



Time Travel Queries and Temporal Data Storage - What you need to know!

Daniel L Luksetich
DanL Database Consulting
danl@db2expert.com
WWW.DB2EXPERT.COM

Dan Luksetich is a senior DB2 DBA consultant. He works as a DBA, application architect, presenter, author, and teacher. Dan has been in the information technology business for over 30 years, and has worked with DB2 for over 25 years. He has been a COBOL and BAL programmer, DB2 system programmer, DB2 DBA, and DB2 application architect. His experience includes major implementations on z/OS, AIX, i Series, and Linux environments.

Dan's experience includes:

Application design and architecture

Database administration

Complex SQL

SQL tuning

DB2 performance audits

Replication

Disaster recovery

Stored procedures, UDFs, and triggers

Dan works everyday on some of the largest and most complex DB2 implementations in the world. He is a certified DB2 DBA, system administrator, and application developer, and has work on the teams that have developed the DB2 for z/OS certification exams. He is the author of several DB2 related articles as well as co-author of the DB2 9 for z/OS Certification Guide and the DB2 10 for z/OS Certification Guide.



IDUG DB2 North America Tech Conference
Philadelphia, Pennsylvania | May 2015

#IDUGDB2

International DB2 Users Group

- INDEPENDENT User Group Founded in 1988
- Web site – IDUG.ORG
- Annual Conferences
 - North American in May
 - EMEA in November
 - Australasia in September
 - India



IDUG DB2 North America Tech Conference
Philadelphia, Pennsylvania | May 2015

#IDUGDB2

IDUG Content Committee

- Dedicated to supporting the IDUG and DB2 user community by providing valuable free content
 - IDUG web site
 - Conferences
- Web Site
 - DB2 news blog
 - DB2 beginner's blog
 - IDUG content blog
 - IDUG content file library
 - Videos
 - IDUG Tech Talk Channel
 - YouTube

Agenda



- The Need for Temporal Data Storage
- The Advantage of Using DB2 for Temporal Data Storage
- Overview of the Types of Automated Temporal Tables
 - Application-Period Temporal Tables
 - System-Period Temporal Tables
 - Bi-Temporal Tables
- Implications for Referential Constraints when using Temporal Tables
- Temporal Data Access
 - Updating Temporal Tables
 - Application-Period
 - System-Period
 - Querying Temporal Tables
 - Application-Period
 - System-Period

The Need for Temporal Storage



- Over the years the requirements for maintaining time based data has increased
- Several factors are contributing to this
 - Auditing
 - Compliance/regulation
 - Staging of data (future settings/values/accounts, etc.)
 - Data analytics
 - Performance?!
- Modern technology is making it easier to store this data
 - Faster machines/processors
 - Significantly cheaper and larger storage devices
 - Ultra-fast storage retrieval times!

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

5

Data volume is increasing at a tremendous rate, and business and regulatory demands are driving the need to maintain large volumes of data, as well as multiple versions of data. Flexibility in an application design is important, especially in the era of the ever expanding internet where adaptability and flexibility dictate whether or not a product, business, or strategy will succeed. We can now stored extremely large volumes of data relatively cheaply, and collecting this data over time can give businesses a tremendous advantage. Being able to quickly access historical information, or query changes to data can allow us to answer complex questions very quickly.

There is a lot of responsibility in managing these large quantities of data and maintaining efficiency and accuracy. Having a strong temporal policy will help meet these requirements successfully.

The Need for Temporal – Business Activity

- Business involves more than just what is happening today
- Take for example an insurance policy
 - I had insurance for my 1999 Ford Escort automobile from June of 2006 until January of 2012
 - I added a 2012 Jeep Liberty in June of 2012
 - In September of 2012 my daughter will be added to my policy
 - Someone filed a claim against my policy for a fender bender with my Escort back in August of 2011
- All of this information about my policy needs to be stored by time
 - When I added or removed vehicles to/from my policy
 - When I will be adding a person to my policy
 - Filing claim “as of” a certain date
 - Was I covered that day?

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

6

A businesses' relationship with its customers can change over time. It's quite simple to maintain “current” information about the state of a customer's account, but what about where their account was a certain point in time? What if the customer wants to change their relationship with the business, but not for another six months? How can that information be stored?

The answer is to include a set of dates or times associated with a set of data. This period of time, called an application-period or business period reflects a period in time in which the business data was in effect or active. This time period becomes part of the key of the data, and thus a database table can hold many rows for a certain customer or account depending upon the number of application-periods.

The Need for Temporal – Business Activity

- Every time a policy changes
 - Terminate the current policy
 - Create a new policy
 - This could get complicated and confusing to the user and/or customer
 - The database design is simple, however
- Add a time dimension to the policy
 - Each policy gets an effective period
 - Each effective period represented by a start and end date
 - The database design becomes more complicated
 - A time period has to become part of the identifier
 - Overlaps in effective periods have to be controlled or avoided

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

7

Without temporal data storage only the current active data could be stored within a database table. This can lead to some difficulties and confusion as one can see what the current data looks like, but not what it looked like 6 months ago, or even at some point in the future. A temporal design overcomes these limitations and so when a change occurs to the business data in a way that is significantly meaningful to the relationship between the business and the customer then that change is added to the database, but does not replace the data that already existed. Business rules, reflected in the application-period, now dictate when that piece of data was or is active.

The Need for Temporal – Auditing/History

- There is a tremendous need to keep a history of all data changes
 - Regulation
 - Audit requirements
 - Analytics
- Lower data storage costs make it possible to store vast quantities of data
- Modern OLTP designs now include historical storage
 - Base table plus associated history table
 - Data moved from base to history any time something changes

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

8

In addition to the need to keep business relative information over different active periods of time there is also the need to record all physical changes that are made to data over time. This need is fueled by an ever increasing amount of regulation that forces a business to keep a detailed audit of any changes made to data. In addition, the lower cost of data storage encourages businesses to store all data as it changes over time in order to learn more about what their customers are doing. The vast storage of data can be analyzed to determine specific patterns of data change, which can ultimately be used to improve marketing or offer better and more customized services.

The Need for Temporal – Performance?



- Keeping history can get expensive
 - Separate base and history tables can be used to improve performance
 - Smaller base table easier to maintain
 - Application processing only acts against base table
 - Data moved either synchronously or asynchronously to a history table
- Most often only “current” information is required
 - Read processes designed to read only from base table most of the time
 - History queries are a separate process and read both tables

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

9

I've worked with several large database customers that had a requirement to store large quantities of historical information for a variety of reasons. Experience and testing showed that as the size of a table increased the database engine experienced some performance challenges. Such things as table maintenance (REORG, COPY, etc.) became more expensive. Inserting data into large indexes can get relatively more expensive, and scanning large tables for various data conditions can be very expensive. In some situations it was beneficial to separate current and historical information into separate data tables. This is especially true if the majority of the business activity involves the most recent image of the data, and transaction volumes are relatively high. In this case a temporal design that maintains a current data image in a base table and historical information in a separate table can be a performance improvement, as long as the majority of the read requests fall only against the base table.

The Challenges of Temporal Data Storage

- Key values are more difficult to maintain
 - Longer development times and more complex programming
 - Challenge to avoid time period overlaps
 - More tables and table maintenance
 - Potential for data movement to history tables means even more programming
- Queries can become much more complicated
 - If multiple tables which table do you query?
 - Each query must consider time

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

10

There are challenges associated with temporal database designs. The most significant of these challenges revolves around the amount of programming involved with the proper maintaining the data in the database. Simple changes to data can become seriously complicated when the change is relative to time. Care must be involved in avoiding time period overlaps that can lead to confusion as to which piece of data is currently in effect. Referential integrity can become a seriously complex issue when various relationships have to be maintained over time. Querying the data can also become seriously more complex as a time component has to be used, or perhaps a determination as to which table to query can come into play.

Before the automated features of DB2 temporal tables this all had to be done programmatically. Performance and accuracy could be seriously compromised if this was not done correctly.



DB2 Temporal Tables

Introducing DB2 Temporal Tables



- DB2 10 for both z/OS and LUW has introduced automated temporal tables
- The automation applies to both table updates and queries
- There are two types of temporal tables
 - Application-period temporal tables
 - System-period temporal tables
- The two types of temporal tables can be combined to create a bi-temporal table

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

12

We have always had some level of automation built into the DB2 database engine. Strong typing, referential integrity, and triggers are all examples of database automation.

Temporal tables take this concept of database automation even further by basically taking an entire portion of application logic and placing it in a relatively simple table structure with seamless automation built in. Application developers can incorporate a time dimension in their design without having to code for the maintenance of the time factor, and can rely on DB2 to maintain the proper consistency of their time dependent data. They will still have to consider time as they read their data, but the maintenance of the time dimension is automated within DB2.

There are two types of DB2 temporal tables, application-period and system-period temporal tables. The application-period temporal tables are useful for the management of data that is active during a certain time period, such as insurance policy information, and allows data to be staged (active in the future), current, and historical in nature. The system-period temporal tables are useful for tracking physical changes to data values, which makes them great for auditing and/or transaction history, and thus important for compliance and quality control and analytics.

The Advantage of Using DB2 for Temporal Data Storage



- Automation, automation, automation
 - Temporal periods automatically maintained
 - Temporal period overlap detection
 - Automatic movement of data from base table to history table
 - Temporal updates and deletes
 - Temporal query access
- This automation is the key
 - Developers need not know about history tables
 - No reason to code for overlap detection/maintenance
 - DB2 is responsible for finding the correct data to SELECT, UPDATE, and DELETE based upon a period-specification

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

13

I've worked on several database projects that involved the use of triggers. The great thing about triggers is that they can be used to replace large sections of application logic. I've used triggers to capture database changes and propagate them to audit tables, perform denormalizations on the fly, and aggregate data to summary tables. Days or weeks of application programming and testing were replaced with just a few hours to set up and test the triggers. Another great thing was that after the process was no longer needed a simple DROP of the triggers removed that process without an application program having to change a single line of code.

DB2 can automatically take this concept of replication via triggers even further with temporal tables. This includes maintaining the time periods, moving data to history tables, and building period specific predicates.

Application-Period Temporal Tables




- Single table definition that contains specific designations
 - Time columns to designation to start and end of a business period
 - Can be DATE data type
 - Can be TIMESTAMP data type
 - The specification of the columns that indicate the business period
 - “PERIOD BUSINESS_TIME” clause
- Application-period temporal table is used to contain data along with the period of time in which the data is effective or active
 - This is the application-period or business period
 - Used to stage data, or reflect the state of data at a specific time
 - Example: this satisfies our insurance policy needs

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

14

An application-period temporal table is a table that has been defined with the appropriate time columns along with the specification of a business period, and can be useful for such things as reference and code tables where descriptions will change over time, or for more involved applications such as the active period of a specific insurance policy. The business period is created via the inclusion of a “PERIOD BUSINESS_TIME” clause that identifies the start and end time columns. These columns can be DATE or TIMESTAMP data types.

Application-Period Temporal Table Definition 

```

CREATE TABLE ACT (
  ACTNO          SMALLINT NOT NULL,
  START_DT      DATE NOT NULL,
  END_DT        DATE NOT NULL,
  ACTKWD        CHAR(6) NOT NULL,
  ACTDESC       VARCHAR(20) NOT NULL,
  PERIOD BUSINESS_TIME (START_DT, END_DT) );

CREATE UNIQUE INDEX XACT1 ON ACT
(ACTNO ASC, BUSINESS_TIME WITHOUT OVERLAPS);

```

Includes start and end columns to indicate the effective time, known as the business period or application-period

PERIOD clause identifies the start and end columns and that this is an application-period temporal table

Optional BUSINESS_TIME WITHOUT OVERLAPS clause makes the application-period part of the primary key and enables the automatic enforcement of overlap detection and prevention.

8/16/2015 © 2015 DanL Database Consulting - presented August 2015 15

When you define the business period for the table DB2 will automatically generate an implicit table check constraint that ensures the start time column value for a row is less than the end time column for that row. The application is still responsible for setting the values for the start and end time columns when new data is inserted into the table. The application can also update start and end time columns. However, DB2 will control the data, including start and end times, when time relevant updates and deletes are applied. Optionally, you can also define a unique constraint and/or index on the table that includes the business period. This optional definition will ensure that the business periods do not overlap across rows in the table.

System-Period Temporal Tables



- A single or multiple table definition that contains specific designations
 - TIMESTAMP columns that used to control the system-period
 - Start timestamp
 - End timestamp
 - Transaction timestamp
 - The specification of columns that represent the system-period
 - “PERIOD SYSTEM_TIME” clause
- There is an option history table that can be associated with the system-period temporal table
- A system-period temporal table holds the current version of data values
 - Only one row per primary key
 - System-period is not part of the key
 - Option history table will contain previous data values
 - This satisfies out audit/history requirements

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

16

A system-period temporal table is a table that has been defined with the appropriate timestamp columns along with the specification of a system period, and can be useful when all changes to a table need to be tracked with a retained history to fulfill auditing or compliance requirements. The system period is created via the inclusion of a “PERIOD SYSTEM_TIME” clause that identifies the start and end timestamp columns. When you define a system period for a table DB2 is in control of the start and end timestamps, and the timestamps reflect the state of the data for the time specified. Within the base table only data that is currently effective is represented. Optionally, you can define a history table and associate it to the base table. Any changes to data then result in before images of the changed data automatically replicated to the history table, with appropriate updates to the timestamps in both the base and history tables.

System-Period Temporal Table Definition **DanL** Database Consulting

```

CREATE TABLE DEPT (
DEPTNO          CHAR(3) FOR SBCS DATA NOT NULL,
START_TS        TIMESTAMP(12)
GENERATED ALWAYS AS ROW BEGIN NOT NULL,
END_TS          TIMESTAMP(12)
GENERATED ALWAYS AS ROW END   NOT NULL,
TRANS_TS        TIMESTAMP(12)
GENERATED ALWAYS
AS TRANSACTION START ID IMPLICITLY HIDDEN,
DEPTNAME        VARCHAR(36) FOR SBCS DATA NOT
NULL,
PERIOD SYSTEM_TIME(START_TS, END_TS));

CREATE UNIQUE INDEX XDEPT1 ON DEPT
(DEPTNO ASC);

CREATE TABLE DEPT_HIST LIKE DEPT;

ALTER TABLE DEPT ADD VERSIONING USE HISTORY
TABLE DEPT_HIST;

```

Includes start and end timestamp columns are always generated by DB2 and start as row begin and end columns

The primary key does not include the system-period

An optional history table can be associated with the base table

8/16/2015 © 2015 DanL Database Consulting - presented August 2015 17

When you define a system-period temporal table DB2 is completely in control of the start and end timestamps. The timestamps reflect the period of time in which the data is current, and are not part of the primary key. This history table is optional, but when defined and related to the base table DB2 will automatically propagate “before” images of any data that changes to the history table and include the proper start and end timestamps.

Bi-Temporal Tables



- Combination of an application-period and system-period temporal table
 - This results in the table possessing the functionality of both types of temporal tables
- A bi-temporal table can include
 - A start and end set of application-period date or timestamp columns
 - A BUSINESS_PERIOD specification
 - A start and end set of system-period timestamp columns
 - A SYSTEM_PERIOD specification
 - An optional index containing controls business period overlaps
 - An optional history table
- Bi-temporal will not be covered in this presentation
 - But we can assume the functionality is a combination of application-period and system-period tables

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

18

An application-period and system-period design can be incorporated into a single table making what is known as a bi-temporal table. This design gives you all the flexibility and power to combine both the business period and system period controls into a single table. We won't cover bi-temporal in this presentation. However, the features of each application-period and system-period can be assumed in a bi-temporal design.



Table Design Considerations

Table Design Performance Considerations

- Special table design considerations should be made for history tables
- The history table will actually be insert-only
 - So it should be designed for insert-only
- Some features that should be considered for history tables, especially for high activity base tables (updates and deletes)
 - APPEND ONLY table definition
 - MEMBER CLUSTER table space definition
 - Larger index page sizes to reduce index page split frequency
 - PCTFREE 0 FREEPAGE 0
- Frequent REORGS for history tables may be necessary
 - Especially if there are frequent history table reads
 - This is regardless of any special history table design

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

20

History table, table space, and index design have to be considered for system-period temporal designs. Unlike its base table counterpart, the history table will actually be an insert-only table. With that in mind, you may want to consider an alternate physical design of the history table that favors a higher level of performance for inserts depending upon the frequency of change to the base table. If the base table experiences a high frequency of updates and/or deletes some history table design considerations would be to make the history table APPEND ONLY and/or the table space MEMBER CLUSTER, and using a larger index page size than 4K to reduce the frequency of index page splits. Frequent REORGS to the history table or index are very important when there is high insert activity, resulting from updates to the base table, regardless of the physical design. If there are also frequent reads from the history table then the REORGS become even more important in that you'll want to respect the clustering that is meaningful for the readers.

Referential Constraints and Temporal Tables



- Time base enforced referential integrity is not supported by DB2
- A primary key can be defined on an application-period temporal table that includes the business time
 - Application-period temporal tables normally include the business time as part of the key
 - However, if cannot be used to establish relationships with other tables
- This can lead to a more complex database design if database enforced referential integrity is desired
 - Need to establish “key” tables
- This does not generally apply to system-period temporal tables
 - The system-period is not part of the key

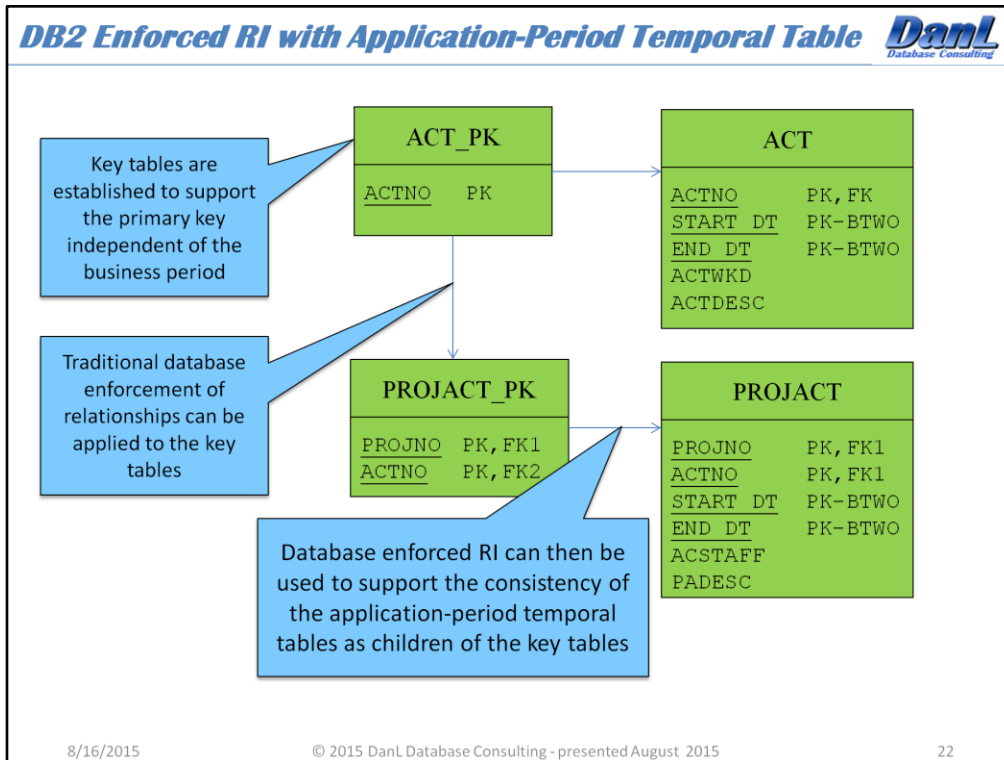
8/16/2015

© 2015 DanL Database Consulting - presented August 2015

21

Time based referential integrity is currently not supported by DB2. So, while you can define a primary key on an application-period temporal table, you cannot establish relationships with other tables. Since the whole point of application-period temporal tables is to add a time component to what was otherwise a primary key this adds some considerations and complexity to a database design that utilizes application-period temporal tables.

Setting up database enforced referential integrity for system-period temporal tables is more traditional and straight forward. Since the system-period columns are not part of the primary key, and there is still only one row per primary key, then tradition parent-child relationships can be enforced for the base table.



One solution to the challenge of database enforced referential integrity for an application-period temporal table is to establish separate key tables that support the database enforced referential integrity, while their application-period temporal children support the storage of the detailed elements.



Temporal Data Access; Updating and Deleting

Updating Application-Period Temporal Tables



- Active or effective period for a row in an application-period temporal table can be controlled
 - By the application
 - Also, by DB2
- Application can make changes to start and end times
 - DB2 will check for overlaps if included in a unique index
- DB2 can also control the active or effective period
 - Dependent upon updates and deletes that are performed by the application
 - SQL extension controls this in the DML statement
 - FOR PORTION OF clause
 - The application specifies the period of time in which a DELETE or UPDATE will apply
 - DB2 adjusts the data accordingly
 - May impact multiple rows

8/16/2015


© 2015 DanL Database Consulting - presented August 2015

24

With an application-period temporal table the changes to the span in time in which a row is active is controlled both by the application and by the database. When making changes to data we can specify start and end times for our data values since the application is ultimately in control of these values. For example, we can change the end date of the data for ACTNO 10 in our ACT table to make the value expire at the end of 2012.

In addition to application provided start and end times, the database can also control the active time range for a key value dependent upon the updates and deletes that are performed. DB2 has provided an extension to the SQL syntax to allow a time specification for temporal modifications to data in the form of the FOR PORTION OF clause. This clause allows to you specify the period in time in which a row for the non-time key columns is active. So, for our data (notice I didn't say row) represented by the ACTNO 10 we can start a new representation of the data for a future period of time. Imagine that beginning in June of 2012 this ACTNO will represent something called "MANAGE/ADVISE2".

Updating Application-Period Temporal Tables



```

SELECT * FROM ACT WHERE ACTNO = 10;

ACTNO  START_DT  END_DT      ACTKWD  ACTDESC
-----
    10  2000-01-01  9999-12-31  MANAGE  MANAGE/ADVISE

UPDATE ACT SET END_DT = '2012-12-31'
WHERE ACTNO = 10;

ACTNO  START_DT  END_DT      ACTKWD  ACTDESC
-----
    10  2000-01-01  2012-12-31  MANAGE  MANAGE/ADVISE

```

Original row value with application-period of 1/1/2000 through 12/31/9999

An update statement can modify the start and end values directly

Here the end date is updated and DB2 ensures the new application-period is accurate

8/16/2015
© 2015 DanL Database Consulting - presented August 2015
25

You can execute normal insert and update statements, but if the unique or primary key of the table includes the business time then your insert can add a row for an already existing non-time based key value and a delete could remove multiple rows representing different time ranges for a value. Our ACT table has a non-time key column of ACTNO and then the time duration as part of its unique key. Inserts, updates, and deletes will have to consider the time duration.

Updating Application-Period Temporal Tables **DanL** Database Consulting

```

SELECT * FROM ACT WHERE ACTNO = 10;

ACTNO  START_DT  END_DT  ACTKWD  ACTDESC
-----
      10 2000-01-01 2012-12-31  MANAGE  MANAGE/ADVISE

UPDATE ACT FOR PORTION OF BUSINESS_TIME
FROM '2012-06-01' TO '2012-12-31'
SET ACTDESC = 'MANAGE/ADVISE2'
WHERE ACTNO = 10;

ACTNO  START_DT  END_DT  ACTKWD  ACTDESC
-----
      10 2012-06-01 2012-12-31  MANAGE  MANAGE/ADVISE2
      10 2000-01-01 2012-06-01  MANAGE  MANAGE/ADVISE
  
```

The result of this time relative update results in the update of one row and the insertion of a second row since the portion of the did not span the entire original period

Original row value with application-period of 1/1/2000 through 12/31/2012

The FOR PORTION OF clause is used to designate the period of time for which an update applies

8/16/2015 © 2015 DanL Database Consulting - presented August 2015 26

In addition to application provided start and end times, the database can also control the active time range for a key value dependent upon the updates and deletes that are performed. DB2 has provided an extension to the SQL syntax to allow a time specification for temporal modifications to data in the form of the FOR PORTION OF clause. This clause allows to you specify the period in time in which a row for the non-time key columns is active. So, for our data (notice I didn't say row) represented by the ACTNO 10 we can start a new representation of the data for a future period of time. Imagine that beginning in June of 2012 this ACTNO will represent something called "MANAGE/ADVISE2".

This update will result in the addition of a row for ACTNO 10 as well as an update such that the data for the two new time periods are properly represented.

Updating System-Period Temporal Tables



- The system-period is NOT part of the key of a system-period temporal table
 - The application CANNOT update the start and end values
 - DB2 is in control of these values
 - This forces the accurate representation of values over time
- Inserts to the base table
 - Add data only to the base table
 - Start value is the time of the insert
 - End value is the maximum time
- Deletes to the base table
 - Remove data from the base table
 - Adds data to the history table
 - Start value in the history table is start value of the base table row
 - End value in the history table is the time of the delete

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

27

With a system-period temporal tables the start and end timestamps are out of the control of the application and completely controlled by the data server. Inserts add data to the base table, updates modify data in the base table, and deletes remove data from the base table. Only updates will result in the changing of start timestamps in the base table, and the end timestamp will always be the maximum value. It's only when a history table is associated with the base table that things become more interesting. Inserts into the base table will not result in any activity to the history table. However, any update or delete will result in "before" image rows being propagated automatically to the history table. In addition, the end timestamp value for the propagated row will be updated to reflect the end period for the data values of the row.

Updating System-Period Temporal Tables



- Updates to the base table
 - Data is updated in the base table
 - The start value in the base table is updated to the time of the update
 - The end value in the base table remains the maximum value
 - Data is inserted into the history table
 - Start value in the history table is start value of the base table row
 - End value in the history table is the time of the update

Employee System-Period Temporal Table



```
CREATE TABLE EMP (  
EMPNO          CHAR(6) FOR SBCS DATA NOT NULL,  
START_TS       TIMESTAMP(12)  
GENERATED ALWAYS AS ROW BEGIN NOT NULL,  
END_TS         TIMESTAMP(12)  
GENERATED ALWAYS AS ROW END NOT NULL,  
TRANS_TS       TIMESTAMP(12)  
GENERATED ALWAYS  
AS TRANSACTION START ID IMPLICITLY HIDDEN,  
LASTNAME       VARCHAR(15) FOR SBCS DATA NOT NULL,  
WORKDEPT       CHAR(3) FOR SBCS DATA WITH DEFAULT  
NULL,  
PERIOD SYSTEM_TIME(START_TS, END_TS));  
  
CREATE TABLE EMP_HIST LIKE EMP;  
ALTER TABLE EMP  
ADD VERSIONING USE HISTORY TABLE EMP_HIST;
```

Includes start and end timestamp columns are always generated by DB2 and start as row begin and end columns

The history table is associated with the base table

Updating System-Period Temporal Tables **DanL** Database Consulting

```

SELECT * FROM EMP WHERE EMPNO='000010';

EMPNO START_TS          END_TS          LN  WORKDEPT
-----
000010 2012-03-27 12:36:54.170022171 9999-12-30 00:00:00.0 HAAS  A00

<history table is empty>

UPDATE EMP SET LASTNAME = 'RADY' WHERE EMPNO='000010';

SELECT * FROM EMP WHERE EMPNO='000010';

EMPNO START_TS          END_TS          LN  WORKDEPT
-----
000010 2012-03-28 15:22:16.928786187 9999-12-30 00:00:00.0 RADY  A00

SELECT * FROM EMP_HIST WHERE EMPNO='000010';

EMPNO START_TS          END_TS          LN  WKD
-----
000010 2012-03-27 12:36:54.170022171 2012-03-28 15:22:16.928786187 HAAS  A00

```

Original EMP row for employee '000010' before update. History table is empty.

Updated EMP row for employee '000010' after update.

After the update to the base table the history table contains the "before" image of the updated row. System-period reflects the period of time in which the data was "current".

8/16/2015 © 2015 DanL Database Consulting - presented August 2015 30

The end timestamp value for the propagated row will be updated to reflect the end period for the data values of the row. For example, if we change the value of the LASTNAME column of employee '000010' as shown here, a row will be inserted into the history table to reflect the condition of the data previous to the update. Likewise, a delete to the DEPT table for DEPTNO 'J22' will result in the row for that department removed from the base table, and a row reflecting the image of the row up to the point the delete in the DEPT_HIST table.



Temporal Data Access; Time Travel Queries

Querying Temporal Tables



- Temporal tables can be accessed via normal SQL statements
 - Application-period table
 - System-period base table
 - System-period history table
- You can specify your own time values against the period columns

```

SELECT * FROM ACT WHERE ACTNO = 10;

ACTNO  START_DT  END_DT    ACTKWD  ACTDESC
-----
      10 2012-06-01 2012-12-31  MANAGE  MANAGE/ADVISE2
      10 2000-01-01 2012-06-01  MANAGE  MANAGE/ADVISE

SELECT * FROM ACT WHERE ACTNO = 10
AND END_DT > '2012-12-30' AND START_DT <= '2012-12-30';

ACTNO  START_DT  END_DT    ACTKWD  ACTDESC
-----
      10 2012-06-01 2012-12-31  MANAGE  MANAGE/ADVISE2
  
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

32

Temporal tables can be accessed via normal SQL statements, and do not require a time specification when accessing them.

Time Traveling using a Period Specification



- A period-specification can also be used to request time relative data
 - For a specific point in time
 - AS OF clause
 - For a range of time values
 - BETWEEN clause
 - FROM/TO clause
- Period specification
 - BUSINESS_TIME specification for application-period
 - SYSTEM_TIME specification for system-period
 - You can use both for bi-temporal tables
- The period specification is part of the FROM clause
 - “Magic” time travel access happens during DB2 query transformation

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

33

Queries against temporal tables can also contain a clause called a period-specification. This period-specification can be for a specific point in time, or for a range of time values. However, it is important to know how the queries will behave both with and without the time specification.

BUSINESS_TIME Period Specification **DanL** Database Consulting

```

SELECT *
FROM ACT FOR BUSINESS_TIME AS OF '2012-12-30'
WHERE ACTNO = 10;

transformed query:
SELECT *
FROM ACT
WHERE ( ACT.ACTNO = 10
AND ACT.END_DT > '2012-12-30'
AND ACT.START_DT <= '2012-12-30' );

result:
ACTNO START_DT  END_DT  ACTKWD ACTDESC
-----
    10 2012-06-01 2012-12-31  MANAGE  MANAGE/ADVISE2

```

Period specification uses the "AS OF" clause to request data for a specific point in time. In this case that is a date.

The query transformation component of DB2 rewrites the period specification as a compound range predicate.

8/16/2015
© 2015 DanL Database Consulting - presented August 2015
34

Let's take a look at a BUSINESS_TIME specification for application-period temporal tables. It is important to note here that the business start and end times for a row in an application-period temporal table are inclusive for the start time and exclusive for the end time. So, if a row has a range of '2012-06-01 to '2012-12-31', as with our ACT table example, then the data is active inclusively for June 1st through December 30th. So, if you consider the ACT table data then you can assume that a query with a period-specification as of '2012-12-31' will return no rows, while one that specifies '2012-12-30' will return the single row that is active for that date.

Also, notice that the period-specification is transformed by the query transformation component of DB2 into a compound range predicate.

Application-Period Datetime Values



```
Data in the ACT table:  
ACTNO  START_DT  END_DT  ACTKWD  ACTDESC  
-----  
10 2012-06-01 2012-12-31  MANAGE  MANAGE/ADVISE2  
10 2000-01-01 2012-06-01  MANAGE  MANAGE/ADVISE
```

You must be mindful of the fact that for an application period the start value is inclusive and end value is exclusive


```
SELECT *  
FROM ACT FOR BUSINESS_TIME AS OF '2012-12-31'  
WHERE ACTNO = 10;
```

The period specification requests the data as of 12/31/2012, which is equal to the end value of one row

```
transformed query:  
SELECT *  
FROM ACT  
WHERE ( ACT.ACTNO = 10  
AND ACT.END_DT > '2012-12-31'  
AND ACT.START_DT <= '2012-12-31' );
```

Because the end value is exclusive of the period no data is returned

```
result:  
<empty set>
```

Between BUSINESS_TIME Period Specification 

```

SELECT *
FROM ACT FOR BUSINESS_TIME
BETWEEN '2012-05-01' AND '2012-06-01'
WHERE ACTNO = 10;

transformed query:
SELECT *
FROM ACT WHERE ( '2012-05-01' <= '2012-06-01'
AND ACT.ACTNO = 10
AND ACT.END_DT > '2012-05-01'
AND ACT.START_DT <= '2012-06-01' );

result:
ACTNO START_DT  END_DT  ACTKWD ACTDESC
-----
  10 2000-01-01 2012-06-01  MANAGE  MANAGE/ADVISE
  10 2012-06-01 2012-12-31  MANAGE  MANAGE/ADVISE2

```

BETWEEN clause locates data that overlaps the period specification

Query transformation includes the start date in the range

8/16/2015 © 2015 DanL Database Consulting - presented August 2015 36

In addition to the “AS OF” period-specification you can query based upon a range of time. There are two choices for specifying range values, the “FROM” and “BETWEEN” clauses. They differ slightly in that the FROM is strictly locating data that was active in the time that intersects the specification, and BETWEEN is locating data that was active in the time that overlaps the specification. To put it in straight forward terms, FROM will return any rows where a start value for a row is less than the second value specified and the end value for a row is greater than the first value specified. BETWEEN will return any rows where a start value for a row is less than or equal to the second value specified and the end value for the row is greater than the first value specified. Notice that for BETWEEN the end value still has to be greater than the first value specified. This is because the end time for a row is exclusive.

It’s important to note this behavior and make sure that developers have the right knowledge and training to be able to accurately code period-specifications. If it’s too much to know, or too complicated to let the database do it, then you can always code the predicates against the time value columns yourself! This could allow you to break the rules and treat the end time value as inclusive in your queries.

From/To BUSINESS_TIME Period Specification



```

SELECT *
FROM ACT FOR BUSINESS_TIME
FROM '2012-05-01' TO '2012-06-01'
WHERE ACTNO = 10;

transformed:
SELECT *
FROM ACT
WHERE ( '2012-05-01' < '2012-06-01'
AND ACT.ACTNO = 10
AND ACT.END_DT > '2012-05-01'
AND ACT.START_DT < '2012-06-01' );

result:
ACTNO START_DT  END_DT  ACTKWD ACTDESC
-----
    10 2000-01-01 2012-06-01  MANAGE  MANAGE/ADVISE

```

FROM/TO clause locates data that intersects the period specification

Query transformation excludes the start date in the range

SYSTEM_TIME Specification

```

SELECT *
FROM EMP FOR SYSTEM_TIME AS OF '2012-03-27-18.00.00.000000000000'
WHERE EMPNO = '000010';

```

Result:

EMPNO	START_TS	END_TS
000010	2012-03-27 12:36:54.170022171	2012-03-28 15:22:16.928786187

Data can be retrieved from either the base table or the history table depending upon where the time relevant data is stored.

Period specification uses the "AS OF" clause to request data for a specific point in time

8/16/2015
© 2015 DanL Database Consulting - presented August 2015
38

System-period temporal base tables and history tables can each be accessed directly via normal SQL. From a performance perspective this is the preferred way to access the tables. If you need current data then you read from the base table directly. If you want historical data then you can read from the history table directly, or from both the base and history tables. You can also include a period-specification in your SQL statement against the base table, much in the same way as is specified for an application-period temporal table. The same rules apply for the various period-specification clauses as with the application-period temporal tables I discussed earlier. The big difference here is that any time travel query against a system-period temporal table will generate access against both the base and history table. The query here will return data from our employee history table when the AS Of period-specification includes a point in time prior to our update.

SYSTEM_TIME Query Transformation



```
SELECT *  
FROM EMP FOR SYSTEM_TIME AS OF '2012-03-27-18.00.00.000000000000'  
WHERE EMPNO = '000010';
```

transformed query:

```
(SELECT * FROM EMP WHERE ( EMP.EMPNO = '000010'  
AND EMP.START_TS <= '2012-03-27-18.00.00.000000000000'  
AND EMP.END_TS > '2012-03-27-18.00.00.000000000000' )  
UNION ALL  
SELECT * FROM EMP_HIST WHERE ( EMP_HIST.EMPNO = '000010'  
AND EMP_HIST.START_TS <= '2012-03-27-18.00.00.000000000000'  
AND EMP_HIST.END_TS > '2012-03-27-18.00.00.000000000000' ));
```

The query transformation component of DB2 will rewrite a time travel query against a system-period temporal table as a UNION ALL against the base and history tables

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

39

Notice that in the transformed query what was written to appear as single table access was rewritten as a union of accesses to both the base and history table. This will happen even for queries in which the AS OF period-specification includes the CURRENT_TIMESTAMP value. Therefore, your applications that deal with system-period time travel queries should perhaps only use them for truly historical requirements. For current period requirements only the base table access without a system-period specification will deliver optimal performance. There may be a change in a future release of DB2 that will allow current time queries to prune access to the history table, but for now you'll have to code for it yourself.

SYSTEM_TIME Query Transformation **DanL** Database Consulting

```

SELECT * FROM EMP FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP E
INNER JOIN DEPT FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP D
ON E.WORKDEPT = D.DEPTNO WHERE E.EMPNO = '000010';

transformed query:
( SELECT * FROM EMP, DEPT
  WHERE ( DEPT.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND DEPT.END_TS > (EXPR) AND EMP.EMPNO = '000010'
        AND EMP.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND EMP.END_TS > (EXPR) AND EMP.WORKDEPT = DEPT.DEPTNO)

  UNION ALL
  SELECT * FROM EMP_HIST, DEPT
  WHERE ( DEPT.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND DEPT.END_TS > (EXPR) AND EMP_HIST.EMPNO = '000010'
        AND EMP_HIST.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND EMP_HIST.END_TS > (EXPR) AND EMP_HIST.WORKDEPT = DEPT.DEPTNO)

  UNION ALL
  SELECT * FROM EMP, DEPT_HIST
  WHERE ( DEPT_HIST.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND DEPT_HIST.END_TS > (EXPR) AND EMP.EMPNO = '000010'
        AND EMP.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND EMP.END_TS > (EXPR) AND EMP.WORKDEPT = DEPT_HIST.DEPTNO)

  UNION ALL
  SELECT * FROM EMP_HIST, DEPT_HIST
  WHERE ( DEPT_HIST.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND DEPT_HIST.END_TS > (EXPR) AND EMP_HIST.EMPNO = '000010'
        AND EMP_HIST.START_TS <= CAST( CURRENT_TIMESTAMP AS TIMESTAMP )
        AND EMP_HIST.END_TS > (EXPR) AND EMP_HIST.WORKDEPT = DEPT_HIST.DEPTNO));

```

Query Transformation and underlying base and history table access can get expensive when you specify joins

The number of tables accessed in a time travel join of system-period temporal tables, if each table reference contains a SYSTEM_TIME period-specification, is equal to $(2^n) \times n$ where n is the number of tables in the join.

8/16/2015
© 2015 DanL Database Consulting - presented August 2015
40

The transformation of a query with a system-period specification into a union of the base and history table can become more complicated when we consider the joining of two tables. No one would think much about a join between the DEPT and EMP tables, and can assume two tables are accessed in the query, but if you include a SYSTEM_TIME period-specification for the tables in the query then the number of tables accessed increases dramatically. The number of tables accessed in a time travel join of system-period temporal tables, if each table reference contains a SYSTEM_TIME period-specification, is equal to $(2^n) \times n$ where n is the number of tables in the join. Therefore the join of DEPT to EMP, each with their own period-specification, results in 8 table accesses. A similar three table join will result in 24 table accesses, and a similar 4 table join will result in 64 table accesses. It therefore goes without saying that joining system-period temporal tables using SYSTEM_TIME period-specifications should be done carefully or perhaps avoided when performance is the primary concern.

Options to Control Temporal Access



- Special Registers Use
 - Can be used to transform an existing application to temporal capable
 - Improves performance without separate queries
- Special Registers
 - CURRENT TEMPORAL SYSTEM_TIME
 - CURRENT TEMPORAL BUSINESS_TIME
- Temporal tables allowed in views
 - Temporal tables can be specified in view syntax
 - Period-specification allowed against the view


Bind Options



- Bind and routine options: the following options are added to control the sensitivity to the settings of the CURRENT TEMPORAL SYSTEM_TIME/BUSINESS_TIME registers
 - SYSTIMESENSITIVE and BUSTIMESENSITIVE (default YES)
 - BIND PACKAGE
 - REBIND PACKAGE
 - REBIND TRIGGER PACKAGE
 - CREATE TRIGGER (implicit trigger package)
- SYSTEM_TIME SENSITIVE and BUSINESS_TIME SENSITIVE (default YES)
 - CREATE FUNCTION (SQL scalar)
 - ALTER FUNCTION (SQL scalar)
 - CREATE PROCEDURE (SQL native)
 - ALTER PROCEDURE (SQL native)

Making an application time sensitive allows for the creation of two access paths for every query coded against temporal tables that does not have a period-specification. Then the special registers control which access path is used at run time.

Special Register Example



- Register value is null by default
- Setting the value sets the “as of” time
- Multiple queries not needed in a program

```
SELECT *
FROM EMP
WHERE EMPNO = '000010';
```

Query runs normal if
CURRENT TEMPORAL
SYSTEM_TIME is null

- Performance improvement for system-period temporal tables
 - Multiple access paths stored
 - One with temporal predicates
 - One without temporal predicates

```
SELECT *
FROM EMP
WHERE EMPNO = '000010' AS OF
CURRENT TEMPORAL SYSTEM_TIME;
```

Query runs as a time
travel query if CURRENT
TEMPORAL SYSTEM_TIME
is not null

8/16/2015
© 2015 DanL Database Consulting - presented August 2015
43

Using these special registers allows for only one query to be coded against a temporal table. Whether or not that query is a normal query or time travel query is then dependent upon the special register setting. Setting the special register to the null value will cause the query to execute as if there was no period-specification, and any valid non-null value will cause the query to execute as an “as of” time travel query.

During statement compile time (prepare for dynamic or bind for static) an extended section is added to the statement access path. So, for queries that do not specify a period-specification DB2 will build two sections for the table access; one without the UNION ALL and temporal predicates, and a second with the UNION ALL and temporal predicates. This will only happen if the package bind parameter SYSTIMESENSITIVE is set to YES, which is the default.

Summary



- Temporal tables
 - Automation of table updates greatly simplifies coding
 - Automation of table updates also guarantees accuracy and performance
 - Enforcing referential integrity in the database can be more complex for application-period temporal tables
- Time travel queries
 - Period specification simplifies and consolidates application coding
 - Programmers must be aware that end values are not inclusive
 - Accessing system-period temporal tables with period specification can get expensive
 - May want to code separate current and history queries
 - You can access any temporal table directly
 - Without using a period specification
 - Specifying your own time values

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

44

DB2 temporal tables are an outstanding database automation feature that can replace significant amounts of application programming. Establishing and maintaining application-period, system-period, or bi-temporal tables is quite straight forward and easy to do. With DB2 managing the integrity of the temporal data, the application is free from the responsibility of properly updating and reading temporal data. However, it is import to take the time to learn exactly how period-specifications are interpreted by query transformation to be certain that these specifications are coded correctly in our application. In addition, performance considerations are critical when using queries containing period-specifications against system-period temporal tables, especially for joins. Utilizing database enforced referential integrity with temporal tables can be a little more involved that with a normal database design as well. Testing and understanding how the tables work is critical for both the DBAs and application developers.



Time Travel Queries and Temporal Data Storage - What you need to know!

Daniel L Luksetich
DanL Database Consulting
danl@db2expert.com
WWW.DB2EXPERT.COM

DanL Database Consulting

- Long and short term consulting
- Experience on multiple platforms (z/OS, AIX, UNIX, Windows, Linux)
- DB2 SQL Education (multi-platform)
- DB2 application and database design
- DB2 performance audits