Using temporal tables in DB2 10 for z/OS and DB2 11 for z/OS

Yan Feng Han
Hao Zhang

October 16, 2014

Temporal tables were introduced in IBM DB2® 10 for z/OS® and enhanced in V11. If you have to maintain historical versions of data over several years, temporal tables can be helpful for period-based data. In this tutorial, explore how your applications can use temporal tables to manage different versions of data, simplify service logic, and provide information for auditing. Learn about when and how to use three types of temporal tables to manage period-based data.

Background

Our world is full of data, and information management is not easy anymore. In some circumstances, each update and deletion of data requires copying and saving old data. New regulatory laws sometimes require maintaining historical versions of data for years, presenting a challenge to manage different versions of application data. Application programmers and database administrators have struggled for years with this issue. The traditional approach is to let the application take the burden of managing different versions of application data. This complicates table design and is error-prone for the application. A better solution was needed to relieve applications from their burden.

DB2 for z/OS has been helping to resolve the problem of maintaining historical versions of data. DB2 10 for z/OS began providing the capability to implement table-level specifications to manage application data based on time. New types of tables, called temporal tables, were introduced. DB2 11 for z/OS provides improvements to temporal tables.

In this article, learn about the temporal tables in DB2 10 for z/OS and DB2 11 for z/OS.

The basis of temporal tables: The period

The period is defined as two columns where the first column represents the beginning of the period and the second column represents the end of the period. The period is the foundation of temporal tables.

DB2 supports two types of periods:
Application period
Also called the BUSINESS_TIME period, the application period consists of a pair of columns with application-maintained values that indicate the period of time when the business data in a row is valid. A table can have only one BUSINESS_TIME period. The application period is used to help an application manage temporal data in its service logic.

To define a table with an application period, you need to include `PERIOD BUSINESS_TIME (begin_column, end_column)` in the table's definition. `begin_column` and `end_column` must identify the columns that exist in the table, the former points when the business deems the row valid, and the latter points when the business deems row validity ends. Use `begin_column` and `end_column` as either `TIMESTAMP(6) NOT NULL` or `DATE NOT NULL`. When a BUSINESS_TIME period is defined for a table, DB2 generates a check constraint in which the end column value must be greater than the begin column value.

System period
Also called SYSTEM_TIME period, the system period consists of a pair of columns with system-maintained values that indicate the period of time when a row is valid. A table can have only one SYSTEM_TIME period. The system period is used to audit data changes by DB2.

To define a table with a system period, you need to include `PERIOD SYSTEM_TIME (begin_column, end_column)` in the table's definition. `begin_column` and `end_column` must identify the columns that exist in the table. `begin_column` records the time one new transaction begins for the data, which is when one new row was inserted into the table or when the existing row is updated. `end_column` records the time one transaction finishes, which is when the existing row is modified or deleted from the table. `begin_column` must be defined as `TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN`, and `end_column` must be defined as `TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END`.

The period is in inclusive-exclusive mode, which means the beginning value is included in the period and end value is not included in the period.

Overview of temporal tables
In DB2 10 for z/OS, you can create three types of temporal tables: an application-period temporal table, a system-period temporal table, and a bitemporal table.

Application-period temporal table
In an application-period temporal table, which is a table defined with an application period, every row has a pair of timestamps (dates) set and managed by the application. The application records them to the BUSINESS_TIME period to indicate the valid period of the row in that business area. For an application-period temporal table, DB2 provides applications with new ways to easily query, update, and delete at the current time, any time prior, or at a future point or period in business time. Figure 1 shows the basic principle.
Using temporal tables in DB2 10 for z/OS and DB2 11 for z/OS

Figure 1. Basic principle of an application-period temporal table

System-period temporal table

In a system-period temporal table, every row has a pair of timestamps, which are set by the DB2 system and recorded in the `begin_column` and `end_column` columns. DB2 uses the `SYSTEM_TIME` period to audit the data changes. It provides the application with the database managed row begin and row end timestamp maintenance, providing non-overlapping time periods with no gaps in time.

A system-period temporal table also provides automatic management of data movement from a system-period temporal table to another table. You can define system-period data versioning on a system-period temporal table, which specifies that old rows are archived into another table. The system-period temporal table only contains the current active rows. The table that contains the archived rows is called the *history* table. You can delete the rows from the history table when those rows are no longer needed.

For a system-period temporal table, DB2 provides applications with new ways to easily query the current period or any prior point or period in system time for auditing. DB2 optimizer presents the system-period temporal table and its history tables as one. When an auditing SQL comes, it pulls answers from both of them as needed and presents as one answer set. Figure 2 shows an example.

Figure 2. Basic principle of a system-period temporal table

Bitemporal table

A bitemporal table is defined with both an application period and a system period. It is both a system-period temporal table and an application-period temporal table. A bitemporal table also includes the information on two dimensional tombstoning of data: business history and audit
history. The business history allows the application to more easily manage data related to time, and the audit history is for catching and correcting operation errors.

You can use a bitemporal table to keep application period information and system-based historical information, and to gain a lot of flexibility in querying data based on periods of time. As shown in Figure 3, in the application logic the SQL for current data takes advantage of application-period supports and the auditing SQL takes advantage of system-period support.

**Figure 3. Basic principle of a bitemporal table**

Using an application-period temporal table

You can use an application-period temporal table to maintain application period information. If the service logic of the application needs to manage different versions of business data, or if it is struggling with validating the date/time predicates or the effective data effective at any point in time, the application-period temporal table is a good choice.

**Generating an application-period temporal table**

You can include an application-period when using the CREATE TABLE statement to create a new table. The table needs to have:

- A begin column defined as TIMESTAM (6) NOT NULL or DATE NOT NULL.
- An end column defined as TIMESTAMP(6) NOT NULL or DATE NOT NULL.
- A period BUSINESS_TIME defined on the two columns above (PERIOD BUSINESS_TIME(begin_column, end_column), with implicit check constraint generated by DB2: end_column > begin_column).
- Optionally, the new index clause BUSINESS_TIME WITHOUT OVERLAPS to enforce unique index keys for a period of time. It is optional for a user to define a unique key with BUSINESS_TIME WITHOUT OVERLAPS. If the clause is not specified, then overlaps in time can occur and DB2 does not do any overlap checking. For example, let's create the application-period temporal table POLICY_ATT to record the payment for clients in different banks. BANK is the bank's name, CLIENT is the client's name, and PAY is the payment. The CREATE SQL statement is:

```sql
CREATE TABLE POLICY_ATT (BANK VARCHAR(4), BUS_BEGIN TIMESTAMP NOT NULL, BUS_END TIMESTAMP NOT NULL, CLIENT CHAR(4), TYPE CHAR(5), PAY INT, PERIOD BUSINESS_TIME (BUS_BEGIN, BUS_END), PRIMARY KEY (BANK, CLIENT, BUSINESS_TIME WITHOUT OVERLAPS) );
```
You can also migrate one existing table to an application-period temporal table with the ALTER TABLE statement, as follows:

- Use ALTER TABLE ADD COLUMN to add `begin_column` and `end_column` for BUSINESS_TIME.
- Use ALTER TABLE ADD PERIOD BUSINESS_TIME to add an application period. For example, given the existing POLICY_ATT the definition is:

```sql
CREATE TABLE POLICY_ATT (BANK VARCHAR(4), CLIENT CHAR(4), TYPE CHAR(5), PAY INT );
```

The statement to migrate the table to the application-period table is:

```sql
ALTER TABLE POLICY_ATT ADD COLUMN BUS_BEGIN TIMESTAMP NOT NULL ADD COLUMN BUS_END TIMESTAMP NOT NULL ADD PERIOD BUSINESS_TIME(BUS_BEGIN, BUS_END);
```

### Querying an application-period temporal table

#### DB2 10 for z/OS

In DB2 10 for z/OS, to query data in a portion of time on an application-period temporal table, you need to query with the period specification explicitly to specify the time criteria. The SELECT statement is:

```sql
SELECT ... FROM ... period specification
```

A period-specification is:

```sql
FOR BUSINESS_TIME AS OF value1 or FROM value1 TO value2 or BETWEEN value1 AND value2.
```

It consists of the following clauses:

- **FOR BUSINESS_TIME** to indicate you want to query an application-period temporal table.
- **AS OF**, **FROM**...**TO**, or **BETWEEN**...**AND** to indicate the time criteria for which you want data.
- **AS OF** value1 specifies the rows existing for the time point of value1, which are: the begin value for the period in the row is less than or equal to value1 and the end value for the period in the row is greater than value1.
- **FROM** value1 **TO** value2 specifies the rows existing for the period specified from value1 up to value2, which are: the begin 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.
- **BETWEEN** value1 **AND** value2 specifies the rows existing for the period specified between value1 and value2, which are: the begin 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.

For example, if the user wants to query payment at the time point of 2010-06-01-00.00.00.000000 for client BUPT in the bank of CCB1, you can query the application-period temporal table as follows:

```sql
SELECT CLIENT, PAY FROM POLICY_ATT FOR BUSINESS_TIME AS OF '2010-06-01-00.00.00.000000' WHERE BANK='CCB1' AND CLIENT='BUPT';
```
DB2 generates implicit predicates, and the query is equivalent to:

```
SELECT CLIENT, PAY FROM POLICY_ATT WHERE BANK='CCB1' AND CLIENT='BUPT' AND 
BUS_BEGIN <= '2010-06-01-00.00.00.000000' AND BUS_END > 
'2010-06-01-00.00.00.000000';
```

### DB2 11 for z/OS

DB2 11 for z/OS offers another way to query an application-period temporal table, as
discussed later in **Temporal query, update, and delete support**.

### Updating and deleting on an application-period temporal table

#### DB2 10 for z/OS

With DB2 10 for z/OS, to delete or update data in a portion of time on an application-
period temporal table, you need to use DELETE/UPDATE with the FOR PORTION OF 
BUSINESS_TIME period clause explicitly.

The DELETE/UPDATE statements are:

- DELETE FROM … FOR PORTION OF BUSINESS_TIME FROM value1 TO value2;
- UPDATE … FOR PORTION OF BUSINESS_TIME FROM value1 TO value2;

For example, if a user wants to delete all business data that is valid from
2010-06-01-00.00.00.000000 to 2011-06-01-00.00.00.000000 in an application-period 
temporal table, the DELETE SQL statement is:

```
DELETE FROM POLICY_ATT FOR PORTION OF BUSINESS_TIME FROM 
'2010-06-01-00.00.00.000000' TO '2011-06-01-00.00.00.000000';
```

DB2 generates implicit predicates. The query is equivalent to:

```
DELETE FROM POLICY_ATT WHERE BUS_BEGIN < 
'2011-06-01-00.00.00.000000' AND BUS_END > 
'2010-06-01-00.00.00.000000';
```

If a user wants to modify payment to 8000 from 2010-06-01-00.00.00.000000 to 
2011-06-01-00.00.00.000000 in an application-period temporal table, the UPDATE SQL 
statement is:

```
UPDATE POLICY_ATT FOR PORTION OF BUSINESS_TIME FROM 
'2010-06-01-00.00.00.000000' TO '2011-06-01-00.00.00.000000' SET PAY = 8000 
WHERE CLIENT = 'BUPT';
```

DB2 generates implicit predicates. The query is equivalent to:

```
UPDATE POLICY_ATT SET PAY = 8000 BUS_BEGIN = MAX (BUS_BEGIN, value1), 
BUS_END = MIN (BUS_END, value2) WHERE CLIENT = 'BUPT' AND BUS_BEGIN < 
'2011-06-01-00.00.00.000000' AND BUS_END > '2010-06-01-00.00.00.000000';
```

DELETE or UPDATE operations may result in row splitting inside the DB2 system, which 
leads to internal INSERTs depending on the relationship between BUSINESS_TIME period
of the existing rows and FOR PORTION OF BUSINESS_TIME period \((\text{value}_1, \text{value}_2)\) in the UPDATE/DELETE statement. Figure 4 shows the four different types of those relationships, which lead to different results of inside INSERT(s). On the time lines, the green bars are the periods that need to be updated or deleted and the red bars are the periods that are needed inside INSERT(s).

**Figure 4. Row splitting inside DB2 system led by temporal UPDATE/DELETE**

For example, in the UPDATE statement above, if one row with BUSINESS_TIME\((2010-01-01-00.00.00.000000, 2012-01-01-00.00.00.000000)\) exists in the table, DB2 will update the payment to 8000 for the period of \((2010-06-01-00.00.00.000000, 2011-06-01-00.00.00.000000)\) and insert two rows with original payment for the period of \((2010-01-01-00.00.00.000000, 2010-06-01-00.00.00.000000)\) and \((2011-06-01-00.00.00.000000, 2012-01-01-00.00.00.000000)\).

**DB2 11 for z/OS**

DB2 11 for z/OS offers another way to update and delete an application-period, as discussed later in UPDATE or DELETE with special registers.

**Using a system-period temporal table**

The system-period temporal table provides an easy way to audit operations and changes on data. If an application requires auditing data changes, and is struggling with the task, consider using the system-period temporal table.

**Generating a system-period temporal table**

To generate and use a system-period temporal table effectively, you need to create a temporal table with a system period and define system-period data versioning on this base table, as follows:

1. Generate a system-period temporal table to store the current data.
2. Generate a history table that receives the old data from the system-period temporal table.
3. Define the system-period data versioning to establish a link between the system-period temporal table and the history table.
Generating a system-period temporal table to store current data

You can include a system-period when using a CREATE TABLE statement to create a new table. The table needs to have:

- A begin column defined as TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN.
- An end column defined as TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END.
- SYSTEM_TIME period defined on the two columns mentioned above: PERIOD SYSTEM_TIME(begin_column, end_column).
- One column defined as TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID.

For example, to create the system-period temporal table POLICY_STT to record the payment for clients in different banks, the CREATE SQL statement is:

```
CREATE TABLE POLICY_STT (BANK VARCHAR(4), SYS_BEGIN TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN, SYS_END TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END, TRANS_ID TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID, CLIENT CHAR(4), TYPE CHAR(5), PAY INT, PERIOD SYSTEM_TIME (SYS_BEGIN, SYS_END), PRIMARY KEY (BANK, CLIENT) );
```

You can also migrate one existing table to a system-period temporal table by using the ALTER TABLE statement, as follows:

- Use ALTER TABLE ADD COLUMN to add begin_column, end_column for SYSTEM_TIME and the transaction ID column.
- Use ALTER TABLE ADD PERIOD SYSTEM_TIME to add a system period.

If the existing POLICY_STT definition is:

```
CREATE TABLE POLICY_STT (BANK VARCHAR(4), CLIENT CHAR(4), TYPE CHAR(5), PAY INT );
```

The statements to migrate the table to system-period table are:

```
ALTER TABLE POLICY_STT ADD COLUMN SYS_BEGIN TIMESTAMP(12) WITHOUT TIME ZONE NOT NULL GENERATED ALWAYS AS ROW BEGIN;
ALTER TABLE POLICY_STT ADD COLUMN SYS_END TIMESTAMP(12) WITHOUT TIME ZONE NOT NULL GENERATED ALWAYS AS ROW END;
ALTER TABLE POLICY_STT ADD COLUMN TRANS_ID TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID;
ALTER TABLE POLICY_STT ADD PERIOD SYSTEM_TIME (SYS_BEGIN, SYS_END);
```

Generating a history table for a system-period temporal table

The history table must have the same number of columns as the system-period temporal table that it corresponds to. The columns should have the same names, data types, null attributes, CCSIDs,
subtypes, hidden attributes, and field procedures as the corresponding system-period temporal table. The DB2 for z/OS Administration Guide has the detailed criteria for a history table.

You can use a CREATE TABLE statement to create a history table, but an easier way to define a history table for a system-period temporal table, which includes the above criteria, is to use the LIKE clause with the CREATE TABLE statement.

For example, you can create the history table for POLICY_STT as follows:

```
CREATE TABLE POLICY_STT_HIST (BANK VARCHAR(4), SYS_BEGIN TIMESTAMP(12) NOT NULL,
SYS_END TIMESTAMP(12) NOT NULL, TRANS_ID TIMESTAMP(12), CLIENT CHAR(4), TYPE
CHAR(5), PAY INT, PRIMARY KEY (BANK, CLIENT) );
```

Or, you can use the following code:

```
CREATE TABLE POLICY_STT_HIST LIKE POLICY_STT ...;
```

### Defining the system-period data versioning

Issue the ALTER TABLE ADD VERSIONING statement with the USE HISTORY TABLE clause to define system-period data versioning on the table, which establishes a link BETWEEN the system-period temporal table and the history table. For example:

```
ALTER TABLE POLICY_STT ADD VERSIONING USE HISTORY TABLE POLICY_STT_HIST;
```

### Querying a system-period temporal table

In DB2 10 for z/OS, to query current and historical data for auditing the data changes, you can use the SELECT statement. Specify that the period specification for the historical data is requested in the FROM clause. The SELECT statement is:

```
SELECT ... FROM ...period specification
```

A period-specification is:

```
FOR SYSTEM_TIME AS OF value1 or FROM value1 TO value2 or BETWEEN value1 AND value2
```

It consists of the following clauses:

- FOR SYSTEM_TIME to indicate you want to query a system-period temporal table.
- AS OF, FROM...TO, or BETWEEN...AND to indicate the time criteria for which you want data. They should have similar definitions to those for BUSNIESS_TIME period.

For example, to query the payment changes during a period from 2010-06-01-00.00.00.000000000000 to 2011-06-01-00.00.00.000000000000, you can query the system-period temporal table as follows:

```
SELECT CLIENT, PAY FROM POLICY_STT FOR SYSTEM_TIME FROM
'2010-06-01-00.00.00.000000000000' TO '2011-06-01-00.00.00.000000000000'
WHERE BANK='CCB1' AND CLIENT='BUPT';
```
DB2 generates implicit predicates; the query is equivalent to:

```
SELECT CLIENT, PAY FROM POLICY_STT FROM (SELECT CLIENT, PAY FROM POLICY_STT
WHERE SYS_BEGIN < '2011-06-01-00.00.00.000000000000' AND SYS_END >
'2010-06-01-00.00.00.000000000000' UNION ALL SELECT CLIENT, PAY FROM
POLICY_STT_HIST WHERE SYS_BEGIN < '2011-06-01-00.00.00.000000000000' AND
SYS_END > '2010-06-01-00.00.00.000000000000') POLICY_STT WHERE
BANK='CCB1' AND CLIENT='BUPT';
```

### Using a bitemporal table

The bitemporal table has the advantages of both the application-period temporal table and the system-period temporal table. If an application needs to manage business temporal data as well as audit data changes, the bitemporal table is recommended.

### Generating a bitemporal table

To generate and use a bitemporal table, you need to define a bitemporal table and its history table then define system-period data versioning on the bitemporal table, as follows:

1. Generate a bitemporal table to store the current data.
2. Generate a history table that receives the old data from the bitemporal table.
3. Define the system-period data versioning to establish a link between the bitemporal table and the history table.

#### Generating a bitemporal table to store the current data

You can create a new table by using the `CREATE TABLE` statement and defining both an application-period and a system-period in it. For example, to create a bitemporal table to record the payment for clients in different banks, use the following code:

```
CREATE TABLE POLICY_BTT ( BANK VARCHAR(4), BUS_BEGIN TIMESTAMP NOT NULL,
BUS_END TIMESTAMP NOT NULL, SYS_BEGIN TIMESTAMP(12) NOT NULL GENERATED
ALWAYS AS ROW BEGIN, SYS_END TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW
END, TRANS_ID TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START
ID, CLIENT CHAR(4), TYPE CHAR(5), PAY INT, PERIOD SYSTEM_TIME (SYS_BEGIN,
SYS_END), PRIMARY KEY (BANK, CLIENT, BUSINESS_TIME WITHOUT OVERLAPS) );
```

You can also migrate an existing table to a bitemporal table using the `ALTER TABLE` statement, as follows:

- Use `ALTER TABLE ADD COLUMN` to add `begin_column` and `end_column` for `BUSINESS_TIME`, and add `begin_column` and `end_column` for `SYSTEM_TIME` and the transaction ID column.
- Use `ALTER TABLE ADD PERIOD BUSINESS_TIME` to add the application period.
- Use `ALTER TABLE ADD PERIOD SYSTEM_TIME` to add the system period.

The subsequent steps to generate a history table and define the system-period data versioning for a bitemporal table are similar to the corresponding steps for a system-period temporal table (see Generating a history table for a system-period temporal table).
Querying, updating, or deleting a bitemporal table

The query to the application temporal table and the system temporal table can be combined for a bitemporal table. For example, to query payments that are valid for a business at the point in time 2010-06-01-00.00.00.000000 as well as all the changes to payments around 2012-01-01-00.00.00.000000000000, use the following SQL:

```
SELECT * FROM POLICY_BTT FOR BUSINESS_TIME AS OF '2010-06-01-00.00.00.000000' FOR SYSTEM_TIME AS OF '2012-01-01-00.00.00.000000000000' WHERE BANK='CCB1' AND CLIENT='BUPT';
```

Updating or deleting rows from a bitemporal table are the same as Updating and deleting on an application-period temporal table.

DB2 11 for z/OS enhancements for temporal tables

DB2 11 for z/OS has enhancements to temporal tables, including:

- Temporal query, update, and delete support for views.
- Temporal query, update, and delete with special registers.

Temporal query, update, and delete support for view

When you query a view that references a temporal table, you can specify the period specifications in the FROM clause of the SELECT statement. If the view references an application temporal table or a bitemporal table, you can also UPDATE or DELETE the view with the period specification. The period specifications include:

- SYSTEM_TIME
- BUSINESS_TIME

For example:

```
CREATE VIEW POLICY_VIEW AS SELECT ... FROM POLICY_BTT;

SELECT ... FROM POLICY_VIEW FOR BUSINESS_TIME AS OF '2010-06-01-00.00.00.000000' FOR SYSTEM_TIME AS OF '2012-01-01-00.00.00.000000000000';

UPDATE POLICY_VIEW FOR PORTION OF BUSINESS_TIME FROM '2010-06-01-00.00.00.000000' TO '2010-07-01-00.00.00.000000' SET PAY = 123;
```

Temporal query with special registers

You can query temporal tables for different points in time without modifying the SQL statement. Instead, use special registers to specify the time criteria. The new method lets you change the time criteria later; you do not have to modify the SQL and then rebind the application. This enhancement is implemented by two new special registers.
New special registers
The CURRENT TEMPORAL BUSINESS_TIME special register specifies a value that is used in the default BUSINESS_TIME period specification for references to application-period temporal tables. The initial value of the special register is null; you can change it by using the SET CURRENT TEMPORAL BUSINESS_TIME statement.

The CURRENT TEMPORAL SYSTEM_TIME special register specifies a TIMESTAMP(12) value that is used in the default SYSTEM_TIME period specification for references to system-period temporal tables. The initial value of the special register is null; you can change it by using the SET CURRENT TEMPORAL SYSTEM_TIME statement.

Implementing the special registers
DB2 11 for z/OS adds some bind options and routine options to control the sensitivity of the settings of the CURRENT TEMPORAL SYSTEM_TIME/BUSINESS_TIME registers.

There are two new bind options for BIND PACKAGE, REBIND PACKAGE, and REBIND TRIGGER PACKAGE commands:

- SYSTIMESSENSITIVE specifies whether references to system-period temporal tables are affected by the value of the CURRENT TEMPORAL SYSTEM_TIME special register. The default is YES.
- BUSTIMESSENSITIVE specifies whether references to application-period temporal tables are affected by the value of the CURRENT TEMPORAL BUSINESS_TIME special register. The default is YES.

There are two new options for the CREATE/ALTER FUNCTION and CREATE/ALTER PROCEDURE statement:

- SYSTEM_TIME SENSITIVE
- BUSINESS_TIME SENSITIVE

Options for the two routines have the same meaning as the new bind options and take effect for the native functions and procedures.

You can also set SYSTEM_TIME SENSITIVE and BUSINESS_TIME SENSITIVE by DB2I panels for the BIND PACKAGE and REBIND PACKAGE options.

Temporal query by specifying the time criteria with special registers
You can query an application-period temporal table by using the special registers, as shown in Figure 5.

Figure 5. Query an application-period temporal table with special registers

```sql
Select ... from policy_att
bustimesensitive = yes
Set current temporal business_time not null
Select ... from policy_att
As of current temporal business_time;
```
For application-period temporal tables:

1. Set the BUSTIMESENSITIVE bind option to YES.
2. Set the CURRENT TEMPORAL BUSINESS_TIME special register to the time point you want to query.
3. Specify FOR BUSINESS_TIME AS OF CURRENT TEMPORAL BUSINESS_TIME as the period specification.

Figure 6 shows a query with special registers for a system-period temporal table.

**Figure 6. Query a system-period temporal table by special registers**

For system-period temporal tables:

1. Set the SYSTIMESENSITIVE bind option to YES.
2. Set the CURRENT TEMPORAL SYSTEM_TIME special register to the time point you want to query.
3. Specify FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME as the period specification.

Because a bitemporal table is both an application temporal table and a system-period temporal table, you can involve both new registers and bind options to query it.

**UPDATE or DELETE by specifying time criteria with special registers**

For an application-period temporal table, you can UPDATE or DELETE all rows whose application-periods include one appointed time by specifying the time criteria with CURRENT TEMPORAL BUSINESS_TIME.

1. Set the BUSTIMESENSITIVE bind option to YES.
2. Set the CURRENT TEMPORAL BUSINESS_TIME special register to the time point you want to UPDATE or DELETE.

For example, to update POLICY_ATT, use:

```
UPDATE POLICY_ATT SET PAY = 3000 WHERE CLIENT = 'BUPT';
```

DB2 generates implicit temporal predicates, so the UPDATE statement is transformed as follows:

```
UPDATE POLICY_ATT SET PAY = 3000 WHERE CLIENT = 'BUPT' AND (CURRENT TEMPORAL BUSINESS_TIME IS NULL OR (BUS_START <= CURRENT TEMPORAL BUSINESS_TIME AND BUS_END > CURRENT TEMPORAL BUSINESS_TIME));
```

The statement updates payment to 3000 in all rows with application-period including the time point set in CURRENT TEMPORAL BUSINESS_TIME.
Summary

In this article, you learned about the temporal table feature that was introduced in DB2 10 for z/OS and enhanced in DB2 11 for z/OS. The three temporal tables in this feature — application-period temporal table, system-temporal table, and bitemporal table — combine all of the features of the former tables. With temporal tables, DB2 can help applications resolve many difficulties in managing different versions of application data. Using the temporal tables can help you simplify application service logic, provide information for auditing data changes, and correct data errors.
Related topics

- For DB2 10 for z/OS, learn more about:
  - Creating temporal tables
  - CREATE TABLE statement
- For DB2 11 for z/OS, learn more about:
  - Creating temporal tables
  - CREATE TABLE statement
  - Improvements to querying temporal tables
  - Support for explicit period specification on views

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)