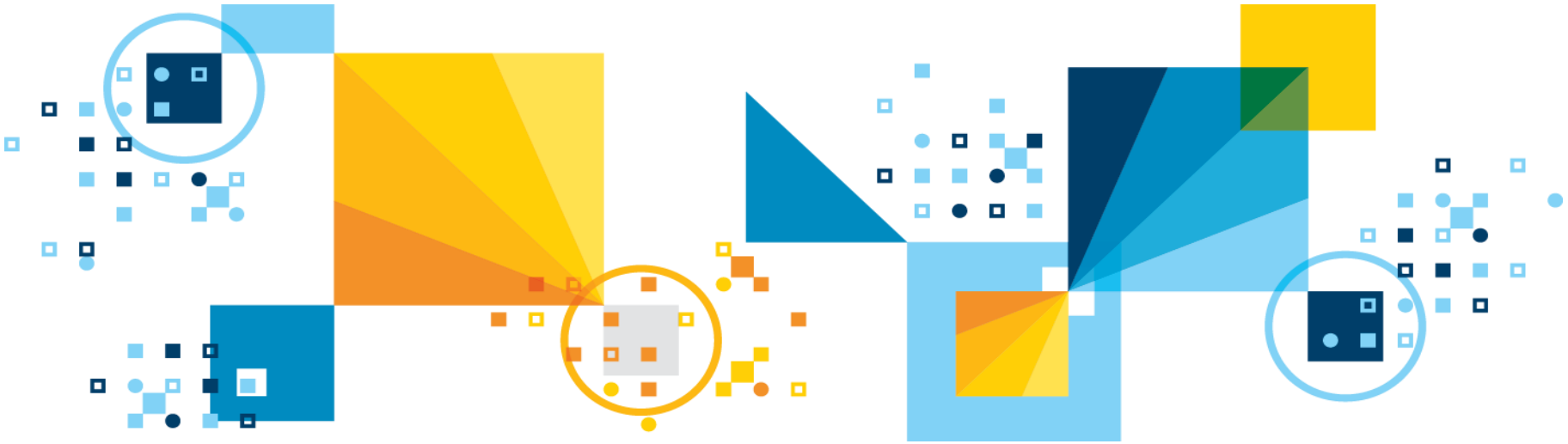
DB2 Temporal Data Management

Module ID | 10115

Length | 1 hour + 1 hour Hands on Lab



For questions about this presentation contact askdata@ca.ibm.com

January 30, 2015

Disclaimer

Module Information

- You should have completed or acquired the necessary knowledge for the following modules in order to complete this module:
  - DB2 Fundamentals

- After completing this module, you should be able to:
  - Explain the concepts of:
    - System-period temporal tables (STTs)
    - Application-period temporal tables (ATTs)
    - Bitemporal tables

Module Content

- Temporal Data Management – Use Cases & Overview
- Benefits
- How do they work?
- System-period temporal tables
- Application-period temporal tables
- Bitemporal: combines system-period and application-period temporal tables
- View support
- Special registers
- Bind options and routine ops
- Moving from existing tables to temporal tables

Use Cases of Temporal Data Management

- **Track and analyze changes in your business**
  - Easily compare data from two points in time
  - Accuracy in time-based reporting

- **Effectively perform and trace data corrections**
  - Easily make data changes in the past, i.e. effective as a past in point time, and record when the change was made

- **Auditing and compliance**
  - Ability to show past data for any point in time
  - Ability to show which information got changed in the same transaction and when, up to pico-second precision

Temporal Tables – What are They?

- **Temporal tables allow you to associate time-based state information to your data**
  - Managed by DB2, not application logic
- **DB2's temporal table allow**
  - Tracking of data changes (versioning)
  - Support for custom business data storage and manipulation
  - A combination of the two

- **Benefits**
  - **Increase business insight**
    - Easily report on data at a given point in time in the past, present, or future
  - **Lower costs**
    - Increase developer productivity – less coding by offloading time management to DB2
    - Simpler code means reduced maintenance costs
  - **Allow you to reduce compliance efforts and better track data changes**
  - **Better manage data**
    - Standards-based technology provides consistency and data quality across the enterprise
    - Seamless and deep integration with all database features such as partitioning and compression

Temporal Tables – Types

- **Built into DB2 – automatic and transparent**
- **Three types of temporal tables**

System-period temporal tables (STTs)	Application-period temporal tables (ATTs)	Bitemporal Tables
▪ DB2 automatically maintains historical versions of the rows in the history table ▪ You can query the past state of your data *Example* Employees who have left the company	▪You assign a date range to each row, indicating the period when the data is valid in the real world ▪ Valid periods can be in the past, present, or future *Example* •Insurance policy valid from Jan 1 to June 30 •4% interest rate is effective from Nov 1 to 20	▪ Combination of STT and ATT ▪ Keep application-based period information as well as system-based historical information

System-Period Temporal Tables (STTs)

- **DB2 automatically maintains historical versions of the rows in the table**
- **Step 1 - Create base table including three specific columns**
  - **Row-begin column**: time at which the row data became current
  - **Row-end column**: time at which the row data was no longer current
  - **Transaction start-ID column**: start time of execution for the transaction impacting the row. Often the same as row-begin (and then set to NULL to save space)
- **Step 2 - Create an identical table that will serve as the history table**
  - History table can be configured to suit your needs, e.g. partitioning, compression, storage location, etc..
- **Step 3 - Associate history table to the base table**
- **Behavior**
  - DB2 automatically migrates rows from base table to history table as changes occur, updating the three columns as required
  - All management of base and history tables is automatic and transparent.
  - Queries only reference the base table, and DB2 will transparently access the history table as needed

How to Define a System-Period Temporal Table

1. Create a table with a SYSTEM_TIME period

```
CREATE TABLE travel(
     trip_name CHAR(30) NOT NULL PRIMARY KEY,
     destination CHAR(12) NOT NULL,
     departure_date DATE NOT NULL,
     price DECIMAL (8,2) NOT NULL,
     sys_start TIMESTAMP(12) NOT NULL
         generated always as row begin implicitly hidden,
     sys_end TIMESTAMP(12) NOT NULL
         generated always as row end implicitly hidden,
     tx_start TIMESTAMP(12)
         generated always as transaction start id implicitly hidden,
         PERIOD SYSTEM_TIME (sys_start, sys_end)
) IN travel_space;
```

Optional:
IMPLICITLY HIDDEN excludes columns from SELECT * result sets

Captures the begin and end times when the data in a row is current

2. Create the history table

```
CREATE TABLE travel_history LIKE travel IN hist_space;
```

3. Add versioning to the base table to establish a link to the history table

```
ALTER TABLE travel ADD VERSIONING USE HISTORY TABLE travel_history;
```

Insert Data into a System-Period Temporal Table

Add new trips:
Amazonia, departing on 10/28/2011 & Ski Heavenly Valley, departing on 3/1/2011

Current Date = January 1, 2011

```
INSERT INTO travel
VALUES ('Amazonia','Brazil','10/28/2011',1000.00);
INSERT INTO travel
VALUES ('Ski Heavenly Valley','California','03/01/2011',400.00);
```

System validity period
(inclusive, exclusive)

TRAVEL

trip_name	destination	departure_ date	price	sys_start	sys_end
Amazonia	Brazil	10/28/2011	1000.00	01/01/2011	12/30/9999
Ski Heavenly Valley	California	03/01/2011	400.00	01/01/2011	12/30/9999

Both `SYS_START` and `SYS_END` columns are inserted by DB2, not the application. For simplicity, they are represented here as DATEs, rather than TIMESTAMPs

Alter and Update a System-Period Temporal Table

Current Date = February 15, 2011

- **Destination name is not explicit enough. Alter the DESTINATION column to make it longer**

```
ALTER TABLE travel ALTER COLUMN destination SET DATA TYPE VARCHAR(50);
```

****History table is automatically modified**

- **Update the destination column for Ski Heavenly Valley to make it clearer:**

```
UPDATE travel SET destination = 'Lake Tahoe, CA'
WHERE trip_name = 'Ski Heavenly Valley';
```

New sys_start date

TRAVEL

trip_name	destination	departure_date	price	sys_start	sys_end
Amazonia	Brazil	10/28/2011	1000.00	01/01/2011	12/30/9999
Ski Heavenly Valley	Lake Tahoe, CA	03/01/2011	400.00	**02/15/2011**	12/30/9999

TRAVEL_HISTORY

trip_name	destination	departure_date	price	sys_start	sys_end
Ski Heavenly Valley	California	03/01/2011	400.00	**01/01/2011**	**02/15/2011**

- **DB2 automatically inserted row into history table and supplied sys_start and sys_end dates**

Delete from a System-Period Temporal Table

Current Date = April 1, 2011

- **We are no longer offering the Ski Heavenly Valley trip – delete it.**

```
DELETE FROM travel WHERE trip_name = 'Ski Heavenly Valley';
```

TRAVEL

trip_name	destination	departure_date	price	sys_start	sys_end
Amazonia	Brazil	10/28/2011	1000.00	01/01/2011	12/30/9999

Ski Heavenly Valley has been removed from base table

System validity period (inclusive, exclusive)

TRAVEL_HISTORY

trip_name	destination	departure_date	price	sys_start	sys_end
Ski Heavenly Valley	California	03/01/2011	400.00	01/01/2011	02/15/2011
Ski Heavenly Valley	Lake Tahoe, CA	03/01/2011	400.00	**02/15/2011**	**04/01/2011**

- **DB2 automatically inserted row into history table and supplied sys_start and sys_end dates**

Query a System-Period Temporal Table

Current Date = May 1, 2011

- **Query the past: what trips were available on 03/01/2011 for less than $500?**

```
SELECT trip_name FROM travel
FOR SYSTEM_TIME AS OF '03/01/2011'
WHERE price < 500.00;
```

```
Result: Ski Heavenly Valley
```

- **Query the present: what trips are currently available to Brazil?**

```
SELECT trip_name FROM travel WHERE destination = 'Brazil';
```

```
Result: Amazonia
```

  - **defaults to the current table only**
    - **functions as if we added *FOR SYSTEM_TIME AS OF CURRENT TIMESTAMP***

- **Query the past and the present: In 2011, how many different tours were offered?**

```
SELECT COUNT (DISTINCT trip_name) FROM travel
FOR SYSTEM_TIME BETWEEN '01/01/2011' AND '01/01/2012';
```

```
Result: 2
```

Application-Period Temporal Tables (ATTs)

- **Allows you to store business time (application's logical notion of time). Examples:**
  - Insurance policy valid from Jan 1 to June 30
  - 4% interest rate is effective from Nov 1 to 20, and on Nov 21 the effective rate *will* increase to 5%
- **Each row has a pair of timestamp or date columns, stored by the application**
  - **Begin column***:* represents the time at which row data begins to be valid
  - **End column***:* represents the time at which row data ceases to be valid
  - Time-sensitive columns are **controlled by the user or application**
- **Unlike System-period temporal tables (STTs), no separate history table is required**
- **Benefits**
  - DB2 adds, splits, or deletes rows transparently as needed
  - Can be used to model data in the past, present and future
  - Constraints can be automatically enforced to disallow overlapping validity periods

System Time vs. Business Time

System Time	Business Time
Captures the time when changes happened to data inside DB2	Captures the time when changes happen(ed) to business artifacts
DB2-generated history of updated or deleted rows	Application-driven changes to the time dimension of business artifacts
History based on DB2 system timestamps	Dates or timestamps provided by the application
DB2's physical view of time	Your application's logical view of time
Spans from the past to the present time	Spans past, present, and future time
System validity ("transaction time")	Business validity ("valid time")
Supports queries such as: *"Which policies were stored in the database on June 30?"*	Supports queries such as: *"Which policies were active on June 30?"*

How to Define an Application-Period Temporal Table

- **Create a table with a BUSINESS_TIME attribute**

```
CREATE TABLE travel (
     trip_name CHAR(25) NOT NULL,
     destination CHAR(8) NOT NULL,
     departure_date DATE NOT NULL,
     price DECIMAL(8,2) NOT NULL,
     bus_start DATE NOT NULL,
     bus_end DATE NOT NULL,
     PERIOD BUSINESS_TIME (bus_start, bus_end),
     PRIMARY KEY (trip_name, BUSINESS_TIME WITHOUT OVERLAPS)
);
```

PERIOD (bus_start, bus_end) is (inclusive, exclusive)
The 'bus_start' column in the PERIOD clause must be less than the 'bus_end' column

trip_name plus the bus_start and bus_end PERIOD form a unique primary key.
DB2 enforces that there are no overlapping PERIODs for trip_name.

Insert Data into a Application-Period Temporal Table

Current Date = May 1, 2011

** Application-period time entries are independent of the current date

- **Add new trip: Manu Wilderness, departing on 08/02/2011**

```
INSERT INTO travel
VALUES ('Manu  Wilderness', 'Peru',
'08/02/2011',1500.00,'05/01/2011', '01/01/2012');
```

bus_start and bus_end columns are inserted by the application, not DB2

BUSINESS_TIME period
(inclusive, exclusive)

trip_name	destination	departure_ date	price	bus_start	bus_end
Manu Wilderness	Peru	08/02/2011	1500.00	05/01/2011	01/01/2012

Update an Application-Period Temporal Table

Current Date = May 15, 2011

- **Manu Wilderness trip isn't selling well, so we'll offer a special price of $1000.00 for the month of June.**

```
UPDATE travel FOR PORTION OF BUSINESS_TIME FROM '06/01/2011' TO '07/01/2011'
SET price = 1000.00 WHERE trip_name = 'Manu Wilderness';
```

Before (Prior to Update)

trip_name	destination	departure_date	price	bus_start	bus_end
Manu Wilderness	Peru	08/02/2011	1500.00	05/01/2011	01/01/2012

BUSINESS_TIME period
(inclusive, exclusive)

After (Updated Table)

trip_name	destination	departure_date	price	bus_start	bus_end
Manu Wilderness	Peru	08/02/2011	1500.00	05/01/2011	**06/01/2011**
Manu Wilderness	Peru	08/02/2011	**1000.00**	**06/01/2011**	**07/01/2011**
Manu Wilderness	Peru	08/02/2011	1500.00	**07/01/2011**	01/01/2012

DB2 inserted 2 rows and updated 1 row.

Application-Period Temporal Table – Unique Enforcement

Current Date = Sept 1, 2011

- **Manu Wilderness trip has sold out, so we'll add another trip departing on 11/2/2011, which is on sale from 10/01/2011 through the end of 2011 (same sale period):**

```
INSERT INTO travel
VALUES ('Manu Wilderness','Peru','11/02/2011',1500.00,
'10/01/2011','01/01/2012');
```

Insert fails: bus_start and bus_end PERIOD of inserted row cannot overlap bus_start and bus_end times of existing rows (they form a unique primary key with trip_name)

BUSINESS_TIME period (inclusive, exclusive)

TRAVEL

trip_name	destination	departure_date	price	bus_start	bus_end
Manu Wilderness	Peru	08/02/2011	1500.00	05/01/2011	06/01/2011
Manu Wilderness	Peru	08/02/2011	1000.00	06/01/2011	07/01/2011
Manu Wilderness	Peru	08/02/2011	1500.00	**07/01/2011**	**01/01/2012**

Application-Period Temporal Table – Valid Insert

Current Date = Sept 1, 2011

- **Solution to INSERT error on previous page: change name of new section to 'Manu Wilderness 2' and re-INSERT:**

```
INSERT INTO travel
VALUES 'Manu  Wilderness 2','Peru',
'11/02/2011',1500.00,'10/01/2011','01/01/2012');
```

SUCCESS!

New line inserted in blue

BUSINESS_TIME period (inclusive, exclusive)

TRAVEL

trip_name	destination	departure_date	price	bus_start	bus_end
Manu Wilderness	Peru	08/02/2011	1500.00	05/01/2011	06/01/2011
Manu Wilderness	Peru	08/02/2011	1000.00	06/01/2011	07/01/2011
Manu Wilderness	Peru	08/02/2011	1500.00	07/01/2011	01/01/2012
Manu Wilderness 2	**Peru**	**11/02/2011**	**1500.00**	**10/01/2011**	**01/01/2012**

Query an Application-Period Temporal Table

These queries access the table on the previous page.

Current Date = Sept 1, 2011

1. **How much did the Manu Wilderness trip cost on 06/01/2011?**

```
SELECT price FROM travel FOR BUSINESS_TIME AS OF '06/01/2011'
WHERE trip_name = 'Manu Wilderness';
```

```
Result: 1000
```

2. **What was the lowest price this year for the Manu Wilderness trip?**

```
SELECT MIN (price) FROM travel
FOR BUSINESS_TIME FROM '01/01/2011' TO '01/01/2012'
WHERE trip_name = 'Manu Wilderness';
```

```
Result: 1000
```

3. **What trips are available for booking in October?**

```
SELECT trip_name FROM travel
FOR BUSINESS_TIME BETWEEN '10/01/2011' AND '11/01/2011';
```

```
Result:
Manu Wilderness
Manu Wilderness 2
```

DELETE from an Application-Period Temporal Table

- **To drive sales, Manu Wilderness is to be temporarily made not available for five days.**

Current Date = Sept 15, 2011

```
DELETE FROM travel
FOR PORTION OF BUSINESS_TIME  FROM '09/15/2011' TO '09/20/2011'
WHERE trip_name LIKE 'Manu Wilderness%';
```

BUSINESS_TIME period
(inclusive, exclusive)

TRAVEL

trip_name	destination	departure_date	price	bus_start	bus_end
Manu Wilderness	Peru	08/02/2011	1500.00	05/01/2011	06/01/2011
Manu Wilderness	Peru	08/02/2011	1000.00	06/01/2011	07/01/2011
Manu Wilderness	Peru	08/02/2011	1500.00	07/01/2011	09/15/2011
Manu Wilderness	Peru	08/02/2011	1500.00	09/20/2011	01/01/2012
Manu Wilderness 2	Peru	11/02/2011	1500.00	10/01/2011	01/01/2012

- **DB2 automatically spilt the rows to accommodate the bus_start and bus_end times to satisfy the condition.**

DELETE from an Application-Period Temporal Table

- **Mudslide has wiped out the Manu Wilderness Lodge. Discontinue the Manu Wilderness trips until they rebuild:**

Current Date = Sep 30, 2011

```
DELETE FROM travel
FOR PORTION OF BUSINESS_TIME  FROM '10/01/2011' TO '12/30/9999'
WHERE trip_name LIKE 'Manu Wilderness%';
```

BUSINESS_TIME period
(inclusive, exclusive)

TRAVEL

trip_name	destination	departure_date	price	bus_start	bus_end
Manu Wilderness	Peru	08/02/2011	1500.00	05/01/2011	06/01/2011
Manu Wilderness	Peru	08/02/2011	1000.00	06/01/2011	07/01/2011
Manu Wilderness	Peru	08/02/2011	1500.00	07/01/2011	09/15/2011
Manu Wilderness	Peru	08/02/2011	1500.00	09/20/2011	10/01/2011

- **DB2 has changed bus_end column for Manu Wilderness trip to 10/01/2011**
- **DB2 has deleted row for Manu Wilderness 2 trip because the entire PERIOD for the row was later than 10/01/2011**

Bitemporal Tables

- **Combine application-period (ATT) and system-period (STT) capabilities**
- **Every row has a pair of:**
  - **timestamps set by DB2 (SYSTEM_TIME period)**
  - **business timestamps or date columns (BUSINESS_TIME period) set by the application**

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	02/15/2011	50.00	02/01/2011	02/16/2011	02/01/2011	12/30/9999

- **Query using both application-period time and system-period time**
  - **Example:** *what trips were offered on June 20, 2011, as recorded in the database on May 10, 2011?*

```
SELECT trip_name, destination FROM TRAVEL
FOR BUSINESS_TIME AS OF '06/20/2011' FOR SYSTEM_TIME AS OF  '2011-05-10';
```

- **INSERT/UPDATE/DELETE behavior is similar to ATTs (rows inserted/split/deleted as required)**
- **UPDATE and DELETE cause automatic insertion into the corresponding history table**
- **SELECT will go to the history table as needed to get rows**

How to Define a Bitemporal Table

```
CREATE TABLE travel(
trip_name CHAR(25) NOT NULL,
destination CHAR(8) NOT NULL,
departure_date DATE NOT NULL,
price DECIMAL(8,2) NOT NULL,
BUS_START DATE NOT NULL,
BUS_END DATE NOT NULL,
SYS_START TIMESTAMP(12) NOT NULL
          GENERATED ALWAYS AS ROW BEGIN IMPLICITLY HIDDEN,
SYS_END TIMESTAMP(12) NOT NULL
          GENERATED ALWAYS AS ROW END IMPLICITLY HIDDEN,
TX_ID TIMESTAMP(12)
          GENERATED ALWAYS AS TRANSACTION START ID IMPLICITLY HIDDEN,
PERIOD SYSTEM_TIME (SYS_START, SYS_END),
PERIOD BUSINESS_TIME (BUS_START, BUS_END),
PRIMARY KEY (trip_name, BUSINESS_TIME WITHOUT OVERLAPS));

CREATE TABLE travel_history LIKE travel;

ALTER TABLE travel ADD VERSIONING USE HISTORY TABLE travel_history;
```

Application-temporal (ATT) keywords

System-temporal (STT) keywords

Insert Data into a Bitemporal Table

Current Date = Feb 1, 2011

- **Add new trip: : Alligator Swamp, departing 3 times in 2011, on 2/15/2011, 5/15/2011, and 10/15/2011:**

```
INSERT INTO TRAVEL VALUES ('Alligator
Swamp','Louisiana','02/15/2011',50.00,'02/01/2011','02/16/2011');
```

(plus 2 more INSERT statements for departure_dates 05/15/2011 and 10/15/2011)

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	02/15/2011	50.00	02/01/2011	02/16/2011	02/01/2011	12/30/9999
Alligator Swamp	Louisiana	05/15/2011	50.00	02/16/2011	05/16/2011	02/01/2011	12/30/9999
Alligator Swamp	Louisiana	10/15/2011	50.00	05/16/2011	10/16/2011	02/01/2011	12/30/9999

- **Both sys_start and sys_end columns are inserted by DB2, not the application.**
  - For simplicity, they are represented here as DATEs, rather than TIMESTAMPs
- **They are also implicitly hidden as well as tx_id (not shown above)**

Update Bitemporal Business Time

Current Date = Feb 2, 2011

- **Change departure date for 3rd section from October 15 to September 15, and update valid period during which trip can be booked (bus_end):**

```
UPDATE travel
SET departure_date = '09/15/2011', bus_end = '09/16/2011'
WHERE trip_name = 'Alligator Swamp'
        and departure_date = '10/15/2011'
```

TRAVEL

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	02/15/2011	50.00	02/01/2011	02/16/2011	02/01/2011	12/30/9999
Alligator Swamp	Louisiana	05/15/2011	50.00	02/16/2011	05/16/2011	02/01/2011	12/30/9999
Alligator Swamp	Louisiana	**09/15/2011**	50.00	05/16/2011	**09/16/2011**	**02/02/2011**	12/30/2099

TRAVEL_HISTORY

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	**10/15/2011**	50.00	05/16/2011	**10/16/2011**	**02/01/2011**	**02/02/2011**

- **Application determined new bus_end time;**
  DB2 sets new sys_start value and inserts old row into history table.

Delete Portion of Business Time in a Bitemporal Table

Current Date = June 1, 2011

- **Alligator Swamp guide quit; remove trip for 3 months while we find a new guide:**

```
DELETE FROM travel
FOR PORTION OF BUSINESS TIME FROM '06/01/2011' TO '09/1/2011'
WHERE trip_name = 'Alligator Swamp'
```

TRAVEL

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	02/15/2011	50.00	02/01/2011	02/16/2011	02/01/2011	12/30/9999
Alligator Swamp	Louisiana	05/15/2011	50.00	02/16/2011	05/16/2011	02/01/2011	12/30/9999
Alligator Swamp	Louisiana	09/15/2011	50.00	05/16/2011	**06/01/2011**	**06/01/2011**	12/30/2099
Alligator Swamp	Louisiana	09/15/2011	50.00	**09/01/2011**	09/16/2011	**06/01/2011**	12/30/9999

TRAVEL_HISTORY

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	10/15/2011	50.00	05/16/2011	10/16/2011	02/01/2011	02/02/2011
Alligator Swamp	Louisiana	**09/15/2011**	50.00	**05/16/2011**	**09/16/2011**	**02/02/2011**	**06/01/2011**

Bitemporal: Query in Both System Time and Business Time

Current Date = June 1, 2011

- **What departure dates for Alligator Swamp were available for booking on 03/01/2011, as recorded in the database on 02/01/2011?**

```
SELECT  departure_date FROM travel
FOR BUSINESS_TIME AS OF '03/01/2011'
FOR SYSTEM_TIME AS OF TIMESTAMP '2011-02-01-00.00.00.000000'
WHERE trip_name = 'Alligator Swamp'
```

Result: 05/15/2011

TRAVEL

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	02/15/2011	50.00	02/01/2011	02/16/2011	02/01/2011	12/30/9999
Alligator Swamp	Louisiana	**05/15/2011**	50.00	**02/16/2011**	**05/16/2011**	**02/01/2011**	**12/30/9999**
Alligator Swamp	Louisiana	09/15/2011	50.00	05/16/2011	06/01/2011	06/01/2011	12/30/2099
Alligator Swamp	Louisiana	09/15/2011	50.00	09/01/2011	09/16/2011	06/01/2011	12/30/9999

TRAVEL_HISTORY

trip_name	destination	departure_date	price	bus_start	bus_end	sys_start	sys_end
Alligator Swamp	Louisiana	10/15/2011	50.00	05/16/2011	10/16/2011	**02/01/2011**	**02/02/2011**
Alligator Swamp	Louisiana	09/15/2011	50.00	05/16/2011	09/16/2011	02/02/2011	06/01/2011

Views on Temporal Table

- Views may be defined on system-period temporal tables (base and history), application-period temporal tables, or bitemporal tables
- All syntax (e.g. FOR PORTION OF, AS OF, FROM…TO, etc.) is supported for views
- **Two types of views may be defined for temporal tables:**
  - Containing **FOR BUSINESS_TIME or FOR SYSTEM_TIME**

```
CREATE VIEW travel_view AS
SELECT * FROM travel

FOR SYSTEM_TIME BETWEEN '06/30/2011' AND '01/01/2012';
SELECT * FROM travel_view;
```

    - Restriction: queries against the view can't also contain FOR BUSINESS TIME and FOR SYSTEM TIME
    - Would lead to ambiguity or conflicts
  - Without **FOR BUSINESS_TIME or FOR SYSTEM_TIME**
    - Data from all periods is available to the query

```
CREATE VIEW travel_view AS SELECT * FROM travel;
SELECT * FROM travel_view
FOR BUSINESS_TIME AS OF '01/01/2011';
```

Special Registers

- You can “set the clock back or forward” to a specific time for a given session
  - No changes required on applications!
- **Special registers:**
  - CURRENT TEMPORAL BUSINESS_TIME
  - CURRENT TEMPORAL SYSTEM_TIME
- **Setting one or both of these registers allows you to query:**
  - Past point in **SYSTEM_TIME**
  - Past or future point in **BUSINESS_TIME**
- e.g.

```
DB2 SET CURRENT TEMPORAL SYSTEM_TIME  = CURRENT TIMESTAMP - 1 YEAR
DB2 SET CURRENT TEMPORAL BUSINESS_TIME = '2012-12-31'
```

- **Implicit period specification attached to SQL statements:**
  - FOR BUSINESS_TIME AS OF **CURRENT TEMPORAL BUSINESS_TIME**
  - FOR SYSTEM_TIME AS OF **CURRENT TEMPORAL SYSTEM_TIME**

Bind Options and Routine Options

- Choose whether special registers affect compiled packages, stored procedures, etc.
  - CURRENT TEMPORAL BUSINESS_TIME
  - CURRENT TEMPORAL SYSTEM_TIME

Action	Option	Value	Explanation
Compilation	SYSTIMESENSITIVE	NO	Default for existing packages, package never affected by register.
		YES	Default for new packages, package observes special register.
	BUSTIMESENSITIVE	NO	Default for existing packages, package never affected by register.
		YES	Default for new packages, package observes special register.

- To create stored procedures that are insensitive to the special register:

BUSINESS_TIME

```
CALL SET_ROUTINE_OPTS('BUSTIMESENSITIVE NO');
```

SYSTEM_TIME

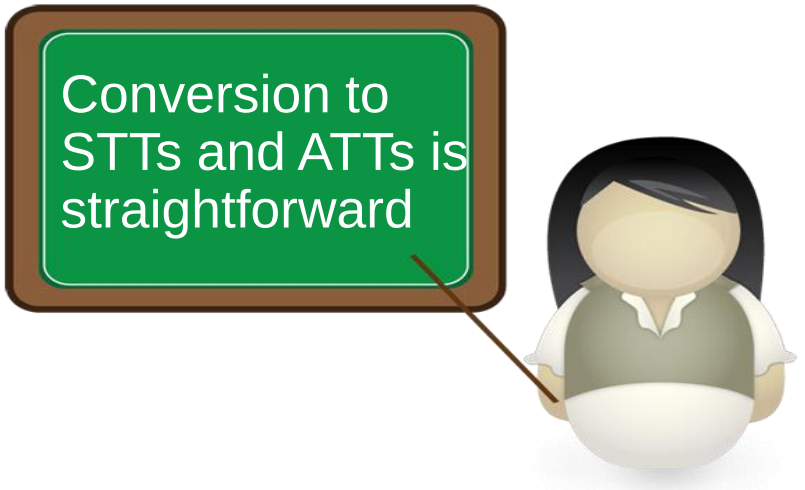
```
CALL SET_ROUTINE_OPTS('SYSTIMESENSITIVE NO');
```

BUSINESS_TIME & SYSTEM_TIME

```
CALL SET_ROUTINE_OPTS('BUSTIMESENSITIVE NO SYSTIMESENSITIVE NO');
```

Turning Existing Tables into STTs and ATTs

- Many users have existing tables which they would like to turn into system-temporal and/or application-temporal tables
- **Benefits include:**
  - **Lower cost**
    - Reduced application logic and shorter application development time
    - Fewer hand-coded triggers and stored procedures
    - Simpler queries
  - **Better data management**
    - Eliminate data input errors
    - Insure data integrity and uniqueness
    - Transparent deployment
  - **Simpler, faster compliance**



Turning Home-Grown Solution into STTs

- **Scenario: current application table exists, with or without history table**
- **To start using STT:**
  - Add columns or alter existing columns that track the version start time and end time
  - Add a column that tracks the transaction start id
- **These columns may be IMPLICITLY HIDDEN**

```
ALTER TABLE customer
        ADD sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN,
        ADD sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
        ADD trans_id TIMESTAMP(12)GENERATED ALWAYS AS TRANSACTION START ID;
```

- **Create history table if it doesn't exist**

```
  CREATE TABLE customer_history LIKE customer;
```

Turning Home-Grown Solution into STTs (Continued)

- **Ensure the 3 TIMESTAMP(12) columns exist in the history table**
  - current and history schemas must match
  - Ensure the 3 columns represent valid dates/times when rows were archived from the current table
    - If not, can LOAD REPLACE STT history table from another table
    - Can override GENERATED ALWAYS clauses for TIMESTAMP columns and use values from the load input table
    - PERIODOVERRIDE and TRANSACTIONIDOVERRIDE clauses
- **Alter the table that has current data to have a SYSTEM_TIME period**

```
ALTER TABLE CUSTOMER
ADD PERIOD SYSTEM_TIME (system_start, system_end);
```

- **Add versioning to the table that has current data referencing the history table**

```
ALTER TABLE customer ADD VERSIONING USE HISTORY TABLE history_table
```

DB2 now assumes control of versioning

- **Modify your application syntax as necessary to take advantage of the new STT tables**
  - e.g. (FOR SYSTEM TIME AS OF…), etc.

Turning Home-Grown Solution into ATTs

- **Scenario: current application-maintained table exists:**
- **To start using ATT:**
  – Alter or add the columns that track the business start time and end time

```
ALTER TABLE customer
        ADD bus_Start DATE NOT NULL DEFAULT CURRENT DATE,
        ADD bus_end DATE NOT NULL DEFAULT '9999-12-30';
```

- **Alter the table to have a BUSINESS_TIME period:**

```
ALTER TABLE customer
        ADD PERIOD BUSINESS_TIME (bus_start, bus_end);
```

- **(Optional) To enforce uniqueness of periods, alter or add a primary key:**

```
ALTER TABLE customer
        ADD PRIMARY KEY (id, BUSINESS_TIME WITHOUT OVERLAPS);
```

- **Modify your application syntax as necessary to take advantage of the new ATT**
  e.g. (FOR BUSINESS TIME BETWEEN…), etc.

Time Travel Tables summarized

- **Temporal tables enable time travel!**
- Temporal tables can be used as:
  - **System-period temporal tables** (STTs)
    - Managed by DB2
    - DB2 maintains a separate history table
  - **Application-period temporal tables** (ATTs)
    - Managed by the application
    - Current and historical rows are all in the base table
  - **Bitemporal tables**
    - Combine System-period and Application-period temporal tables
- Views can be created on STTs or ATTs for select or update
- **Special registers** allow querying past or future points in time
- **Current tables can be converted** to STTs or ATTs

The next steps…

The Next Steps…

- Complete the Hands on Lab for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module
  - Download the workbook and the virtual machine image
  - Follow the instructions in the workbook to complete the lab
- Complete the online quiz for this module
  - Log onto SKI, go to “My Learning” page, and select the “In Progress” tab.
  - Find the module and select the quiz
- Provide feedback on the module
  - Log onto SKI, go to “My Learning” page
  - Find the module and select the “Leave Feedback” button to leave your comments

Questions?
askdata@ca.ibm.com