Triton
CONSULTING

Time Travel Gotchas

By Mark Gillis

Time Travel Gotchas

Part 1

Time Travel query offers a quick and intuitive way of querying your data for historical scenarios, as well as the traditional current picture. There are a number of good articles out there showing how to enable the feature and some suggestions on how to use it. But like nearly every option, it's got pitfalls and overheads that will catch you out. I'm going to share a few that I've come across, specifically with reference to System Time. Business time has much less overhead than System Time, as there is no History table to be concerned about.

Many of the articles I've read and some of the instructions in the manuals suggest using the

CREATE TABLE history table **LIKE** base table

command to ensure you get a history table with the required matching attributes that will allow you to enable versioning. The problem with this is that the history table will be very simple and will not be optimized for the access or maintenance.

Unrestricted Growth

The trouble with storing historical data is that it will drastically increase the footprint of your data. Without invoking SYSTEM TIME versioning on your base table, the action of updating a row doesn't change the amount of data stored (ignoring the overhead of VARCHAR columns for the sake of argument) and deleting a row will reduce the amount of data. Before you enable SYSTEM TIME versioning, the actions of inserting, updating and deleting a single row would result in a 0 increase in data footprint. Once you have enabled SYSTEM TIME versioning, the same set of actions would leave you with 2 rows still in existence, stored in the history table. As a simple illustration (with an extra UPDATE for the hell of it) it will go like this:

Action	Rows in Base table	Rows in History table
INSERT	1	0
UPDATE	1	1
UPDATE	1	2
DELETE	0	3

The only action that won't populate the History table is the original insert so you can see that a table with a high-volume of Update activity is going to generate a lot of data in the history table.

And it will continue to grow until there is a mechanism or maintenance job implemented to reduce it. Regular scheduled DELETE jobs are OK if you're dealing with fairly low-volumes and can make use of a low activity window in your application schedule. But if you're anticipating higher-volumes and a more limited, or non-existent, maintenance window then defining your history table with Partitioning might be the answer. There's no reason that the History table shouldn't be organized in a completely different way to the Base table as long as the column attributes are identical. I recently set up a History table with the following options

```
PARTITION BY RANGE( SYSTEM_END )
(     STARTING   '2011-01-01-01.01.01.000000'
      ENDING     '2014-01-01-01.01.01.000000'
      EVERY 3 MONTH
)
Organize by Dimensions
(     SYSTEM_BEGIN_YEAR     )
```

which has given us the benefit of being able to detach and add partitions as needed. By using DETACH we can roll-out a section of the oldest data without impacting the on-going activity on the base table and can also ADD in a new partition to service the anticipated versioning for the future, with no performance impact either.

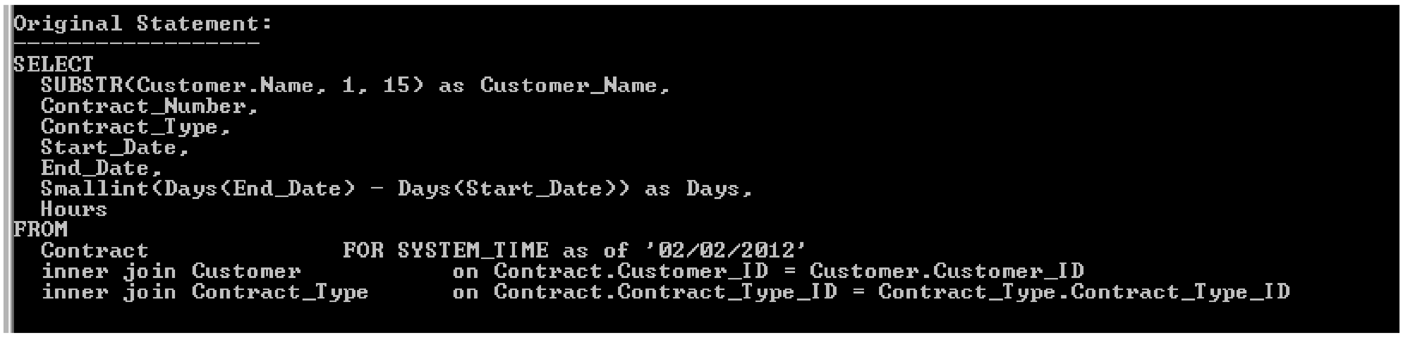
```
ALTER TABLE CONTRACT_HISTORY_MDC DETACH PARTITION PART0 INTO CONTRACT_HISTORY_PART0

ALTER TABLE CONTRACT_HISTORY_MDC ADD PARTITION STARTING FROM '2014-01-01-
01.01.01.000000000000' inclusive ENDING AT '2014-04-01' exclusive
```

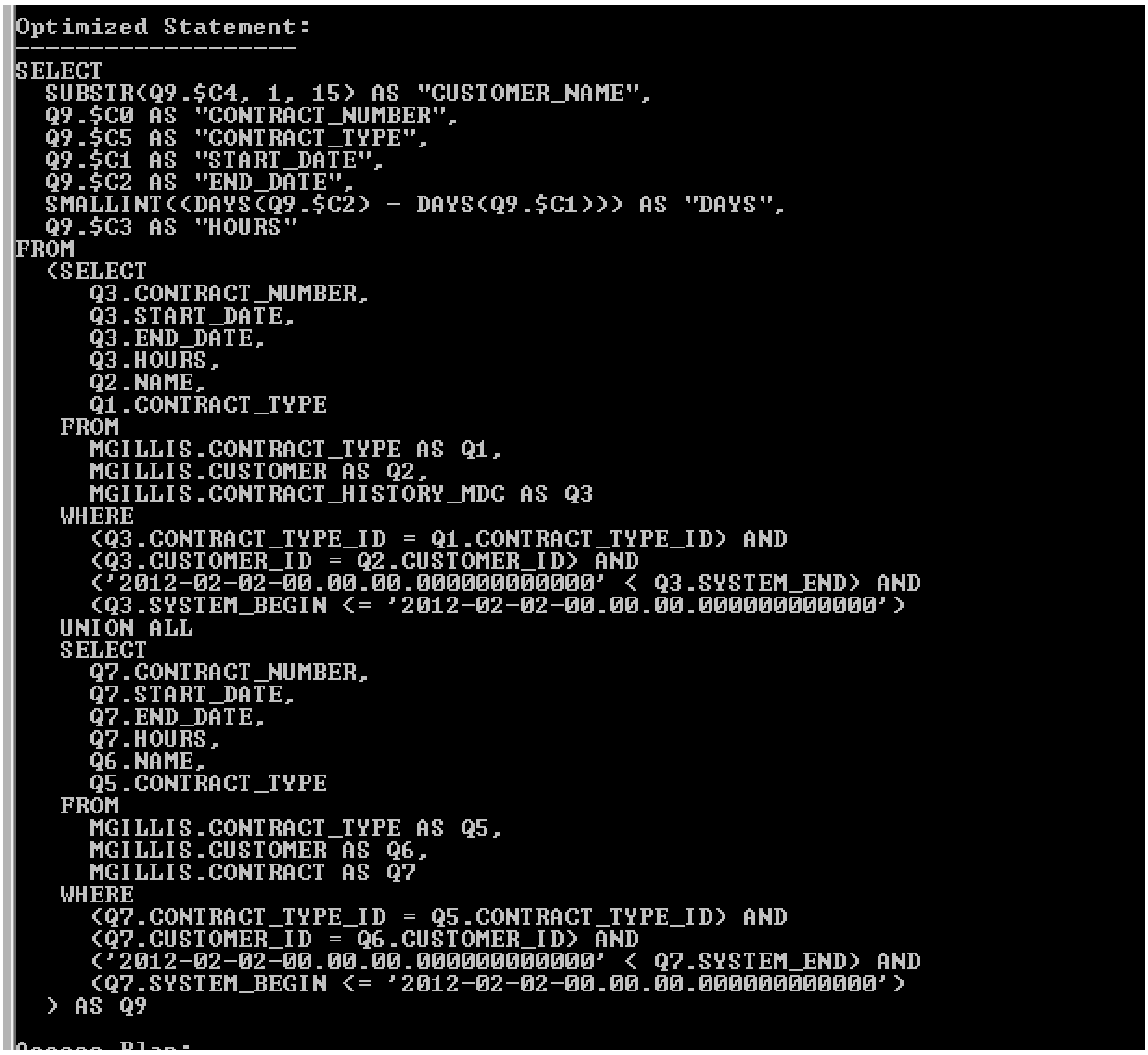
Part 2 – Access Paths

You will have noticed that my previous example also built the history table as a Multi-Dimensional Cluster (MDC). That doesn't give any benefits in terms of space management but does address a problem with the access path. It's not the only way; using insert time clustering (ITC), or just some well-built indexes and altering the table to be APPEND ON might work. But you will need to spend some time on the access strategy.

When you have defined the table as a SYSTEM Temporal table and you issue a query with a FOR SYSTEM_TIME clause in it, you'll find the optimizer has done something like this to your SQL.

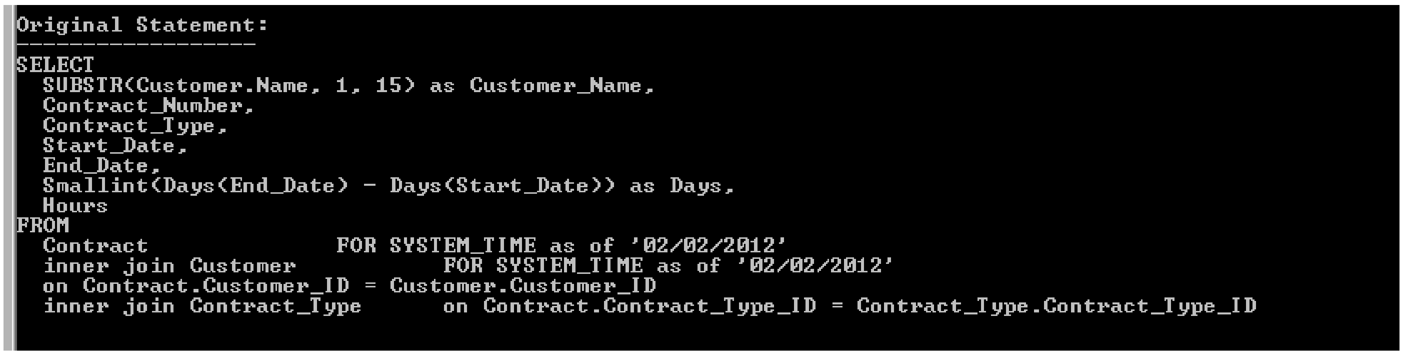


```
Original Statement:
------------------
SELECT
  SUBSTR(Customer.Name, 1, 15) as Customer_Name,
  Contract_Number,
  Contract_Type,
  Start_Date,
  End_Date,
  Smallint(Days(End_Date) - Days(Start_Date)) as Days,
  Hours
FROM
  Contract               FOR SYSTEM_TIME as of '02/02/2012'
  inner join Customer            on Contract.Customer_ID = Customer.Customer_ID
  inner join Contract_Type       on Contract.Contract_Type_ID = Contract_Type.Contract_Type_ID
```



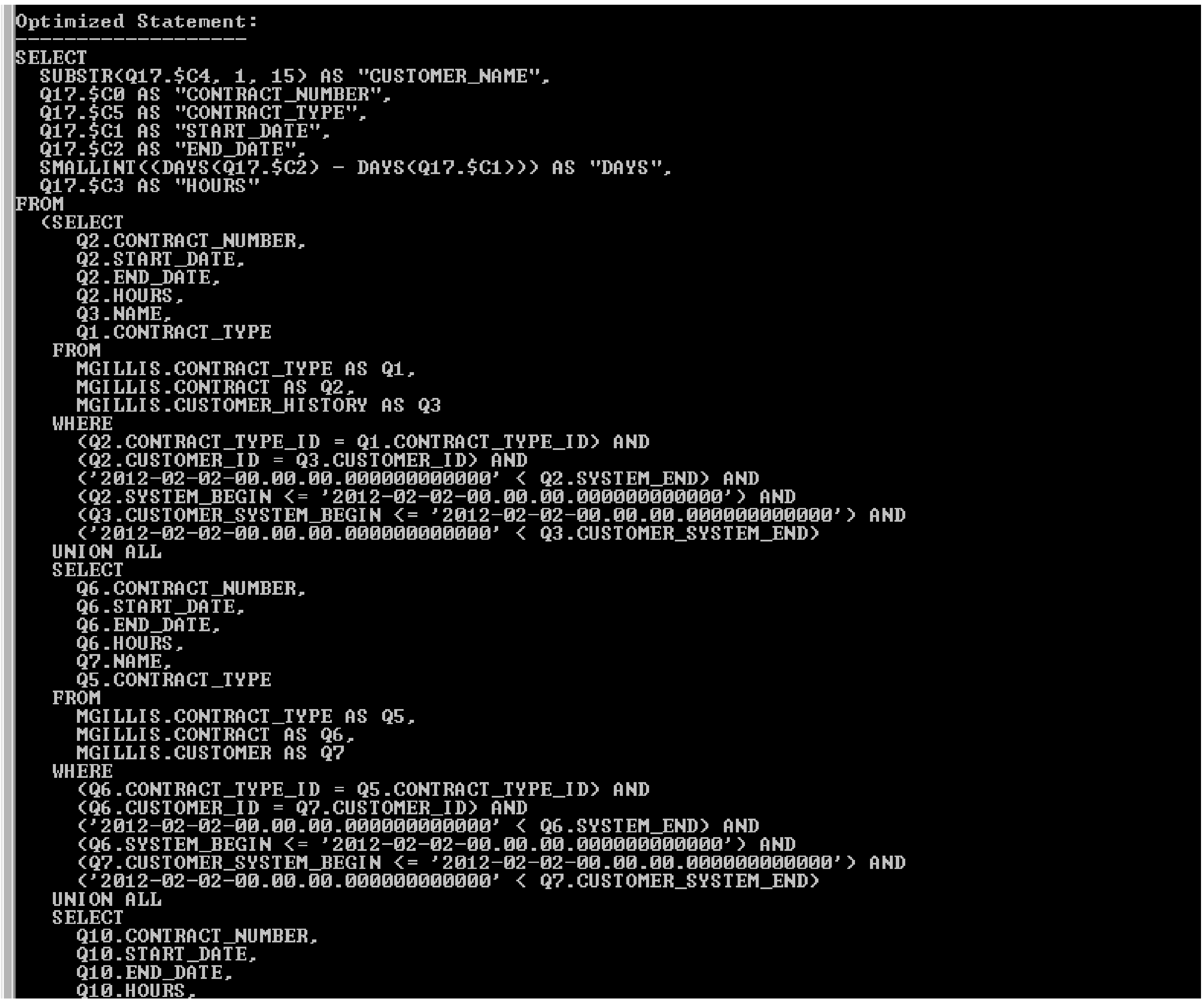
```
Optimized Statement:
-------------------
SELECT
  SUBSTR(Q9.$C4, 1, 15) AS "CUSTOMER_NAME",
  Q9.$C0 AS "CONTRACT_NUMBER",
  Q9.$C5 AS "CONTRACT_TYPE",
  Q9.$C1 AS "START_DATE",
  Q9.$C2 AS "END_DATE",
  SMALLINT((DAYS(Q9.$C2) - DAYS(Q9.$C1))) AS "DAYS",
  Q9.$C3 AS "HOURS"
FROM
  (SELECT
      Q3.CONTRACT_NUMBER,
      Q3.START_DATE,
      Q3.END_DATE,
      Q3.HOURS,
      Q2.NAME,
      Q1.CONTRACT_TYPE
   FROM
      MGILLIS.CONTRACT_TYPE AS Q1,
      MGILLIS.CUSTOMER AS Q2,
      MGILLIS.CONTRACT_HISTORY_MDC AS Q3
   WHERE
      (Q3.CONTRACT_TYPE_ID = Q1.CONTRACT_TYPE_ID) AND
      (Q3.CUSTOMER_ID = Q2.CUSTOMER_ID) AND
      ('2012-02-02-00.00.00.000000000000' < Q3.SYSTEM_END) AND
      (Q3.SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000')
   UNION ALL
   SELECT
      Q7.CONTRACT_NUMBER,
      Q7.START_DATE,
      Q7.END_DATE,
      Q7.HOURS,
      Q6.NAME,
      Q5.CONTRACT_TYPE
   FROM
      MGILLIS.CONTRACT_TYPE AS Q5,
      MGILLIS.CUSTOMER AS Q6,
      MGILLIS.CONTRACT AS Q7
   WHERE
      (Q7.CONTRACT_TYPE_ID = Q5.CONTRACT_TYPE_ID) AND
      (Q7.CUSTOMER_ID = Q6.CUSTOMER_ID) AND
      ('2012-02-02-00.00.00.000000000000' < Q7.SYSTEM_END) AND
      (Q7.SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000')
  ) AS Q9
```

You'll see that what you now have is two queries UNIONed together to interrogate the contents of the Base and History tables. In my example the Total Cost went from 20.33 timerons to 52.50. That's a fairly insignificant overhead but my test tables have minimal data in them (less than 20 rows) and the history table is only twice the size of the base table. I would anticipate the history table being far larger than the base table eventually. And look what happens when you enable one of the other tables in the same query as a System temporal table. It will go from an original statement (note the extra FOR SYSTEM TIME against the Customer table now) of

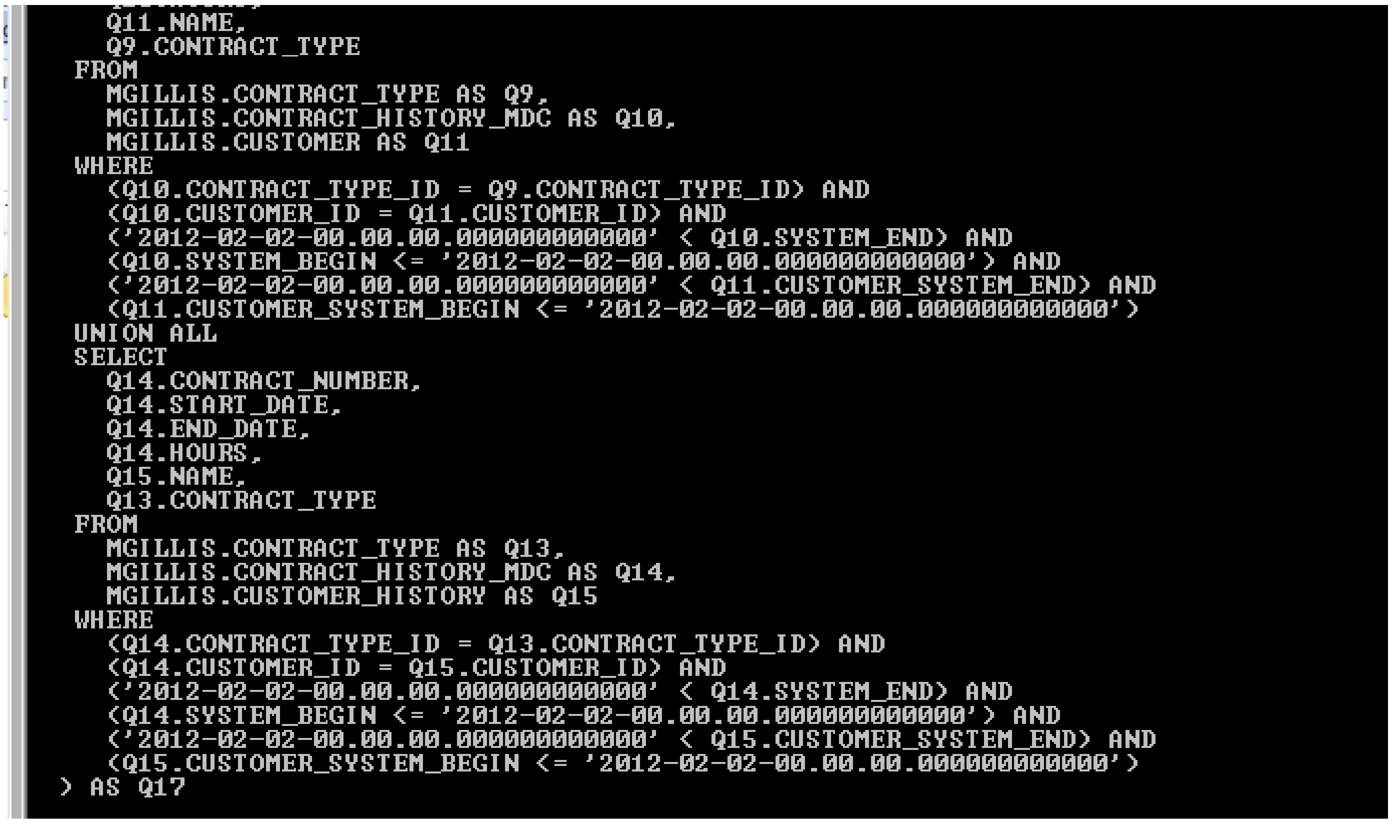


```
Original Statement:
------------------
SELECT
  SUBSTR(Customer.Name, 1, 15) as Customer_Name,
  Contract_Number,
  Contract_Type,
  Start_Date,
  End_Date,
  Smallint(Days(End_Date) - Days(Start_Date)) as Days,
  Hours
FROM
  Contract              FOR SYSTEM_TIME as of '02/02/2012'
  inner join Customer           FOR SYSTEM_TIME as of '02/02/2012'
  on Contract.Customer_ID = Customer.Customer_ID
  inner join Contract_Type      on Contract.Contract_Type_ID = Contract_Type.Contract_Type_ID
```

to this



```
Optimized Statement:
-------------------
SELECT
  SUBSTR(Q17.$C4, 1, 15) AS "CUSTOMER_NAME",
  Q17.$C0 AS "CONTRACT_NUMBER",
  Q17.$C5 AS "CONTRACT_TYPE",
  Q17.$C1 AS "START_DATE",
  Q17.$C2 AS "END_DATE",
  SMALLINT((DAYS(Q17.$C2) - DAYS(Q17.$C1))) AS "DAYS",
  Q17.$C3 AS "HOURS"
FROM
  (SELECT
      Q2.CONTRACT_NUMBER,
      Q2.START_DATE,
      Q2.END_DATE,
      Q2.HOURS,
      Q3.NAME,
      Q1.CONTRACT_TYPE
    FROM
      MGILLIS.CONTRACT_TYPE AS Q1,
      MGILLIS.CONTRACT AS Q2,
      MGILLIS.CUSTOMER_HISTORY AS Q3
    WHERE
      (Q2.CONTRACT_TYPE_ID = Q1.CONTRACT_TYPE_ID) AND
      (Q2.CUSTOMER_ID = Q3.CUSTOMER_ID) AND
      ('2012-02-02-00.00.00.000000000000' < Q2.SYSTEM_END) AND
      (Q2.SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000') AND
      (Q3.CUSTOMER_SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000') AND
      ('2012-02-02-00.00.00.000000000000' < Q3.CUSTOMER_SYSTEM_END)
    UNION ALL
    SELECT
      Q6.CONTRACT_NUMBER,
      Q6.START_DATE,
      Q6.END_DATE,
      Q6.HOURS,
      Q7.NAME,
      Q5.CONTRACT_TYPE
    FROM
      MGILLIS.CONTRACT_TYPE AS Q5,
      MGILLIS.CONTRACT AS Q6,
      MGILLIS.CUSTOMER AS Q7
    WHERE
      (Q6.CONTRACT_TYPE_ID = Q5.CONTRACT_TYPE_ID) AND
      (Q6.CUSTOMER_ID = Q7.CUSTOMER_ID) AND
      ('2012-02-02-00.00.00.000000000000' < Q6.SYSTEM_END) AND
      (Q6.SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000') AND
      (Q7.CUSTOMER_SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000') AND
      ('2012-02-02-00.00.00.000000000000' < Q7.CUSTOMER_SYSTEM_END)
    UNION ALL
    SELECT
      Q10.CONTRACT_NUMBER,
      Q10.START_DATE,
      Q10.END_DATE,
      Q10.HOURS,
```



```
      Q11.NAME,
      Q9.CONTRACT_TYPE
    FROM
      MGILLIS.CONTRACT_TYPE AS Q9,
      MGILLIS.CONTRACT_HISTORY_MDC AS Q10,
      MGILLIS.CUSTOMER AS Q11
    WHERE
      (Q10.CONTRACT_TYPE_ID = Q9.CONTRACT_TYPE_ID) AND
      (Q10.CUSTOMER_ID = Q11.CUSTOMER_ID) AND
      ('2012-02-02-00.00.00.000000000000' < Q10.SYSTEM_END) AND
      (Q10.SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000') AND
      ('2012-02-02-00.00.00.000000000000' < Q11.CUSTOMER_SYSTEM_END) AND
      (Q11.CUSTOMER_SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000')
    UNION ALL
    SELECT
      Q14.CONTRACT_NUMBER,
      Q14.START_DATE,
      Q14.END_DATE,
      Q14.HOURS,
      Q15.NAME,
      Q13.CONTRACT_TYPE
    FROM
      MGILLIS.CONTRACT_TYPE AS Q13,
      MGILLIS.CONTRACT_HISTORY_MDC AS Q14,
      MGILLIS.CUSTOMER_HISTORY AS Q15
    WHERE
      (Q14.CONTRACT_TYPE_ID = Q13.CONTRACT_TYPE_ID) AND
      (Q14.CUSTOMER_ID = Q15.CUSTOMER_ID) AND
      ('2012-02-02-00.00.00.000000000000' < Q14.SYSTEM_END) AND
      (Q14.SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000') AND
      ('2012-02-02-00.00.00.000000000000' < Q15.CUSTOMER_SYSTEM_END) AND
      (Q15.CUSTOMER_SYSTEM_BEGIN <= '2012-02-02-00.00.00.000000000000')
  ) AS Q17
```

You now have 4 queries joined by UNION ALL, the estimated Total Cost is 105 and the access path is becoming pretty complex. The optimizer has determined that it now has to have a query for each possible combination (ignoring CONTRACT_TYPE as this is not enabled as a System temporal table)

CUSTOMER	CONTRACT
CUSTOMER	CONTRACT_HISTORY
CUSTOMER_HISTORY	CONTRACT
CUSTOMER_HISTORY	CONTRACT_HISTORY

Adding another System temporal table, or converting CONTRACT_TYPE in this example, would invoke 8 sub-queries and a jump in estimated Total Cost to 672

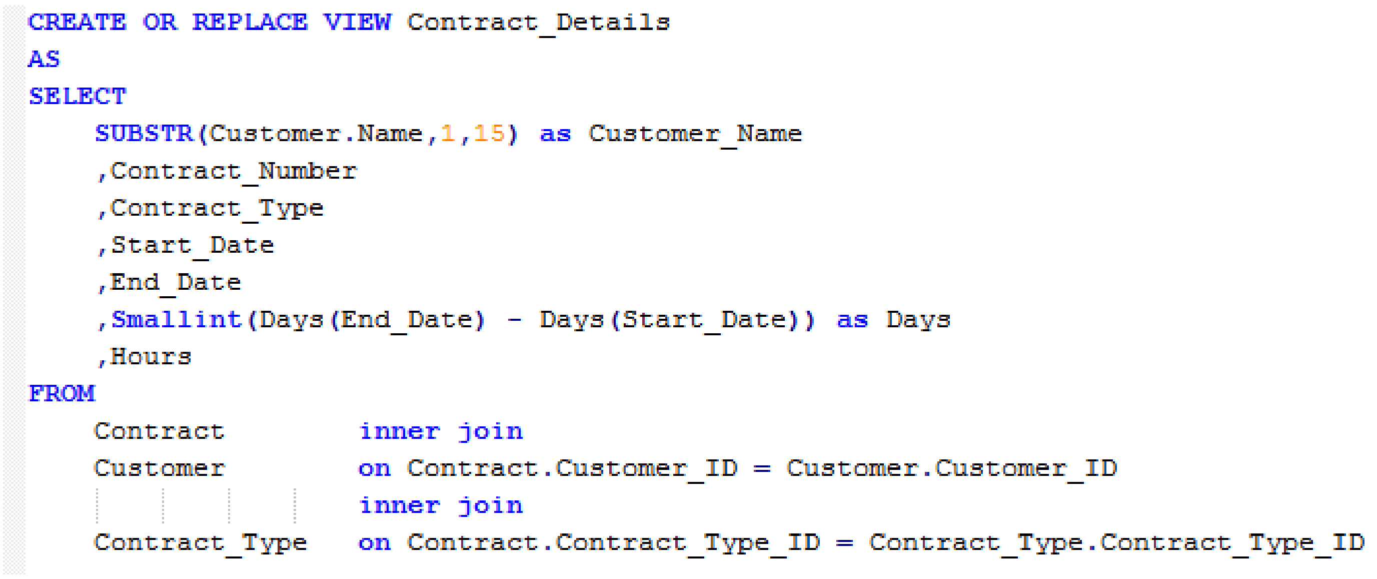
CUSTOMER	CONTRACT_TYPE	CONTRACT_HISTORY_MDC
CUSTOMER	CONTRACT_TYPE	CONTRACT
CUSTOMER_HISTORY	CONTRACT_TYPE	CONTRACT
CUSTOMER_HISTORY	CONTRACT_TYPE	CONTRACT_HISTORY_MDC
CUSTOMER_HISTORY	CONTRACT_TYPE_HISTORY	CONTRACT
CUSTOMER_HISTORY	CONTRACT_TYPE_HISTORY	CONTRACT_HISTORY_MDC
CUSTOMER	CONTRACT_TYPE_HISTORY	CONTRACT_HISTORY_MDC
CUSTOMER	CONTRACT_TYPE_HISTORY	CONTRACT

This has the potential to get out of hand very quickly. There's no hard and fast way of solving this, as for any performance issue, and I'm not going to presume to lecture experienced DBAs on how to design an efficient table and index strategy. I'm merely attempting to illustrate that there might be a problem with your queries once you enable System Time versioning, even if they ran like wildfire before you did the conversion.

Part 3 – Views

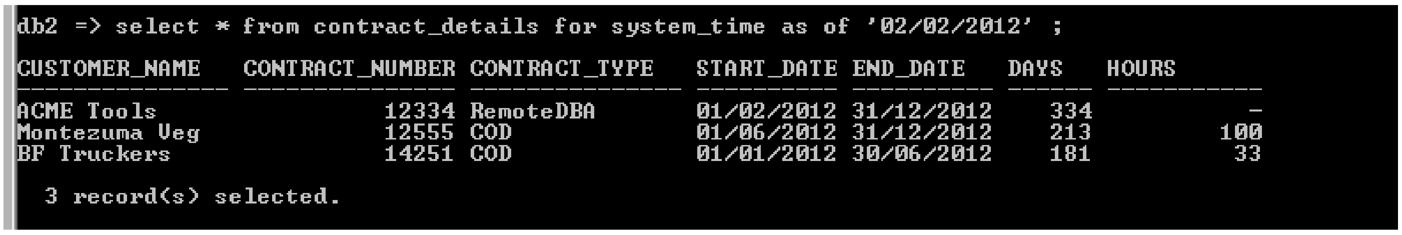
One of the features that I really like about Time Travel Query is an issue that's related to the one above. The optimizer, being the canny chunk of software that it is, will take your simple SYSTEM_TIME query and convert it into the necessary sub-queries to make sure all required historical data is accessed to.

What it will also do is accept a query against a view that contains System temporal table(s) in its definition. So I have taken the query used in the previous section, removed all SYSTEM_TIME clauses and used it to create a view:



```
CREATE OR REPLACE VIEW Contract_Details
AS
SELECT
    SUBSTR(Customer.Name,1,15) as Customer_Name
    ,Contract_Number
    ,Contract_Type
    ,Start_Date
    ,End_Date
    ,Smallint(Days(End_Date) - Days(Start_Date)) as Days
    ,Hours
FROM
    Contract        inner join
    Customer        on Contract.Customer_ID = Customer.Customer_ID
                    inner join
    Contract_Type   on Contract.Contract_Type_ID = Contract_Type.Contract_Type_ID
```

With the tables as they were at the end of my previous example, this query works fine



```
db2 => select * from contract_details for system_time as of '02/02/2012' ;

CUSTOMER_NAME   CONTRACT_NUMBER CONTRACT_TYPE   START_DATE END_DATE   DAYS   HOURS
--------------- --------------- --------------- ---------- ---------- ------ ----------
ACME Tools                12334 RemoteDBA       01/02/2012 31/12/2012    334          -
Montezuma Veg             12555 COD             01/06/2012 31/12/2012    213        100
BF Truckers               14251 COD             01/01/2012 30/06/2012    181         33

  3 record(s) selected.
```

You avoid the need to put a FOR SYSTEM_TIME AS OF against each table, but you lose the ability to specify a different time against each table. Now I've yet to find a situation where I'd want to do that but I guess it's possible. What might be more likely is that you would want to specify a SYSTEM TIME for just one of the tables, or a sub-set of the tables within the view.
That's the only gotcha there; your specified AS OF date gets applied to each and every System temporal table within the view. I still think it's a neat bit of short-hand.

So there are a couple of things that have caused me some headaches. It's not an exhaustive list and it's not got a cast-iron set of solutions but forewarned is forearmed and I hope you'll get to spot some of the pitfalls before they become a problem.

About Triton Consulting
Triton Consulting specialises in Data Management and has been an IBM Premier Business Partner since 1998. Specialising in DB2 for both the mainframe and distributed systems, Triton provides a full range of services from consultancy through to education and 24/7 DB2 support.

For more information visit www.triton.co.uk