OData Extension for Temporal Data

A Directional White Paper

Introduction

This paper documents some use cases, initial requirements, examples and design principles for an OData extension for temporal data. It is non-normative and is intended to seed discussion in the OASIS OData TC for the development of an OASIS standard OData extension defining representation and semantics of temporal data.

Both application time period tables and system time period tables were added to SQL/Foundation:2011.

Tables specified with an application time period have start and end columns that identify the period of time for which rows apply. A salary table with application time period can indicate an employee's salary in the past, in the present, or projected for the future. The rows of such a table can be changed at any time.

Tables specified with system time period have start and end columns that indicate the period of time for which the row existed. Historic rows are created automatically when current rows are deleted or updated. Historic rows cannot be changed once they have been created.

The representation of temporal data is not currently supported in OData. We suggest that an OData extension be defined to add this support. Entity types that represent temporal data will be identified as such, and additional operations will be made available on such entities and entity sets.

Status

Version 1.0 (May 18, 2012)

Authors

Andrew Eisenberg, IBM
Ralf Handl, SAP
Michael Pizzo, Microsoft

Background

SQL:2011 allows tables to be defined with application time periods. In this example, "business_time" is the name of the application time period:

```
CREATE TABLE emp_a
   (emp_id VARCHAR(30),
    dept_id VARCHAR(30),
    bus_start DATE NOT NULL,
    bus_end DATE NOT NULL,
    PERIOD FOR business_time (bus_start, bus_end),
    PRIMARY KEY (emp_id, business_time WITHOUT OVERLAPS)
    FOREIGN KEY (dept_id, PERIOD business_time)
       REFERENCES dept_a (dept_id, PERIOD business_time)
   );

CREATE TABLE dept_a
   (dept_id VARCHAR(30),
    budget DECIMAL(7,2),
    bus_start DATE NOT NULL,
    bus_end DATE NOT NULL,
    PERIOD FOR business_time (bus_start, bus_end),
    PRIMARY KEY (dept_id, business_time WITHOUT OVERLAPS)
   );
```

The emp_a table could be populated with the following row:

```
INSERT INTO emp_a
VALUES ('McDevitt', 'Help Desk', DATE '2011-01-01', DATE '2015-01-01')
```

emp_a would now have the following rows:

emp_id	dept_id	bus_start	bus_end
McDevitt	Help Desk	2011-01-01	2015-01-01

Changes can be made to employee rows that apply to a specific period of time:

```
UPDATE emp_a
          FOR PORTION OF business_time
              FROM DATE '2012-07-01' TO DATE '2013-01-01'
SET    dept_id = 'Business Services'
WHERE  emp_id = 'McDevitt'
```

emp_a would now have the following rows:

emp_id	dept_id	bus_start	bus_end
McDevitt	Help Desk	2011-01-01	2012-07-01
McDevitt	Business Services	2012-07-01	2013-01-01
McDevitt	Help Desk	2013-01-01	2015-01-01

Deletes made to employee rows can also apply to a specific period of time:

```
DELETE FROM emp_a
               FOR PORTION OF business_time
                   FROM DATE '2012-01-01' TO DATE '2012-04-01'
WHERE  emp_id = 'McDevitt'
```

emp_a would now have the following rows:

emp_id	dept_id	bus_start	bus_end
McDevitt	Help Desk	2011-01-01	2012-01-01
McDevitt	Help Desk	2012-04-01	2012-07-01
McDevitt	Business Services	2012-07-01	2013-01-01
McDevitt	Help Desk	2013-01-01	2015-01-01

Employee rows that apply to all dates can be retrieved:

```
SELECT   *
FROM     emp_a
```

Employee rows that apply to the current date can be retrieved:

```
SELECT   *
FROM     emp_a FOR BUSINESS_TIME AS OF CURRENT_DATE()
```

or:

```
SELECT   *
FROM     emp_a
WHERE    bus_start <= CURRENT_DATE()
         AND bus_end > CURRENT_DATE()
```

Employee rows that apply to a specific time period can be retrieved:

```
SELECT   *
FROM     emp_a FOR BUSINESS_TIME
                   FROM DATE('2010-01-01') TO DATE('2011-01-01')
```

or:

```
SELECT   *
FROM     emp_a
WHERE    bus_end > DATE('2010-01-01')
         AND bus_start < DATE('2011-01-01')
```

or:

```
SELECT   *
FROM     emp_a FOR BUSINESS_TIME
                   BETWEEN DATE('2010-01-01') AND DATE('2011-01-01')
```

or:

```
SELECT   *
FROM     emp_a
WHERE    bus_end > DATE('2010-01-01')
         AND bus_start <= DATE('2011-01-01')
```

“BUSINESS_TIME AS OF …”, “BUSINESS_TIME BETWEEN … AND …”, “BUSINESS_TIME FROM … TO …” are not part of SQL/2011. They are extensions supported by some SQL implementations.

“FROM value1 TO value2” returns rows where bus_start < value2 and bus_end > value1. “BETWEEN value1 AND value2” returns rows where bus_start <= value2 and bus_end > value1.

SQL:2011 allows tables to be defined with system time periods.

```
CREATE TABLE emp_s
   (emp_id VARCHAR(30),
    dept_id VARCHAR(30),
    system_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
    system_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
    PERIOD FOR SYSTEM_TIME (system_start, system_end),
    PRIMARY KEY (emp_id),
    FOREIGN KEY (dept_id) REFERENCES dept_s (dept_id)
   ) WITH SYSTEM VERSIONING;

CREATE TABLE dept_s
   (dept_id VARCHAR(30),
    budget DECIMAL(7,2),
    system_start DATE NOT NULL,
    system_end DATE NOT NULL,
    PERIOD FOR SYSTEM_TIME (system_start, system_end),
    PRIMARY KEY (dept_id)
   ) WITH SYSTEM VERSIONING;
```

In order to simplify the examples that follow, the system_start and system_end columns that are specified as TIMESTAMP will be treated as if they were of type DATE.

On Jan. 1, 2012 the following insert might be issued on an empty emp_s table:

```
INSERT INTO emp_s (emp_id, dept_id)
VALUES ('McDevitt', 'Landcsaping')
```

emp_a would then have the following rows:

emp_id	dept_id	system_start	system_end
McDevitt	Landscaping	2012-01-01	9999-12-31

On Feb. 1, 2012 the following update might be issued on emp_s:

```
UPDATE  emp_s
SET     dept_id = 'Strategic Planning'
WHERE   emp_id = 'McDevitt'
```

emp_a would then have the following rows:

emp_id	dept_id	system_start	system_end
McDevitt	Strategic Planning	2012-02-01	9999-12-31
McDevitt	Landscaping	2012-01-01	2012-02-01

On March 1, 2012 the following delete might be issued on emp_s:

```
DELETE  emp_s
WHERE   emp_id = 'McDevitt'
```

emp_a would then have the following rows:

emp_id	dept_id	system_start	system_end
McDevitt	Strategic Planning	2012-02-01	2012-03-01
McDevitt	Landscaping	2012-01-01	2012-02-01

Current employee rows can be retrieved:

```
SELECT   *
FROM     emp_s
```

Employee rows from a specific time period can be retrieved:

```
SELECT   *
FROM     emp_s FOR SYSTEM_TIME
                  FROM DATE('2010-01-01') TO DATE('2011-01-01')
```

or:

```
SELECT   *
FROM     emp_s FOR SYSTEM_TIME
                  BETWEEN DATE('2010-01-01') AND DATE('2011-01-01')
```

A database designer could choose to create an emp table that specifies an application time period, a system time period, or both types of time periods (this is known as a bitemporal table).

Motivation

An OData service might publish an Employees_a entity set that reflects the emp_a table:

```
<Schema
      xmlns="http://schemas.microsoft.com/ado/2008/09/edm"
      Namespace="Personnel">
   <Using Namespace="org.odata.temporal" Alias="Temp" />
   <EntityContainer Name="MyCompany">
      <EntitySet Name="Employees_a" EntityType="Employee_a"/>
   </EntityContainer>
   <EntityType Name="Employee_a">
      <Key>
         <PropertyRef Name="emp_id"/>
      </Key>
      <Property Name="emp_id" Type="Edm.String" />
      <Property Name="dept_id" Type="Edm.String" />
      <Property Name="bus_start" Type="Edm.DateTime" />
      <Property Name="bus_end" Type="Edm.DateTime" />
   </EntityType>
</Schema>
```

An OData service might publish an Employees_s entity set that reflects the emp_s table:

```
<Schema
      xmlns="http://schemas.microsoft.com/ado/2008/09/edm"
      Namespace="Personnel">
   <Using Namespace="org.odata.temporal" Alias="Temp" />
   <EntityContainer Name="MyCompany">
      <EntitySet Name="Employees_s" EntityType="Employee_s"/>
   </EntityContainer>
   <EntityType Name="Employee_s">
      <Key>
         <PropertyRef Name="emp_id"/>
      </Key>
      <Property Name="emp_id" Type="Edm.String" />
      <Property Name="dept_id" Type="Edm.String" />
      <Property Name="system_start" Type="Edm.DateTime" />
      <Property Name="system_end" Type="Edm.DateTime" />
   </EntityType>
</Schema>
```

Edm.DateTime has been used to represent the start and end values for these time periods. Edm.DateTimeOffset could also have been used. The reflection in OData of date/time values without explicit time zones needs further investigation.

The addition of Edm.Date to the OData primitive data types might be considered.

A client might wish to query theses entity sets in a number of ways.

1. Retrieve a current employee, showing the value of an employee's department.
2. Retrieve employees that worked for the "Performance Analysis" department in 2010.
3. Retrieve an employee as of a particular moment, including the employee's department as of that same moment.
4. Retrieve all versions of an entity that represents an employee in the past, present, and future.
5. Retrieve all versions of the department that a specific version of an employee works for, within the validity period of that employee version.
6. Retrieve all versions of an employee for a given time period, including all versions of the related departments within the validity period of each employee version that occurred within that time period.
7. Retrieve all versions of an employee, including all versions of the department related to each version of the employee within the validity period of that employee version.
8. Change an employee's department during an application time period already in existence.
9. Change an employee's department for a new application time period.

10. Change an employee's department in the current value of a system time employee entity.

Requirements

The following capabilities must be supported in this extension to OData:

- Annotate OData entity types that expose temporal data
- Support both application time periods and system time periods
- The result of queries on temporal data must be represented in OData
- For system time periods, allow only current entities to be returned
- Support AS OF, FROM, and BETWEEN operations on these entities
- Allow entities reflecting application time temporal data to be updated
- Allow entities reflecting current entities of system time temporal data to be updated

Examples

The following examples describe possible annotations and extensions to OData to support temporal data. Although concrete annotations, functions, and behavior are described, they are intended to be purely illustrative and not prescriptive.

The Employees_a entity set might now be published as:

```
<Schema
      xmlns="http://schemas.microsoft.com/ado/2008/09/edm"
      Namespace="Personnel">
   <Using Namespace="org.odata.temporal" Alias="Temporal" />
   <EntityContainer Name="MyCompany">
      <EntitySet Name="Employees_a" EntityType="Employee_a"/>
   </EntityContainer>
   <EntityType Name="Employee_a">
      <Key>
         <PropertyRef Name="emp_id"/>
      </Key>
      <Property Name="emp_id" Type="Edm.String" />
      <Property Name="dept_id" Type="Edm.String" />
      <Property Name="bus_start" Type="Edm.DateTime" />
      <Property Name="bus_end" Type="Edm.DateTime" />
      <TypeAnnotation Term="Temporal.ApplicationPeriod">
         <PropertyValue Name="StartPeriod" String="bus_start" />
         <PropertyValue Name="EndPeriod" String="bus_end" />
      </TypeAnnotation>
   </EntityType>
</Schema>
```

The Temporal.ApplicationPeriod annotation identifies this entity type as representing an application time period and specifies the names of the start and end properties.

The Employees_s entity set might now be published as:

```
<Schema
      xmlns="http://schemas.microsoft.com/ado/2008/09/edm"
      Namespace="Personnel">
   <Using Namespace="org.odata.temporal" Alias="Temporal" />
   <EntityContainer Name="MyCompany">
      <EntitySet Name="Employees_s" EntityType="Employee_s"/>
   </EntityContainer>
   <EntityType Name="Employee_s">
      <Key>
         <PropertyRef Name="emp_id"/>
      </Key>
      <Property Name="emp_id" Type="Edm.String" />
      <Property Name="dept_id" Type="Edm.String" />
      <Property Name="system_start" Type="Edm.DateTime" />
      <Property Name="system_end" Type="Edm.DateTime" />
      <TypeAnnotation Term="Temporal.SystemPeriod">
         <PropertyValue Name="StartPeriod" String="system_start" />
         <PropertyValue Name="EndPeriod" String="system_end" />
      </TypeAnnotation>
   </EntityType>
</Schema>
```

The Temporal.SystemPeriod annotation identifies this entity type as representing a system time period and specifies the names of the start and end properties.

To retrieve a current employee, showing the value of an employee’s department, one might submit the standard OData request:

```
http://www.ibm.com/temporal/Employees_a(emp_id = 'McDevitt')
```

This query might return:

```
<entry ...>
   <id>
      Employees_a('McDevitt';2012-01-01T00:00:00;2013-01-01T00:00:00)
   </id>
   <link
      rel="edit"
      title="Employees_a"
      href="http://www.ibm.com/temporal
                /application_time_as_of(datetime'2012-05-18T12:00:00')
                /Employees_a('McDevitt')" />
   <m:properties>
      <d:emp_id>McDevitt</d:emp_id>
      <d:dept_id>Standards</d:dept_id>
      <d:bus_start>2012-01-01T00:00:00</d:bus_start>
      <d:bus_end>2013-01-01T00:00:00</d:bus_end>
   </m:properties>
</entry>
```

The “edit” link that has been returned can be used to request changes to this entity. In the interest of brevity, these “edit” links will not be shown in subsequent examples.

This query is equivalent to:

```
http://www.ibm.com/temporal
         /application_time_as_of(current_date_time())
         /Employees_a('McDevitt')
```

To retrieve employees that worked for the “Performance Analysis” department in 2010, one might submit:

```
http://www.ibm.com/temporal/application_time_between
                                (datetime'2010-01-01T00:00:00',
                                 datetime'2011-01-01T00:00:00')
                           /Employees_a
?$filter=dept_id eq 'Performance Analysis'
```

To retrieve all versions of an entity that represents an employee in the past, present, and future, one might submit:

```
http://www.ibm.com/temporal/application_time_between
                                (datetime'0000-01-01T00:00:00',
                                 datetime'9999-12-31T00:00:00')
                           /Employees_a('McDevitt')
```

This query has used the system function application_time_between() to request employees for a specific application time period. This query might return:

```
<feed>
   <entry ...>
      <id>
         Employees_a('McDevitt';2012-01-01T00:00:00
                                ;2013-01-01T00:00:00)
      </id>
      <m:properties>
         <d:emp_id>McDevitt</d:emp_id>
         <d:dept_id>Standards</d:dept_id>
         <d:bus_start>2012-01-01T00:00:00</d:bus_start>
         <d:bus_end>2013-01-01T00:00:00</d:bus_end>
      </m:properties>
   </entry>
   <entry ... >
      <id>
         Employees_a('McDevitt';2011-01-01T00:00:00
                                ;2012-01-01T00:00:00)
      </id>
      <m:properties>
         <d:emp_id>McDevitt</d:emp_id>
         <d:dept_id>Software Engineering</d:dept_id>
         <d:bus_start>2011-01-01T00:00:00</d:bus_start>
         <d:bus_end>2012-01-01T00:00:00</d:bus_end>
      </m:properties>
   </entry>
</feed>
```

The context of queries on temporal data can be set by the functions:

```
application_time_as_of()
application_time_from()
application_time_between()
system_time_as_of()
system_time_from()
system_time_between()
```

Function such as application_time_all() and system_time_all() could be defined to simplify asking for entities in all time periods.

Functions such as min_date_time() and max_date_time() could be defined. These would isolate OData queries from the limits supported by each OData server.

The application_time_between() function establishes an application time period context that is used to retrieve employees. This context is also used for subsequent items in the navigation path.

The relationship between employees and departments might be reflected by the following navigation property and by the associations for Employee_a:

```
<Schema ...>
   <EntityType Name="Employee_a">
      .
      .
      .
      <NavigationProperty
         Name="departments"
         Relationship="E_D"
         FromRole="E_D_Source"
         ToRole="E_D_Target" />
   </EntityType>
   <Association Name="E_D">
      <End Role="E_D_Source" Type="Employee_a" Multiplicity="*" />
      <End Role="E_D_Target" Type="Department_a" Multiplicity="*" />
   </Association>
   <AssociationSet Name="E_D" Association="E_D">
      <End Role="E_D_Source" EntitySet="Employees_a" />
      <End Role="E_D_Target" EntitySet="Departments_a" />
   </AssociationSet>
</Schema>
```

Note that since navigation properties may return multiple versions for the same entity, all associations to temporal entity types must have a Multiplicity of many ("*").

To retrieve departments that an employee works for, one might submit:

```
http://www.ibm.com/temporal/Employees_a('McDevitt')
/departments
```

This query would return a single Department_a entity. Both the employee entity and its department entity are applicable for the current date and time:

```
<feed>
   <entry ...>
      <id>
         Department_a('Standards';'2012-01-01T00:00:00'
                                 ;'2012-07-01T00:00:00')
      </id>
      <m:properties>
         <d:dept_id>Standards</d:dept_id>
         <d:budget>80000.00</d:budget>
         <d:bus_start>2012-01-01T00:00:00</d:bus_start>
         <d:bus_end>2012-07-01T00:00:00</d:bus_end>
      </m:properties>
   </entry>
</feed>
```

Another query that retrieves departments that an employee works for is:

```
http://www.ibm.com/temporal
          /application_time_between(datetime'2012-01-01T00:00:00',
                                    datetime'2013-01-01T00:00:00')
          /Employees_a('McDevitt')
          /departments
```

This query might return several Department_a entities, as the department's budget may have changed during the application period that has been specified:

```
<feed>
   <entry ...>
      <id>
         Department_a('Standards';'2012-01-01T00:00:00'
                                 ;'2012-07-01T00:00:00')
      </id>
      <m:properties>
         <d:dept_id>Standards</d:dept_id>
         <d:budget>80000.00</d:budget>
         <d:bus_start>2012-01-01T00:00:00</d:bus_start>
         <d:bus_end>2012-07-01T00:00:00</d:bus_end>
      </m:properties>
   </entry>
   <entry ...>
      <id>
         Department_a('Standards';'2012-07-01T00:00:00'
                                 ;'2013-01-01T00:00:00')
      </id>
      <m:properties>
         <d:dept_id>Standards</d:dept_id>
         <d:budget>90000.00</d:budget>
         <d:bus_start>2012-07-01T00:00:00</d:bus_start>
         <d:bus_end>2013-01-01T00:00:00</d:bus_end>
      </m:properties>
   </entry>
</feed>
```

The query above reflects an employee that has worked for a department, where the values of some of the department's properties have changed. If an employee worked for different departments over time, then this would be reflected in multiple employee entities.

Navigations, either in the request (i.e., through $expand) or in the results (i.e., through links associated with navigation properties) preserve the temporal nature of the request. Conceptually, navigation links for each version of an entity must encode the overlap of the time specified in the request with the validity time for the version of the entity that contains the navigation property.

For example, a request for an employee as of a particular moment in time:

```
http://www.ibm.com/temporal
          /application_time_as_of('2012-06-01T00:00:00')
          /Employees_a('McDevitt')
```

Would return a relationship link for Departments that encoded the moment in time of the request, such as:

```
<link rel="http://org.oasis.odata/related/Department"
   type="application/atom+xml;type=feed"
   title="Department"
   href="application_time_as_of('2012-06-01T00:00:00')
         /Employees_a('McDevitt')/departments" />
```

A request for an employee as of a time period:

```
http://www.ibm.com/temporal
          /application_time_between(datetime'2012-01-01T00:00:00',
                                    datetime'2013-01-01T00:00:00')
          /Employees_a('McDevitt')
          /departments
```

Would return a relationship link for Departments that encoded the overlap of the time period specified in the request with the valid time of that employee, such as:

```
<link rel="http://org.oasis.odata/related/Department"
   type="application/atom+xml;type=feed"
   title="Department"
   href="application_time_between(datetime'2012-01-01T00:00:00',
                                  datetime'2013-01-01T00:00:00')
         /Employees_a('McDevitt')/departments" />
```

A second navigation property or function such as “overlapping_departments” might be defined to return all department entities that have an application time period that overlaps with the application time period of the employee. This navigation might return more department entities than “departments”, as the application time period for an employee might extend before or after the application time period that was set as the context for the query.

To change an employee’s department during an application time period already in existence, one might submit:

```
PUT/temporal/application_time_as_of(datetime'2012-05-18T12:00:00')
            /Employees_a('McDevitt')/dept_id/$value HTTP/1.1
Host: www.ibm.com
DataServiceVersion: 1.0
MaxDataServiceVersion: 2.0
accept: application/xml
content-type: text/plain
Content-Length: 6
SW Eng
```

This request uses the edit link that was retrieved earlier.

To change an employee’s department for a new application time period, one might submit:

```
PUT/temporal/application_time_as_of(datetime'2012-05-18T12:00:00')
            /Employees_a('McDevitt')/dept_id/$value
   ?start='2012-07-01T00:00:00' & end='2012-10-01T00:00:00' HTTP/1.1
Host: www.ibm.com
DataServiceVersion: 1.0
MaxDataServiceVersion: 2.0
accept: application/xml
content-type: text/plain
Content-Length: 11
Landscaping
```

To change an employee’s department in the current value of a system time employee entity, one might submit:

```
PUT/temporal/Employees_s('McDevitt')/dept_id/$value HTTP/1.1
Host: www.ibm.com
DataServiceVersion: 1.0
MaxDataServiceVersion: 2.0
accept: application/xml
content-type: text/plain
Content-Length: 7
Finance
```

Design Principles

OData is an application-level protocol for interacting with data via RESTful web services. An OData Service's contract is defined by simple, well-defined conventions and semantics applied to the data model exposed by the service, providing a high level of semantic interoperability between loosely coupled clients and services.

The design principles of OData are to:

- Make it easy to implement and consume a basic OData service over a variety of data sources. Rather than try and expose the full functionality of all stores, define common features for core data retrieval and update scenarios and incremental, optional features for more advanced scenarios.
- Leverage Data Models to guide clients through common interaction patterns rather than force clients to write complex queries against raw data
- Define consistency across the protocol and a single way to express each piece of functionality

The design principles of OData extensions are to:

- Ensure extensions do not violate the core semantics of OData
- Avoid defining different representations for common concepts across extensions
- Ensure independent extensions compose well
- Ensure clients can ignore extended functionality and still query and consume data correctly

Design Principles for Temporal Extensions

- The entire result of a request has a consistent temporal value (either "current", or a specified moment or time period)
- Navigating relationships on entities returned from a temporal query should have the same behavior as expressing $expand within a temporal query

Technical Direction

The design of this extension to OData should take the following direction:

- An OData vocabulary for Temporal Data shall be defined.
- An annotation from the Temporal Data vocabulary should be applied to entity types that reflect data from an application time period table, a system time period table, or a bitemporal table that contains both periods.
- Functions will be defined that allow entities to be retrieved "as of" a certain time, or "between" or "from" a certain period of time.
- These functions will be based on functions found in SQL/Foundation:2011.

Open questions, issues and work items

- The reflection in OData of date/time values without explicit time zones needs further investigation.
- The Edm.String data type can be used to define a type annotation property when the property will hold the name of some other property. A more specific data type could be added to OData for this purpose.
- It may be desirable to allow a query to apply “as of”, “from”, and “between” to an entire request, an entity set, or possibly at some finer level of granularity.
- OData might be extended to allow functions to apply to entity sets of any entity type. This could greatly reduce the number of functions that need to be defined.
- A function that returns the current date and time is needed.
- The addition of Edm.Date to the OData primitive data types might be considered.
- It is unclear what value of precision should be used for functions that have Edm.DateTime or Edm.DateTimeOffset parameters. Some databases support precisions as high as 12 digits.
- How do we differentiate between POST and PUT/PATCH for temporal entities? If we say that PUT may create a new entity for the specified time if one does not already exist for that time, do I use POST only for a new key value? Can I use PUT for new key values and say it always does an upsert?
- The current approach requires that any relationship to a temporal type be many in order to handle cases where navigating the relationship could return different versions of the entity (for example, if an employee was requested using a time range, rather than a point in time, it may have multiple departments during that time).

  An alternative design would be to define relationships as always being tied to a moment in time. This would allow more natural navigations, both in query (i.e., Employee('McDevitt')/Department/City eq 'London') and in generating strongly typed results (String city = employee.Department.City).

  Fixing the relationships at a moment in time is not an issue for "current" or "as of" queries, which anchor navigation to the moment in time specified in the initial request (in fact, it is what we propose), but it is an issue for navigating relationships from entities retrieved using "from" and "between" requests. One proposal was to define relationships for entities requested with a time span as returning the related version as of the last valid time for the parent entity within that queried timespan (i.e., the most recent department version for the given employee version within the specified timespan).

- This document suggests that the edit link and id for the entity returned without specifying a temporal context encodes the current time as the temporal context. That is, /Employees('McDevitt') is the same as application_time_as_of(currenttime())/Employees('McDevitt'). An alternate design that would be closer to existing OData semantics for the default case would be that queries with no temporal context specified would return ids and edit links that also had no temporal context. That is, the edit link would always update the current entity.

Additional Notes

- We considered making application start/end time part of the key so that versions of each entity were unique. In this case we could have more comfortably exposed all versions in an application time entity set and uniquely identified each. However, clients may want to update the start and end times and OData does not allow updating keys.
- We considered the following options for specifying a temporal period in the request:

    1. Use new system query option(s). The primary issue with this approach was composability. We wanted the entry to be able to return a self/edit link that the client could compose on top of, which favors the temporal modifier to be in the path portion of the request
    2. Use top-level Functions. We could have separate top level entry points (functions?) for each temporal entity set (i.e., EmployeesByApplicationTime(…), EmployeesBySystemTime(…), etc.) Once we made the simplification that a temporal period applied to an entire request, this implied that any temporal request would have to be rooted in one of these functions. This doesn't work well for composing temporal aspects to an existing URL (i.e., if I wanted to see all departments related to a particular employee for a specified period of time independent of how that employee was retrieved, or if I wanted to navigate from a non-temporal entity to a particular version of a temporal entity).
    3. Use Functions. We discussed applying functions to collections in order to apply temporal aspects to the set. This made for an elegant composable navigation model syntactically, but it seemed a little weird to define a function on a collection which changed the membership of that collection. Also, once we made the simplifying assumption that the entire request was as of a particular temporal period, exposing as composable functions provided more flexibility than we wanted.

4. Path Modifiers. We discussed including system-defined “function-like” operators in the path that would mean “interpret the path from the preceding collection on as being of this time”. The semantics of the method affecting the membership of the collection identified by the previous segment was a little strange as it wasn't a filter over the membership but rather changed the version of the entities exposed by that collection.

References

1. ISO/IEC 9075-2:2011 Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation).
2. Temporal Features in SQL standard, Krishna Kulkarni, May 13, 2011, http://metadata-standards.org/Document-library/Documents-by-number/WG2-N1501-N1550/WG2_N1536_koa046-Temporal-features-in-SQL-standard.pdf.
3. Go Back in Time, IBM DM Magazine, Sasirekha Rameshkumar, July 15, 2011, http://ibmdatamag.com/2011/07/go-back-in-time/.
4. A matter of time: Temporal data management in DB2 10, IBM developerWorks, Cynthia M. Saracco, Matthias Nicola, Lenisha Gandhi, April 3, 2012, http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata/dm-1204db2temporaldata-pdf.pdf.