Living in a Temporal World

Chris Muncan

Manulife

Session code: F08

Tue, Jun 04, 2019 (02:30 PM - 03:30 PM)

Db2 for z/OS

IDUG
Leading the Db2 User Community since 1988

F08 - Living in a Temporal World

Chris Muncan

Christopher_Muncan@Manulife.com

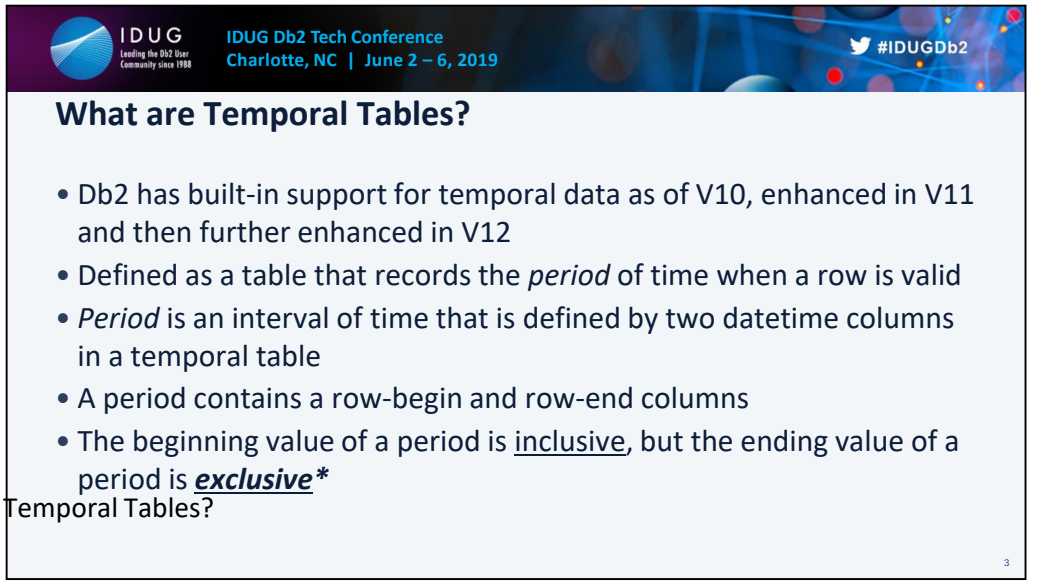
My 2nd IDUG presentation!!!

Agenda

- What are Temporal Tables
- Various Types
- Special Registers
- Creating and Using Them
- Restrictions
- Where Temporal Tables Can Help You

- My agenda for today:
- What are temporal tables
- The different types
- Special registers
- Creating and using temporal tables
- Db2 restrictions around them
- Where they can help you in your shop

- So before we begin, who is in Db2 V12?
- Anyone still on V11? Some features/enhancements to temporal tables you need to be in V12+ to use them. I'll identify those as we go along.
- Now the embarrassing question, who is on V10?



Db2's initial release of temporal tables was back in V10, enhanced in V11 and then even further enhanced in V12 with added functionality.

A temporal table is a table that records the period of time when a row is valid.

And a PERIOD is an interval of time defined by 2 date/time columns that are either system maintained or user maintained, depending on what type of temporal table you have setup

Every temporal table contains a period of 2 datetime columns to indicate when that particular row is valid.

The begin and end datetime periods are INCLUSIVE / EXCLUSIVE by default and in V12 they allowed you to modify the period END as INCLUSIVE when creating the temporal table.

Temporal tables have greatly been expanded upon with new functionality to make it easier to use since their initial release back in V10. So, if you are using temporal tables and not on V12…this is a huge selling feature of upgrading to V12 for sure!

Now that we've covered off what they are, lets look at the types of temporal tables

Types of Temporal Tables

3 types of temporal tables

1. System-Period Temporal Table (STT)
   - Db2 maintains the row start and end values
2. Application-period or business-period temporal table (ATT)
   - The period is maintained by business rules and is end-user controlled
3. Bitemporal table
   - which is a table with both a system-period AND business-period

Next lets looks at the special registers that can be used to aid you in temporal table navigation

Special Registers

- CURRENT TEMPORAL SYSTEM_TIME
  - TIMESTAMP(12)
  - If set, the period specification is implicit:

    FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME

- CURRENT TEMPORAL BUSINESS_TIME
  - DATE or TIMESTAMP
  - If set, the period specification is implicit:

    If BUSINESS_TIME columns are defined as TIMESTAMP
    FOR BUSINESS_TIME AS OF CURRENT TEMPORAL BUSINESS_TIME
    If BUSINESS_TIME columns are defined as DATE
    FOR BUSINESS_TIME AS OF CAST(CURRENT TEMPORAL BUSINESS_TIME AS DATE)

5

Db2 comes with 2 special registers, CURRENT TEMPORAL SYSTEM_TIME and BUSINESS_TIME

These special registers aid you when selecting or modifying data from your temporal tables.

SYSTEM_TIME is a TIMESTAMP(12)

- Always precision 12

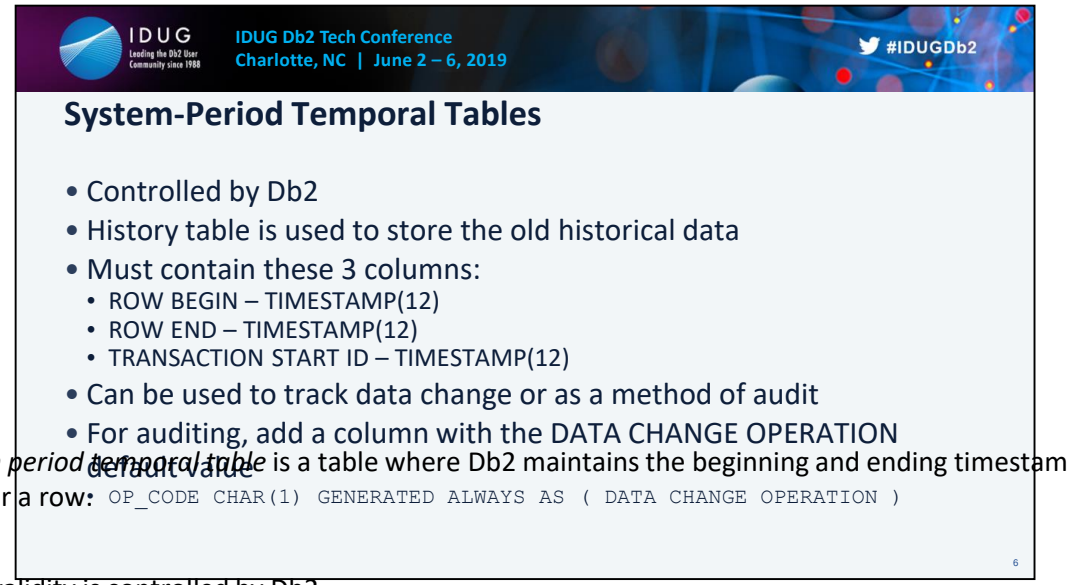
System time involves tracking when any changes are made to contents of a row.

BUSINESS_TIME is either a DATE or TIMESTAMP, depending on how you've defined your application temporal table

Business time 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.”)

If you don't modify the SYSTEM_TIME or BUSINESS _TIME special registers before running your query, Db2 will default to current timestamp or current date

Let's take a closer look at STTs and ATTs



A *system period temporal table* is a table where Db2 maintains the beginning and ending timestamp values for a row.

- Row validity is controlled by Db2
- A history table contains the previous or old rows
- 3 columns are required:
  - The *row-begin column* of the system period contains the timestamp value for when a row is created.
  - The *row-end column* contains the timestamp value for when a row is removed.
  - A *transaction-start-ID column* contains a unique timestamp value that Db2 assigns per transaction, or the null value.

- Can also add a column non-required column which can be used for table auditing such as

op_code CHAR(1) **GENERATED ALWAYS AS (DATA CHANGE OPERATION)**

How to Create a System-Period Temporal Table (1|2)

- Create the STT

```
CREATE TABLE SYSTEM_TEMPORAL_TABLE (
POLICY_ID   CHAR(10)      NOT NULL,
BENEFIT     CHAR(10)      NOT NULL,
COVERAGE    INT           NOT NULL,
USER_ID     VARCHAR(128)          GENERATED ALWAYS AS (SESSION_USER),
SYS_START   TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
SYS_END     TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
CREATE_TS   TIMESTAMP(12)         GENERATED ALWAYS AS TRANSACTION START ID,
PERIOD SYSTEM_TIME(SYS_START,SYS_END));
```

7

How to create a STT?

How to Create a System-Period Temporal Table (2|2)

- Create the History Table

```
CREATE TABLE SYSTEM_TEMPORAL_TABLE_HIST (
POLICY_ID   CHAR(10)      NOT NULL,
BENEFIT     CHAR(10)      NOT NULL,
COVERAGE    INT           NOT NULL,
USER_ID     VARCHAR(128),
SYS_START   TIMESTAMP(12) NOT NULL,
SYS_END     TIMESTAMP(12) NOT NULL,
CREATE_TS   TIMESTAMP(12);
```

- Link the STT to the History

```
ALTER TABLE SYSTEM_TEMPORAL_TABLE
ADD VERSIONING USE HISTORY TABLE SYSTEM_TEMPORAL_TABLE_HIST;
```

8

How to create a STT?

If you are wanting to track changes such as for auditing, then in the ADD VERSIONING, you would include: "**ON DELETE ADD EXTRA ROW"**

Elaborate on meaning of "versioning" not to be confused with package version or table versioning

In Db2's eyes, the concept of versioning is where a related table that is in essence a clone of the base table, same columns (names, definitions, etc.) and this is where Db2 stores the old historical information. When a row changes, the original row is copied into the history table.

How to Use a System-Period Temporal Table (1|4)

- Add rows to the STT

```
INSERT INTO SYSTEM_TEMPORAL_TABLE (POLICY_ID, BENEFIT, COVERAGE) VALUES('A111','HEALTH', 100);
INSERT INTO SYSTEM_TEMPORAL_TABLE (POLICY_ID, BENEFIT, COVERAGE) VALUES('A111','LTD', 5000);
INSERT INTO SYSTEM_TEMPORAL_TABLE (POLICY_ID, BENEFIT, COVERAGE) VALUES('Z999','DENTAL', 500);
```

```
POLICY_ID  BENEFIT    COVERAGE USER_ID SYS_START                  SYS_END                    CREATE_TS
---------- ---------- -------- ------- -------------------------- -------------------------- ----------
A111       HEALTH          100 MUNCACH 2019-03-15 11:00:00.0      9999-12-30 00:00:00.0      NULL
A111       LTD            5000 MUNCACH 2019-03-15 11:00:00.0      9999-12-30 00:00:00.0      NULL
Z999       DENTAL          500 MUNCACH 2019-03-15 11:00:00.0      9999-12-30 00:00:00.0      NULL
```

9

How to use a STT?

In this example, I am inserting 3 rows into a system-period temporal table.

Policy A111 has 2 benefits: Health & LTD and Policy Z999 has just Dental

How to Use a System-Period Temporal Table (2|4)

- Update a row

```
UPDATE SYSTEM_TEMPORAL_TABLE
SET COVERAGE = 0
WHERE POLICY_ID = 'A111' AND BENEFIT = 'HEALTH';
```

- Select from the STT

```
POLICY_ID  BENEFIT    COVERAGE USER_ID SYS_START                  SYS_END                    CREATE_TS
---------- ---------- -------- ------- -------------------------- -------------------------- ---------
A111       HEALTH            0 MUNCACH 2019-03-15 12:00:00.0      9999-12-30 00:00:00.0      NULL
A111       LTD            5000 MUNCACH 2019-03-15 11:00:00.0      9999-12-30 00:00:00.0      NULL
Z999       DENTAL          500 MUNCACH 2019-03-15 11:00:00.0      9999-12-30 00:00:00.0      NULL
```

- Select from the STT_HIST

```
POLICY_ID  BENEFIT    COVERAGE USER_ID SYS_START                  SYS_END                    CREATE_TS
---------- ---------- -------- ------- -------------------------- -------------------------- ---------
A111       HEALTH          100 MUNCACH 2019-03-15 11:00:00.0      2019-03-15 12:00:00.0      NULL
```

10

How to use a STT?

We then update the health coverage of A111 to have coverage amount of $0

After the update, you can select from the STT and see the updated row. Notice the SYS_START & SYS_END columns

You can also select from the STT_HIST table and see that the original row was copied into there and the SYS_END is populated with the same timestamp as the updated rows' SYS_START

Now that we have some usable data in our table, lets see how we can use the global variables and do some time travel on our table to see how rows looked at a point in time.

How to Use a System-Period Temporal Table (3|4)

How to Time Travel

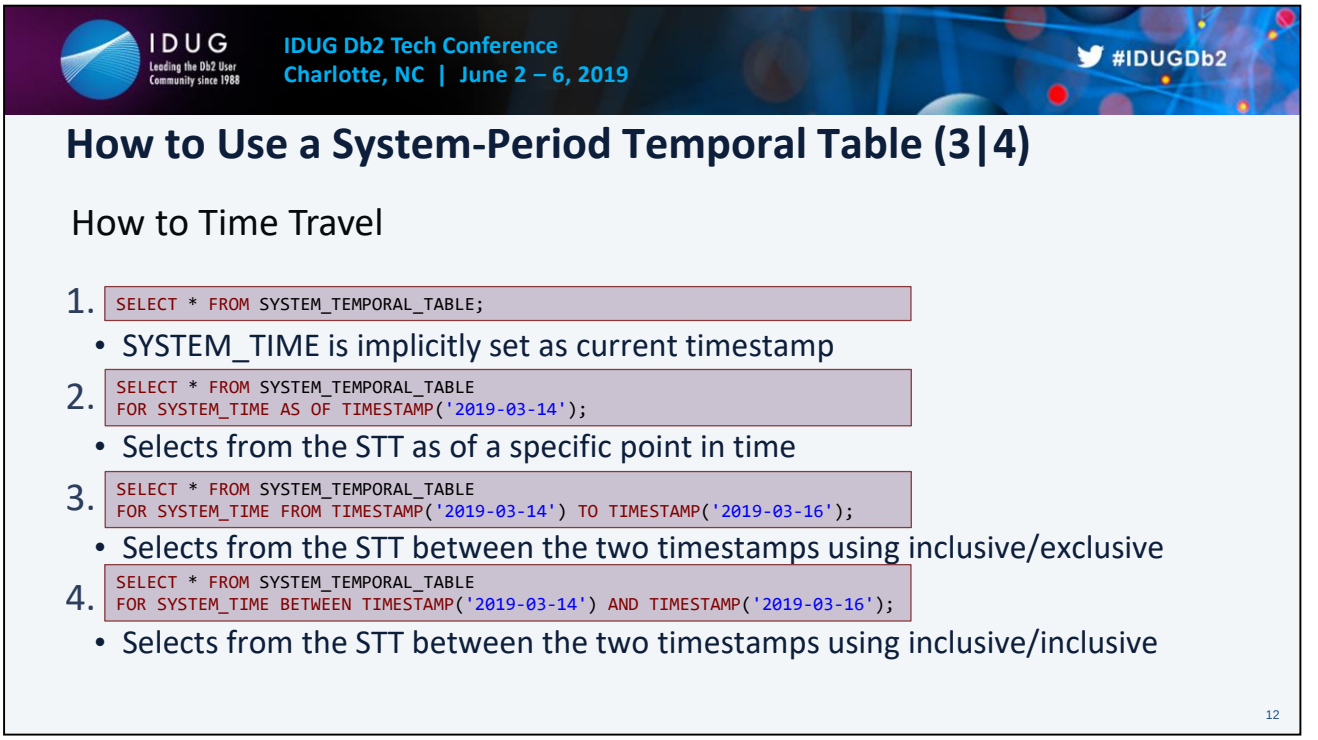
11

How to use a STT?

Now will cover off 4 basic ways of how to navigate through your temporal table using the SYSTEM_TIME special register

We are setting the CURRENT TEMPORAL SYSTEM_TIME **after** the original inserts but **BEFORE** the first round of updates. Notice how the HEALTH benefit row now has a coverage of $100 rather than the updated $0.

The main intention for setting the current temporal system_time is so you can easily travel back in time to see what state the database tables were in as of a particular point in time! Think of the RTS tables in Db2. If you have temporal tables enabled, you can travel through time to see your RTS over time to see how the tablespaces have changed.



How to use a STT?

Now will cover off 4 basic ways of how to navigate through your temporal table using the SYSTEM_TIME special register

We are setting the CURRENT TEMPORAL SYSTEM_TIME **after** the original inserts but **BEFORE** the first round of updates. Notice how the HEALTH benefit row now has a coverage of $100 rather than the updated $0.

The main intention for setting the current temporal system_time is so you can easily travel back in time to see what state the database tables were in as of a particular point in time! Think of the RTS tables in Db2. If you have temporal tables enabled, you can travel through time to see your RTS over time to see how the tablespaces have changed.

How to Use a System-Period Temporal Table (4|4)

- Select from STT setting the CURRENT TEMPORAL SYSTEM_TIME

```
SET CURRENT TEMPORAL SYSTEM_TIME = TIMESTAMP '2019-03-15 11:30:00.000000';
SELECT * FROM SYSTEM_TEMPORAL_TABLE
FOR SYSTEM_TIME AS OF CURRENT TEMPORAL SYSTEM_TIME --implicit
;
```

POLICY_ID	BENEFIT	COVERAGE	USER_ID	SYS_START	SYS_END	CREATE_TS
Z999	DENTAL	500	MUNCACH	2019-03-15 11:00:00.0	9999-12-30 00:00:00.0	NULL
A111	LTD	5000	MUNCACH	2019-03-15 11:00:00.0	9999-12-30 00:00:00.0	NULL
A111	**HEALTH**	**100**	**MUNCACH**	**2019-03-15 11:00:00.0**	**2019-03-15 12:00:00.0**	**NULL**

13

How to create a STT?

Here, we are setting the CURRENT TEMPORAL SYSTEM_TIME to a time **AFTER** the original inserts but **BEFORE** the first round of updates. Notice how the HEALTH benefit row now has a coverage of $100 rather than the updated $0.

The main intention for setting the current temporal system_time is so you can easily travel back in time to see what state the database tables were in as of a particular point in time!

A perfect example of where this can be very useful are the RTS tables that come shipped with V12. If you have temporal tables enabled, you can travel through time to see your RTS over time to see how the tablespaces have changed.

How to Turn Off Versioning for a STT

```
ALTER TABLE table-name DROP VERSIONING;
```

- Takes effect immediately
- No need for a reorg
- Cannot specify with any other alter statements
- Versioning cannot be dropped if there are any views, MQTs , or SQL table functions that depend on the SYSTEM_TIME period
- It ***may*** invalidate packages which depend on the target object

14

How do I turn off versioning of my STT?

It's easy…just ALTER TABLE <table-name> DROP VERSIONING

YES, it is really that simple!

And this statement takes effect immediately! No need for a reorg.

There are only 2 caveats:

1. Cannot specify any other ALTER statements with DROP VERSIONING
2. Can't drop the versioning if there are views, MQTs, or functions that depend on SYSTEM_TIME
3. It ***may*** invalidate packages that depend on the target object, or packages that depend on related objects through cascading effects

Restrictions of Using System Temporal Tables

- System Temporal Table & History tablespaces must be recovered as a set when performing PIT recovery
- To recover the tablespaces individually, specify the VERIFYSET NO option
- You cannot alter the schema of a system-period temporal table or history table
- History or base table cannot be dropped while versioning is turned on

15

Db2 Restrictions of Using Temporal Tables

Lastly I want to review some of the restrictions of a STT

This may not be a complete list. For a full list, refer to the IBM Knowledge Center

Key ones to make note of:
- PIT RECOVERY of both STT BASE & History tablespaces must be recovered as a SET
- You can recover them individually but you must explicitly specify VERIFYSET NO in the recovery
- Can't alter the schema in any way EXCEPT
- Columns can be added to your base table and they will be automatically added to your history table

Now that we've covered off SYSTEM TEMPORAL TABLES, let's look at Application Temporal Tables

Application-Period Temporal Tables

- Controlled by the user
- No history table used
- ROW-BEGIN & ROW-END – TIMESTAMP(6) or DATE

An *application-period temporal table* is a type of temporal table where ***you*** maintain the values that indicate when a row is valid.

Similar constructs as a SYSTEM temporal table but there is no history table and the row begin & end can be either timestamps or dates

How to Create an Application-Period Temporal Table

- Create the ATT

```
CREATE TABLE APPLICATION_TEMPORAL_TABLE (
POLICY_ID   CHAR(10) NOT NULL,
BENEFIT     CHAR(10) NOT NULL,
COVERAGE    INT      NOT NULL,
BUS_START   DATE     NOT NULL,
BUS_END     DATE     NOT NULL,
PERIOD BUSINESS_TIME(BUS_START, BUS_END INCLUSIVE))
```

- Add a unique index

```
CREATE UNIQUE INDEX X01ATT
ON APPLICATION_TEMPORAL_TABLE (POLICY_ID,BENEFIT,
BUSINESS_TIME WITHOUT OVERLAPS);
```

17

How to create an ATT?

PERIOD BUSINESS_TIME default is INCLUSIVE/EXCLUSIVE but with the activation of V12 M5xx, you can now override the period to be INCLUSIVE/INCLUSIVE

Creating an index on the table can be done a few different ways:

BUSINESS_TIME WITHOUT OVERLAPS – UNIQUE indexes must use this

BUSINESS_TIME WITH OVERLAPS – non-unique indexes only

WITH OR WITHOUT OVERLAPS

Indicates whether multiple rows may exist with the same values for the non-period columns and expressions of the index key for a row, with overlapping time periods.

WITH OVERLAPS

Indicates that multiple rows may exist with the same values for the non-period columns and expressions of the index key for a row, with overlapping time periods. The BUSINESS_TIME WITH OVERLAPS clause is intended for use in defining an index for the foreign key of a temporal referential constraint.

- BUSINESS_TIME WITH OVERLAPS must not be specified when UNIQUE is specified for the index definition.
- BUSINESS_TIME WITHOUT OVERLAPS must not be specified if the table is defined with a partitioning key that includes any columns of the BUSINESS_TIME period.

WITHOUT OVERLAPS

Indicates that the values for the non-period columns and expressions of the index key for a row must be unique with respect to the time represented by the BUSINESS_TIME period for the row. Db2 enforces that multiple rows do not exist with the same key values for the columns or expressions of the index, with overlapping time periods. The BUSINESS_TIME WITHOUT OVERLAPS clause is intended for use in defining a unique index to enforce a primary key or unique constraint.

- BUSINESS_TIME WITHOUT OVERLAPS can only be specified for an index that is defined as UNIQUE.

How to Use an Application-Period Temporal Table (1|6)

- Add rows to the ATT

```
INSERT INTO APPLICATION_TEMPORAL_TABLE
...
VALUES('A123', 'HEALTH', 100, '2019-01-01', '2019-12-31');
VALUES('A123', 'DENTAL', 100, '2019-01-01', '2019-12-31');
VALUES('Z999', 'HEALTH',1000, '2019-01-01', '2019-12-31');
```

POLICY_ID	BENEFIT	COVERAGE	BUS_START	BUS_END
A123	HEALTH	100	2019-01-01	2019-12-31
A123	DENTAL	100	2019-01-01	2019-12-31
Z999	HEALTH	1000	2019-01-01	2019-12-31

adding rows is simple. Just like adding a normal row to a table

How to Use an Application-Period Temporal Table (2|6)

- Updating the data the old way

```
POLICY_ID  BENEFIT    COVERAGE BUS_START  BUS_END
---------- ---------- -------- ---------- ----------
A123       DENTAL          100 2019-01-01 2019-12-31

A123       HEALTH          100 2019-01-01 2019-04-30
A123       HEALTH          500 2019-05-01 2019-07-31
A123       HEALTH          100 2019-08-01 2019-12-31

Z999       HEALTH         1000 2019-01-01 2019-12-31
```

```
UPDATE TABLE
     SET BUS_END = '2019-04-30'
     WHERE POLICY_ID = 'A123' AND BENEFIT = 'HEALTH';

INSERT INTO TABLE VALUES('A123', 'HEALTH', 500, '2019-05-01', '2019-07-31');

INSERT INTO TABLE VALUES('A123', 'HEALTH', 100, '2019-08-01', '2019-12-31');
```

19

Take for example you are NOT using an ATT but you have your own in-house built solution. Here's what you would need to do if you want to update a portion of a row in the table…

How to Use an Application-Period Temporal Table (3|6)

- Updating the data using an ATT

```
POLICY_ID  BENEFIT    COVERAGE BUS_START  BUS_END
---------- ---------- -------- ---------- ----------
A123       DENTAL          100 2019-01-01 2019-12-31

A123       HEALTH          100 2019-01-01 2019-04-30
A123       HEALTH          500 2019-05-01 2019-07-31
A123       HEALTH          100 2019-08-01 2019-12-31

Z999       HEALTH         1000 2019-01-01 2019-12-31
```

```
UPDATE APPLICATION_TEMPORAL_TABLE
FOR PORTION OF BUSINESS_TIME BETWEEN '2019-05-01' AND '2019-07-31'
SET COVERAGE = 500 WHERE POLICY_ID = 'A123' AND BENEFIT = 'HEALTH';
```


Now you we can do the exact same thing using an ATT in just 1 update command.

How to Use an Application-Period Temporal Table (4|6)

- Deleting data

```
DELETE FROM APPLICATION_TEMPORAL_TABLE
FOR PORTION OF BUSINESS_TIME BETWEEN '2019-04-01' AND '2019-08-30'
WHERE POLICY_ID = 'A123' AND BENEFIT = 'HEALTH';
```

```
POLICY_ID  BENEFIT    COVERAGE BUS_START  BUS_END
---------- ---------- -------- ---------- ----------
A123       DENTAL          100 2019-01-01 2019-12-31

A123       HEALTH          100 2019-01-01 2019-03-31
A123       HEALTH          100 2019-09-01 2019-12-31

Z999       HEALTH         1000 2019-01-01 2019-12-31
```

How to Use an Application-Period Temporal Table (5|6)

Data Retrieval of an ATT

1.
```
SELECT * FROM APPLICATION_TEMPORAL_TABLE;
```
   - BUSINESS_TIME is implicitly set as current date
2.
```
SELECT * FROM APPLICATION_TEMPORAL_TABLE
FOR BUSINESS_TIME AS OF '2019-03-14';
```
   - Selects from the ATT as of a specific point in time
3.
```
SELECT * FROM APPLICATION_TEMPORAL_TABLE
FOR BUSINESS_TIME FROM '2019-03-14' TO '2019-03-16';
```
   - Selects from the ATT between the two timestamps using inclusive/***exclusive***
4.
```
SELECT * FROM APPLICATION_TEMPORAL_TABLE
FOR BUSINESS_TIME BETWEEN '2019-03-14' AND '2019-03-16';
```
   - Selects from the ATT between the two timestamps using inclusive/***inclusive***

22

Data Retrieval of an ATT

Just like SYSTEM Temporal Tables, Application Temporal Tables have 4 different ways to select data

1. Generic select
2. Select setting BUSINESS_TIME “AS OF”
3. Select setting BUSINESS_TIME “FROM”
4. Select setting BUSINESS_TIME “BETWEEN”

One of the main principles of specifying the BUSINESS_TIME is so you can easily travel back in time to see what state the database tables were in as of a particular point in time!

Think of insurance rates, you want to know what your coverage was as of a particular point in time, not what it is today, or what it was when you first received your benefits

How to Use an Application-Period Temporal Table (6|6)

Data Manipulation of an ATT

```
DELETE FROM ... FOR PORTION OF BUSINESS_TIME FROM value1 TO value2;
UPDATE      ... FOR PORTION OF BUSINESS_TIME FROM value1 TO value2;
```

```
DELETE FROM APPLICATION_TEMPORAL_TABLE
FOR PORTION OF BUSINESS_TIME FROM '2019-03-14' TO '2019-03-16'
WHERE POLICY_ID = 'A123' AND BENEFIT = 'HEALTH';
```

```
UPDATE APPLICATION_TEMPORAL_TABLE
FOR PORTION OF BUSINESS_TIME FROM '2019-03-01' TO '2019-03-31'
SET COVERAGE = 0
WHERE POLICY_ID = 'A123' AND BENEFIT = 'HEALTH';
```


Data Manipulation

Adding RI to a Temporal Table

- RI can exist between Child & Parent temporal tables
- Each child row must reside within a parent row's period or multiple contiguous parent rows

You can even have RI between a temporal table and a child table.

Compared to traditional RI, temporal RI is an even stronger condition that can be enforced between two tables with business time periods.

For every row in a child table, one or more corresponding rows in a parent table exist with matching key and the business time period(s) of the parent row(s) completely cover the business time period of the child row without gaps.

For example, an administration system may need to ensure that every employee at every point in time during his/her employment belongs to a department that actually exists at that point.

Adding RI to a Temporal Table

```
CREATE TABLE DEPT
(
 DEPT_NO       INT       NOT NULL
,DEPT_NAME     CHAR(20)  NOT NULL
,DEPT_START    DATE      NOT NULL
,DEPT_END      DATE      NOT NULL
PERIOD BUSINESS_TIME (DEPT_START, DEPT_END),
PRIMARY KEY (DEPT_NO, BUSINESS_TIME WITHOUT OVERLAPS)
)
```

```
CREATE TABLE EMP
(
 EMP_NO        INT       NOT NULL
,EMP_NAME      CHAR(20)  NOT NULL
,EMP_DEPT_NO   INT       NOT NULL
,EMP_START     DATE      NOT NULL
,EMP_END       DATE      NOT NULL
PERIOD BUSINESS_TIME (EMP_START, EMP_END)
FOREIGN KEY (EMP_DEPT_NO, PERIOD BUSINESS_TIME)
REFERENCES DEPT (DEPT_NO, PERIOD BUSINESS_TIME)
)
```

Index on parent table

```
CREATE UNIQUE INDEX X01DEPT
ON DEPT (DEPT_NO, BUSINESS_TIME WITHOUT OVERLAPS);
```

Index on child table

```
CREATE INDEX X02EMP_FK
ON EMP  (EMP_NO,  BUSINESS_TIME WITH    OVERLAPS);
```

25

You can even have RI between a temporal table and a child table.

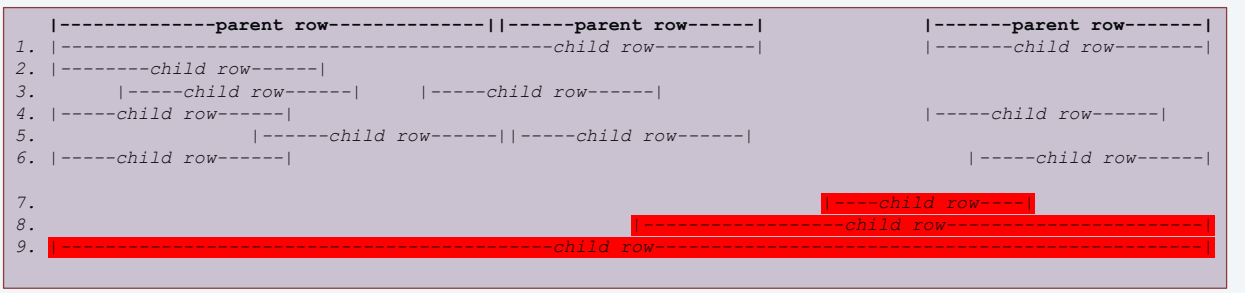
Compared to traditional RI, temporal RI is an even stronger condition that can be enforced between two tables with business time periods. For every row in a child table, one or more corresponding rows in a parent table exist with matching key and the business time period(s) of the parent row(s) completely cover the business time period of the child row without gaps. For example, an administration system may need to ensure that every employee at every point in time during his/her employment belongs to a department that actually exists at that point.

GO BACK TO THE POINT ABOUT THE **WITH OVERLAPS** VERSUS **WITHOUT OVERLAPS**

Adding RI to a Temporal Table

Adding RI to a Temporal Table

- A child row can also reside within multiple contiguous parent periods
- Eg:



This covers all of the details on an APPLICATION TEMPORAL TABLE, I'll go over some restrictions of temporal tables

Where Temporal Tables Can Help You

Application Temporal Tables

- Insurance Coverage
- Vehicle Rental Agency

System Temporal Tables

- Db2 catalog tables
  - V12 now ships the RTS tables with matching history tables for versioning

28

Db2 Restrictions of Using Temporal Tables

ATT

- Insurance Coverage
  - Travel Insurance
  - Annual Maximums
  - Verifying Claims
  - etc
- Vehicle Rental
  - Track who has it when
  - Who had the vehicle
  - Again, type of insurance that was purchased on it
  - Mileage in/out
  - Maintenance schedule
  - And the list can go on for this one too

STT

Db2 V12 is deployed with history tables just waiting for you to run on versioning for the RTS tables

You can use the historical information to analyze, predict, and help prevent specific conditions in a subsystem. Such as identifying the need for a REORG of the TS or IX data is growing to near the max dataset size so you need to add a new partition, etc.

I highly recommend BEFORE enabling versioning of the RTS tables that you first develop and test a purge strategy!

SYSIBM.SYSTABSPACESTATS_H, which is described in Temporal versioning for Db2 catalog tables

The following catalog tables are added but not used:

- SYSAUDITPOLICIES_H
- SYSCOLAUTH_H
- SYSCONTEXT_H
- SYSCONTEXTAUTHID_H
- SYSCONTROLS_H
- SYSCTXTTRUSTATTR_H
- SYSDBAUTH_H
- SYSPACKAUTH_H
- SYSPLANAUTH_H
- SYSRESAUTH_H
- SYSROUTINEAUTH_H
- SYSSCHEMAAUTH_H
- SYSSEQUENCEAUTH_H
- SYSTABAUTH_H
- SYSUSERAUTH_H
- SYSVIEWDEP_H

References

- Creating a system-period temporal table
  - https://www.ibm.com/support/knowledgecenter/SSEPEK_12.0.0/admin/src/tpc/db2z_creatingtemptableversion.html
- Creating an application-period temporal table
  - https://www.ibm.com/support/knowledgecenter/SSEPEK_12.0.0/admin/src/tpc/db2z_creatingbusitimeperiod.html
- Creating bitemporal tables
  - https://www.ibm.com/support/knowledgecenter/SSEPEK_12.0.0/admin/src/tpc/db2z_creatingbitempversioning.html
- Temporal versioning for Db2 catalog tables
  - https://www.ibm.com/support/knowledgecenter/en/SSEPEK_12.0.0/wnew/src/tpc/db2z_cataloghistorytables.html
- IDUG: Using the new DB2 for z/OS Temporal Special Registers by Claire McFeely
  - https://www.idug.org/p/bl/et/blogid=278&blogaid=468
- IDUG: Temporal Support of Db2 for z/OS – Part III Temporal RI
  - https://www.idug.org/p/bl/et/blogid=278&blogaid=798

This concludes my presentation. I want to thank you all for attending and taking part in this amazing experience for me.

I hope that you'll walk out of here with some excellent ideas and now that you know how you can use Temporal Tables and perhaps you can go back to your architects and start using them in your shop

If you have any questions, please feel free contact me.