DB2 10 for z/OS
Temporal Overview

Paul Wirth
wirthp@us.ibm.com
Disclaimer and Trademarks

Information contained in this material has not been submitted to any formal IBM review and is distributed on "as is" basis without any warranty either expressed or implied. Measurements data have been obtained in laboratory environment. Information in this presentation about IBM's future plans reflect current thinking and is subject to change at IBM's business discretion. You should not rely on such information to make business plans. The use of this information is a customer responsibility.

IBM MAY HAVE PATENTS OR PENDING PATENT APPLICATIONS COVERING SUBJECT MATTER IN THIS DOCUMENT. THE FURNISHING OF THIS DOCUMENT DOES NOT IMPLY GIVING LICENSE TO THESE PATENTS.

TRADEMARKS: THE FOLLOWING TERMS ARE TRADEMARKS OR ® REGISTERED TRADEMARKS OF THE IBM CORPORATION IN THE UNITED STATES AND/OR OTHER COUNTRIES: AIX, DATABASE 2, DB2, Enterprise Storage Server, FICON, FlashCopy, Netfinity, RISC, RISC SYSTEM/6000, System i, System p, System x, System z, IBM, Lotus, NOTES, WebSphere, z/Architecture, z/OS, zSeries

The FOLLOWING TERMS ARE TRADEMARKS OR REGISTERED TRADEMARKS OF THE MICROSOFT CORPORATION IN THE UNITED STATES AND/OR OTHER COUNTRIES: MICROSOFT, WINDOWS, WINDOWS NT, ODBC, WINDOWS 95

For additional information see ibm.com/legal/copytrade.phtml
DB2 10 for z/OS
Temporal Data – Time Travel Query

- What is temporal data?
  - Concepts
  - Defining temporal tables
  - Temporal usage examples
  - Performance
  - My Observations On Temporal Tables
  - References
Temporal Executive Summary

“...New DB2 temporal data management technology enables firms to track and query historical, current, and future conditions in a straightforward and efficient manner. The result is a simpler way to implement auditing and compliance initiatives, to pinpoint (and correct) human errors, to ensure the integrity of data over time, and to assess changing business conditions. Instead of hard-coding greater awareness of time into database applications, triggers, and stored procedures, firms can use simple SQL statements to instruct DB2 to automatically manage multiple versions of their data as well as track effective dates for changing business conditions, such as insurance coverage, product prices, medical prescriptions, and interest rates on credit cards...”

Source: A Matter of Time: Temporal Data Management in DB2 for z/OS
Temporal (tem·po·ral)?

Adjective:
1) of or pertaining to time.
2) pertaining to or concerned with the present life or this world; worldly: temporal joys.
3) enduring for a time only; temporary; transitory (opposed to eternal).
4) Grammar
   a) of, pertaining to, or expressing time: a temporal adverb.
   b) of or pertaining to the tenses of a verb.
5) secular, lay, or civil, as opposed to ecclesiastical.

Noun: Usually, temporals.
6) a temporal possession, estate, or the like; temporality.
7) something that is temporal; a temporal matter or affair.
Temporal

- If “time is money,” then DB2’s temporal data management technology is designed to save you some of both. Built-in support for managing multiple versions of data and tracking effective business dates can save database administrators and application developers considerable time and effort.

- In prior releases, database professionals were forced to create triggers or complex application logic to manage time-dependent conditions. Now, DB2 10 for z/OS minimizes or eliminates such efforts through the introduction of new table design options, new query syntax and semantics (derived from emerging ANSI/ISO SQL standardization efforts), and other new features.
What Is Temporal Data?

- Data that you need to keep a record of for any given point in time
- Data that you may need to look at for the past, current or future situation
- One of the major improvements in DB2 10 will be the ability for the database to reduce the complexity and amount of coding needed to implement “versioned” data, data that has different values at different points in time.
- The ability to support history or auditing queries
- Supporting Business Time and System Time
Benefits Of Using Temporal Tables…

- Move the logic from the application layer to the database layer
  - Consistent handling of temporal data
- Reduce Application development time by up to 10x
  - Application development can focus on business functions
- Run current applications with no code change
  - For System Time working with the current version of data
- Preserve execution time for current queries going after current data (System Time)
- You probably have these types of applications running in your shop
Benefits Of Using Temporal Tables...

- Business Problems you can solve with temporal tables
  - Ensure that a customer only has one financial position at a given time
  - Was an insured covered for a procedure on a specific date?
    - Was that information correct at the time the claim was processed?
  - Establish prices for a catalog ahead of time, so that they are completed before the change needs to be made
  - Answer a customer complaint about an old bill
  - ... and many, many more
Temporal Data – Time Travel Query

- What is temporal data?
- **Concepts**
  - Defining temporal tables
  - Temporal usage examples
- Performance
- My Observations On Temporal Tables
- References
Basic Temporal Concepts

- **System time (system period)** involves tracking when changes are made to the state of a table, such as when an insurance policy is modified or a loan is created.

- **Business time (application period)** involves tracking the effective dates of certain business conditions, such as the terms of an insurance policy or the interest rate of a loan. (Business time is sometimes referred to as “valid time” or “application time.”)

- Indeed, some organizations need to track both types of temporal data in a single table; such tables would be considered **bitemporal**.
Basic Temporal Concepts

- DB2 defines a notion of a period, which is a time interval represented by a start time and an end time. Note that we use the term time here in a general sense, denoting some form of a time representation, be that a date or a timestamp. Also note that there are other possible time interval representations, such as a start time along with a duration; however, DB2 only supports time intervals that are expressed in terms of the starting and ending times.

- A start time of a period is considered to be included into that period, and the end time of the period is considered to be excluded from that period.

- Important time interval note: It is important to reiterate that a period in DB2 for z/OS is a (closed, opened) time interval, also referred to as half-open time interval or (inclusive, exclusive) time interval. Be cautious when developing applications (or porting existing applications), as your organization might be using a different convention (for example, another popular convention is (closed, closed) time interval where the end time is included into the interval).
Basic Temporal Concepts - System Period

- As its name implies, system period is managed by the system itself rather than by an application or a database administrator. System period is used by DB2 for automatic maintenance of historical data. That is, DB2 automatically keeps track of when a row is inserted, updated, or deleted from a table. The old rows (deleted or updated) get archived into another table. This automatic maintenance is referred to as system-period data versioning. A table that has the system-period data versioning enabled is called system-period temporal table. A system-period temporal table contains current (active) rows of the table, and the table that contains the archived old rows is called a history table.

- The system period has a name of SYSTEM_TIME and is defined on a row-begin column and a row-end column. Both of these columns must be defined as TIMESTAMP(12) WITHOUT TIME ZONE NOT NULL with a GENERATED attribute that designates these columns as row-begin and row-end.
Basic Temporal Concepts - Application Period

- As its name implies, **application period is managed by the application or a user**. The users, not the DB2 system, control what values are stored for the application period. The application period can be considered as a period showing when a row is valid to the user. It has the property of modeling data in the past, present, and future, as the data values are controlled by the application.

- With application period, DB2 **also provides functionality to create a UNIQUE index that will be unique over a period of time**. DB2 also extends UPDATE and DELETE syntax to provide capability to modify rows based upon the application period. And, of course, rows can be fetched based upon the application period as well.

- The application period has a name of **BUSINESS_TIME** and is defined on a **begin column** and an **end column**. Both of these columns must be defined either as TIMESTAMP(6) WITHOUT TIME ZONE NOT NULL or as DATE NOT NOT NULL. The data type for the begin column and end column must be the same.
Basic Temporal Concepts

- **System Time (Assertion Dates, Knowledge Dates, Transaction Time, Audit Time, In/Out-dates)**
  - Every row has a pair of `TIMESTAMP(12)` columns set by DBMS
    - Begin time: when the row was inserted in the DBMS
    - End Time: when the row was modified/deleted
  - Every base row has a Transaction Start ID timestamp
  - Query at current or any prior point/period in system time

- **Business Time (Effective Dates, Valid Time, From/To-dates)**
  - Every row has a pair of `TIMESTAMP(6)` or `DATE` columns set by Application
    - Begin time: when the business deems the row valid
    - End Time: when the business deems row validity ends
  - Constraint created to ensure Begin time < End time
  - Query at current, any prior, or future point/period in business time

- Times are inclusive for start time and exclusive for end times
Basic Temporal Concepts

- **History Table**
  - Table to save “old” rows when using System Time

- **Temporal Uniqueness**
  - PK or Unique Key with BUSINESS_TIME WITHOUT OVERLAPS
  - Support for a unique constraint for a point in time
  - This is optional, however without it:
    - Unique constraints will likely return errors due to multiple rows per key

- **Bitemporal**
  - Inclusion of both System Time and Business Time in row
Temporal Data – Time Travel Query

- What is temporal data?
- Concepts

- Defining temporal tables
  - System-period Table and History Table
  - Application-period Table
  - Bitemporal-period Table and History Table

- Temporal usage examples
- Performance
- My Observations On Temporal Tables
- References
Period Table Defined using System time
Creating A New System-period Table

CREATE TABLE TRYTEMPORAL_S(
  POLICY INTEGER,
  ATTRIBUTE VARCHAR(30),
  STATUS VARCHAR(20),
  P_FROM TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
  P_TO TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
  TRANS_ID TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID,
  PERIOD SYSTEM_TIME(P_FROM, P_TO)
);
Period Table Defined Using System Time
Adding System-period To An Existing Table

ALTER TABLE TRYTEMPORAL_E
ADD COLUMN SYS_BEGIN TIMESTAMP(12) NOT NULL
GENERATED ALWAYS AS ROW BEGIN
ADD COLUMN SYS_END TIMESTAMP(12) NOT NULL
GENERATED ALWAYS AS ROW END
ADD COLUMN TRANS_ID TIMESTAMP(12) NOT NULL
GENERATED ALWAYS AS TRANSACTION START ID
ADD PERIOD SYSTEM_TIME(SYS_BEGIN, SYS_END);
Period Table Defined using System Time
Adding System-period To An Existing Table...

• Note that we used one ALTER statement to make multiple alternations to the table, but using separate alters works also. After the alter takes effect, all existing rows in the table will have the same initial value of 0001-01-01-00.00.00.000000000000 in the SYS_BEGIN column (which is the row-begin column) and the same initial value of 9999-12-31-24.00.00.000000000000 in the SYS_END column (which is the row-end column).

• Alternatively, you might have existing timestamp columns that you want to become the basis for SYSTEM_TIME period. In this case, you would use ALTER COLUMN clause of the ALTER statement to change the existing column to the row-begin or row-end column. Note that in this case you might need to perform the following alterations.

  1) First, change the data type from timestamp(6) to timestamp(12), because 6 is the default precision and the only precision that is available prior to DB2 10 for z/OS.

  2) Second, change the column to be GENERATED ALWAYS as either ROW BEGIN or ROW END. After such alter takes effect, the values in the SYS_BEGIN and SYS_END in all existing rows would not be affected (would be preserved). Because these two columns become un-updatable by the user, you might want to ensure they contain expected values prior to altering these columns to row-begin or row-end columns.
History Table Defined Using System Time

CREATE TABLE HIST_TRYTEMPORAL_S(
  POLICY INTEGER,
  ATTRIBUTE VARCHAR(30),
  STATUS VARCHAR(20),
  P_FROM TIMESTAMP(12) NOT NULL,
  P_TO TIMESTAMP(12) NOT NULL,
  TRANS_ID TIMESTAMP(12) NOT NULL
);

Enable System-period Data Versioning

ALTER TABLE TRYTEMPORAL_S
ADD VERSIONING USE HISTORY TABLE HIST_TRYTEMPORAL_S;
History Table Defined Using System Time Requirements

To enable system-period data versioning requires the conditions that we list here. The system-period temporal table must have:

1) A column defined as TIMESTAMP(12) WITHOUT TIME ZONE, NOT NULL, with the attribute for a begin column of a SYSTEM_TIME period, namely, GENERATED ALWAYS AS ROW BEGIN. Note that this column is not updatable.

2) A column defined as TIMESTAMP(12) WITHOUT TIME ZONE, NOT NULL, with the attribute for an end column of a SYSTEM_TIME period, namely, GENERATED ALWAYS AS ROW END. Note that this column is not updatable.

3) A column defined as TIMESTAMP(12) WITHOUT TIME ZONE, NOT NULL, with the attribute for a transaction-start-ID, namely, GENERATED ALWAYS AS TRANSACTION_START_ID. Note that this column is not updatable.

4) A period, SYSTEM_TIME, specified on the first two timestamp columns described earlier in which the first column is a row-begin column and the second column is a row-end column.
History Table Defined Using System Time Requirements...

The system-period temporal table must not have:
- A security label column
- Any row permissions
- Any column masks
- A clone table defined on it
- A materialized query table definition
- An incomplete table definition
- Any other tables defined in the same table space

The history table must have:
1) The same number of columns as the system-period temporal table, minus the PERIOD statement and its parameters.
2) The columns must have the same corresponding names, data type, null attribute, CCSID, FOR BIT or SBCS, or mixed data attribute, hidden attribute and fieldproc as the system-period temporal table.
History Table Defined Using System Time Requirements...

The *history* table must **not** have:
- A security label, identity, row change timestamp, row-begin, row-end, or transaction-start-ID column
- A **generated column** other than a ROWID. Note that ROWID values in the history table will have new values, and values from the associated system-period temporal table will not be preserved.
- A period definition
- Any row permissions
- Any column masks
- A clone table defined on it
- Only one table defined in the table space.
- A materialized query table definition
- An incomplete table definition
History Table Defined Using System Time Requirements...

Additional restrictions for system-period data versioning include:

- For point-in-time recovery to keep the system-period temporal table and history data in sync, the **history table space and system-period temporal table space must be recovered as a set** and cannot be done individually unless the keyword VERIFYSET NO is used.
- No utility operation is allowed that will delete data from the system-period temporal table. This includes LOAD REPLACE, REORG DISCARD, and CHECK DELETE YES.
- A schema attribute (data type, add column, and other such attributes) cannot be altered for a system-period temporal table or history table.
- A history table, table space containing a history table, or database containing a history table cannot be explicitly dropped. A history table is implicitly dropped when the associated system-period temporal table is dropped.
- A clone table cannot be defined for a system-period temporal table or history table.
- A table cannot be created in a table space that contains a system-period temporal table or history table. In other words, both tables must be created in their own separate table spaces, and there could be no other tables in those table spaces.
- A system-period temporal table, or a column of a system-period temporal table, cannot be renamed.
- A history table, or a column of a history table, cannot be renamed.
Period Table Defined Using Business Time
Creating A New Application-period Table

CREATE TABLE TRYTEMPORAL_B(
POLICY INTEGER NOT NULL
,ATTRIBUTE VARCHAR(30)
,STATUS VARCHAR(20)
,P_FROM DATE NOT NULL
,P_TO DATE NOT NULL
,PERIOD BUSINESS_TIME(P_FROM, P_TO)
,PRIMARY KEY(POLICY, BUSINESS_TIME WITHOUT OVERLAPS);
);
Period Table Defined Using Business Time
Define The Business Time No-overlap Enforcement

```
ALTER TABLE TRYTEMPORAL_B
ADD UNIQUE(POLICY, BUSINESS_TIME WITHOUT OVERLAPS);

CREATE UNIQUE INDEX IDX_TRYTEMPORAL_B ON
TRYTEMPORAL_B(POLICY, BUSINESS_TIME WITHOUT OVERLAPS);
```

You can also define the temporal table with BUSINESS_TIME period to have a unique attribute that is unique for a period of time by specifying the BUSINESS_TIME WITHOUT OVERLAPS clause in CREATE TABLE, ALTER TABLE, or CREATE UNIQUE INDEX statements. This clause instructs DB2 to add the end column and begin column of the BUSINESS_TIME period to the index in ascending order and to enforce that there are no overlaps in time with respect to the specified index keys.
Period Table Defined Using Business Time Restrictions

- The application period has a name of BUSINESS_TIME and is defined on a begin column and an end column.
- Both of these columns **must** be defined either as TIMESTAMP(6) WITHOUT TIME ZONE NOT NULL or as DATE NOT NULL.
- The data type for the begin column and end column **must** be the same.
Bitemporal Table Defined Using System And Business Time

CREATE TABLE TRYTEMPORAL_SB(
  POLICY INTEGER,
  ATTRIBUTE VARCHAR(30),
  STATUS VARCHAR(20),
  S_FROM TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
  S_TO TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
  TRANS_ID TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID,
  B_FROM DATE NOT NULL,
  B_TO DATE NOT NULL,
  PERIOD SYSTEM_TIME(S_FROM, S_TO),
  PERIOD BUSINESS_TIME(B_FROM, B_TO)
PRIMARY KEY(POLICY, BUSINESS_TIME WITHOUT OVERLAPS);
);
History Table Defined Using System Time

CREATE TABLE HIST_TRYTEMPORAL_SB(
  POLICY INTEGER,
  ATTRIBUTE VARCHAR(30),
  STATUS VARCHAR(20),
  S_FROM TIMESTAMP(12) NOT NULL,
  S_TO TIMESTAMP(12) NOT NULL,
  TRANS_ID TIMESTAMP(12) NOT NULL,
  B_FROM DATE NOT NULL,
  B_TO DATE NOT NULL
);

Enable System-period Data Versioning

ALTER TABLE TRYTEMPORAL_SB
ADD VERSIONING USE HISTORY TABLE HIST_TRYTEMPORAL_SB;
History Table Defined Using System Time

CREATE TABLE HIST_TRYTEMPORAL_SB(
  POLICY INTEGER,
  ATTRIBUTE VARCHAR(30),
  STATUS VARCHAR(20),
  P_FROM TIMESTAMP(12) NOT NULL,
  P_TO TIMESTAMP(12) NOT NULL,
  TRANS_ID TIMESTAMP(12) NOT NULL,
  P_FROM DATE NOT NULL,
  P_TO DATE NOT NULL
);

Enable System-Period Data Versioning

ALTER TABLE TRYTEMPORAL_SB
ADD VERSIONING USE HISTORY TABLE HIST_TRYTEMPORAL_SB;
Temporal Information In The DB2 Catalog

- You can query the SYSIBM.SYSCOLUMNS catalog table to see the information about a column that is part of a period. The DEFAULT column of SYSIBM.SYSCOLUMNS contains a value of ‘Q’ for any column defined with the AS ROW BEGIN attribute, and a value of ‘R’ for any column defined with the AS ROW END attribute. The PERIOD column of SYSIBM.SYSCOLUMNS contains a value of ‘S’ for a column that is the start of the SYSTEM_TIME period and a value of ‘T’ for the column that is the end of the SYSTEM_TIME period.
- You can query the SYSIBM.SYSTABLES catalog table to verify that the system-period data versioning is enabled for the table. The columns VERSIONING_SCHEMA and VERSIONING_TABLE for the row associated with the system-period temporal table contain the schema name and the table name of the corresponding history table. In addition, the VERSIONING_SCHEMA and VERSIONING_TABLE columns for the row that is associated with the history table contain the schema name and the table name of the corresponding system-period temporal table. In addition, the TYPE column for the row associated with a history table has a value of ‘H’.
- You can query the SYSIBM.SYSCOLUMNS catalog table to see the information about a column that is part of a period. The PERIOD column of SYSIBM.SYSCOLUMNS contains a value of ‘B’ for a column that is the start of the BUSINESS_TIME period and a value of ‘C’ for the column that is the end of the BUSINESS_TIME period.
Temporal Data – Time Travel Query

- What is temporal data?
- Concepts
- Defining temporal tables

**Temporal usage examples**
- “AS OF”, “FROM ... TO ...” and “BETWEEN ... AND”
- Application-period Tables – Updates And Deletes

- Performance
- My Observations On Temporal Tables

- References
Ways To Access Data In System-period Tables

To improve the usability and to simplify accessing the historical data, DB2 extends the FROM clause of the table-reference. You can specify that historical data is requested from the system-period temporal table by using the following period-specification clauses after the table name in the FROM clause:

1) FROM FOR SYSTEM_TIME AS OF timestamp-expression
   a) Specifies that the table-reference includes each row for which the begin value for the specified period is less than or equal to timestamp-expression and the end value for the period is greater than timestamp-expression.

2) FOR SYSTEM_TIME FROM timestamp-expression1 TO timestamp-expression2
   a) Specifies that the table-reference includes rows that exist for the period that is specified from timestamp-expression1 up to timestamp-expression2. A row is included in the table-reference if the start value for the period in the row is less than timestamp-expression2 and if the end value for the period in the row is greater than timestamp-expression1.
   b) Note that the timestamp-expression2 is not inclusive, which means that any row that starts with timestamp-expression2 will not qualify in the result.
Ways To Access Data In System-period Tables...

3) FOR SYSTEM_TIME BETWEEN timestamp-expression1 AND timestamp-expression2
   a) Specifies that the table-reference includes rows in which the specified period overlaps at any point in time between timestamp-expression1 and timestamp-expression2. A row is included in the table-reference if the start value for the period in the row is less than or equal to timestamp-expression2 and if the end value for the period in the row is greater than timestamp-expression1.
   b) The table reference contains zero rows if timestamp-expression1 is greater than timestamp-expression2. If timestamp-expression1 = timestamp-expression2, the expression is equivalent to AS OF timestamp-expression1. If timestamp-expression1 or timestamp-expression2 is the null value, the table reference is an empty table.
   c) The timestamp-expression2 is inclusive, which means that any row that starts with timestamp-expression2 will qualify in the result.
Ways To Access Data In System-period Tables...

Note that you specify the new period-specification clauses for a system-period temporal table, and not the history table. The history table is not referenced directly by the query. DB2 rewrites the query to include data from the history table with a `UNION ALL` operator.

If the table is a system-period temporal table and a period-specification for the SYSTEM_TIME period is not specified, the table reference includes all current rows of the table and does not include any historical rows of the table.

The period-specification on a FROM clause of a table reference should simplify coding of applications because only the system-period temporal table needs to be referenced by a query, regardless of whether you need to access current or historical data or both.
DML With System-period Data Versioning: Insert

```sql
-- assume we perform this INSERT at the following time: 2010-01-01 08:30:00
INSERT INTO TRYTEMPORAL_S(POLICY, ATTRIBUTE, STATUS) VALUES(1, 'Purple', 'Under review');
-- assume we perform this INSERT at the following time: 2010-08-01 10:45:30
INSERT INTO TRYTEMPORAL_S(POLICY, ATTRIBUTE, STATUS) VALUES(2, 'Blue', 'Under review');
-- assume we perform this INSERT at the following time: 2010-05-01 15:05:51
INSERT INTO TRYTEMPORAL_S(POLICY, ATTRIBUTE, STATUS) VALUES(3, 'Yellow', 'Under review');

SELECT POLICY, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO FROM TRYTEMPORAL_S;
-- result: POLICY FROM TO
--   1 2010-01-01-08.30.00 9999-12-31-24.00.00
--   2 2010-08-01-10.45.30 9999-12-31-24.00.00
--   3 2010-05-01-15.05.51 9999-12-31-24.00.00

SELECT POLICY, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO FROM HIST_TRYTEMPORAL_S;
-- result: no rows returned
```

Note: History Table
-- assume we perform this UPDATE at the following time: 2010-09-01 16:00:00
UPDATE TRYTEMPORAL_S
  SET STATUS = 'Approved'
WHERE POLICY = 2;

SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM TRYTEMPORAL_S;
-- result: POLICY | STATUS        | FROM            | TO
-- 1  Under review | 2010-01-01-08.30.00 | 9999-12-31-24.00.00
-- 2  Approved      | 2010-09-01-16.00.00 | 9999-12-31-24.00.00
-- 3  Under review  | 2010-05-01-15.05.51 | 9999-12-31-24.00.00

SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM HIST_TRYTEMPORAL_S;
-- result: POLICY | STATUS        | FROM            | TO
-- 2  Under review | 2010-08-01-10.45.30 | 2010-09-01-16.00.00

Note: Before Image In The History Table
DML With System-period Data Versioning: Delete

```
-- assume we perform this DELETE at the following time: 2010-09-02 12:34:56
DELETE FROM TRYTEMPORAL_S
WHERE POLICY = 1;

SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM TRYTEMPORAL_S;
-- result: POLICY  STATUS FROM TO
-- 2  Approved  2010-09-01-16.00.00 9999-12-31-24.00.00
-- 3  Under reiew  2010-05-01-15.05.51 9999-12-31-24.00.00

SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM HIST_TRYTEMPORAL_S;
-- result: POLICY  STATUS FROM TO
-- 2  Under reiew  2010-08-01-10.45.30 2010-09-01-16.00.00
-- 1  Under reiew  2010-01-01-08.30.00 2010-09-02 12:34:56

Note: Deleted Record In The History Table
```
Example of FOR SYSTEM_TIME AS OF

-- assume table TRYTEMPORAL_S has the following data in POLICY, P_FROM, PT_TO:
-- POLICY
-- | P_FROM | P_TO |
-- |--------|------|
-- | 1 2001-01-01-00.00.00.0000000000000000 | 2008-01-01-00.00.00.0000000000000000 |
-- | 2 2002-01-01-00.00.00.0000000000000000 | 2006-01-01-00.00.00.0000000000000000 |
-- | 3 2003-01-01-00.00.00.0000000000000000 | 2004-01-01-00.00.00.0000000000000000 |
-- | 4 2005-01-01-00.00.00.0000000000000000 | 2007-01-01-00.00.00.0000000000000000 |

SELECT POLICY FROM TRYTEMPORAL_S
    FOR SYSTEM_TIME AS OF TIMESTAMP '2006-08-08 00.00.00';
-- result: POLICY
-- | 1 |
-- | 4 |

SELECT POLICY FROM TRYTEMPORAL_S
    FOR SYSTEM_TIME AS OF TIMESTAMP '2005-08-08 00.00.00';
-- result: POLICY
-- | 1 |
-- | 2 |
-- | 4 |
Example Of FOR SYSTEM_TIME FROM... TO

-- assume table TRYTEMPORAL_S has the following data in POLICY, P_FROM, PT_TO:
-- POLICY
-- | P_FROM  | P_TO             |
-- |--------|------------------|
-- | 1 2001-01-01 00.00.00.0000000000000000 | 2008-01-01 00.00.00.0000000000000000 |
-- | 2 2002-01-01 00.00.00.0000000000000000 | 2006-01-01 00.00.00.0000000000000000 |
-- | 3 2003-01-01 00.00.00.0000000000000000 | 2004-01-01 00.00.00.0000000000000000 |
-- | 4 2005-01-01 00.00.00.0000000000000000 | 2007-01-01 00.00.00.0000000000000000 |

SELECT POLICY FROM TRYTEMPORAL_S
FOR SYSTEM_TIME FROM TIMESTAMP '2003-01-01 00.00.00.00'
  TO TIMESTAMP '2005-01-01 00.00.00.00';

-- result: POLICY
--  1
--  2
--  3

SELECT POLICY FROM TRYTEMPORAL_S
FOR SYSTEM_TIME FROM TIMESTAMP '2004-01-01 00.00.00.00'
  TO TIMESTAMP '2007-01-01 00.00.00.00';

-- result: POLICY
--  1
--  2
--  4
Example Of FOR SYSTEM_TIME BETWEEN... AND

-- assume table TRYTEMPORAL_S has the following data in POLICY, P_FROM, PT_TO:

```
-- POLICY   P_FROM                     P_TO
--  1  2001-01-01-00.00.00.0000000000000000  2008-01-01-00.00.00.0000000000000000
--  2  2002-01-01-00.00.00.0000000000000000  2006-01-01-00.00.00.0000000000000000
--  3  2003-01-01-00.00.00.0000000000000000  2004-01-01-00.00.00.0000000000000000
--  4  2005-01-01-00.00.00.0000000000000000  2007-01-01-00.00.00.0000000000000000
```

```
SELECT POLICY FROM TRYTEMPORAL_S
   FOR SYSTEM_TIME BETWEEN TIMESTAMP '2003-01-01 00.00.00'
                        AND   TIMESTAMP '2005-01-01 00.00.00'

```

-- result: POLICY

```
--  1
--  2
--  3
--  4
```

```
SELECT POLICY FROM TRYTEMPORAL_S
   FOR SYSTEM_TIME BETWEEN TIMESTAMP '2004-01-01 00.00.00'
                        AND   TIMESTAMP '2007-01-01 00.00.00';

```

-- result: POLICY

```
--  1
--  2
--  4
```
Example Of Accessing History Data Using Versioning

Original Data:

```
SELECT POLICY, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM TRYTEMPORAL_S
```

Query execution time => 82 ms

<table>
<thead>
<tr>
<th>Status</th>
<th>POLICY</th>
<th>FROM</th>
<th>TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2011-06-01 13:12:46.0</td>
<td>10000-01-01 00:00:00.0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2011-06-01 13:12:47.0</td>
<td>10000-01-01 00:00:00.0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2011-06-01 13:12:47.0</td>
<td>10000-01-01 00:00:00.0</td>
</tr>
</tbody>
</table>

Update Some Data:

```
UPDATE TRYTEMPORAL_S
SET STATUS = 'Approved'
WHERE POLICY = 2
```

Query execution time => 90 ms

Delete Some Data:

```
DELETE FROM TRYTEMPORAL_S
WHERE POLICY = 1
```

Query execution time => 112 ms
Example Of Accessing History Data Using Versioning...

**Period Table Data:**

<table>
<thead>
<tr>
<th>Status</th>
<th>Result1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`-- Select from Period TBL
SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM TRYTEMPORAL_S`

Query execution time => 126 ms

<table>
<thead>
<tr>
<th>POLICY</th>
<th>STATUS</th>
<th>FROM</th>
<th>TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Approved</td>
<td>2011-06-01 13:37:34</td>
<td>10000-01-01 00:00:00.0</td>
</tr>
<tr>
<td>2</td>
<td>Under reiew</td>
<td>2011-06-01 13:37:34</td>
<td>10000-01-01 00:00:00.0</td>
</tr>
</tbody>
</table>

**History Table Data:**

<table>
<thead>
<tr>
<th>Status</th>
<th>Result1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`-- Select from History TBL
SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM HIST_TRYTEMPORAL_S`

Query execution time => 95 ms

<table>
<thead>
<tr>
<th>POLICY</th>
<th>STATUS</th>
<th>FROM</th>
<th>TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Under reiew</td>
<td>2011-06-01 13:12:47.0</td>
<td>2011-06-01 13:37:34.0</td>
</tr>
<tr>
<td>2</td>
<td>Under reiew</td>
<td>2011-06-01 13:12:46.0</td>
<td>2011-06-01 13:54:05.0</td>
</tr>
</tbody>
</table>
Example Of Accessing History Data Using Versioning...

Temporal Query

Period Table - Update  
History Table - Update  
History Table - Delete

Explain Showing Union of Period and History Tables - Automatic

```sql
SELECT POLICY, STATUS, TIMESTAMP(P_FROM, 0) AS FROM, TIMESTAMP(P_TO, 0) AS TO
FROM TRYTEMPORAL_S
FOR SYSTEM_TIME BETWEEN TIMESTAMP '2011-06-01 13.00.00'
AND TIMESTAMP '2011-06-01 14.45.00'
```
Ways To Access Data In Application-period Tables

After the BUSINESS_TIME period is established, you can query the business time data and update and delete data based upon a period of time. You need to make a similar extension to the FROM clause to allow the querying of the data based on the BUSINESS_TIME period:

1) FOR BUSINESS_TIME AS OF value
   a) Specifies that the table-reference includes each row for which the begin value for the specified period is less than or equal to value and the end value for the period is greater than value.

2) FOR BUSINESS_TIME FROM value1 TO value2
   a) Specifies that the table-reference includes rows that exist for the period that is specified from value1 up to value2. A row is included in the table-reference if the start value for the period in the row is less than value2 and the end value for the period in the row is greater than value1.
   b) Note that the value2 is not inclusive, which means that any row that starts with value2 will not qualify in the result.
Ways To Access Data In Application-period Tables...

3) FOR BUSINESS_TIME BETWEEN value1 AND value2
   a) Specifies that the table-reference includes rows in which the specified period overlaps at any point in time between value1 and value2. A row is included in the table-reference if the start value for the period in the row is less than or equal to value2 and the end value for the period in the row is greater than value1. The table reference contains zero rows if value1 is greater than value2. If value1 = value2, the expression is equivalent to AS OF value1. If value1 or value2 is the null value, the table reference is an empty table.
   b) Note that the value2 is inclusive, which means that any row that starts with value2 will qualify in the result.
Example Of FOR BUSINESS_TIME clauses

-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY   P_FROM     P_TO
--   1 2001-01-01 2008-01-01
--   2 2002-01-01 2006-01-01
--   3 2003-01-01 2004-01-01
--   4 2005-01-01 2007-01-01

SELECT POLICY FROM TRYTEMPORAL_B
FOR BUSINESS_TIME FROM DATE '2003-01-01'
   TO    DATE '2005-01-01';

-- result: POLICY
--   1
--   2
--   3
Application-period Tables – Updates And Deletes

The rows in a table for which the UPDATE or DELETE is issued can fall into one of the following categories:

- The row might be not contained in the specified period, which occurs when both columns of BUSINESS_TIME period are less than or equal to value1 or are greater than or equal to value2.
- The row might be fully contained in the specified period, which occurs when the value for the begin column for BUSINESS_TIME period in the row is greater than or equal to value1 and the value for the corresponding end column in the row is less than value2.
- The row might be partially contained in the specified period, which occurs when row overlaps the beginning of the specified period or the end of the specified period, but not both. We say that the row overlaps the beginning of the specified period, if the value of the begin column is less than value1 and the value of the end column is greater than value1. Similarly, we say that the row overlaps the end of the specified period, if the value of the end column is greater than or equal to value2 and the value of the begin column is less than value2.
- The row can fully overlap the specified period, which occurs when the row overlaps the beginning of the specified period and overlaps the end of the specified period.
Application-period Tables – Updates And Deletes...

When updating or deleting a row, DB2 takes different actions, depending on the category to which the row belongs:
• If the row is **not contained** in the specified period, no update/delete occurs for that row.
• If the row is **fully contained** in the specified period, the row is updated, deleted.
• If the row is **partially contained** in the specified period, the row is updated/deleted and a new row is inserted using the original values from the row, except that:
  ▪ If the row overlaps the beginning of the specified period, the end column is set to **value1**.
  ▪ If the row overlaps the end of the specified period, the begin column is set to **value2**.
• If the row **fully overlaps** the specified period, the row is update/deleted and two new rows are inserted using the original values from the row, except that the begin column of one row is set to **value2** and the end column of another row is set to **value1**.
Example Of FOR PORTION OF BUSINESS_TIME
Semantics: Case 1 “Not Contained”

```
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY     P_FROM       P_TO
-- 1 2001-01-01  2008-01-01
-- 2 2002-01-01  2006-01-01
-- 3 2003-01-01  2004-01-01
-- 4 2005-01-01  2007-01-01
DELETE FROM TRYTEMPORAL_B
  FOR PORTION OF BUSINESS_TIME FROM DATE '1999-01-01'
                      TO  DATE '2000-01-01';
SELECT POLICY, P_FROM, P_TO
FROM TRYTEMPORAL_B;
-- result: POLICY     P_FROM       P_TO
-- 1 2001-01-01  2008-01-01
-- 2 2002-01-01  2006-01-01
-- 3 2003-01-01  2004-01-01
-- 4 2005-01-01  2007-01-01
```
Example Of FOR PORTION OF BUSINESS_TIME Semantics: Case 2 "Partially Contained"

```
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY   P_FROM     P_TO
-- 1 2001-01-01 2008-01-01
-- 2 2002-01-01 2006-01-01
-- 3 2003-01-01 2004-01-01
-- 4 2005-01-01 2007-01-01

DELETE FROM TRYTEMPORAL_B
    FOR PORTION OF BUSINESS_TIME FROM DATE'2002-01-01'
                          TO   DATE'2002-02-02'
WHERE POLICY > 1;

SELECT POLICY, P_FROM, P_TO
FROM TRYTEMPORAL_B;
```

Note: Deleted record portion of record

```
-- result: POLICY   P_FROM     P_TO
-- 1 2001-01-01 2008-01-01
-- 3 2003-01-01 2004-01-01
-- 4 2005-01-01 2007-01-01
-- 2 2002-02-02 2006-01-01
```
Example of FOR PORTION OF BUSINESS_TIME Semantics: Case 3 "Fully Overlaps"

```sql
-- assume table TRYTEMPORAL_B has the following data in POLICY, P_FROM, P_TO:
-- POLICY   P_FROM    P_TO
-- 1 2001-01-01 2008-01-01
-- 2 2002-01-01 2006-01-01
-- 3 2003-01-01 2004-01-01
-- 4 2005-01-01 2007-01-01
DELETE FROM TRYTEMPORAL_B
FOR PORTION OF BUSINESS_TIME FROM DATE'2003-03-03'
TO DATE'2004-04-04'
WHERE POLICY = 1;
SELECT POLICY, P_FROM, P_TO
FROM TRYTEMPORAL_B;
-- result: POLICY  P_FROM    P_TO
-- 2 2002-01-01 2006-01-01
-- 3 2003-01-01 2004-01-01
-- 4 2005-01-01 2007-01-01
-- 1 2001-01-01 2003-03-03
-- 1 2004-04-04 2008-01-01
```

Note: Deleted record portion of record split
Temporal Data – Time Travel Query

- What is temporal data?
- Concepts
- Defining temporal tables
- Temporal usage examples
- **Performance**
- My Observations On Temporal Tables
- References
All three tests were carried out with the same workload, which is a transaction mix of 70% read (SELECT), 30% write (10% INSERT + 20% UPDATE/DELETE).

The trigger CPU overhead can be calculated as 78.31 - 49.86 = 28.45, and the CPU overhead associated with temporal history maintenance can be calculated as 62.33 - 49.86 = 12.47. So, the percentage savings in CPU overhead can be calculated as \((28.45 - 12.47) / 28.47 = 56\%\) on this mixed workload.
UPDATE Performance With SYSTEM TIME
Temporal vs. Trigger Solution

Even though, significant UPDATE path length and CPU time overhead is expected with the data versioning (temporal) function enabled, UPDATE with SYSTEM TIME support performed better than using trigger for history row backup. For the test scenario described in this section, DB2-provided system time temporal support for UPDATE out-performed user defined trigger solution by 30% to 39%.
DELETE Performance With SYSTEM TIME
Temporal vs. Trigger Solution

Even though, significant DELETE path length and CPU time overhead is expected with the data versioning (i.e., temporal) function enabled, DELETE with SYSTEM TIME support performed better than that of using a trigger solution for history row backup.

For the test scenario described in this section, DB2-provided system time temporal support for DELETE out-performed user defined trigger solution by 10% to 19%.
UPDATE Performance On BUSINESS TIME Temporal vs. Stored Procedure Solution

Two tests were run to measure the UPDATE performance difference on a Business Time Temporal Table as described below. UPDATE statements were ran against a temporal table with a business time period defined and the update operation caused row splits.
Temporal Performance - Queries With Explicit And Implicit UNION ALL With History Tables

The table below shows three sample queries which return identical rows, along with their respective run time. We compare the execution of RDS transformed query with base table UNION ALL with history table as a result of SYSTEM_TIME AS OF syntax vs. user specified query with explicit base table UNION ALL with history table.

<table>
<thead>
<tr>
<th>No</th>
<th>Query</th>
<th>UNION ALL</th>
<th>DB2 elapsed time</th>
<th>DB2 CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>SELECT * FROM LINEITEM WHERE L_ORDERKEY=80001 AND SYS_BEGIN &lt; '2009-10-06-02.00.58' UNION ALL SELECT * FROM LINEITEM_H WHERE L_ORDERKEY=80001 AND SYS_END &lt; '2009-10-06-02.00.58'</code></td>
<td>Explicit</td>
<td>1.382320</td>
<td>0.009238</td>
</tr>
<tr>
<td>2</td>
<td><code>SELECT * FROM LINEITEM FOR SYSTEM_TIME AS OF '2009-10-06-02.00.58' WHERE L_ORDERKEY = 80001;</code></td>
<td>Implicit - due to AS OF predicate</td>
<td>1.372669</td>
<td>0.009321</td>
</tr>
<tr>
<td>3</td>
<td><code>SELECT * FROM LINEITEM FOR SYSTEM_TIME BETWEEN '2001-01-01-02.00.58' AND '2009-10-06-02.00.58' WHERE L_ORDERKEY = 80001;</code></td>
<td>Implicit - due to date range predicate</td>
<td>1.608537</td>
<td>0.009156</td>
</tr>
</tbody>
</table>
Temporal Performance - Queries With Explicit And Implicit UNION ALL With History Tables – Query #1

The access path for query #1 shows the explicit UNION ALL in the access path information.
Temporal Performance - Queries With Explicit And Implicit UNION ALL With History Tables – Query #2

The access path for query #2 shows the implicit UNION ALL in the access path information.
Temporal Performance - Queries With Explicit And Implicit UNION ALL With History Tables – Query #3

The access path for query #3 shows the implicit UNION ALL in the access path information.
Productivity Improvements Using Temporal Capability Observed In The Proceeding Benchmark

<table>
<thead>
<tr>
<th>Comparison</th>
<th>User application solution</th>
<th>DB2 supplied solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>System time support</td>
<td>2 triggers for each table</td>
<td>Versioning capability</td>
</tr>
<tr>
<td>Business time support</td>
<td>2 stored procedures for each table</td>
<td>SQL statements</td>
</tr>
<tr>
<td>Period overlap detection</td>
<td>1 trigger for each table</td>
<td>Part of primary key index for each table</td>
</tr>
<tr>
<td>Total number of lines of code</td>
<td>650</td>
<td>13</td>
</tr>
<tr>
<td>Total number of SQL statements</td>
<td>27</td>
<td>3</td>
</tr>
<tr>
<td>Time to develop and test</td>
<td>7 weeks</td>
<td>&lt;1 hour</td>
</tr>
</tbody>
</table>
Summary Of Temporal Benchmark

The performance expectations along with overhead associated with both System and Business Period temporal table usage are summarized below (your mileage may vary depending on which method is being used as the baseline to compare):

System period temporal tables:
1) DB2-provided system time support for UPDATE out-performs user defined triggers by 30 – 39%, and by 10-19% for DELETE respectively in DB2 CPU time.
2) UPDATE and DELETE against base table performance is affected by the history insert performance, which in turn is controlled by the number of indexes, number of columns in the index, clustering order, space map search, etc on the history table.
3) INSERT and UPDATE of current data (i.e., from the base table) may perform slower than table not performing data versioning.
4) SELECT against base tables (i.e., current data) would perform similar to tables not performing data versioning.
5) SELECT of historical data using the new approach may be slower. The performance overhead of temporal query statement is determined by efficiency of indexing on both base and history table, type of join, and number of table joining.
Summary On Temporal Benchmark

Business period temporal tables
1) 1-3% overhead of maintaining business periods without overlaps can be expected.
2) DB2-provided business time support for row splitting out-performs user defined stored procedure by 57% to 68% in DB2 CPU time.
Temporal Data – Time Travel Query

- What is temporal data?
- Concepts
- Defining temporal tables
- Temporal usage examples
- Performance
- My Observations On Temporal Tables
- References
My Observations On Temporal Tables

- System-period Table and History Table
  - Easiest to implement with a design point of adding to existing applications
  - History table data will need to be purge or archived
  - Recovery utility processing needs to support both System-period and History tables, not automatic
  - Can save CPU over user implemented solutions

- Application-period Table
  - Application coding tightly coupled to application-period tables
  - Updates and Deletes must be well understood by application development and users
My Observations On Temporal Tables...

- Bitemporal-period Table and History Table
  - Combines System-period and Application-period features into a single period table and history table
Temporal Data – Time Travel Query

- What is temporal data?
- Concepts
- Defining temporal tables
- Temporal usage examples
- Performance
- My Observations On Temporal Tables
- References
References and Resources

- DB2 10 for z/OS Administration Guide
- DB2 10 for z/OS SQL Reference
- *DB2 10 for z/OS Technical Overview - SG24-7892-00
- **DB2 10 for z/OS Performance Topics – SG24-7942-00
- A Matter of Time: Temporal Data Management in DB2 for z/OS  
  lang=en_US&source=sw-infomgt&S_PKG=db2z-temporal-tables-wp
- DB2 10 for z/OS Migration Planning Workshop

* Provided sample DDL and DML used in presentation
** Provided performance statistics using in presentation
The End!
Thank you!