Extender are removed.
- Net.Data, stabilized at V7, is removed.
  - WebSphere is IBM’s strategic solution for getting DB2 data to Web applications.
- DB2 Estimator is not provided for DB2 9.
- DB2-managed stored procedure support is removed.
- Simple table space creation support is removed.
- JDBC/SQLJ Driver for OS/390 and z/OS support is removed.
DB2 10: EXPLAIN & Access Path Stuff

• Ability to EXPLAIN existing packages
  – Command: BIND PACKAGE...EXPLAIN (ONLY)
    • Works for all packages.
  – SQL: EXPLAIN PACKAGE
    • The package being explained must have been created in DB2 9 or above.
      – Older packages cannot be explained like this!

• CURRENT EXPLAIN MODE special register
  – Controls how EXPLAIN behaves for eligible dynamic SQL (SELECT, INSERT, and the searched form of UPDATE and DELETE).
    • **NO** - turns off capture of EXPLAIN information. (default)
    • **YES** - turns on capture of EXPLAIN information; plan tables will be updated.
    • **EXPLAIN** - this option is the same of YES. However, the dynamic SQL statements are not executed.

• New authority that can be granted: EXPLAIN
New DB2 10 feature: Safe Query Optimization

The DB2 Optimizer now considers the uncertainty of predicate filtering when selecting an index.

- Previously, overestimating the filtering of a predicate was possible when the value of a host variable was not known (i.e. no literal).
- As of DB2 10, when indexes have a close cost estimate, the DB2 Optimizer factors in the uncertainty of matching and screening predicates.
DB2 10: New Hint Mechanism

- Prior to DB2 10 QUERYNO is required for matching a hint to SQL statement
- New DB2 10 mechanism allows you to associate the text of a query with a hint... much easier
  - Populate DSN_USERGQUERY_TABLE (new) with the query text and QUERYNO
  - Populate the PLAN_TABLE with the hints
  - Run the new DSN command BIND QUERY to integrate the hint into the repository
    - FREE QUERY removes hints from repository
As we all should know already, in order to take advantage of the DSC, the dynamic SQL statement must be exactly the same.

If a literal changes, it is not the same. For example:

**NOT THE SAME**

```
SELECT NAME, ADDRESS
FROM CUST
WHERE CUSTNO = 1234;
```

```
SELECT NAME, ADDRESS
FROM CUST
WHERE CUSTNO = 5678
```

As of DB2 V10, dynamic SQL with literals can be reused in the DSC.

- It is still generally better to use parameter markers for dynamic SQL than to use literals and rely on this update though.
Access Path Changes

• Access Paths
  – This is perhaps the most time-consuming and worrisome aspects of DB2 version migration.
  – A new version typically means Optimizer enhancements.
  – And that means changes to access paths!
  – Changes that occur are not always predictable
    • There may be trends that you can anticipate, though
  – Other migration issues can cause access path changes, too
    • Statistics changes
    • Environment changes
Version Migration Concerns

Typically, it is not a hard & fast requirement to REBIND all of your packages and plans when you move to a new version of DB2.

› However, it is a really good idea to REBIND the most critical (if not all of them).

There are a lot of optimizer enhancements and performance improvements that you won’t get without a REBIND.

And there are some REBINDs you cannot avoid.

› For DB2 9, plans and packages from DB2 V4 or earlier will be automatically rebound when accessed by DB2 9.

› For DB2 10:
  ▪ Plans and packages from DB2 V5 or earlier will be automatically rebound when accessed by DB2 10.
  ▪ Plans bound with DBRMs are automatically rebound to convert it to a PKLIST and the DBRMs are bound into packages.
DB2 10 - REBIND Issues

- Incremental Rebinds
  - DB2 incrementally rebinds static SQL statements that use parallelism after migration to Version 10.
    - Incremental rebinds can cause performance degradation, so you should manually rebind those statements.
  - A query in DSNTIJPM is provided that you can run to determine which statements can use parallelism.

- Thread Constraint Relief
  - DB2 10 improves the number of concurrent threads that can be running. To get this relief, though, you must REBIND.
    - After you REBIND, the work areas are moved above the bar.
DB2 Version 8 had many *typical* access path changes that could be anticipated based on trends

Most organization had the same type of changes, for example:

- Matching Index Scan → TS Scan (for small tables)
- Non-matching Index Scan → TS Scan (for large indexes)

So special rules could be applied based on the trends

But DB2 10 is like DB2 9; some access path changes but...

- Different types of changes in every installation
- A general set of special rules is not really applicable
Access Paths Rely on Statistics

• Always remember that access paths rely on statistics
• The better the statistics, the better the DB2 Optimizer can be at determining the appropriate access paths
• If you are going from V8 to 10: keep in mind
  – Between DB2 V8 and DB2 9
    • The algorithm used for RUNSTATS changed.
    • Different “opinion” as to what is clustered.
    • When will you update RUNSTATS?
• Always a good idea, though:
  – Save performance and access path information
  – To be able to compare before and after
RELEASE DEALLOCATE Improvements

- RELEASE DEALLOCATE requires more memory than RELEASE COMMIT
  - But memory improvements in DB2 V10 makes RELEASE DEALLOCATE more generally applicable
  - Consider using it for well-behaved applications, particularly for high volume transactions (e.g. CICS) with few short running SQL statements

- Do not consider if:
  - Long-running with no COMMITs
  - Using LOCK TABLE
High Performance DBATs

• High performance DBATs can be thought of as similar to CICS DB2 protected entry threads. Benefits include:
  – High-performance DBATs persist through multiple transaction executions
  – RELEASE(DEALLOCATE) can be chosen for distributed packages at BIND time.

• Pre-V10: Distributed packages always executed with RELEASE(COMMIT) behavior
  – DB2 V10 allows RELEASE(DEALLOCATE)
  – Pre-V10 it was possible to BIND a package for remote specifying RELEASE(DEALLOCATE), but DB2 would not honor that specification
Access to Currently Committed Data

• New BIND option to enable read transactions to access currently committed and consistent images of rows
  – Improves flexibility and increases performance for applications that require only available and committed data to be returned from DB2 tables.

• New BIND parameter: CONCURRENTACCESSRESOLUTION
  – There are two options for this parameter:
    – USECURRENTLYCOMMITTED - Specifies that the database manager can use the currently committed version of the data for applicable scans when data is in the process of being updated or deleted.
      • Rows that are in the process of being inserted can be skipped.
    – WAITFOROUTCOME - Specifies that applicable scans must wait for a commit or rollback operation to complete when data is in the process of being updated or deleted.
      • Rows that are in the process of being inserted are not skipped.
When was that Package Used Last?

• At times, you may want to identify programs that are no longer being run
• But it can be difficult to determine when the last time a package was actually used (pre-V10)
• New DB2 Catalog column: **LASTUSED**
  – **SYSPKAGE**
  – **SYSPLAN**
• DATE column that is set when DB2 loads the package header from the EDM
Time for a Change?

DB2 V10
Supports Temporal Data!
One of the biggest and most anticipated new features of DB2 V10 is temporal data support, but what is it?

- Many types of data change over time, and different users and applications have requirements to access the data at different points in time.
- Instead of creating separate history tables, using triggers, and/or implementing snapshot tables, DB2 10 features built-in temporal support.

There are two types of temporal data supported:

- Business Time
- System Time
Temporal Data: Business Time vs. System Time

- **Business Time** (aka application time or valid time)
  - Specifies when the facts stored in the database are true with respect to the real world.
  - These are the dates of interest to the business user interacting with the data.
  - Business time is useful for only certain types of data that change over time and the validity of the data is relevant to the application and users.

- **System Time** (aka transaction time)
  - Denotes the time when the fact became current in the database.
  - System time can be used to track the insertion and modification history of the data.
  - Unlike business time, transaction time may be associated with any database entity.
DB2 Supports Both Business Time and System Time

• Both are implemented via a time period specification
• **Business Time** is tracked in a single table.
  – Beginning and Ending time periods indicate which rows apply to which time period
• **System Time** is tracked using two tables.
  – One table contains the current data.
  – Another, history table, contains the non-current data.
  – Still requires Beginning and Ending times to indicate which rows apply to which time period
• A single “logical” DB2 table can be setup for both business and system time
  – Examples are coming up (in a few slides)
Implementing Business Time

- **Let’s set up a table to track business time.**
  - This requires allocating columns for the start and end of the business time period and identifying these columns to DB2.

```sql
CREATE TABLE COURSE
  (COURSENO INTEGER NOT NULL,
   TITLE VARCHAR(10) NOT NULL,
   CREDITS SMALLINT NOT NULL WITH DEFAULT 3,
   PRICE DECIMAL(7,2) NOT NULL,
   BUS_START DATE NOT NULL,
   BUS_END DATE NOT NULL,
   PERIOD BUSINESS_TIME(BUS_START, BUS_END),
   PRIMARY KEY(COURSENO, BUSINESS_TIME WITHOUT OVERLAPS)
);
```
An index is required to support the unique constraint on the BUSINESS_TIME WITH OVERLAPS clause.

- For example, the following index can be created to support the primary key definition of the COURSE table with BUSINESS_TIME:

```sql
CREATE UNIQUE INDEX XCOURSE
    ON COURSE
    (COURSENO, BUSINESS_TIME WITHOUT OVERLAPS)
    USING STOGROUP DSN8G1010
    PRIQTY 360 SECQTY 36
    ERASE NO
    COPY YES;
```
Let’s start with some sample COURSE data for the temporal table we just created:

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>BUS_START</th>
<th>BUS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>300.00</td>
<td>2011-10-01</td>
<td>2012-01-01</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>350.00</td>
<td>2012-01-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>150</td>
<td>FINANCE</td>
<td>3</td>
<td>300.00</td>
<td>2000-01-01</td>
<td>2005-01-01</td>
</tr>
<tr>
<td>150</td>
<td>FINANCE</td>
<td>3</td>
<td>250.00</td>
<td>2005-01-01</td>
<td>2005-12-31</td>
</tr>
<tr>
<td>200</td>
<td>INTRO TO DB2</td>
<td>1</td>
<td>150.00</td>
<td>2005-01-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>3</td>
<td>500.00</td>
<td>2009-01-01</td>
<td>2010-11-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>500.00</td>
<td>2010-11-01</td>
<td>2011-03-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>550.00</td>
<td>2011-03-01</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>
Query Business Time Data: New SQL Clauses

- To query temporal data as of a specific point-in-time:
  \[\text{FOR \ [SYSTEM\_TIME\ |\ BUSINESS\_TIME]\ AS\ OF\ \text{xxx}}\]

- To query temporal data from one point-in-time to another (remembering that the specified start time is included in the period but the specified end time is not):
  \[\text{FOR \ [SYSTEM\_TIME\ |\ BUSINESS\_TIME]\ FROM\ \text{xxx\ TO\ yyy}}\]

- To query temporal data as of a specific between a range of beginning and ending times (again, remembering that the specified start time is included in the period but the specified end time is not):
  \[\text{FOR \ [SYSTEM\_TIME\ |\ BUSINESS\_TIME]\ BETWEEN\ \text{xxx\ AND\ yyy}}\]
What is the price of the Ethics course on Christmas day 2011?

```
SELECT PRICE
FROM COURSE FOR BUSINESS_TIME AS OF '2011-12-25'
WHERE TITLE = 'ETHICS';
```

How would we write this without temporal support?

```
SELECT PRICE
FROM COURSE
WHERE BUS_START < '2011-12-25'
AND BUS_END > '2011-12-25'
AND TITLE = 'ETHICS';
```
What are the course terms on offer from April 30, 2009 until June 30, 2012?

```
SELECT COURSENO, TITLE, CREDITS, PRICE, 
    BUS_START, BUS_END 
FROM     COURSE FOR BUSINESS TIME FROM '2009-04-30' TO '2012-06-30';
```

This query returns the following results:

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>BUS_START</th>
<th>BUS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>300.00</td>
<td>2011-10-01</td>
<td>2012-01-01</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>350.00</td>
<td>2012-01-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>200</td>
<td>INTRO TO DB2</td>
<td>1</td>
<td>150.00</td>
<td>2005-01-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>3</td>
<td>500.00</td>
<td>2009-01-01</td>
<td>2010-11-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>500.00</td>
<td>2010-11-01</td>
<td>2011-03-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>550.00</td>
<td>2011-03-01</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>
And, of course, you can write non-temporal queries against temporal tables. For example:

```
SELECT *
FROM COURSE
WHERE COURSENO = 220;
```

This query returns three rows because there are three rows in the COURSE table for course number 220.

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>BUS_START</th>
<th>BUS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>DB2</td>
<td>SQL</td>
<td>3</td>
<td>500.00</td>
<td>2009-01-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2</td>
<td>SQL</td>
<td>4</td>
<td>500.00</td>
<td>2010-11-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2</td>
<td>SQL</td>
<td>4</td>
<td>550.00</td>
<td>2011-03-01</td>
</tr>
</tbody>
</table>
Modifying Business Time Data: INSERT

• Here we successfully INSERT to the temporal table:

```
INSERT INTO COURSE
    (COURSENO, TITLE, CREDITS, PRICE, BUS_START, BUS_END)
VALUES
    (300, 'INTRO TO IMS', 3, 300.00, '2011-07-01', '9999-12-31');
```

• But this INSERT fails. Why?

```
INSERT INTO COURSE
    (COURSENO, TITLE, CREDITS, PRICE, BUS_START, BUS_END)
VALUES
    (200, 'INTRO TO DB2', 1, 175.00, '2011-07-01', '9999-12-31');
```
### Modifying Business Time Data: UPDATE

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>BUS START</th>
<th>BUS END</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>300.00</td>
<td>2011-10-01</td>
<td>2012-01-01</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>350.00</td>
<td>2012-01-01</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

- An **UPDATE** to change the price for the Ethics course for the time period from November 1, 2011 to February 2, 2012:

  ```sql
  UPDATE COURSE
  FOR PORTION OF BUSINESS_TIME FROM '2011-11-01' TO '2012-02-02'
  SET PRICE = 325.00
  WHERE COURSENO = 100;
  ```

  - Both rows are impacted by the **UPDATE** statement, because the portion of business time that is being updated overlaps partially with the business period of each row. To make this change, DB2 splits each of the two original rows into two rows.
After Issuing the UPDATE

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>BUS_START</th>
<th>BUS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>300.00</td>
<td>2011-10-01</td>
<td>2011-11-01</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>325.00</td>
<td>2011-11-01</td>
<td>2012-01-01</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>325.00</td>
<td>2012-01-01</td>
<td>2012-02-02</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>350.00</td>
<td>2012-02-02</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>150</td>
<td>FINANCE</td>
<td>3</td>
<td>300.00</td>
<td>2000-01-01</td>
<td>2005-01-01</td>
</tr>
<tr>
<td>150</td>
<td>FINANCE</td>
<td>3</td>
<td>250.00</td>
<td>2005-01-01</td>
<td>2005-12-31</td>
</tr>
<tr>
<td>200</td>
<td>INTRO TO DB2</td>
<td>1</td>
<td>150.00</td>
<td>2005-01-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>3</td>
<td>500.00</td>
<td>2009-01-01</td>
<td>2010-11-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>500.00</td>
<td>2010-11-01</td>
<td>2011-03-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>550.00</td>
<td>2011-03-01</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

- This is the data after running the **UPDATE** statement on the previous slide.
Modifying Business Time Data: DELETE

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>BUS_START</th>
<th>BUS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>550.00</td>
<td>2011-03-01</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

- A DELETE of the DB2 SQL course for the time period from July 3, 2011 to July 9, 2011:

```
DELETE FROM COURSE
FOR PORTION OF BUSINESS_TIME FROM '2011-07-03' TO '2011-07-09'
WHERE COURSENO = 220;
```

- Both rows are impacted by the UPDATE statement, because the portion of business time that is being updated overlaps partially with the business period of each row. To make this change, DB2 splits each of the two original rows into two rows.
After Issuing the DELETE

Note: the week of July 3 thru July 9 is no longer represented in the table for COURSENO 220

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>BUS_START</th>
<th>BUS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>300.00</td>
<td>2011-10-01</td>
<td>2011-11-01</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>325.00</td>
<td>2011-11-01</td>
<td>2012-01-01</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>325.00</td>
<td>2012-01-01</td>
<td>2012-02-02</td>
</tr>
<tr>
<td>100</td>
<td>ETHICS</td>
<td>3</td>
<td>350.00</td>
<td>2012-02-02</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>150</td>
<td>FINANCE</td>
<td>3</td>
<td>300.00</td>
<td>2000-01-01</td>
<td>2005-01-01</td>
</tr>
<tr>
<td>150</td>
<td>FINANCE</td>
<td>3</td>
<td>250.00</td>
<td>2005-01-01</td>
<td>2005-12-31</td>
</tr>
<tr>
<td>200</td>
<td>INTRO TO DB2</td>
<td>1</td>
<td>150.00</td>
<td>2005-01-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>3</td>
<td>500.00</td>
<td>2009-01-01</td>
<td>2010-11-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>500.00</td>
<td>2010-11-01</td>
<td>2011-03-01</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>550.00</td>
<td>2011-03-01</td>
<td>2011-07-03</td>
</tr>
<tr>
<td>220</td>
<td>DB2 SQL</td>
<td>4</td>
<td>550.00</td>
<td>2011-07-09</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>
Implementing System Time

• OK, we’ve walked through business time examples, but what about system time?

• Setting up system time support requires a 3 step process:
  1. Create the base table and include three `TIMESTAMP(12)` columns. Two of these columns are familiar to us -- the starting and ending points for the system time period (similar to how business time support). The additional timestamp is for the transaction start time. DB2 uses the transaction start time column to track when the transaction first executed a statement that changes the table’s data.
  2. Create the history table with an identical structure, preferably by using a `CREATE TABLE . . . LIKE` statement.
  3. `ALTER` the current table to enable versioning and identify the history table.
Let’s set up a table to track system time.

- Start by creating the COURSE table with three TIMESTAMP(12) columns and a SYSTEM_TIME specification:

```sql
CREATE TABLE COURSE
(COURSENO INTEGER PRIMARY KEY NOT NULL,
TITLE VARCHAR(10) NOT NULL,
CREDITS SMALLINT NOT NULL WITH DEFAULT 3,
PRICE DECIMAL(7,2) NOT NULL,
SYS_START TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
SYS_END TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
TXN_START TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID IMPLICITLY HIDDEN,

PERIOD SYSTEM_TIME(SYS_START, SYS_END)
);
```
2. Define the history table. Remember system time saves history in a separate table:

```
CREATE TABLE COURSE_HIST
    LIKE COURSE;
```

3. Then we ALTER the table to add versioning specifying the newly created history table:

```
ALTER TABLE COURSE
    ADD VERSIONING
    USE HISTORY TABLE COURSE_HIST;
```
Let’s Start by Adding Data to our System Time “Table”

– After inserting these three rows →

```sql
INSERT INTO COURSE (COURSENO, TITLE, CREDITS, PRICE)
VALUES (500, 'INTRO TO COBOL', 2, 200.00);

INSERT INTO COURSE (COURSENO, TITLE, CREDITS, PRICE)
VALUES (600, 'INTRO TO JAVA', 2, 250.00);

INSERT INTO COURSE (COURSENO, TITLE, CREDITS, PRICE)
VALUES (650, 'ADVANCED JAVA', 3, 400.00);
```

– Our tables look like this...

**COURSE table**

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>INTRO TO COBOL</td>
<td>2</td>
<td>200.00</td>
<td>2012-01-10</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>600</td>
<td>INTRO TO JAVA</td>
<td>2</td>
<td>250.00</td>
<td>2012-01-10</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>400.00</td>
<td>2012-01-10</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

**COURSE_HIST table**

The table contains three rows

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table is empty
Then we UPDATE our System Time “Table”

- After issuing this
  
  UPDATE statement →

  ```sql
  UPDATE COURSE
  SET PRICE = 375.00
  WHERE COURSENO = 650;
  ```

- Our tables now look like this...

**COURSE table**

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>INTRO TO COBOL</td>
<td>2</td>
<td>200.00</td>
<td>2012-01-10</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>600</td>
<td>INTRO TO JAVA</td>
<td>2</td>
<td>250.00</td>
<td>2012-01-10</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>375.00</td>
<td>2012-01-15</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

**COURSE_HIST table**

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>400.00</td>
<td>2012-01-10</td>
<td>2012-01-15</td>
</tr>
</tbody>
</table>

The table contains three rows

The table contains one row
And then let’s DELETE from our System Time “Table”

– After issuing this DELETE statement

```
DELETE FROM COURSE
WHERE COURSENO = 600;
```

– Our tables now look like this...

**COURSE table**

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>INTRO TO COBOL</td>
<td>2</td>
<td>200.00</td>
<td>2012-01-10</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>375.00</td>
<td>2012-01-15</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

The table contains two rows

**COURSE_HIST table**

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>400.00</td>
<td>2012-01-10</td>
<td>2012-01-15</td>
</tr>
<tr>
<td>600</td>
<td>INTRO TO JAVA</td>
<td>2</td>
<td>250.00</td>
<td>2012-01-10</td>
<td>2012-02-05</td>
</tr>
</tbody>
</table>

The table contains two rows
Before We Query System Time, Let’s Change More Data

```
INSERT INTO COURSE
  (COURSENO, TITLE, CREDITS, PRICE)
VALUES
  (700, 'INTRO TO BASIC', 2, 150.00);

UPDATE COURSE
SET PRICE = 385.00
WHERE COURSENO = 650;

UPDATE COURSE
SET CREDITS = 1
WHERE COURSENO = 500;
```
Our Sample System Time Data (for queries)

COURSE table

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>INTRO TO COBOL</td>
<td>1</td>
<td>200.00</td>
<td>2012-03-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>385.00</td>
<td>2012-03-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>700</td>
<td>INTRO TO BASIC</td>
<td>1</td>
<td>175.00</td>
<td>2012-03-01</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

The table contains three rows

COURSE_HIST table

<table>
<thead>
<tr>
<th>COURSENO</th>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
<th>SYS_START</th>
<th>SYS_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>400.00</td>
<td>2012-01-10</td>
<td>2012-01-15</td>
</tr>
<tr>
<td>600</td>
<td>INTRO TO JAVA</td>
<td>2</td>
<td>250.00</td>
<td>2012-01-10</td>
<td>2012-02-05</td>
</tr>
<tr>
<td>650</td>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>375.00</td>
<td>2012-01-15</td>
<td>2012-03-01</td>
</tr>
<tr>
<td>500</td>
<td>INTRO TO COBOL</td>
<td>2</td>
<td>200.00</td>
<td>2012-01-10</td>
<td>2012-03-01</td>
</tr>
</tbody>
</table>

The table contains four rows
• Simply retrieve the current info for Advanced Java course:

```
SELECT * FROM COURSE WHERE COURSENO = 650;
```

• Access info about Advanced Java course for January 16, 2012:

```
SELECT TITLE, CREDITS, PRICE
FROM COURSE FOR SYSTEM_TIME AS OF TIMESTAMP('2012-01-16')
WHERE COURSENO = 650;
```

- Returns one row →

<table>
<thead>
<tr>
<th>TITLE</th>
<th>CREDITS</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVANCED JAVA</td>
<td>3</td>
<td>375.00</td>
</tr>
</tbody>
</table>
• Retrieve the info for COURSENO 600 (Intro to Java)
  – But it was deleted on 2/5/12
  – No problem, DB2 finds it in the history table

```
SELECT TITLE
FROM COURSE FOR SYSTEM_TIME AS OF TIMESTAMP('2012-02-01')
WHERE COURSENO = 600;
```
Bitemporal Tables

- DB2 supports tables with both a business time and a system time
  - This is known as a bitemporal table

```sql
CREATE TABLE COURSE
(
  COURSENO INTEGER NOT NULL,
  TITLE VARCHAR(10) NOT NULL,
  CREDITS SMALLINT NOT NULL WITH DEFAULT 3,
  PRICE DECIMAL(7,2) NOT NULL,
  BUS_START DATE NOT NULL,
  BUS_END DATE NOT NULL,
  SYS_START TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
  SYS_END TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
  TXN_START TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID IMPLICITLY HIDDEN,

  PERIOD SYSTEM_TIME(SYS_START, SYS_END),
  PERIOD BUSINESS_TIME(BUS_START, BUS_END),

  PRIMARY KEY (COURSENO, BUSINESS_TIME WITHOUT OVERLAPS)
);CREATE TABLE COURSE_HIST
LIKE COURSE;
ALTER TABLE COURSE
ADD VERSIONING
USE HISTORY TABLE COURSE_HIST;
```
Temporal Guidelines

• Choose Business Time Period Data Types Wisely

• Favor Using WITHOUT OVERLAPS for Business Time

• Index the temporal columns

• Take Your Time Before Implementing Temporal Tables
Some Additional SQL Changes in DB2 V10

- Indicator Variables
- TIMESTAMP Improvements
  - precision and time zones
- Moving sums and averages
- Functions
  - precision and time zones
  - precision and time zones
- Implicit Casting
- Other “Stuff”
Extended Indicator Variables

Traditionally used to indicate if a column is NULL
DB2 V10 extends their use for conditional programming
Consider: program being written to modify data.
  • Multiple combinations of columns can be modified; perhaps, the program is editing customer data.
  • The customer has multiple columns that could be modified: name, address, telephone number, credit rating, etc.
  • What happens when ENTER is hit?
Use extended indicator variables (on host variables and parameter markers) to assist
  • EXTENDEDINDICATOR option of BIND PACKAGE
  • WITH EXTENDED INDICATORS on PREPARE statement

http://en.wikipedia.org/wiki/Factorial
Extended Indicator Variable Values

Set the extended indicator value as desired:

• **0 (zero) or a positive integer**: This indicates the first host identifier provides the value of this host variable reference and it is not null.

• **-1, -2, -3, -4, or -6**: This indicates a null.

• **-5**: If extended indicator variables are not enabled, this indicates a null; otherwise, a value of -5 indicates that the DEFAULT value is to be used for the target column for this host variable.

• **-7**: If extended indicator variables are not enabled, this indicates a null; otherwise, a value of -7 indicates that the UNASSIGNED value is to be used for the target column for this host variable (in other words, treat it as if it were not specified in this statement).
TIMESTAMP data with greater precision

• Prior to V10, all TIMESTAMP data supported a fractional second of 6 digits in the format:
  2002-10-20-12.00.00.000000

• As of V10, you can specify a precision to indicate the amount of fractional seconds.
  • The default precision is 6.

TIMESTAMP with TIME ZONE

• Prior to V10, no time zone information could be stored

• As of V10, you can optionally specify a TIMESTAMP WITH TIME ZONE to add time zone information to your data.
V10 gives us OLAP specifications for moving sums and moving averages.

• Here is an example calculating the 30 day moving average for the stocks 'ABC' and 'XYX' during 2005:

```sql
CREATE VIEW V1 AS
SELECT SYMBOL, TRADINGDATE,
  AVG(CLOSINGPRICE) OVER (PARTITION BY SYMBOL
  ORDER BY TRADINGDATE
  ROWS BETWEEN 29 PRECEDING AND CURRENT ROW)
FROM DAILystockdata
WHERE SYMBOL IN ('ABC', 'XYZ')
  AND TRADINGDATE BETWEEN DATE('2005-01-01') - 2 MONTHS AND '2005-12-31';

SELECT SYMBOL, TRADINGDATE, MOVINGAVG30DAY
FROM V1
WHERE TRADINGDATE BETWEEN '2005-01-01' AND '2005-12-31'
ORDER BY SYMBOL, TRADINGDATE;
```
Function Improvements

• Table Functions
  • A function that returns a table as its result
  • Use a single RETURN control statement in function body
  • No package generated

• Inline and Non-inline SQL scalar functions
  • As of DB2 10, there are inline and non-inline scalar functions
  • Inline functions do not generate a package; the reference is replaced by the expression
  • Non-inline functions generate a package

• Scalar function versioning
  • Similar to native SQL stored procedure versioning
  • Can have multiple versions using ADD VERSION keyword in CREATE/ALTER FUNCTION
  • Can activate a specific version using ALTER FUNCTION...ACTIVATE VERSION
DB2 V10 offers support for implicit casting of strings and numerics

- DB2 can perform an implicit cast between character or graphic strings and numeric data types (Except for LOB or non-Unicode graphic strings)
- Of course, using CAST still offers more control and is preferred
- The implicit cast is performed according to the following chart:

<table>
<thead>
<tr>
<th>Source Data Type</th>
<th>Target Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>VARCHAR(6)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>VARCHAR(11)</td>
</tr>
<tr>
<td>BIGINT</td>
<td>VARCHAR(20)</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>VARCHAR(p+2)</td>
</tr>
<tr>
<td>REAL, FLOAR, DOUBLE</td>
<td>VARCHAR(24)</td>
</tr>
<tr>
<td>DECFLOAT</td>
<td>VARCHAR(42)</td>
</tr>
<tr>
<td>CHAR, CARCHAR</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>GRAPHIC, VARGRAPHIC</td>
<td>DECFLOAT(34)</td>
</tr>
</tbody>
</table>
Other “Stuff”

- XML Changes and Improvements
- Enhanced support for native SQL procedures
- IBM Data Server Driver for JDBC and SQLJ Type 2 connectivity enhancements
- New Universal Language Interface module (DSNULI)
Summary

Learn the changes
Understand the impact
Plan ahead
REBIND for performance
Do one thing at a time and;
You *can* migrate to DB2 10!
Contact Information


Craig S. Mullins
Mullins Consulting, Inc.
15 Coventry Court
Sugar Land, TX 77479

http://www.craigsmullins.com

craig@craigsmullins.com

Phone: (281) 494-6153

랙 6th edition available now... covering up thru DB2 V10!

http://www.craigsmullins.com/cm-book.htm

© 2012 Mullins Consulting, Inc.