What’s New in the TEMPORAL World of Db2

Steen Rasmussen, Principal Engineering Services Architect

CA technologies A Broadcom Company
What’s New in the TEMPORAL World of Db2

Steen Rasmussen

CA technologies A Broadcom Company

Session code: F05
06 November 2018  10:20 – 11:20

Db2 for z/OS
The Temporal Table design was introduced in Db2 10 and there were quite a few restrictions. Some of these were lifted in Db2 11 and Db2 12 has lifted more – but the temporal design has really started to be adopted by Db2 itself, so we will cover the restrictions removed using live examples as well as a detailed walkthrough of how the Db2 engine itself is starting to offer this functionality and predictions for the future.
Temporal Basics
Db2 10 for z/OS: Temporal Tables Introduced

Three different types of Temporal tables offered in DB2 10

- **SYSTEM TIME TEMPORAL**
  - Controlled by DB2 100%
  - History table get’s “old” data while current data is in base
  - Good approach to see how data has changed over time

- **BUSINESS TIME TEMPORAL**
  - Controlled by the user/application
  - No history table – date-range-columns describe when data is “valid”
  - Good approach to see the status of a row at a specific point-in-time

- **BI-TEMPORAL** = combination of the two

Db2 supports three different types of temporal tables:

1) System Time where a history table is associated to a base table. Upon UPDATES and DELETES, the base row is inserted into the history table along with a start/end timestamp when the row existed in the base table. The application has NO influence on the start/end time which is controlled by Db2 100%.

2) Business Time where the history is kept in the base table. The start/end date/time is 100% controlled by the application and SQL controls the start/end time by using FOR PORTION OF in the SQL.

3) BI Temporal which is a hybrid of the two other types.
Db2 10 for z/OS: Temporal Tables Introduced

• Tables need additional columns to enable time travel queries

```
ALTER TABLE RASST02.STEEN_LIFE
  ADD ROW_START TIMESTAMP (12) NOT NULL GENERATED ALWAYS AS ROW BEGIN;
ALTER TABLE RASST02.STEEN_LIFE
  ADD ROW_END TIMESTAMP (12) NOT NULL GENERATED AS ROW END;
ALTER TABLE RASST02.STEEN_LIFE
  ADD RW_ID TIMESTAMP (12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID;
ALTER TABLE RASST02.STEEN_LIFE
  ADD PERIOD SYSTEM_TIME(ROW_START, ROW_END);

ALTER TABLE STEEN_LIFE ADD VERSIONING USE HISTORY TABLE STEEN_LIFE_H;

DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

• For SYSTEM TIME Temporal table – create LIKE table and associate these two tables

Changing a table (or creating a new table) to be the BASE for a Business Time temporal table requires four additional columns.
The HISTORY table has to be 100% identical – and the temporal design is activated by executing an ALTER of the base ADDING VERSIONING by specifying the HISTORY table.
**Updates to BASE table -> previous data inserted into HISTORY**

```sql
INSERT INTO STEEN_LIFE
( LASTNAME , SSN , STREET , HOUSENUMBER , CITY , ZIP)
VALUES ( 'RASMUSSEN' , '1304610367' , 'VANLOSE ALLE' , '8 ST TH' , 'VANLOSE' , '1340' ) ;
```

<table>
<thead>
<tr>
<th>FIELD</th>
<th>BASE (ROW_ID)</th>
<th>HISTORY (ROW_ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRSTNAME</td>
<td>null</td>
<td>STEEN</td>
</tr>
<tr>
<td>STREET</td>
<td>VANLOSE ALLE</td>
<td>VANLOSE ALLE</td>
</tr>
<tr>
<td>ROW_END</td>
<td>1961-12-30-00.00.00.000000000000</td>
<td>1961-04-14-15.12.46.843716604000</td>
</tr>
</tbody>
</table>

```sql
UPDATE STEEN_LIFE SET FIRSTNAME = 'STEEN' WHERE SSN='1304610367' ;
```

<table>
<thead>
<tr>
<th>FIELD</th>
<th>BASE (ROW_ID)</th>
<th>HISTORY (ROW_ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRSTNAME</td>
<td>STEEN</td>
<td>null</td>
</tr>
<tr>
<td>STREET</td>
<td>VANLOSE ALLE</td>
<td>VANLOSE ALLE</td>
</tr>
<tr>
<td>ROW_START</td>
<td>1961-06-22-07.06.10.71886777700</td>
<td>1961-04-14-15.12.46.843716604000</td>
</tr>
<tr>
<td>ROW_END</td>
<td>1961-12-30-00.00.00.000000000000</td>
<td>1961-04-14-15.12.46.843716604000</td>
</tr>
</tbody>
</table>

Once a row in the base table is updated, the old values are inserted into the HISTORY table. The END timestamp of the row in the history table is the time the row was updated in the base table – so basically the start/end timestamp in the history table is when the row was in the base table.
Db2 10 for z/OS: Temporal Tables Introduced

- Methods to retrieve data (using the temporal idea)
  - Of course SELECT is possible from both like “normal tables”
  - AS OF from BASE: might retrieve data from BASE or HISTORY

```
SELECT * FROM STEEN_LIFE FOR SYSTEM_TIME AS OF '1961-05-09-00.00.000000000000'
WHERE SSN='1304610367';
```

- Data returned based on START-END timestamp matching **AS OF** criteria

<table>
<thead>
<tr>
<th>FIRSTNAME</th>
<th>STEEN_LIFE</th>
<th>STEEN_LIFE_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREET</td>
<td>VANLOSE ALLE</td>
<td>VANLOSE ALLE</td>
</tr>
<tr>
<td>ROW_START</td>
<td>1961-06-22-07.06.10.718867777000</td>
<td>1961-04-14-15.12.46.843716604000</td>
</tr>
<tr>
<td>ROW_END</td>
<td>1999-12-30-00.00.000000000000</td>
<td>1961-06-22-07.06.10.718867777000</td>
</tr>
<tr>
<td>RW_ID</td>
<td>1961-06-22-07.06.10.718867777000</td>
<td>1961-04-14-15.12.46.843716604000</td>
</tr>
</tbody>
</table>

For sure you can retrieve rows from both the base table and the history table by using “normal” SQL on each table.
The savings come into play for the application programmer when the SQL Temporal extensions are used.
Basically the SQL will use AS OF TIMESTAMP or the like, and Db2 will “UNION” the two tables and retrieve zero, one or more rows depending on the predicate.
An interesting “gotcha”

- No commit’s between updates to base - NO HISTORY
- Db2 must consider this ONE UOW (this is Db2 10 . . . . .)

```
INSERT INTO STEEN_LIFE2 (LASTNAME, SSN, STREET, HOUSENUMBER, CITY, ZIP)
VALUES ('RASMUSSEN', '1304610367', 'VANLOSE ALLE', '8 ST TH', 'VANLOSE', '1340');
UPDATE STEEN_LIFE2 SET FIRSTNAME = 'STEEN' WHERE SSN='1304610367';
UPDATE STEEN_LIFE2 SET STREET = 'RAVNAGER', HOUSENUMBER = '22', ZIP='2600'
, CITY='GLOSTRUP' WHERE SSN='1304610367';
UPDATE STEEN_LIFE2 SET STREET = 'STOREBJERG', HOUSENUMBER = '63', ZIP='2670', CITY='GREVE'
WHERE SSN='1304610367';
COMMIT;
SELECT * FROM STEEN_LIFE2_H ;
DSNT404I SQLCODE = 100, NOT FOUND: ROW NOT FOUND FOR FETCH, UPDATE, OR DELETE, OR THE RESULT OF A QUERY IS AN EMPTY TABLE
```

One very interesting scenario the application has to be aware of is when rows are inserted into the history table. If the BASE table are updated multiple times within one commit scope, you will only get the latest update.

The scenario outlined here – does an INSERT into the BASE table followed by multiple updates, but since it’s within one UOW, the history table isn’t impacted.

Hold your thoughts a few slides since this can change in Db2 12 if you so want to.
Db2 10 for z/OS: Temporal Tables Introduced

- Views can be created referencing TEMPORAL tables
- BUT . . . . Views cannot be used with TEMPORAL SQL extensions – only difference is that we reference a VIEW and not the BASE table

```sql
CREATE VIEW TEMPORAL_V AS SELECT * FROM STEEN_LIFE2 ;
COMMIT ;
```

- DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
- SELECT * FROM TEMPORAL_V
- FOR SYSTEM_TIME AS OF '2012-06-27-07.06.10.000000000000'
- WHERE SSN='1304610367' ;
- DSNT408I SQLCODE = -20524, ERROR: INVALID PERIOD SPECIFICATION OR PERIOD CLAUSE FOR PERIOD SYSTEM_TIME. REASON CODE = 04

-20524 INVALID PERIOD SPECIFICATION OR PERIOD CLAUSE FOR PERIOD period-name.
REASON CODE = reason-code.
Explanation: A period specification or period clause is The period specification or period clause was specified for a view.

A lot of shops have a mandate to use views for various reasons. This does open up an issue in Db2 10 since views cannot be used, but are not supported for TEMPORAL TIME TRAVELLING.
Index design is different for BUSINESS time compared to SYSTEM time since HISTORY is built into BASE.

Appending WITHOUT OVERLAPS on a UNIQUE index:

- Allows the index-columns to remain UNIQUE within timeframe of BUSINESS_TIME.

```sql
CREATE UNIQUE INDEX RASST02.STEEN_LIFE3_IX
ON RASST02.STEEN_LIFE3
( SSN ASC, BUSINESS_TIME WITHOUT OVERLAPS )
```

<table>
<thead>
<tr>
<th>#1</th>
<th>SSN</th>
<th>#2</th>
<th>STA_DATE</th>
<th>END_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEEN</td>
<td>1304610367</td>
<td>GRONNEGAARDEN</td>
<td>1985-07-01</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>STEEN</td>
<td>1304610367</td>
<td>VANLOSE ALLE</td>
<td>1961-04-13</td>
<td>1985-07-01</td>
</tr>
</tbody>
</table>

Last section related to the initial delivery of Temporal tables in Db2 10 is related to schema changes.

The index design might have to be considered when UNIQUE indexes exist. For HISTORY tables the same column-value can exist multiple times. For example, consider SSN is a unique value, but when one SSN can be updated multiple times in the life of the row, you will have to define these indexes on the history table as duplicate.

The same goes for BUSINESS TIME and BI-TEMPORAL where history rows co-exist in the base table. In order to retain UNIQUENESS you will have to use WITHOUT OVERLAPS – meaning the row can exist multiple times but is unique within one date/time period.
Db2 10 for z/OS: Temporal Tables Introduced

- Table DDL must be identical -> schema changes difficult
  - RENAME COLUMN, Column attribute changes etc. not supported
  - ADD COLUMN is allowed on BASE table but not on HISTORY
- DB2 automatically ADDs the new column to the HISTORY
  - For TEMPORAL to adopt other BASE changes
  - Drop VERSIONING
  - Alter BASE and HISTORY
  - Establish VERSIONING
  - (think about data - load back or insert) - and remember to retain START/END timestamps prior to re-loading data)

```
ALTER TABLE STEEN_LIFE2 DROP VERSIONING
DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

Implementing schema changes for temporal tables can be a challenge – basically the only schema change which is easy is adding a new column to the BASE table and Db2 will add this column to the history as well.

For all other schema changes you will have to remove VERSIONING – perform the schema changes for both base and history and then ADD VERSIONING back

If the schema change involves Unload-Drop-Create-Load, please remember to retain the START/END timestamps using PERIODOVERRIDE in the load syntax.
Changes to Include / Exclude / Include
When is a row considered part of the temporal period?

• Basically FOUR different options when retrieving rows for START/END system time *(using AS OF , BETWEEN , FROM-TO)*
  1) EXCLUSIVE – EXCLUSIVE : not supported natively
  2) EXCLUSIVE – INCLUSIVE : not supported natively
  3) INCLUSIVE – EXCLUSIVE : supported in Db2 10 natively
  4) INCLUSIVE – INCLUSIVE : requires Db2 12 natively

• You could retrieve rows by coding SELECT/UPDATE again BASE and HISTORY *(kind of defeats the purpose)*

• Method is NOT embedded in SQL - but DDL

```
ROW_START TIMESTAMP (12),
ROW_END    TIMESTAMP (12),
ROW_ID        TIMESTAMP (12),
PERIOD          SYSTEM_TIME
(ROW_START , ROW_END INCLUSIVE)
```

You will have to make sure you understand how SQL retrieval works with the temporal start/end date/timestamp.

Basically there are for combinations, but Db2 10 only supported INCLUSIVE – EXCLUSIVE, so if you are using END DATE and expects to get rows where the END-DATE is equal to what you specified in the SQL, then you are fooled.

Db2 12 has an option to use INCLUSIVE-INCLUSIVE, but the “interesting issue” is that it’s not controlled by SQL but the DDL – BE AWARE !!!!
Referential Integrity - how does it all work
Referential Integrity

- The initial delivery of Temporal Tables in Db2 10 didn’t allow RI
- Db2 12 delivered this capability for Business Time Temporal Tables
  - Also know as Application Controlled Temporal Tables
  - History information in the Base Table (unlike System Time)
  - System Time Temporal not supported

- More complex than “regular RI”
  - Remember HISTORY and CURRENT data is in the same table !!!!
- Cyclical RI on Business Time temporal NOT supported
- DELETE RULE can only be RESTRICT (SET NULL not supported)
- UPDATE/DELETE using FOR PORTION OF not supported

Another initial restriction was that RI wasn’t allowed on a table participating in a Temporal design. This restriction has been lifted for BUSINESS TIME TEMPORAL tables ONLY.

First view into this makes you scratch your head since the implementation seems more complex than “normal RI”.

There are several restrictions for what can be done using SQL, so let’s take a closer look at some examples to get a better understanding of the moving pieces.
Deprecated “Feature”

- **Triggers**
  - Some customers used triggers to “implement RI”
  - Deprecated in Db2 11 NFM


In DB2 V10 and DB2 V11 conversion mode, there was no problem. It was possible to define a trigger referencing a SYSTEM TEMPORAL or BI-TEMPORAL table in the WHEN clause. Since we are in DB2 V11, we receive an SQLCODE=-270 each time that the package trigger is rebound. In DB2 V11, there is a procedure to circumvent this issue, we have to alter the BI-TEMPORAL table to DROP the VERSIONING, DROP/CREATE the trigger, add the VERSIONING and finally REBIND the TRIGGER PACKAGE with bind option SYSTIMESENSITIVE NO. Even if this procedure is less obvious than before, it works. According to IBM, in DB2 V12, this solution will not work anymore and we will have to review/drop the triggers that are impacted and implement the RI in the programs that are updating the tables. This solution will work too but the RI will not be verified if the update is done via spufl, datastudio, ... With triggers it was also verified in such cases. This solution is than a lot less secured, and more difficult to maintain (one TRIGGER per type of modification vs multiple programs doing insert/update/delete).
Referential Integrity

- New syntax needed for Index AND Primary Key:

```sql
CREATE  TABLE RASST02.STEEN_LIFE3
  ( FIRSTNAME VARCHAR ( 30 )
  , LASTNAME VARCHAR ( 30 )
  , SSN CHAR ( 10 ) NOT NULL
  , STREET VARCHAR ( 30 )
  , HOUSENUMBER CHAR ( 10 )
  , CITY CHAR ( 20 )
  , ZIP CHAR ( 6 )
  , STA_DATE DATE NOT NULL WITH DEFAULT
  , END_DATE DATE NOT NULL WITH DEFAULT
  , PERIOD BUSINESS_TIME(STA_DATE , END_DATE) )
IN IDUGNA18.TSSYB18 ;

CREATE UNIQUE INDEX STEEN_LIFE3_IX ON STEEN_LIFE3
  (SSN , BUSINESS_TIME WITHOUT OVERLAPS) ;

ALTER  TABLE RASST02.STEEN_LIFE3
ADD PRIMARY KEY (SSN , BUSINESS_TIME WITHOUT OVERLAPS) ;

DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

Makes a lot of sense since history resides in the same tables - have to ensure uniqueness

Business / Application Temporal table created and unique index WITHOUT OVERLAPS created since history rows reside in the same table unlike System Time Temporal tables. Finally the PK is added – also using WITHOUT OVERLAPS – just like the supporting index.
Child table is created and several components need to be generated for the RI to be accepted by Db2.

1) In this case the child table also has a primary key: ACCTVALUE with Business Time WITHOUT OVERLAPS

2) The foreign key is pointing to the table on the previous slide – also BUSINESS TIME.

3) Unique index defined for the local PK.

4) Remember to create an index on the FOREIGN KEY – WITH OVERLAPS since you can have multiple children pointing to the PK table.
Referential Integrity

- Child table creation and association with parent table:

```sql
INSERT INTO STEEN_LIFE3
(LASTNAME, SSN, STREET, HOUSENUMBER, CITY, ZIP, STA_DATE, END_DATE)
VALUES ('RASMUSSEN', '1304610367', 'VANLOSE ALLE', '8 ST TH',
        'VANLOSE', '1340', '1961-04-13', '9999-12-31');

INSERT INTO STEEN_ACCOUNTS
VALUES ('1304610367', '00001111', 10000, '1961-04-14', '9999-12-31');
```

For Table => RASST02.STEEN_ACCOUNTS
            > Row number=> 1 OF 1
Browse Mode => C > Max Char => 070
SSID: D12A

---
FETCH STATUS: COMPLETE--------------------------

<table>
<thead>
<tr>
<th>CHILD_SSN</th>
<th>ACCTNO</th>
<th>ACCTVALUE</th>
<th>CHILD_START</th>
<th>CHILD_END</th>
</tr>
</thead>
<tbody>
<tr>
<td>1304610367</td>
<td>00001111</td>
<td>10,000.00</td>
<td>1961-04-14</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

First child row inserted – start/end dates have to be contained in the Parent DATE-RANGE

Last task to do is to test if it works 😊

1) A row is inserted into the parent table
2) A row is inserted into the child table with the same SSN
3) Everything looks beautiful and as expected.
Referential Integrity

- Be aware NULL value for Foreign Key doesn’t validate constraint

```sql
INSERT INTO STEEN_ACCOUNTS
VALUES ('1304610367', 0001111, 500, '1960-03-31', '1960-04-03');
```

DSNT408I SQLCODE = -803, ERROR: AN INSERTED OR UPDATED VALUE IS INVALID BECAUSE INDEX IN INDEX SPACE STEENRAC CONSTRAINTS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS.

RID OF EXISTING ROW IS X'0000000202'.

DSNT418I SQLSTATE = 23505 SQLSTATE RETURN CODE

```sql
INSERT INTO STEEN_ACCOUNTS
VALUES (NULL, 00001111, 500, '1960-03-31', '1960-04-03');
```

DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION

- Note that START-DATE is outside the Parent Table Row – but NULL value for foreign-key column causes constraint check to be bypassed (in this case unique index on account-value).

Another interesting scenario is that duplicate keys are not allowed – unless the PK value is NULL
Referential Integrity

• Rules to obey:
  1) Child rows start/end date range MUST be within Parent start-end range.
  2) Can not update parent rows where child rows have dependent START/END dates unless child rows stay within start-end date range.
  3) Like “normal” temporal tables : UPDATE FOR PORTION OF can cause multiple rows to be inserted/updated.

```sql
UPDATE RASST02.STEEN_LIFE3
SET END_DATE = '1995-10-13' ;
```

```sql
DSNT408I SQLCODE = -531, ERROR: PARENT KEY IN A PARENT ROW CANNOT BE UPDATED BECAUSE IT HAS ONE OR MORE DEPENDENT ROWS IN RELATIONSHIP CHILD$SS
DSNT418I SQLSTATE = 23504 SQLSTATE RETURN CODE

Child row has END_DATE '9999-12-31' (more next page). In this case, one row is violated.
```

There are some very specific “rules” to obey for Temporal RI:

You cant update PARENT rows when child rows will be impacted in terms of “date validity”
Referential Integrity

• Rules to obey:

Once Parent table is updated with an END_DATE which is OUTSIDE Child row, the UPDATE works. If END_DATE is set to a for example 2000-01-01, the last row’s END_DATE is now outside the parent and update will fail.

Once the PARENT table is updated to a date range NOT violating the child rows – everything is good.
Temporal Referential Integrity

- More complex than “regular RI”
  - Remember both CURRENT and HISTORY data is in the same table !!!!

- The Business time of the Child row must be fully covered by the parent table
  - May be multiple rows in Parent provided there are no gaps in the Timeline (DB2 checks this)

- For Inclusive – Inclusive the smallest time unit is used for gap checking
  - 1 Day for DATE data type Period columns
  - 1 Microsecond for TIMESTAMP(6) Period columns
Auditing Extension to System Time Temporal Tables
ON DELETE ADD EXTRA ROW

- UPDATE operations adds “old image” to the associated history table
  - DELETE operation is not recorded
  - Who did the update/delete/insert and what caused the row to be inserted into HISTORY

- New syntax when HISTORY table associated:
  
  ```sql
  ALTER TABLE c.t ADD VERSIONING USE HISTORY TABLE c.t_hist ON DELETE ADD EXTRA ROW
  ```

- Two new columns / expressions can be added to track operations
  - USER_ID VARCHAR(128) GENERATED ALWAYS AS ( SESSION_USER )
  - SQL_OPERATION CHAR(1) GENERATED ALWAYS AS ( DATA CHANGE OPERATION )
  - Available in DB2 11 via APARs in Service Stream (PM99683, PI15298 & PI15666)
Other Changes since Db2 10
Temporal Special Registers and associated BIND options

- SET CURRENT TEMPORAL BUSINESS TIME & SET CURRENT TEMPORAL SYSTEM TIME
- If set - DB2 acts as if the respective clause was added to each SQL statement
  - Without the need to modify the SQL
- Package Sensitivity to these provided by new BIND options
  - SYSTIMESSENSITIVE
  - BUSTIMESSENSITIVE
- Available in DB2 11
Deprecated “Feature”

- Triggers
  - Some customers used triggers to “implement RI”
  - Deprecated in Db2 11 NFM

In DB2 V10 and DB2 V11 conversion mode, there was no problem. It was possible to define a trigger referencing a SYSTEM TEMPORAL or BI-TEMPORAL table in the WHEN clause. Since we are in DB2 V11, we receive an SQLCODE=-270 each time that the package trigger is rebound. In DB2 V11, there is a procedure to circumvent this issue, we have to alter the BI-TEMPORAL table to DROP the VERSIONING, DROP/CREATE the trigger, add the VERSIONING and finally REBIND the TRIGGER PACKAGE with bind option SYSTIMESENSITIVE NO. Even if this procedure is less obvious than before, it works. According to IBM, in DB2 V12, this solution will not work anymore and we will have to review/drop the triggers that are impacted and implement the RI in the programs that are updating the tables. This solution will work too but the RI will not be verified if the update is done via spufl, datastudio, ... With triggers it was also verified in such cases. This solution is than a lot less secured, and more difficult to maintain (one TRIGGER per type of modification vs multiple programs doing insert/update/delete).
VIEWS

- Db2 10 allows views to reference tables in a temporal design
- Could NOT use Views for Time Travel Queries.
- Many shops have a standard in place to use views
  - Application based views
  - “old” standards
  - . . . .
- Db2 12 introduced the ability to use views for Time Travel Queries
  - Retrofitted to Db2 11

Remember back in the Db2 10 days – views were NOT allowed to update temporal tables – this restriction has been lifted !!!
Logical / Physical Transactions: System Time Temporal tables

- Db2 10 only supported physical
  - Remember the earlier slide “interesting gotcha” below
    
    ```sql
    INSERT INTO STEEN_LIFE2 (LASTNAME, SSN, STREET, HOUSENUMBER, CITY, ZIP)
    VALUES ('RASMUSSEN', '1304610367', 'VANLOSE ALLE', '8 ST TH', 'VANLOSE', '1340');
    UPDATE STEEN_LIFE2 SET FIRSTNAME = 'STEEN' WHERE SSN='1304610367';
    UPDATE STEEN_LIFE2 SET STREET = 'RAVNAGER', HOUSENUMBER = '22', ZIP='2600', CITY='GLOSTRUP' WHERE SSN='1304610367';
    UPDATE STEEN_LIFE2 SET STREET = 'STOREBJERG', HOUSENUMBER = '63', ZIP='2670', CITY='GREVE' WHERE SSN='1304610367';
    COMMIT;
    SELECT * FROM STEEN_LIFE2_H;
    DSNT404I SQLCODE = 100, NOT FOUND: ROW NOT FOUND FOR FETCH, UPDATE, OR DELETE, OR THE RESULT OF A QUERY IS AN EMPTY TABLE
    ```

  - Controlled 100% by Db2
  - Controlled when commit executed using time-of-execution

Another “interesting” piece in Db2 10 was the fact that the history wasn’t being maintained for System Time Temporal Tables when multiple transactions/updates were made in one UOW – this has been changed – if you so want so !!

Move on to the next slide
Logical / Physical Transactions : System Time Temporal tables

- Db2 12 introduced Logical
  - New global variable : SYSIBM.TEMPORAL_LOGICAL_TRANSACTION_TIME
    - Default value for TS(12) column is NULL (Db2 10 behavior)
  - New global variable : SYSIBM.TEMPORAL_LOGICAL_TRANSACTIONS
    - Value 0 / 1 : don’t / do allow more than one logical transaction in one commit-scope
  - System START/END timestamp must append :
    - IMPLICITLY HIDDEN GENERATED ALWAYS AS ROW BEGIN/END
  - Multiple UPDATES within one commit will get different start/end based on :
    - SET TEMPORAL_LOGICAL_TRANSACTION_TIME = CURRENT_TIMESTAMP
    - Multiple INSERTS into HISTORY table
    - Limitations do exist like : two concurrent executing transactions – one can not set transaction time to what another transaction already did

There’s a new option to optionally have every update within a UOW to act as a separate transaction. However, concurrent transaction processing is an issue which has to be taken into the application design.

A new Global Variable has been introduced to either keep the old behavior or adopt the new one where every “logical transaction is treated as a physical one – causing history rows to be generated.

In order to use this new feature the start/end timestamp columns must be appended with IMPLICITLY HIDDEN.
RTS Autonomics - no more need to offload RTS ?
System Time Temporal available for RTS

- Optional to enable history of RTS tables.
  - Two new tablespaces and tables
    
    | TABLE_NAME                | CREATOR  | DATABASE | TBLSPACE |
    |---------------------------|----------|----------|----------|
    | SYSIXSPACESTATS_H         | SYSIBM   | DSNDB06  | SYSTSISH |
    | SYSTABSPACESTATS_H        | SYSIBM   | DSNDB06  | SYSTSTSH |

- History table name length kept less than 18 byte (unfortunately ?)

```sql
ALTER TABLE SYSIBM.SYSINDEXSPACE
ADD VERSIONING USE HISTORY TABLE SYSIBM.SYSIXSPACESTATS_H;

ALTER TABLE SYSIBM.SYSTABLESPACE
ADD VERSIONING USE HISTORY TABLE SYSIBM.SYSTABSPACESTATS_H;
```

DB2 10 introduced three types of Temporal Tables – and now DB2 12 offers the same functionality for two catalog tables: The RTS (Real Time Statistics) tables.

This is an optional feature not being mandated.

In order to activate SYSTEM TIME temporal tables for the RTS tables, two simple ALTER commands are needed (described in the SQL REFERENCE GUIDE).

From my opinion, I would have preferred to simply add “ _H " to the RTS tables for the history tables instead of shortening them.
System Time Temporal available for RTS

- Many Db2 users are offloading RTS on a periodic basis:
  - Once a day before housekeeping processes (Copy, Reorg etc.)
  - Selective objects prior to REORG’ing or other activity for the specific object
- What is the resource consumption when RTS tables updated by the timer?
  - Try it out when “things are quiet” on a Sunday
  - Bigger Db2 job than externalizing in-memory RTS metrics?
    - Twice the #INSERTS since START/END timestamp needs to be modified in history
    - Newly externalized rows into RTS tables
    - BUT . . . Nobody have reported RTS externalization performance issues (AFAIK)
- Make sure the System Time History Table can grow
Catalog Preparation for Autonomic Auditing
System Time Temporal – future planning?

• What is IBM planning? Maybe this provides a clue?!

• Catalog Table changes in the past has not very often indicated what we can expect in the future – DB2 12 may be different

• We have had history tables for several releases covering statistics history:
  SYSIBM.SYSxxxxxx_HIST

We just mentioned RTS can be enabled with system time temporal tables, but what’s quite interesting is to have a closer look at which tablespaces and tables exist in the DB2 12 catalog but are not referenced in the SQL Reference Guide’s Appendix A section describing the catalog tables.

This is probably the first time that the DB2 catalog tables are prepared for the future and kind of eludes what to expect.

The _HIST tables have existed for many releases providing the ability to store historical data for RUNSTATS. There are a complete new set of DIFFERENT history tables – let’s have a closer look.
System Time Temporal – future planning?

- One common component for these OLD and NEW objects – all have the same columns appended (see next slide).

<table>
<thead>
<tr>
<th>Tablespace</th>
<th>Table</th>
<th>AUX</th>
<th>Suggested Base Table</th>
<th>Suggested Base Tablespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTAH</td>
<td>SYSAUDITPOLICIES_H</td>
<td></td>
<td>SYSAUDITPOLICIES</td>
<td>SYSTSADT</td>
</tr>
<tr>
<td>SYSSTFAH</td>
<td>SYSCOLAUTH_H</td>
<td></td>
<td>SYSCOLAUTH</td>
<td>SYSTSFAU</td>
</tr>
<tr>
<td>SYSSTCH</td>
<td>SYSCONTTEXTAUTHID_H</td>
<td></td>
<td>SYSCONTTEXTAUTHIDS</td>
<td>SYSCONTX</td>
</tr>
<tr>
<td>SYSSTCNH</td>
<td>SYSCONTTEXT_H</td>
<td></td>
<td>SYSCONTTEXT</td>
<td>SYSTSCHN</td>
</tr>
<tr>
<td>SYSSTCOTH</td>
<td>SYSCONTROLS_H</td>
<td></td>
<td>SYSCONTROLS</td>
<td>SYSTSCOTH</td>
</tr>
<tr>
<td>SYSSTCDO</td>
<td>SYSCONTROLS_DESC_H</td>
<td>X</td>
<td>SYSCONTROLS_DESC</td>
<td>SYSTSCDO</td>
</tr>
<tr>
<td>SYSSTCRO</td>
<td>SYSCONTROLS_RTXT_H</td>
<td>X</td>
<td>SYSCONTROLS_RTXT</td>
<td>SYSTSCRO</td>
</tr>
<tr>
<td>SYSSTSAH</td>
<td>SYSCTXTTRUSTATTRIB_H</td>
<td></td>
<td>SYSCTXTTRUSTATTRIBS</td>
<td>SYSCONTX</td>
</tr>
<tr>
<td>SYSSTDDBH</td>
<td>SYSDBAUTH_H</td>
<td></td>
<td>SYSDBAUTH</td>
<td>SYSTDDBH</td>
</tr>
<tr>
<td>SYSSTPKH</td>
<td>SYSPACKAUTH_H</td>
<td></td>
<td>SYSPACKAUTH</td>
<td>SYSTSPKA</td>
</tr>
<tr>
<td>SYSSTPLH</td>
<td>SYSPLANAUTH_H</td>
<td></td>
<td>SYSPLANAUTH</td>
<td>SYSTSPLA</td>
</tr>
<tr>
<td>SYSSTSAUH</td>
<td>SYSRESAUTH_H</td>
<td></td>
<td>SYSRESAUTH</td>
<td>SYGSTSAU</td>
</tr>
<tr>
<td>SYSSTRAH</td>
<td>SYSRoutineAUTH_H</td>
<td></td>
<td>SYSRoutineAUTH</td>
<td>SYSTRAU</td>
</tr>
<tr>
<td>SYSSTSCIH</td>
<td>SYSSCHEMAAUTH_H</td>
<td></td>
<td>SYSSCHEMAAUTH</td>
<td>SYSTSCIC</td>
</tr>
<tr>
<td>SYSSTSCAH</td>
<td>SYSSEQUENCEAUTH_H</td>
<td></td>
<td>SYSSEQUENCEAUTH</td>
<td>SYSTSEQ2</td>
</tr>
<tr>
<td>SYSSTSBH</td>
<td>SYSTBAUTH_H</td>
<td></td>
<td>SYSTBAUTH</td>
<td>SYSTSBH</td>
</tr>
<tr>
<td>SYSSTSAH</td>
<td>SYSUSERAUTH_H</td>
<td></td>
<td>SYSUSERAUTH</td>
<td>SYSTSAU</td>
</tr>
<tr>
<td>SYSSTVDH</td>
<td>SYSVIEWDEP_H</td>
<td></td>
<td>SYSVIEWDEP</td>
<td>SYSTSVWD</td>
</tr>
</tbody>
</table>

To me these tables seem to indicate that we can expect all the tables holding authorizations (GRANT’s) to be temporal as well – they all have a couple of attributes in common (please see next page).
System Time Temporal – future planning?

- These appended columns could indicate a potential enablement of System Time Temporal Table (just like the RTS tables can be enabled in DB2 12)

```
CREATE TABLE "SYSIBM"."SYSSEQUENCEAUTH"
    (GRANTOR VARCHAR(128) FOR MIXED DATA NOT NULL,
     GRANTEE VARCHAR(128) FOR MIXED DATA NOT NULL,
     "SCHEMA" VARCHAR(128) FOR MIXED DATA NOT NULL,
     "NAME" VARCHAR(128) FOR MIXED DATA NOT NULL,
     GRANTEETYPE CHARACTER(1) FOR MIXED DATA NOT NULL,
     AUTHHOWGOT CHARACTER(1) FOR MIXED DATA NOT NULL,
     USEAUTH CHARACTER(1) FOR MIXED DATA NOT NULL,
     "COLLID" VARCHAR(128) FOR MIXED DATA NOT NULL,
     CONTOKEN CHARACTER(8) FOR BIT DATA NOT NULL,
     GRANTEDTS TIMESTAMP (6) WITHOUT TIME ZONE NOT NULL,
     IBMREQD CHARACTER(1) FOR MIXED DATA NOT NULL,
     GRANTORTYPE CHARACTER(1) FOR MIXED DATA NOT NULL,
     SYS_START TIMESTAMP (12) WITHOUT TIME ZONE NOT NULL,
     SYS_END TIMESTAMP (12) WITHOUT TIME ZONE NOT NULL,
     TRANS_START TIMESTAMP (12) WITHOUT TIME ZONE NOT NULL,
     PERIOD SYSTEM_TIME
    )
IN DSNDB06.SYSSEQ2
```

All these tables have had the special columns/attributes appended which are needed to turn them into temporal tables. However – if you try to enable history, DB2 will block you from doing so.
System Time Temporal – future planning?
• Another view of “preparation for the future”
• System Time changes – but not enabled to history tables.

```
-- SYSXXX.SYSAUDITPOLICIES WILL BE ALTERED VIA THESE NATIVE DB2 COMMANDS
ALTER TABLE SYSXXX.SYSAUDITPOLICIES
   ADD SYS_START TIMESTAMP (12)
      NOT NULL GENERATED ALWAYS AS ROW BEGIN;

ALTER TABLE SYSXXX.SYSAUDITPOLICIES
   ADD SYS_END TIMESTAMP (12)
      NOT NULL GENERATED ALWAYS AS ROW END;

ALTER TABLE SYSXXX.SYSAUDITPOLICIES
   ADD TRANS_START TIMESTAMP (12)
      NOT NULL GENERATED ALWAYS AS TRANSACTION START ID;

ALTER TABLE SYSXXX.SYSAUDITPOLICIES
   ADD PERIOD SYSTEM_TIME (SYS_START, SYS_END);
```

Doing a schema compare between these tables in DB2 11 and DB2 12 illustrated that the DB2 upgrade process has appended the needed columns to the base catalog tables.
System Time Temporal – future planning?

- Maybe we are getting ready to get an “autonomic audit trail” of GRANT/REVOKE
- If so – changes are required for some catalog objects.
  - SYSSEQUENCEAUTH sits in a tablespace with TWO tables
  - VCAT defined tablespace – not yet converted to PBG (OK but requires one table-tablespace)

My personal opinion is we’re getting prepared to provide an automatic method to audit who is granting/revoking which auth’s from who.

If this is going to happen, there are a couple of schema changes needed since temporal tables have to reside in single table-tablespaces so violations do exist:

1. VCAT defined tablespace not yet converted to PBG
2. Some tablespaces have more than one table.
Questions and/or Comments Please
Steen Rasmussen
CA technologies
steen.rasmussen@broadcom.com  or
DB2steen@yahoo.com

Session code: