System Versioned Tables
MetaEd 2.x Plug-in

Contents
Requirements................................................................................................................................................ 1
Overview ....................................................................................................................................................... 1
  What is a system-versioned temporal table? ........................................................................................... 1
  How does this work with the ODS database? ........................................................................................... 1
  What about future upgrades/schema changes? ...................................................................................... 1
Plugin Installation and Integration ................................................................................................................ 2
  Installation ................................................................................................................................................ 2
    Atom MetaEd-IDE setup ....................................................................................................................... 2
    Setup registry for Certica-Public ........................................................................................................... 2
    Install Certica SVT plugin....................................................................................................................... 2
  Integration ................................................................................................................................................ 2
    Ed-Fi ODS/API Changes ......................................................................................................................... 2
System Version Table Usage ......................................................................................................................... 3
  Point In Time Queries ............................................................................................................................... 3
    Basic Queries......................................................................................................................................... 3
    Hybrid Temporal Query ........................................................................................................................ 4
    Full Temporal Query (time traveling).................................................................................................... 4
  Range Queries ........................................................................................................................................... 5
Limitations .................................................................................................................................................... 6
  Recording time....................................................................................................................................... 6
  Ed-Fi ODS API Behavior ........................................................................................................................ 6
Requirements

- SQL Server 2017 Standard, or Higher
- MetaEd 2.0
- Ed-Fi ODS/API 2.x

Overview

What is a system-versioned temporal table?

A system-versioned temporal table is a type of user table designed to keep a full history of data changes and allow easy point in time analysis. This type of temporal table is referred to as a system-versioned temporal table because the period of validity for each row is managed by the system (i.e. database engine).

Every temporal table has two explicitly defined columns, each with a datetime2 data type. These columns are referred to as period columns. These period columns are used exclusively by the system to record period of validity for each row whenever a row is modified.

In addition to these period columns, a temporal table also contains a reference to another table with a mirrored schema. The system uses this table to automatically store the previous version of the row each time a row in the temporal table gets updated or deleted. This additional table is referred to as the history table, while the main table that stores current (actual) row versions is referred to as the current table or simply as the temporal table. During temporal table creation users can specify existing history table (must be schema compliant) or let system create default history table.

The table structures of the current and history tables must be identical.

How does this work with the ODS database?

System Versioned tables allow for temporal queries to be run on the ODS. These queries are ANSI SQL 2011 compliant. In order to enable this, the plugin generates SQL scripts that do the following:

1. Alters the tables in the source schema (e.g. edfi) to add two required columns SysStartTime and SysEndTime which are added as hidden.
2. Creates a history schema for each source schema defined in MetaEd e.g [edfi_history] for [edfi].
3. In the history schema, creates the tables that are present in the source schema as bare tables i.e. without constraints, identity, triggers, etc.

What about future upgrades/schema changes?

Some changes, when applied to the source schema table will also update the history table. Other changes need to be applied to the history table directly. The recommendation is to manage changes through a migration process that can account for the nuances of the process.


This plugin can be used to generate a new set of history tables based on a new source MetaEd model, however the schema comparison with an existing deployment of the history tables will need to be done separately.
Plugin Installation and Integration

Installation

Atom MetaEd-IDE setup

1. From Atom, Navigate to MetaEd > Settings
2. Make sure Ed-Fi Data Standard Version and Ed-Fi ODS/API version are 2.x

Setup registry for Certica-Public

1. Browse to the folder where you have MetaEd-IDE
2. Create a file with name .yarnrc (if you don’t already have one)
3. Add Certica Public registry (as scoped registry) on a new line to the .yarnrc file.
   • "@certica:registry" "https://www.myget.org/F/certica-public/npm/"

Install Certica SVT plugin

1. MetaEd IDE end users who want to include a plugin in their environment can do so with a "yarn" command similar in usage to upgrading their MetaEd IDE version. Plugins can be installed with the "yarn add" command.
   • yarn add metaed-plugin-certica-systemversiontables@npm:@certica/metaed-plugin-certica-systemversiontables
2. Plugins can be upgraded to newer versions with the "yarn upgrade" command.
   • yarn upgrade metaed-plugin-certica-systemversiontables@npm:@certica/metaed-plugin-certica-systemversiontables

Integration

Ed-Fi ODS/API Changes

If a precompiled version of the Ed-Fi ODS API 2.x is used, such as the Cloud ODS, the following changes may not be required, however please note this scenario has not been tested.

Ed-Fi-ODS Repository

1. Code Changes to ignore hidden columns during code generation.
   b. Update the private string constant COLUMN_SQL, just above the ORDER BY statement, add the following WHERE clause
      • WHERE COALESCE(sc.is_hidden,0) = 0
2. Test Changes (optional)
   Ignore broken XML bulk load tests.
   Bulk load works, however, the presence of the SystemStart and End columns in the Bulk database causes an issue for the tests.
   b. Add an Ignore attribute to the function
      • Should_have_a_record_in_ODS_that_matches_inserted_data
3. After building a MetaEd-IDE project with the **metaed-plugin-certica-systemversiontables** plugin
   a. System Version scripts will be generated as a part of MetaEdOutput for the project you are working under
   b. Scripts location
      - EdFi SVT: \MetaEdOutput\EdFi\Database\SQLServer\ODS\Structure
      - Extension SVT: \MetaEdOutput\Extension\Database\SQLServer\ODS\Structure
   c. Copy these scripts to separate temporary working folders
   d. Renumber the scripts across both folders in the sequence they should be executed.
      - The first script should have a sequence higher than the current scripts in the target `Ed-Fi-ODS-Implementation\Database\Structure\EdFi` folder.
   e. After renaming these scripts, move them to:
      - Ed-Fi-ODS-Implementation\Database\Structure\EdFi
      - Make sure the scripts from both the locations mentioned in step b are here and not in separate folders.
   f. These scripts will be picked up when running `initdev`

**System Version Table Usage**

There are several categories of temporal queries that can be run with different temporal sub-clauses such as:

- Point in time
- Range

**Point In Time Queries**

Point in time queries enable data to be returned 'as of' a given point in time. They are powerful in their ability to allow data to be viewed as it was (or if the historical data has gone through a major migration, as it would have been). It is also one of the simpler ways to query historical data as it can be easier to manage cross table joins to ensure lookups (like descriptors) have a match, and not no match (the lookup no longer or doesn't yet exist), or multiple matches (the look up has changed over time). For this reason, point in time or partial point in time queries are recommended for basic temporal analysis.

**Basic Queries**

This query returns all the current student table data for students that have disciplinary incidents in the current table.

**Code 1- Current**

```sql
SELECT edfi.Student.*
FROM edfi.Student
inner join edfi.StudentDisciplineIncidentAssociation d on d.StudentUSI = edfi.Student.StudentUSI
```
This query adds a temporal dimension but drops the join. It returns all the students in the student table with a last name that starts with 'S' as of a given date. It is a 'full' temporal query because all the tables involved in the query are being accessed at the same point in time.

*Code 2- Full Temporal, Single Table*

```
SELECT *
FROM edfi.Student
For SYSTEM_TIME as of '2019-04-11 05:00'
Where LastSurname like 'S%'
```

**Hybrid Temporal Query**

These queries blend tables of different temporal periods (current and a previous point in time in these examples), to show results for the current student table data for all current students that had disciplinary incidents in the associated discipline table as of a previous date.

```
-- Current student, historical association.
SELECT *
FROM edfi.Student s
where s.StudentUsi in (select Studentusi from edfi.StudentDisciplineIncidentAssociation For SYSTEM_TIME as of '2019-04-11 05:00');

SELECT s.*
FROM edfi.Student s
inner join edfi.StudentDisciplineIncidentAssociation
For SYSTEM_TIME as of '2019-04-11 05:00' d
on d.StudentUSI = s.StudentUSI;

SELECT s.*
FROM edfi.Student s,
edfi.StudentDisciplineIncidentAssociation
For SYSTEM_TIME as of '2019-04-11 05:00' d
where d.StudentUSI = s.StudentUSI;
```

These types of queries can answer interesting questions, however the greater the number of temporal periods the more complicated and error prone the queries become. The recommendation to help keep queries simple and manageable is to keep queries to a very small number of temporal periods (1 or 2) and only join lookups (descriptors) to base tables in the same temporal period. The temporal intersections should occur on shared keys and typically filter, not merge results.

**Full Temporal Query (time traveling)**

These queries allow for time traveling to a specific point in time and show results for the student table data that had disciplinary incidents in the table as of a previous date. There is not a chance of missing relational data since all tables are being examined at the same point in time by using a single SQL variable that ensures consistency.
It's important to note that all the system times that are selected against are UTC. If one wants to use local time zones, the point in time variables can be defined as follows to adjust from the source time zone to UTC.

```
Declare @pit as Datetime = '2019-04-11 05:00'

SELECT *
FROM edfi.Student
For SYSTEM_TIME as of @pit s
where s.StudentUsi in (select Studentusi from edfi.StudentDisciplineIncidentAssociation For SYSTEM_TIME as of @pit);

SELECT s.*
FROM edfi.Student
For SYSTEM_TIME as of @pit s
inner join edfi.StudentDisciplineIncidentAssociation
For SYSTEM_TIME as of @pit d
on d.StudentUSI = s.StudentUSI;

SELECT s.*
FROM edfi.Student For SYSTEM_TIME as of @pit s,
edfi.StudentDisciplineIncidentAssociation
For SYSTEM_TIME as of @pit d
where d.StudentUSI = s.StudentUSI;
```

```
Declare @pit as DATETIMEOFFSET = CAST('2019-04-10 23:00' as datetime) AT TIME ZONE 'Central Standard Time'
SET @pit = @pit AT TIME ZONE 'UTC'
```

Range Queries

Range queries can help you see how a specifically keyed piece of data has changed over time but can be more difficult to align with other data changes due to there being a possibility of multiple records for the same key, over the time examined. One must be careful with joining multiple tables when using ranged queries because relational integrity is not enforced across time and can result in unexpected results unless all temporal implications are considered.

For additional information about ranged queries please see:


Limitations

Recording time

When doing temporal queries remember that this is what the data looked like at a point in time. If an event occurred during that time, but a record was not present for it in the table, then it won't be returned by the query.
For example in the queries above if a student discipline incident occurred on 04/05/2019 but the record was not sent by the system of record to the ODS until 04/15/2019 then the as of query would not contain the record for that student even though the event actually took place before the as-of time specified.

The temporal tables represent what was in the database at a point in time, not events at a point in time.

**Ed-Fi ODS API Behavior**

The Ed-Fi ODS API has, by default, checks to prevent unchanged resources from being persisted in to the ODS. This helps prevent extraneous change record from being generated in the history tables.

However, an Ed-Fi resource can span multiple table entities in the database. This means that there can be small variations in the time between when a resource started to be processed into the database and when all the constituent table rows were persisted. Additionally, even if the root entity for a resource (e.g. Staff or Student) was not updated, if a child entity was updated (e.g. StudentAddress or StaffAddress) the root record is updated to show that the resource was modified. These variations need to be accounted for when querying the temporal tables.