This presentation provides an early look at some DB2 10 features which will be important to you as the Application Developer or DBA. We will start with the basics and demonstrate how they can be an integral part of a good design strategy. This will allow you to start the planning process early and see if/how they fit in your organization.

We will not just scratch the surface but take a deep dive into features most important to you as a developer and a DBA.
Temporal tables
• Business driver
• Set up
• Access
• Performance
• Recommendations

Hash access
• Business driver
• Set up
• Access
• Performance
• Recommendations

New SQL features
• Recap of ROW_NUMBER, RANK and DENSE_RANK
• Moving sum
• Moving average
• Generic insert/update (extended indicator variables)
• Implicit casting
• Greater precision for timestamp

Access path optimization
• Literal replacement
• PTC for IN lists
• Scrolling
• Safe query optimization
  • Dealing with uncertainty
  • RID List failures
• Explaining dynamic SQL

Currently committed data
• Business driver
• Access
• Interaction with other zparms
• Comparison and recommendations
Suresh Sane is an IDUG Hall of Fame speaker with two Best User Speaker awards and numerous top 10 finishes. He has lectured worldwide and co-authored 4 IBM Redbooks (Dynamic SQL, Stored Procedures, Data Integrity and DB2 Packages). He was recognized as an IBM Information Champion in 2009 and in 2010.

He is a long time IDUG volunteer and was the Conference Chair for IDUG NA 2008. He currently serves on the IDUG Board of Directors.

Contact Information:

sssane@dstsystems.com

Suresh Sane
DST Systems, Inc.
1055 Broadway
Kansas City, MO 64105
USA

(816) 435-3803
If you have ever invested in a mutual fund, have had a prescription filled, or are a cable or satellite television subscriber, you may have already had dealings with our company.

DST Systems, Inc. is a publicly traded company (NYSE: DST) with headquarters in Kansas City, MO. Founded in 1969, it employs about 10,000 associates domestically and internationally.

The three operating segments - Financial Services, Output Solutions and Customer Management - are further enhanced by DST’s advanced technology and e-commerce solutions.
In this section, we will discuss how temporal tables can simplify very complex business logic in use today.
While the name “temporal table” applies to both types of tables, they deal with different business problems – system-time simplifies versioning and consists of two tables while business time consists of only one table with start and end.
Temporal tables with System Time

- Useful for auditing and compliance
- Two tables – main and history
- Each row on the main table has a pair of timestamps set by DB2 - start time & end time – must be TIMESTAMP(12)
- Main table defines a PERIOD SYSTEM_TIME with these columns
- History table identical in structure, connected to main table via the ADD VERSIONING USE HISTORY TABLE clause
Sample DDL for creating the base table and its associated history table. They are linked via the “ADD VERSIONING USE HISTORY TABLE” clause.
Let’s explore how this works in practice.

Note that the end time is NOT inclusive – for example June 15th is not included in the row ending on June 15th (but start time is).
The logic to determine “As of” condition is now built into the DB2 engine itself.
Versioning (tables with SYSTEM_TIME) restrictions

- No alter of schema (data type, add column, etc.) allowed on both tables - i.e. DROP TABLE required!
- Cannot drop the history table its tablespace
- Cannot define a clone table to either of the tables
- Each table must be the only table in tablespace
- Cannot RENAME column or table
- For point-in-time recovery, both must be recovered as a set (individual recovery disallowed unless the keywords ENFORCE NO is used)
- No utility operation allowed that will delete data from the system period temporal table
- Third-party tool support?

Some restrictions. Most important at this time is probably the tool support.
Temporal tables with Business Time

- Useful for tracking of business events over time
- Only 1 table (no history table as in System Time)
- Each row has a pair of dates set by application – start date and end date (can be future dates) – must be TIMESTAMP(6) NOT NULL or DATE NOT NULL
- The table defines a PERIOD BUSINESS_TIME with these columns
- Unique index possible on period to prevent overlaps
DDL to create a table with business time.

```
CREATE TABLE PRICES
( ITEMNO SMALLINT NOT NULL,
  PRICE DEC(9,2) NOT NULL,
  REASON CHAR(10) NOT NULL,
  START_DT DATE NOT NULL,
  END_DT DATE NOT NULL,
  PERIOD BUSINESS_TIME
  (START_DT,END_DT))

CREATE UNIQUE INDEX PRICESK0
ON PRICES
(ITEMNO, BUSINESS_TIME WITHOUT OVERLAPS)
```
An example of how this would work in practice. Notice all the activity which is automatically generated by the update.

Also note that the end-date is NOT included as part of the interval (but start date is).
Example of how “As of” queries are supported.
Business time restrictions

- No ALTER INDEX ADD BUSINESS_TIME WITHOUT OVERLAPS
- No SELECT FROM DELETE or SELECT FROM UPDATE when UPDATE or DELETE with FOR PORTION OF specified

Some restrictions.
Bi-temporal tables

- Combines features of system time as well as business time
- System time:
  - Useful for auditing and compliance
  - Two tables — main and history
- Business time:
  - Each row has a pair of dates set by application – start date and end date
- Need to determine if added complexity provides benefit

Bi-temporal tables benefit from the auditing capability provided by system time as well as the “as of” capability provided by business time.
Sample DDL for creating the base table and its associated history table. They are linked via the “ADD VERSIONING USE HISTORY TABLE” clause. In this case, each table also has the business times.
Recommendations

- RI – need to version all related code tables? Lack of declarative RI (range vs. value based)?
- Single table OK but versioning for a group – more complex?
- Bi-temporal may have limited use (my opinion only)
- For SYSTEM_TIME, increased storage for the table (depending on update/delete activity)
- For SYSTEM_TIME, degraded database performance for UPDATE and DELETE
  - Querying historical data involves a UNION ALL process and period search will be with multiple range predicates.
- For BUSINESS_TIME, understanding the functionality of UPDATE and DELETE FOR PORTION OF.

A few considerations and my preliminary recommendations (I reserve the right to change my mind as we get more experience).
In this section we will discuss how access works and when it might be appropriate.
A typical B-tree structure for a large index consisting of 5 levels.
General guideline on when you should consider hash access.

**Overview**

- **Good for**
  - Tables with a unique key, whose index is many levels (>= 5?)
  - Applications (such as OLTP) needing single row access via the unique key
  - Known approximate table size
  - Equal predicate on all hash key columns (or IN-List)

- **Not good for**
  - Sequential processing
  - Using range predicates e.g. BETWEEN or > and <
DDL to create a hash table.

```
CREATE TABLE...
...
PARTITION BY RANGE...
PARTITION 1 ...
....
PARTITION 5 .....HASH SPACE 1G
...
ORGANIZE BY HASH UNIQUE
(lastname,firstname)
HASH SPACE 2G
```
How hashing works and how collisions are handled.
Monitoring

- Important Real Time Statistics (RTS) values are:
  - SYSTABLESPACESTATS.TOTALROWS - actual number of rows in the table
  - SYSTABLESPACESTATS.DATASIZE - total number of bytes used for rows
  - SYSINDEXSPACESTATS.TOTALENTRIES - number of overflow records with keys in the overflow index
- Ideally, HASH SPACE (in DDL) should be close to DATASIZE (in RTS)
- TOTENTRIES as a percentage of TOTALROWS should be (ideally zero) less than 10%
- ALTER HASH SPACE and REORG or let DB2 decide during REORG

Key parameters to watch.
Usage notes

- A hash access table must be the only table in the table space (dictated by UTS rules)
- Hash access cannot support clustering indexes and member cluster
- Cloned tables cannot be altered to add Hash organization
- LOAD performance may be slower because the data will not be sorted in page order
- Hash key column values cannot be updated - the row must be deleted and re-inserted with the new key value
- If the fixed hash space is too small then performance may suffer

Some considerations and restrictions.
### Catalog Impact

<table>
<thead>
<tr>
<th>Table</th>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCOLUMNS</td>
<td>HASHKEYCOLSEQ</td>
<td>Numeric position of column in table’s hash key</td>
</tr>
<tr>
<td>SYSCOPY</td>
<td>STYPE</td>
<td>H = Hash organization alloned</td>
</tr>
<tr>
<td></td>
<td>TTYPE</td>
<td>For retyr, and stype=M, prev. value of HASHIDATAPAGES</td>
</tr>
<tr>
<td>SYSINDEXES</td>
<td>HASH</td>
<td>Y = index is hash overflow index for the table</td>
</tr>
<tr>
<td>SYSINDEKSPACESTATS</td>
<td>REORGINDEXACCESS</td>
<td>(# of times index was used) - for hash overflow indexes - # of overflow uses</td>
</tr>
<tr>
<td>(RTS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTABLEPART</td>
<td>HASHSPACE</td>
<td>For PBR, space (KB) override for table level</td>
</tr>
<tr>
<td>SYSTABLES</td>
<td>HASHDATAPAGES</td>
<td>For PBR, space (pages) for part</td>
</tr>
<tr>
<td>SYSTABLESPACE</td>
<td>HASHKEYCOLUMNS</td>
<td># of columns in hash key</td>
</tr>
<tr>
<td>SYSTABLESPACE</td>
<td>ORGANIZATIONTYPE</td>
<td>H=hash</td>
</tr>
<tr>
<td></td>
<td>HASHSPACE</td>
<td>Space (KB)</td>
</tr>
<tr>
<td></td>
<td>HASHDATAPAGES</td>
<td>Space (pages)</td>
</tr>
<tr>
<td>SYSTABLESPACESTATS</td>
<td>REORGHASHACCESS</td>
<td># of times hash access was used</td>
</tr>
<tr>
<td>(RTS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HASNLASTUSED</td>
<td>Date when hash access was last used</td>
</tr>
</tbody>
</table>

The relevant columns in the catalog for hash access.
In this section we will cover some of the new SQL features offered in DB2 10.
Recap of rownum, rank and dense rank - SQL

```
SELECT ID, FIRSTNAME, LASTNAME, SALARY, RANK() OVER (ORDER BY SALARY DESC) AS SAL_RANK
FROM Y915
ORDER BY SAL_RANK
```

```
SELECT ID, FIRSTNAME, LASTNAME, SALARY, ROW_NUMBER() OVER (ORDER BY SALARY DESC) AS SAL_ROWNUM
FROM Y914
ORDER BY SAL_ROWNUM
```

```
SELECT ID, FIRSTNAME, LASTNAME, SALARY, DENSE_RANK() OVER (ORDER BY SALARY DESC) AS SAL_DRANK
FROM Y910
ORDER BY SAL_DRANK
```

This is not new, but covered here for completeness only.
How these 3 features work – an example.
We will cover moving sum and moving average only, but other aggregate functions (e.g. CORRELATION, STDDEV etc. can also be used).

Choices include where to start (UNBOUNDED PRECEDING, n PRECEDING or CURRENT ROW) and where to stop (UNBOUNDED FOLLOWING or n FOLLOWING).
### Moving sum - example

<table>
<thead>
<tr>
<th>CITY</th>
<th>YEAR</th>
<th>MONTH</th>
<th>RAINFALL</th>
<th>SUM_RAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>3</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>4</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>5</td>
<td>9</td>
<td>23</td>
</tr>
<tr>
<td>SFATTI F</td>
<td>2010</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>2</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>3</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>4</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>5</td>
<td>7</td>
<td>39</td>
</tr>
</tbody>
</table>

An example of how it works.
Moving average - SQL

```
SELECT CITY, YEAR, MONTH, RAINFALL,
    AVG(RAINFALL) OVER
    (PARTITION BY CITY
     ORDER BY YEAR ASC, MONTH ASC
     ROWS 2 PRECEDING)
    AS AVG_RAIN
FROM X914
ORDER BY CITY, YEAR, MONTH
```

Choices include where to start (UNBOUNDED PRECEDING, n PRECEDING or CURRENT ROW) and where to stop (UNBOUNDED FOLLOWING or n FOLLOWING).
Moving average - example

<table>
<thead>
<tr>
<th>CITY</th>
<th>YEAR</th>
<th>MONTH</th>
<th>RAINFALL</th>
<th>AVG_RAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>KANSAS CITY</td>
<td>2010</td>
<td>5</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>SFATTI F</td>
<td>2010</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>2</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>3</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>2010</td>
<td>5</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

An example of how it works.
A typical problem scenario, especially for an update screen.
Business challenges

- For handling a “generic” insert/update (see previous slide), a user has three main choices:
  - Always update all columns – even when not changed
    - “Lazy” programmer’s choice perhaps most common
    - DB2 resources wasted
  - Create static SQL for each combination
    - Efficient
    - Number of combinations grows exponentially
    - No practical
  - Use dynamic SQL
    - Complex
    - Dynamic SQL cache reuse may be limited (even with parameter markers, if large number of combinations)

Choices which exist today.
Extended NULL indicators to the rescue!

- Enabled as a bind option for static SQL (EXTENDEDINDICATOR(YES)) or by PREPARE for dynamic SQL – default is NO!
  - -5 means:
    - For insert/merge insert: use the default value for the column
    - For update/merge update: use the default value for the column
  - -7 means:
    - For insert/merge insert: use the default value for the column
    - For update/merge update: no-op (column did not change)
  - Application logic sets the indicator variable for each host variable appropriately (see next slide)
  - One generic insert/update/merge statement can be now be used safely

A new choice now available in DB2 10.
A coding example of how the indicators will be set by the application and used in an insert statement.
A coding example of how the indicators will be set by the application and used in an update statement.
Implicit casting

- DB2 9 implicitly casts in many cases - e.g. INTEGER to DECIMAL
- DB2 10 extends this to casting between a character or graphic string type and a numeric type
- Allows index access
### Implicit casting from numeric to string

<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>NUMERIC/DECIMAL</td>
<td>VARCHAR(precision+2)</td>
</tr>
<tr>
<td>REAL</td>
<td>VARCHAR(24)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>VARCHAR(24)</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>VARCHAR(24)</td>
</tr>
<tr>
<td>DECFLOAT</td>
<td>VARCHAR(42)</td>
</tr>
</tbody>
</table>
Note that DB2 always attempts to covert a string to DECFLOAT. This means that an invalid string e.g. '12AB34' (embedded blanks) will cause an SQLCODE -420. The message text may be confusing since you may be attempting to covert to DECIMAL or INTEGER.

<table>
<thead>
<tr>
<th>Source data type</th>
<th>Target data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>GRAPHIC</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>VARCHAR2</td>
<td>DECFLOAT(34)</td>
</tr>
</tbody>
</table>
Greater precision for timestamp

- Number of digits for the fractional second in a timestamp extended
  - Range supported in DB2 10 is 0 to 12 digits
    - Maximum of \texttt{TIMESTAMP(12)}
    - String representation: \texttt{yyyy-mm-dd- hh.mm.ss.nnnnnnnnnn}
- Greater precision may be useful when timestamp is used as a unique key
  - Reduces chance of duplicate
  - Still need logic to handle duplicates in application
- The DB2 9 default of 6 digits remains
  - \texttt{TIMESTAMP} is the same as \texttt{TIMESTAMP(6)}
  - String representation: \texttt{yyyy-mm-dd- hh.mm.ss.nnnnnn}

Also available is “timestamp with timezone” but I see limited use for this feature and I will not cover it here.
This section will cover the improvements made in access path selection and additional features to stabilize and/or fallback easily.
Full caching combines the two benefits (ability to issue an EXECUTE without PREPARE – small impact) + (the ability reuse the statement prepared by another thread – big impact).

Activated by KEEPDYNAMIC(YES) bind parameter and zparm MAXKEEPD > 0 and zparm CACHEDYN=YES.
No need to issue a prepare after a commit.
Local cache and Global cache both shown.
Literal replacement

- Advantages of dynamic statement cache
  - Avoids Full PREPARE
  - Requires the user id and statement to be identical
- Overcoming the problem with literals
  - WHERE EMPNO = 123 different from WHERE EMPNO = 456
  - Parameters markers preferred e.g. WHERE EMPNO = ?
  - Literals replaced with & (like ? but different)
  - Enables cache re-use
- Performance impact
  - Biggest gain for complex SQL (high PREPARE time) with literals which now has a cache hit
  - Beware of NUD/Correlation: REOPT will be needed – else worse performance!

How literal replacement can provide better performance (in most cases).
Literal replacement – How does it work?

- Enablement (either one of..)
  - Put CONCENTRATE STATEMENTS WITH LITERALS in the ATTRSTRING in the PREPARE – (yes, but code change!)
  - Set LITERALREPLACEMENT in the ODBC initialization file
  - Set the keyword enableLiteralReplacement='YES' in the JDBC Driver

- Lookup sequence
  - Original SQL with literals is searched in the cache
    - If not found, literals replaced and searched again
    - Must have same attribute – i.e. ? does not match &
    - If not found, new SQL is prepared and stored in the cache

Details on how the replacement can be specified. The choice requiring code change seems the least attractive to me.
Simple example of how transitive closure works.
A common problem that haunts all restartable cursors! This feature spells R-E-L-I-E-F!!! I am expecting a huge performance gain for cursors accessing large tables in CICS programs which use such restartable cursors.

**Scrolling/Re-positioning issues**

- Most scrolling predicates provide no filtering at all and easily mislead the optimizer – example:

```
WHERE (FUND > :WS-FUND
OR (FUND = :WS-FUND AND
   ACCT > :WS-ACCT)
OR (FUND = :WS-FUND AND
   ACCT = :WS-ACCT AND
   DATE > :WS-DATE))
AND FUND >= :WS-FUND
ORDER BY FUND, ACCT, DATE
```

Index on: FUND
          ACCT
          DATE

Redundant predicate                   Matching index access
Needed in DB2 9                        Possible in DB2 10
For matching index access             3 probes – access type NR 49
Safe query optimization

• Common reasons for bad access paths
  • Host variables or parameter markers with non uniform distribution of data – most common in our environment
  • Missing stats or stats not current
  • Unpredictable runtime resource availability – especially, RID pool usage

• Access path based purely on cost-based optimization needs a “reality check” (just like degree of parallelism)
  • Dealing with reality at run time
  • Dealing with uncertainty at bind time

Features of the “safe optimization” initiative.
Dealing with RID pool failures

- If a RID limit is reached
  - Overflow RIDs to workfile and continue processing
  - Avoids fallback to table space scan as in DB2 9
- Work-file usage may increase
  - Need to monitor
  - Default increased in DB2 10

Some implications of what safe optimization means at run time.
Dealing with uncertainty

- Optimizer evaluates the “risk” associated with each predicate
- Compare access paths with close cost and chooses lowest risk plan
- Full details of “uncertainty” are NOT externalized!
- A good first step but not there yet (my opinion) – see next slide

Some implications of what safe optimization means at bind time.
You want minimum cost, but what cost? Average or worst-case?

This tradeoff between performance and predictability has been discussed in various papers, most notably by Brian Babcock and Surajit Chaudhuri (see ref #3).

In their own words:

“..a robust query optimizer is one that generates plans that work reasonably well even when optimizer assumptions fail to hold.” – yes, DB2 10 attempts to do so.

“Because robustness sometimes comes at the cost of performance, users should be allowed to prioritize these competing objectives.” – no, this is hidden within DB2.
Impact on the catalog tables.

Catalog impact

- SYSIBM.SYSPACKCOPY
  - New catalog table
  - Holds SYSPACKAGE-style metadata for any previous or original package copies
  - Information for inactive copies was visible in DB2 9 only after a SWITCH

- EXPLAIN PACKAGE
  - Extract PLAN_TABLE information for one or more packages and their copies
  - The package/copy must be created on DB2 9 or later
  - Useful if you did not specify EXPLAIN(YES) during BIND
Obtaining EXPLAIN information for dynamic SQL

- For host-based languages (e.g. COBOL), modify the source to add the following prior to PREPARE:
  - EXEC SQL SET CURRENT EXPLAIN MODE = YES END-EXEC.
- For JDBC and SQLJ
  - Set currentExplainmode connection property
  - Automatically sets the CURRENT EXPLAIN MODE special register
- For ODBC and CLI
  - For system-wide setting: Set keyword DB2EXPLAIN in USNAOINI file
  - For specific application: set SQL-ATTR-DB2EXPLAIN using SQLSetConnectAttr() function
- Compatible with DB2 LUW

New options to obtain EXPLAIN information.
The choices.

A couple of things to be aware of.

First, the user must have “explain monitored statements” privilege – not typically granted to developers.

Secondly, if you place an explainable statement and then select from the plan_table, DB2 runs (and explains the selects) also – beware of the extra rows!
In this final section, we will present a new option to provide more concurrency. It comes at a price and we will discuss all the “gotcha’s” also.
Why and How?

- Isolation UR avoids contention, but does not return committed data
- Applications ported from other DBMSs (e.g., Oracle) particularly prone to timeouts
- Returns currently committed data without waiting for locks
- Supported for uncommitted inserts or deletes
- **No support in for uncommitted updates**

The business driver.
How does it work?

- Applies only when accessing UTS on DB2 10 NFM
- If contention is with uncommitted **insert**, it applies to Isolation CS or RS only
- If contention is with uncommitted **delete**, it applies to Isolation CS with CURRENTDATA(NO) only
- Statement level overrides
  - package/plan level which overrides
    - system level
- If lock not available & held by inserter – skip row
- If lock not available & held by deleter – return row

Some considerations and restrictions.
A simple example of how each option works.

Simple example (assumes row-level locking)

User A

Insert 50 and
Not committed

User B

Delete 20 and
Not committed

User C: SELECT *
FROM...

Default (wait for outcome): (10) and wait/timeout on 20
Skip locked data: 10, 30, 40
With UR: 10, 30, 40, 50
Currently committed: 10, 20, 30, 40

Note: UPDATE not supported - causes wait for outcome
Syntax

- Defaults are today’s “wait for outcome” behavior
- New BIND Option
  - CONCURRENTACCESSRESOLUTION(USECURRENTLYCOMMITTED | WAITFOROUTCOME)
- New PREPARE Attribute
  - PREPARE … USE CURRENTLY COMMITTED | WAIT FOR OUTCOME
- New bind option in CREATE/ALTER of PROCEDURE, FUNCTION
  - CONCURRENT ACCESS RESOLUTION USING CURRENTLY COMMITTED | WAIT FOR OUTCOME
When does it apply?

- Applies to row and page locking
  - Page locking
    - IRLM tracks up to 8 rows on a page
    - Page lock request for insert/delete specifies row
    - Page lock request for read specifies row
    - If IRLM knows about it then returns to say deleter/insserter
    - If IRLM doesn’t know about it, then reader will wait for lock to be released
  - Does not apply to table, partition or table space locks
    - Not applicable when LOCK TABLE IN EXCLUSIVE used
    - Not applicable when lock holder is performing mass delete
    - Not applicable if lock holder has escalated

Cases when it applies and when it does not.
What about other zparms?

- SKIPUNCI (skip uncommitted insert) BIND/PREPARE option takes precedence
  - BIND/PREPARE WAIT FOR OUTCOME: zparm is ignored
- EVALUNCZPARM takes precedence
  - Predicate evaluation performed then BIND/PREPARE option is honored
- ZPARMs behave as in DB2 9, if not overridden via BIND/PREPARE options
How do I track how often it happens?

- New counter QISTRCCI (part of Data Manager Statistics) - (IFCID 002)
  - Shows the number of rows skipped by read transactions using currently committed option which finds uncommitted inserts
- Similarly, new counter QISTRCCD
  - Shows the number of rows skipped by read transactions using currently committed option which finds uncommitted deletes

New counter to see how often the feature is activated.
Gotcha’s

- Currently committed may allow committed data to be returned without waiting
- BUT – *does not guarantee that DB2 will do so* - in some cases DB2 may revert to unconditional locking (e.g. more than 8 rows locked on a page)
- Updates are NOT supported (where it is needed most!)

Some things to watch out for. In my opinion, the most troublesome area is update logic on control tables (e.g. next account number) – this is where DB2 10 does NOT support it...oh well, there will be DB2 11...
Summary

1. Temporal tables
   - Business driver, Set up, Access, Performance, Recommendations

2. Hash access
   - Business driver, Set up, Access, Performance, Recommendations

3. New SQL features
   - Moving sum, Moving average, Generic update, Implicit casting, Greater precision for timestamp

4. Access path optimization
   - Literal replacement, PTC for IN list, Scrolling, Dealing with RID pool failures and uncertainty, Explaining dynamic SQL

5. Currently committed data
   - Business driver, Access, Interaction with other zparms, Comparison and recommendations

A summary of the features we discussed.
I trust this session has empowered you with the knowledge to exploit the new application features of DB2 10. Good Luck!
Some of the useful references. At the time of writing this, anxiously waiting a good Redbook...

<table>
<thead>
<tr>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. DB2 10 for z/OS SQL Reference</td>
</tr>
<tr>
<td>2. DB2 10 for z/OS Application Programming and SQL Guide</td>
</tr>
<tr>
<td>4. Various IRM presentations available on public websites</td>
</tr>
</tbody>
</table>
Thank you and good luck with DB2 10!