Traveling Through Time With DB2 LUW Time Travel Query

Philip K. Gunning

Gunning Technology Solutions, LLC

Temporal Tables

- Introduced in DB2 10
- As defined by the ANSI/ISO SQL 2011 Standard
- DB2 LUW first commercially available database to include temporal table support as defined by the standard

The Need for Temporal Tables

- Does your company have full traceability of all data changes and retroactive data corrections?
- Can you delineate the price of an item during any specific current or historical period of time?
- What were the prices of stock used in a valuation or initial public offering three years ago?
- When the company was merged with another company, what were the financials as per due diligence at the time of the merger?
- How long was Policy B offered by the company at a certain rate?
- How much was the deductible for customer X during a certain period of time relative to a filed claim?

The Need for Temporal Tables

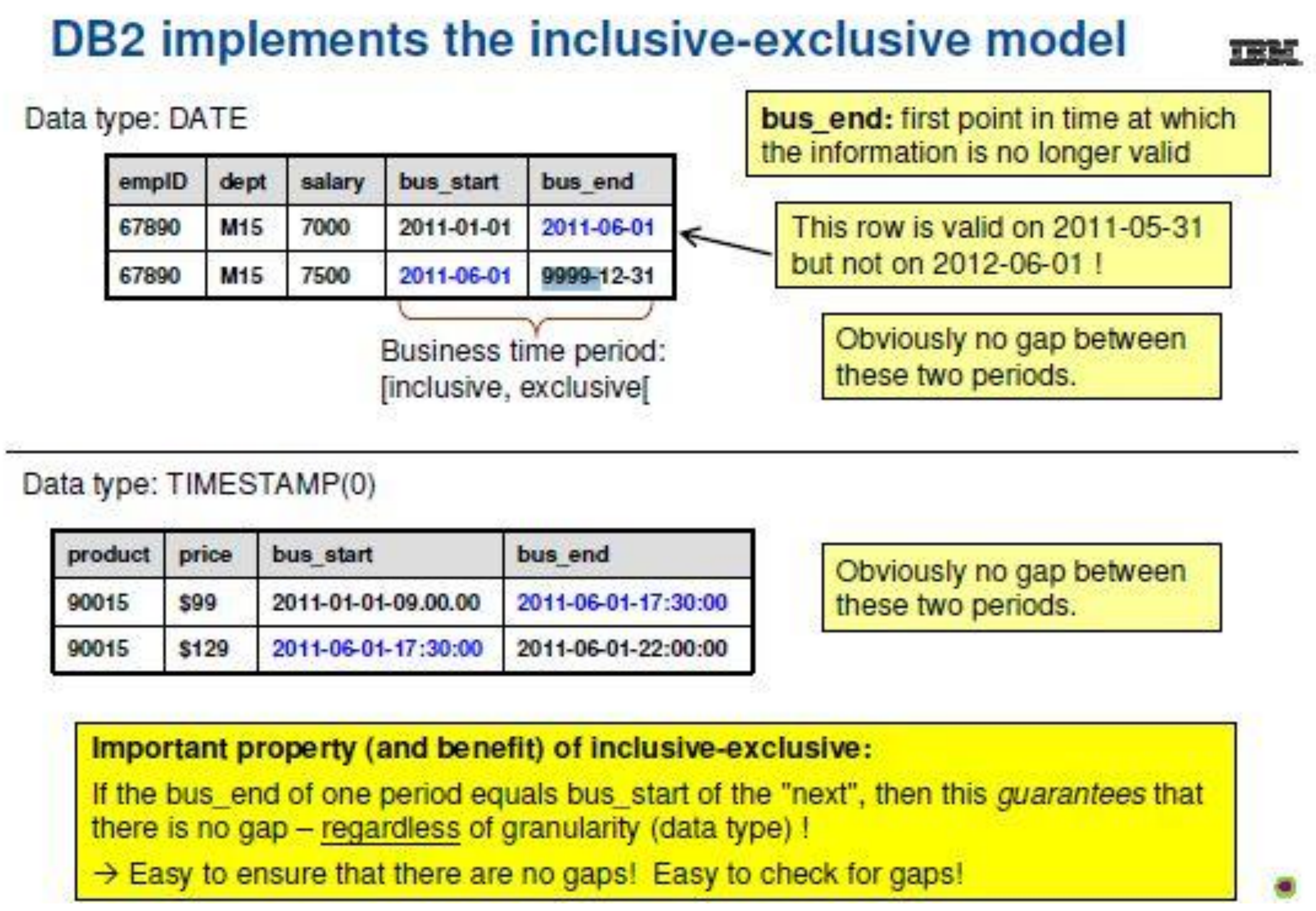
- Companies have a need to store, analyze and managed time-based data
- Amount of data growing significantly and requires increased manpower to manage
- Time-to-market for new applications or enhancements much shorter
- Inherent temporal table capability in the database engine provides much needed relief

Types of Temporal Tables

- System-period
  - Managed automatically by DB2
- Application-period
  - Based on user defined business periods
    - No history table
- Bitemporal Tables
  - Combines benefits of both System-period and Application-period temporal tables

Types of Temporal Tables

- **System-period**
  - Also known as "**Transaction Time**"
  - Tracks when changes are made to the state of the table itself
    - Example: When a mortgage is modified or when a line of credit is created
  - **Timestamps assigned by DB2**
- **Business Time**
  - Also known as "**Valid Time**"
  - Tracks the effective dates of business conditions
    - E.g. policy terms or line of credit term
  - **Timestamps assigned by the application**



DB2 implements the inclusive-exclusive model

IBM

Data type: DATE

empID	dept	salary	bus_start	bus_end
67890	M15	7000	2011-01-01	2011-06-01
67890	M15	7500	2011-06-01	9999-12-31

Business time period:
[inclusive, exclusive[

bus_end: first point in time at which the information is no longer valid

This row is valid on 2011-05-31 but not on 2012-06-01 !

Obviously no gap between these two periods.

Data type: TIMESTAMP(0)

product	price	bus_start	bus_end
90015	$99	2011-01-01-09.00.00	2011-06-01-17:30:00
90015	$129	2011-06-01-17:30:00	2011-06-01-22:00:00

Obviously no gap between these two periods.

Important property (and benefit) of inclusive-exclusive:

If the bus_end of one period equals bus_start of the "next", then this *guarantees* that there is no gap – regardless of granularity (data type) !

→ Easy to ensure that there are no gaps! Easy to check for gaps!

Benefits of System-period Temporal Tables

- DB2 generates beginning and ending time periods for inserts or updates them automatically with versioning
- Uses “versioning” and stores historical data in separate history tables
- Saves developers and DBAs countless hours of development and implementation time

Benefits of System-period Temporal Tables

- System-period temporal supported in all editions of DB2 10
  - Database Partitioning Feature
  - DB2 PureScale
  - High Availability Disaster Recovery (HADR)
  - Base and history table can have the following:
    - Range Partitioned
      - Roll-in/Roll-out of partitions
    - Multidimensional Clustering
    - Compression
- Improved performance of utilities against base table since historical data is contained in separate history table

NEW DB2 CATALOG VIEW

- You can now use the new SYSCAT.PERIODS catalog view to query the characteristics of any temporal table defined in the database.
- Querying the SYSCAT.PERIODS allows you to find easily:
  - All temporal tables and their period columns
  - The names of the associated history tables
- A new column, TEMPORALTYPE, has been added to the view SYSCAT.TABLES to identify temporal tables. Table types are indicated by the following codes:
  - A=Application-period temporal table
  - B=Bitemporal table
  - S=System-period temporal table
  - N=Not a temporal table.

CREATING SYSTEM TIME TEMPORAL TABLES

```
CREATE TABLE GTS1.item  (
                        SKU  INTEGER NOT NULL PRIMARY KEY ,
                        DESCRIPTION  VARCHAR(52) NOT NULL ,
                        SIZE  VARCHAR(10) ,
                        PRICE  DECIMAL(9,2) NOT NULL ,
                        COLOR   VARCHAR(25) ,
                        WEIGHT  DECIMAL(5,2) ,
                        CUBE   DECIMAL(5,2) ,
                        RESTRICT  CHAR(3) ,
                        SYSTEM_START  TIMESTAMP(12) NOT NULL GENERATED ALWAYS
AS ROW BEGIN ,
                        SYSTEM_END  TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS
ROW END ,
                        TRANS_ID  TIMESTAMP(12) GENERATED ALWAYS AS
TRANSACTION START ID IMPLICITLY HIDDEN ,
PERIOD SYSTEM_TIME (system_start, system_end)) IN  SKU_SPACE;
```

Enable Versioning

- Enable Versioning:

  “CREATE TABLE item_history

LIKE item in SKU_SPACE COMPRESS YES WITH **RESTRICT ON DROP”**;

- Associate the history table with the base table:

  “ALTER TABLE item **ADD VERSIONING**

  USE HISTORY TABLE item_history”;

SYSTEM PERIOD – INSERT EXAMPLE

Insert works as before with DB2 generating the
defaults for the period start and end columns.

On 11/1/2012, two new items were added to the retailer's existing inventory as per
The following INSERT:

Table: Item

Insert into item (sku, description, price) Values (18934, 'Jacket', 194.18),
(21340, 'socks', 14.99)

SKU	Description	Price	….	System_start	System_end
18934	Jacket	194.18	….	11/01/2012	12/30/9999
21340	Socks	14.99	….	11/01/2012	12/30/9999
58126	Hat	9.99	….	09/01/2012	12/30/9999
08129	Gloves	29.99	….	09/15/2012	12/30/9999

SYSTEM PERIOD – UPDATE EXAMPLE

On 12/07/2012, SKU 18934 went on sale for $149.99 and the Item table is updated as follows:
Update item set price = 149.99 where sku = 18934;

Table: Item

SKU	Description	Price	….	System_start	System_end
18934	Jacket	149.99	….	12/07/2012	12/30/9999
21340	Socks	14.99	….	11/01/2012	12/30/9999
58126	Hat	9.99	….	09/01/2012	12/30/9999
08129	Gloves	29.99	….	09/15/2012	12/30/9999

Table: Item_history

NOTE: The **Before** image is copied to the Item_history table.

SKU	Description	Price	….	System_start	System_end
18934	Jacket	194.18	….	11/01/2012	12/07/2012

System validity period: inclusive, exclusive

RUNNNING EXISTING QUERIES AGAINST PAST POINTS IN TIME

Table: Item

SKU	Description	Price	….	System_start	System_end
18934	Jacket	149.99	….	12/07/2012	12/30/9999
21340	Socks	14.99	….	11/01/2012	12/30/9999
58126	Hat	9.99	….	09/01/2012	12/30/9999
08129	Gloves	29.99	….	09/15/2012	12/30/9999

Table: Item_history

SKU	Description	Price	….	System_start	System_end
18934	Jacket	194.18	….	11/01/2012	12/07/2012

A. What is the price of SKU 18934 right now?

SELECT sku, price from ITEM where sku = 18934; **Answer: 149.99 (current table accessed)**

B. What was the price of sku 18934 on 11/14/2012?

"SELECT sku, price from ITEM **FOR SYSTEM TIME** AS OF '11/11/2012' where sku = 18934";

Answer: 194.18 (history table accessed)

CURRENT TEMPORAL SYSTEM_TIME SPECIAL REGISTER

- For example to see what the price was on 11/14/2012, you can set this new special register to the period of interest by issuing the following statement:
- “SET CURRENT TEMPORAL SYSTEM_TIME ‘11/14/2012’;
- **Same as adding FOR SYSTEM_TIME AS OF to all queries in the session**

RUNNNING EXISTING QUERIES AGAINST PAST POINTS IN TIME

Table: Item

SKU	Description	Price	….	System_start	System_end
18934	Jacket	149.99	….	12/07/2012	12/30/9999
21340	Socks	14.99	….	11/01/2012	12/30/9999
58126	Hat	9.99	….	09/01/2012	12/30/9999
08129	Gloves	29.99	….	09/15/2012	12/30/9999

Table: Item_history

SKU	Description	Price	….	System_start	System_end
18934	Jacket	194.18	….	11/01/2012	12/07/2012

A. What is the price of SKU 18934 right now?

SELECT sku, price from ITEM where sku = 18934; **Answer: 149.99 (current table accessed)**

B. What was the price of sku 18934 on 11/14/2012?

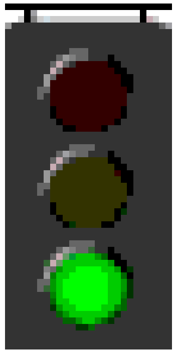
"SELECT sku, price from ITEM **FOR SYSTEM_TIME** AS OF '2012-11-11' where sku = 18934";

Answer: 194.18 (history table accessed)

SPECIAL REGISTER RESTRICTIONS

- When the current TEMPORAL SYSTEM_TIME Special Register is set to a non-null value, Insert, Update, Deletes operations and LOAD utilities on system period temporal tables are blocked.
- Can only specify the desired system time either in the query or with the special register, but not both. Doing so will result in an error being returned.

SCHEMA EVOLUTION



- Schema changes that can't cause loss of history are automatically propagated from the base table to the history table.
- Adding a new column to the base table results in the column being automatically added to the history table.
- Increasing the size of a VARCHAR column is applied to the base table and the history table.
- Changes such as reducing the length of a VARCHAR column or dropping a column from the base table are blocked due to potential data loss. In order to make those changes, you must stop versioning before making the changes.

HISTORY TABLE RESTRICTIONS

- A history table can't explicitly be dropped. It can only implicitly be dropped when the associated system period temporal table is dropped.
- A table space containing a history table, but not its associated system period temporal table, can't be dropped unless dropped at same time in same DROP statement.
- History table columns can't explicitly be added, dropped, or changed.
- A history table must not be defined as a parent, child, or self-referencing in a referential constraint. Referential constraints involving the history table are restricted to prevent cascaded actions to the history table.

HISTORY TABLE PERFORMANCE TIPS

- Under normal operations, a history table experiences mostly insert and read activities
- Use **APPEND ON**
- When enabled, there is no search for free space during insert processing and this can speed up inserts into the history table
- A primary key index of the base table primary key plus temporal dates should be considered
  - This can often improve the **FOR SYSTEM TIME AS OF** queries
  - In lieu of this, consider a non-unique key that is similar to the primary key of the base table

HISTORY TABLE BENEFITS

- Can have different indexes on the current table vs. history table
- Can recover current data and history data independently
- Can have different constraints on current table versus history table
- Can use different storage options for base and history table such as compression, range partitioning, clustering and storage group and physical location on disk

LOAD UTILITY

- LOAD Utility
  - PERIODOVERRIDE: new MODIFIER option for LOAD to load existingtimestamps into the period columns
  - PERIODIGNORE MODIFIER in the LOAD command instructs DB2 to ignore the timestamps in the exported data and instead generate new timestamps during load
  - PERIODMISSING MODIFIER is used when periods are missing, all period column values are generated by the utility.

Temporal Tables and VIEWS

- Views on Temporal Tables
- IF View contains **FOR SYSTEM TIME** or **FOR BUSINESS TIME**
  - **Data is restricted to just the rows constrained by that clause**
  - **Regular queries on the view should not contain that clause**
- **IF neither clause is specified**
  - **Data is un-constrained and all rows are exposed to the view**
  - **Queries on the view therefore require a clause**

BUSINESS TIME

- Application period or Business Time temporal tables differ from system period temporal tables in that they are based on your applications logical notion of time. Some examples of the use of application periods are as follows:
  – Variable rate mortgage products
  – Insurance policies
  – Credit cards with expiration dates.

The Need for APPLICATION PERIOD Temporal Tables

- Business time temporal tables enable developers to manage data according to business time and business related application logic
- This provides more flexibility for developers to meet the needs of their end users

CREATING BUSINESS TIME TEMPORAL TABLES

- To create a table with Business Time specify the PERIOD BUSINESS_TIME with **bus_start** and **bus_end** columns of type DATE or TIMESTAMP.
- To prevent overlapping periods for the same object (employee in our case) specify the **BUSINESS_TIME WITHOUT OVERLAPS** clause on the **PRIMARY KEY** definition.

CREATING BUSINESS TIME TEMPORAL TABLES

```
CREATE TABLE GTS1.employees  (
                    EMP_ID  INTEGER NOT NULL,
                    DEPT  VARCHAR (25) ,
                    SALARY DOUBLE ,
                    …
                    BUS_START   DATE NOT NULL,
                    BUS_END      DATE NOT NULL,
    PERIOD BUSINESS_TIME  (bus_start,  bus_end),
    PRIMARY KEY (EMPID, BUSINESS_TIME WITHOUT
OVERLAPS)  );
```

NOTE: For a given point in time, only one salary figure can be valid for a given employee

BUSINESS TIME

- Inserting data into an application-period temporal table is similar to inserting data into a regular table, the only special consideration is the need for the application to include the row-begin value and row-end values that capture when the row is valid from the perspective of the business applications.
- This valid period is called the BUSINESS_TIME period

INSERT WITH BUSINESS TIME

- We start out in this example with the issuance of a new variable rate mortgage to a customer with ACCT_ID 4389105 shown in following slides

INSERTING AND UPDATING WITH BUSINESS TIME

We start out in the example with the issuance of a new variable rate mortgage to acct_id 4389105 as follows:

```
INSERT INTO cust_mortgage VALUES (4389105, 'PA', 2.79, '2011-09-22', '2012-09-23');
```

Table: Cust_Mortgage

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.79	….	2011-09-22	2012-09-23

From this

In this example, the mortgage has an initial rate which initially is valid until 2012-09-22. However, periodically the rate can be updated based on changes to the US treasury bill. On 2011-12-23, the treasury bill rate went up so the mortgage Vrate needs to be adjusted with the following UPDATE:

```
UPDATE cust_mortgage FOR PORTION OF BUSINESS_TIME FROM '2011-12-23'
TO '2012-03-23' SET Vrate = 2.86 WHERE acct_id = 4389105;
```

State of table after UPDATE

"ROW SPLIT"

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.86	….	2011-12-23	2012-03-23
4389105	PA	2.79	….	2011-09-22	2011-12-23
4389105	PA	2.79	….	2012-03-23	2012-09-23

TO this

Both before and after rows in same table since Business Time tables have no history.

Inserting and Updating with Business Time, Continued

It was discovered that the initial rate was in error as it should have been set to the introductory rate of 2.74 for the first 30 days. A correction is issued as follows:

```
UPDATE cust_mortgage FOR PORTION OF BUSINESS_TIME FROM '2011-09-22'
 to '2011-10-23 '
SET Vrate = 2.74 where acct_id = 4389105;
```

From this

Table: Cust_Mortgage

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.86	….	2011-12-23	2012-03-23
4389105	PA	2.79	….	2011-09-22	2011-12-23
4389105	PA	2.79	….	2012-03-23	2012-09-23

AFTER UPDATE: CORRECTION

"ROW SPLIT"

Table: Cust_Mortgage

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.86	….	2011-12-23	2012-03-23
4389105	PA	2.74	….	2011-09-22	2011-10-23
4389105	PA	2.79	….	2012-03-23	2012-09-23
4389105	PA	2.79	….	2011-10-23	2011-12-23

TO this

DELETE with Business Time

The account holder with acct_id 4389105 pays a fee to convert from a variable rate mortgage to a new permanent fixed rate mortgage beginning 2012-03-23. Continuing with our example we record the change from variable to fixed rate as follows:

```
DELETE cust_mortgage
for PORTION OF BUSINESS_TIME FROM ‘2012-03-23’ TO ‘2012-09-23’
 WHERE acct_id = 4389105;
```

From this

Table: Cust_mortgage

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.86	….	2011-12-23	2012-03-23
4389105	PA	2.74	….	2011-09-22	2011-10-23
4389105	PA	2.79	….	2012-03-23	2012-09-23
4389105	PA	2.79	….	2011-10-23	2011-12-23

State of table after DELETE

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.86	….	2011-12-23	2012-03-23
4389105	PA	2.74	….	2011-09-22	2011-10-23
4389105	PA	2.79	….	2011-10-23	2011-12-23

TO this

SELECT WITH BUSINESS TIME

- The fact that different time periods are contained in the same table can present some challenges when writing queries and when trying not to have to change existing application code. Just like with system time, DB2 has a special register for business time.

Using the BUSINESS_TIME SPECIAL REGISTER

The fact that different time periods are contained in the same table can present some challenges when writing queries and when trying not to have to change existing application code. Just like with system time, DB2 has a special register for business time. You set the special register to the business time period of interest as follows. In our example we set it to ‘2011-12-28’ as follows:

```
“SET CURRENT TEMPORAL BUSINESS_TIME ‘2011-12-28’;
```

```
“SELECT acct_id, bus_start, bus_end
FROM Cust_Mortgage
WHERE acct_id = 4389105”;
```

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.86	….	2011-12-23	2012-03-23
4389105	PA	2.74	….	2011-09-22	2011-10-23
4389105	PA	2.79	….	2011-10-23	2011-12-23

BUSINESS_TIME SPECIAL REGISTER

- **Note:** Setting the CURRENT TEMPORAL BUSINESS TIME special register doesn’t affect regular tables. Only queries against temporal tables with a BUSINESS_TIME period enabled use the time set in the special register. There’s no effect on DDL statements.
- When setting the CURRENT TEMPORAL BUSINESS TIME special register it is the same as adding **FOR BUSINESS_TIME AS OF to all queries in the session**

FOR_BUSINESS_TIME CLAUSE

- You can also use the FOR_BUSINESS_TIME clause in a query to specify your period of interest as follows:

“SELECT acct_id, bus_start, bus_end

FROM Cust_Mortgage FOR BUSINESS_TIME AS OF ‘2011-11-14’ WHERE acct_id = 4389105”;

USING THE BUSINESS_TIME CLAUSE

Again, continuing with our example using the CUST_MORTGAGE table, the rows that fall within the business time period of interest will be returned. In this case, the row with the *bus_start* date of ‘2011-10-23’ and *bus_end* date of ‘2011-12-23’ is returned.

```
“SELECT acct_id, bus_start, bus_end
FROM Cust_Mortgage FOR BUSINESS_TIME AS OF ‘2011-
11-14’ WHERE acct_id = 4389105”;
```

ACCT_ID	State	Vrate	….	Bus_start	Bus_end
4389105	PA	2.86	….	2011-12-23	2012-03-23
4389105	PA	2.74	….	2011-09-22	2011-10-23
4389105	PA	2.79	….	2011-10-23	2011-12-23

BUSINESS_TIME_CLAUSE

- DB2 supports the following period specification options in the FOR BUSINESS_TIME clause as follows:

- FOR BUSINESS_TIME AS OF <value>
- FOR BUSINESS_TIME FROM <value1> TO <value2>
- FOR BUSINESS_TIME BETWEEN <value1> AND <value2>.

CURRENT TEMPORAL BUSINESS_TIME SPECIAL REGISTER BEHAVIOR

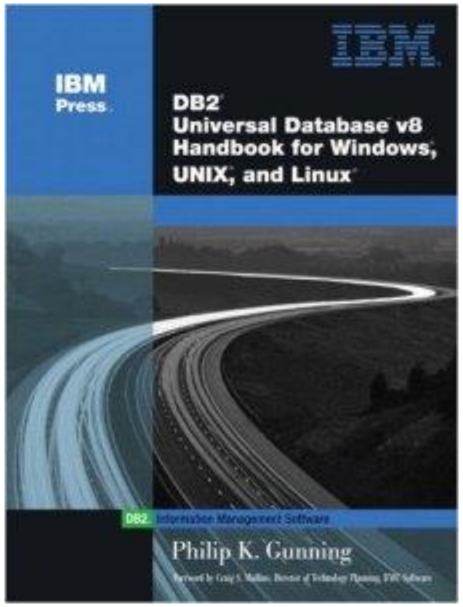
- When the CURRENT TEMPORAL BUSINESS_TIME special register is set to a non-null value, unlike with the SYSTEM_TIME special register, data modification statements like INSERT, UPDATE, DELETE and MERGE against application-period temporal tables are supported.

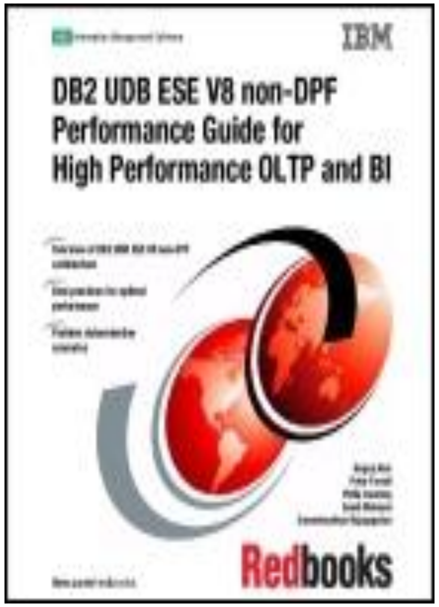
THANK YOU

Traveling Through Time With DB2 LUW Time Travel Query

Philip K. Gunning
Gunning Technology Solutions, LLC
pgunning@gts1consulting.com
www.gts1consulting.com

DB2 Books by Phil


IBM
IBM Press
DB2 Universal Database v8 Handbook for Windows, UNIX, and Linux
DB2 Information Management Software
Philip K. Gunning


IBM
DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI
Redbooks