Migration Utility

Migration Utility v1.1.0

May 2020: On behalf of the Ed-Fi community, we are pleased to announce the release of Migration utility 1.1.0 which includes support for upgrade to the latest Ed-Fi ODS / API v3.4 version.

Overview

The Migration Utility is a command-line tool built to upgrade the schema of an ODS instance up to the latest version.

It currently supports data migration from Ed-Fi Data Standard v2.0 and Ed-Fi Data Standard v2.2 to Ed-Fi Data Standard v3.2. The utility has out-of-the-box support for migrating an as-shipped ODS to the latest version. With additional customized scripting, the Migration Utility can be easily adapted and used to migrate extended ODS instances. ODS shared instances may take advantage of this utility.

Contents:

- Usage Scenarios
- Developer Quick Start
- Usage Walkthrough

Usage Scenarios

The following table summarizes the supported scenarios for the migration utility:

<table>
<thead>
<tr>
<th>Database Type</th>
<th>Databases</th>
<th>Upgrade/Migration strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Databases</td>
<td>EdFi_Ods, EdFi_Ods_YYYY, EdFi_Ods_Sandbox</td>
<td>The current migration utility release supports an in-place upgrade for the following upgrade paths: 2.4 -&gt; 3.4, 2.5 -&gt; 3.4, 3.0 -&gt; 3.4. The migration utility also supports migrating extensions with additional custom scripts.</td>
</tr>
<tr>
<td>Support Databases</td>
<td>EdFi_Admin, EdFi_Security, EdFi_Bulk</td>
<td>These databases can either be recreated, or existing Entity Framework Migrations can be used to migrate them.</td>
</tr>
<tr>
<td>Transient Databases</td>
<td>EdFi_ODS_Empty, EdFi_Ods_Minimal_Template, EdFi_Ods_Populated_Template</td>
<td>No upgrade supported.</td>
</tr>
</tbody>
</table>

Developer Quick Start

The basic steps are simple:

- Restore a backup copy of the target ODS to your local SQL Server instance. We recommend:
  - Start with the basic Version 2.x: Sample ODS Download: EdFi.Samples.Ods/2.0.0.21. This is a small sample ODS, ideal for initial development and testing.
  - Move on to the larger Version 2.x: Glendale 2.0 ODS Backup (created 2018-06-13). This is a much larger ODS containing sample data, useful for validation and QA.
- Choose one of the two options below to launch the migration utility:

  **Step 1.** Install the Ed-Fi ODS Migration tool:
Step 1. Clone git repository: https://github.com/Ed-Fi-Alliance/Ed-Fi-MigrationUtility

Step 2. Open the Visual Studio solution file, Migration.sln.

Step 3. Set up command line input for debugging in Visual Studio 2017:
- Right click the EdFi.Ods.UtilitiesMigration project.
- Select properties.
- Select debug.
- Add command line arguments:

```
Example Command Line Arguments: Grand Bend ODS

.
```

Step 4. Set the EdFi.Ods.UtilitiesMigration project as your startup project.

Step 5. Launch in debug mode (F5).

### Development Overview: The Basics

The table below describes files and folders used by the Migration Utility along with a description and purpose for each resource.

<table>
<thead>
<tr>
<th>Overview Item</th>
<th>Needed by Whom?</th>
<th>Brief Description &amp; Purpose</th>
</tr>
</thead>
</table>
| **Script Directory** | Users writing custom upgrade scripts, Maintainers of the upgrade utility | All database upgrade scripts go here, including:  
- Ed-Fi upgrade scripts.  
- Custom upgrade scripts (user extensions).  
- Compatibility checks.  
- Dynamic / SQL-based validation.  
- Subdirectories contain code for each supported ODS upgrade version. |
| **Directory** | Maintainers of the upgrade utility only | Ed-Fi-Standard XML files containing Descriptors for each ODS version.  
- These XML files are imported directly by the scripting in the Utilities\Migration\Scripts directory above. |
| **Library/Console** | Maintainers of the upgrade utility only | Small, reusable library making use of DbUp to drive the main upgrade.  
- Executes the SQL scripts contained in the Utilities\Migration\Scripts directory above.  
- Takes a configuration object as input, and chooses the appropriate scripts to execute based on ODS version and current conventions. |
DevelopmentTroubleshooting

This section outlines general troubleshooting procedures.

Compatibility Errors

- Before the schema is updated, the ODS data is checked for compatibility. If changes are required, the upgrade will stop and exception will be thrown with instructions on how to proceed.
  An example error message follows:

  ![Error message]

  - After making the required changes (or writing custom scripts), simply launch the upgrade utility again. The upgrade will proceed where it left off and retry with the last script that failed.

Other Exceptions During Development

- Similar to compatibility error events, the upgrade will halt and an exception message will be generated during development if a problem is encountered.
- After making updates to the script that failed, simply re-launch the update tool. The upgrade will proceed where it left off starting with the last script that failed.
- If you are testing a version that is not yet released, or if you need to re-execute scripts that were renamed or modified during active development: restore your database from backup and start over.
- Similar to other database migration tools, a log of scripts successfully executed will be stored in the default journal table. DbUp's default location is [dbo].[SchemaVersions].
- A log file containing errors/warnings from the most recent run may be found by default in C:\ProgramData\Ed-Fi-ODS-Migration\Migration.log.

Additional Troubleshooting

- The step-by-step usage guide below contains runtime troubleshooting information.

Design/Convention Overview

The table below outlines some important conventions in the as-shiped Migration Utility code.

<table>
<thead>
<tr>
<th>What</th>
<th>Why</th>
<th>Optional Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-place upgrade</td>
<td>Database upgrades are performed in place rather than creating a new database copy</td>
<td>Users who have extension tables (or any other schema with foreign key dependencies on the edfi schema) will be notified, and must explicitly acknowledge it by adding the BypassExtensionValidationCheck option at the command line when upgrading. This will ensure that the installer is aware that custom upgrade scripts may be required.</td>
</tr>
<tr>
<td>Extensions</td>
<td>Preserve all unknown data in extension tables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ensure errors and exceptions are properly generated and brought to the upgrader’s attention during migration if the upgrade conflicts with an extension or any other customization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>As a secondary concern, this upgrade method was chosen to ease the upgrade process for a Cloud-based ODS (e.g., on Azure).</td>
<td></td>
</tr>
<tr>
<td>Sequence of events that occur during upgrade</td>
<td>Minimize the number of scripts with complex dependencies on other scripts in the same directory upgrade step.</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Specifics differ for each version, but in general the upgrade sequence executes as follows</td>
<td>Compatibility checks are designed to run before any changes to the edfi schema have been made. This prevents the user from having to deal with a half-upgraded database while making updates.</td>
<td></td>
</tr>
<tr>
<td>1. Validate user input at the command line</td>
<td>Initial hash codes used for data validation also must be generated before touching the edfi schema to ensure accuracy.</td>
<td></td>
</tr>
<tr>
<td>2. Create tempdata (tables/stored procedures) that will be used for upgrade</td>
<td>• It is also better for performance to do this step while all of our indexes are still present.</td>
<td></td>
</tr>
<tr>
<td>3. Check current ODS data for upgrade compatibility, and display action messages to the user if existing data requires changes</td>
<td>• Dropping of constraints, views, etc is taken care of before making any schema changes to prevent unexpected sql exceptions.</td>
<td></td>
</tr>
<tr>
<td>4. Before modifying the edfi schema: calculate and store hash values for primary key data that expected NOT to change during upgrade</td>
<td>• New descriptors are imported as an initial step before making changes to the core tables. This ensures that all new descriptor data is available in advance for reference during updates.</td>
<td></td>
</tr>
<tr>
<td>5. Drop views, constraints, stored procs</td>
<td>• After creation of descriptors, the sequence of the next steps is designed to ensure that all data sources exist unmodified on the old schema where we expect it to.</td>
<td></td>
</tr>
<tr>
<td>6. Import descriptor data from XML</td>
<td>1. Create tables that are brand new to the schema only (and populate them with existing data)</td>
<td></td>
</tr>
<tr>
<td>7. Create all new tables for this version that did not previously exist</td>
<td>2. Modify existing tables (add/drop columns, etc)</td>
<td></td>
</tr>
<tr>
<td>8. Update data in existing tables</td>
<td>3. Drop old tables no longer needed</td>
<td></td>
</tr>
<tr>
<td>9. Drop old tables</td>
<td>• Foreign keys, constraints, etc are all added back in once the new table structure is fully in place.</td>
<td></td>
</tr>
<tr>
<td>10. Create views /constraints/stored procs for the new version</td>
<td>• Once the edfi schema is fully upgraded and will receive no further changes, we can perform the final data validation check.</td>
<td></td>
</tr>
<tr>
<td>11. Validation check: recalculate the hash codes generated previously, and make sure that all data that is not supposed to change was not mistakenly modified</td>
<td>The v2x to v3 upgrade is a good example case to demonstrate the upgrade steps working together due its larger scale:</td>
<td></td>
</tr>
<tr>
<td>12. Drop all temporary migration data</td>
<td>• For v2x to v3: All foreign keys and other constraints were dropped during this upgrade in order to adopt the new naming conventions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Also for v2x to v3: ODS types were replaced with new descriptors. This change impacted nearly every table on the existing schema.</td>
<td></td>
</tr>
<tr>
<td>One script per table in each directory, where possible</td>
<td>Troubleshooting, Timeout prevention</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Scripts are named in the format:</td>
<td>Custom, unknown extensions on the ODS are common. As part of the process of upgrading a highly-customized ODS, an installer is likely to run into a sql exception somewhere in the middle of upgrade (usually caused by a foreign key dependency, schema bound view, etc).</td>
<td></td>
</tr>
<tr>
<td>### TableName [optional_tags].sql</td>
<td>In general, we do not want to attempt to modify an unknown/custom extension on an installer's behalf to try and prevent this from happening. It is important that a installer be aware of each and every change applied to their custom tables. Migration of custom extensions will be handled by the installer.</td>
<td></td>
</tr>
<tr>
<td>This convention does not apply to operations that are performed dynamically</td>
<td>Considering the above, in the event an exception does occur during upgrade, we want to make the troubleshooting process as easy as possible. If an exception is thrown, an installer should immediately be able to tell:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Which table was being upgraded when it occurred (from the file name)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• What were the major changes being applied (from the file tags)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• What went wrong (from the exception message)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Where to find the code that caused it</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Many issues may be fixable from the above information alone. If more detail is needed, the installer can view the code in the referenced script file. By separating script changes by table, we make an effort to ensure that there are only a few lines to look through (rather than hundreds)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>In addition, each script will be executed in a separate transaction. Operations such as index creation can take a long time on some tables with a large ODS. Splitting the code into separate transactions helps prevent unexpected timeout events.</td>
<td></td>
</tr>
<tr>
<td>Most change logic is held in sql scripts (as of V3)</td>
<td>The major downside of this approach is the large number of files it can produce. For example, the v2x to v3 upgrade was a case where all existing tables saw modifications. This convention generates a change script for every table in more than one directory.</td>
<td></td>
</tr>
<tr>
<td>As of v3: Most of the upgrade logic is performed from the SQL scripts, rather than using .NET based upgrade utility to write database changes directly</td>
<td>With updates becoming more frequent in the future, future versions should not be impacted as heavily.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>As of v3, most upgrade tasks are simple enough where they can be executed straight from SQL (given a few stored procedures to get started).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Given this advantage, effort was made to ensure that each part of the migration tool (console utility, library, integration tests) could be replaced individually as needed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The current upgrade utility contains a library making use of DbUp to drive the upgrade process. In the future, if/when this tool no longer suits our needs, we should be able to take existing scripting and port it over to an alternative upgrade tool (such as RoundhousE); or even a custom built tool if the need ever arises.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This convention could (and should) change in the future if upgrade requirements become too complex to execute from SQL scripting alone.</td>
<td></td>
</tr>
</tbody>
</table>
## Two types of data validation/testing options

<table>
<thead>
<tr>
<th>Option</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dynamic, SQL based</strong></td>
<td>- Ensures data that is expected to remain the same does not change</td>
</tr>
<tr>
<td></td>
<td>- Can run on any ODS in the field even if the data is unknown</td>
</tr>
<tr>
<td>Integration tests</td>
<td>- Runs on a known, given set of inputs</td>
</tr>
<tr>
<td></td>
<td>- Used to test logic in areas where changes should occur</td>
</tr>
<tr>
<td><strong>Prevent data loss</strong></td>
<td>The first type of validation, (dynamic, sql based) is executed on data that we know should not change during the upgrade.</td>
</tr>
<tr>
<td></td>
<td>- The source and destination tables do not need to be the same. This validation type is most commonly used to verify that data was correctly moved to the expected destination table during upgrade.</td>
</tr>
<tr>
<td></td>
<td>- Can be executed on any field ODS to ensure that unknown datasets do not cause unexpected data loss during upgrade.</td>
</tr>
<tr>
<td></td>
<td>- The data in these tables does not need to be known</td>
</tr>
<tr>
<td><strong>Integration tests</strong></td>
<td>The second type of data validation, integration test based, is used to test the logic and transformations where we know the data should change:</td>
</tr>
<tr>
<td></td>
<td>- For example, during the v2x to v3 upgrade, descriptor namespaces are converted from the 2.x &quot;<a href="http://EdOrg/Descriptor/Name.xml">http://EdOrg/Descriptor/Name.xml</a>&quot; format to the 3.x &quot;uri://EdOrg/Name&quot; format. Integration tests are created to ensure that the upgrade logic is functioning correctly for several known inputs.</td>
</tr>
<tr>
<td></td>
<td>Together, the two validation types (validation of data that changes, and validation of data that does not change) can be used to create test coverage wherever it is needed for a given ODS upgrade.</td>
</tr>
</tbody>
</table>

## Upgrade Issue Resolution Approach

It is common to encounter scenarios where data cannot be upgraded directly from one major version to another due to schema changes. A common example is that a primary key change causes records allowable by the previous version schema to be considered duplicates on the upgrade version schema and therefore not allowed.

The general approaches included here are a result of collaboration with the community on how to resolve these common situations, and are documented here as a reference for utility developers to apply in their own work.

1. **Approach 1**: Make non-breaking, safe changes to data on the user's behalf. This is the preferred option when practical and safe to do so.
   - Example: Duplicate session names in the v2.x schema that will break the new v3.x primary key will get a term name appended during upgrade to meet schema requirements.
   - Consider logging a warning for the user to review to inform them that a minor change has taken place, and mention the log file in documentation.

2. **Approach 2**: Throw a compatibility exception asking the user to intervene with updates. Used when we are unable to safely make assumptions on the user's behalf to perform the update for them.
   - See the troubleshooting section of this document for an example of this.
   - The returned user-facing message should explain what updates are required, and very briefly include the reason why if it is not immediately obvious.
   - Compatibility exceptions should be thrown before proceeding with core upgrade tasks.
   - We want to make sure that we are not asking someone to make updates to a database in an intermediate upgrade state.
   - Each message class has a corresponding error code, mirrored in code by an enumeration.
   - This is done so that specific compatibility scenarios can be covered by integration tests where needed.

3. **Approach 3**: Back up deprecated objects that cannot be upgraded into another schema before dropping them. Last resort option that should be mainly restricted to deprecated items that are no longer have a place in the new model.
   - As a general approach, it is preferred to avoid dropping data items without a way to recover them in case of disaster. The user may choose to delete the backup if they desire.
   - Avoid this option for tables that exist in both models but simply cannot be upgraded. Should this (hopefully rare) situation occur, consider the option throwing a compatibility exception instead and ask the user back up/empty the table before proceeding.

In general, the option requiring the least amount of user effort while safely preserving all data has been chosen in order to reduce user burden as much as we are able.
Usage Walkthrough

This section explains how to upgrade an existing v2.x ODS / API to version 3.4. The ODS v3.4 database update includes the latest enhancements based on feedback from the Ed-Fi Community.

The steps can be summarized as:

- Step 1. Read the Ed-Fi ODS v3.4 Upgrade Overview
- Step 2. Install Required Tools
- Step 3. Install the Ed-Fi ODS Migration Utility
- Step 4. Back Up and Create a Working Copy of the 2.x Target Database
- Step 5. (Multi-Year ODS Only) Create a Calendar Configuration File
- Step 6. (ODS with NO Extensions Only) Run the Migration Utility
- Step 7. (Extended/Modified ODS Only) - Perform a Test Migration Without Extensions
- Step 8. (Extended/Modified ODS Only) - Write Custom Migration Scripts for Extensions
- Step 9. Post-Upgrade: Review Data Changes

A compatibility reference chart, a command-line parameter list, and a troubleshooting guide are included. Each upgrade step is outlined in detail below.

Step 1. Read the Ed-Fi ODS v3.4 Upgrade Overview

Before you get started, you should review and understand the information in this section.

Target Audience

These instructions have been created for technical professionals, including software developers and database administrators. The reader should be comfortable with performing the following types of tasks:

- Creating and managing SQL Server database backups
- Performing direct SQL commands and queries
- Execution of a command-line based tool that will perform direct database modifications
- Creating a configuration file for upgrade (.csv format)
- Writing custom database migration scripts (Extended ODS only)

General Notes

- Your v2.x of the Ed-Fi ODS / API will be checked for compatibility automatically during the migration process. If changes are needed, you will be prompted at the command line by the migration utility. A summary of commonly encountered compatibility conditions has been included in this section for reference.
- The new v3.4 schema contains upgrades to the structure of primary keys on several tables. In most instances, these new uniqueness requirements will be resolved automatically for you with no action required.
- There are some areas where new identities cannot be generated automatically on your behalf during upgrade. These tables will need to be updated manually.

Compatibility Conditions

This section describes compatibility conditions (i.e., requirements that may need intervention for the compatibility tool to function properly) and suggested remediation.

<table>
<thead>
<tr>
<th>Table</th>
<th>Data Compatibility Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[edfi].[Assessment]</code></td>
<td>All assessments must have a <code>[Namespace]</code> set. (This data may be found in <code>[edfi].[Assessment]</code> or <code>[edfi].[AssessmentFamily]</code>).</td>
</tr>
</tbody>
</table>
The upgrade utility must be able to locate an \[EducationOrganizationId\] for every student with data in the listed tables to proceed.

The easiest way to meet this requirement is to ensure that every student has a corresponding record in \[edfi\].\{StudentSchoolAssociation\} or \[edfi\].\{StudentEducationOrganizationAssociation\}.

The column \[StateOfIssueStateAbbreviationTypeId\] must be non-null for all records.

This is the abbreviation for the name of the state (within the United States) or extra-state jurisdiction in which a license/credential was issued.

(Any extension table)

Additional steps are required when extensions are present. Please review the upgrade process detailed below for additional guidance.

<table>
<thead>
<tr>
<th>Table</th>
<th>Data Compatibility Requirement</th>
</tr>
</thead>
</table>
| \[edfi\].\{GradingPeriod\} | • There must be no duplicate \[PeriodSequence\] values for the same school during the same grading period.  
• If prompted by the upgrade utility, all \[PeriodSequence\] values must be non-null  

**Technical Details:**

This compatibility requirement is a result of a primary key change between 2.x and v3.1  

• Old 2.0 Primary Key: \{GradingPeriodDescriptorId\}, \{SchoolId\}, \{BeginDate\}  
• New 3.1 Primary Key: \{GradingPeriodDescriptorId\}, \{SchoolId\}, \{PeriodSequence\} (new), \{SchoolYear\} (new), \{BeginDate\} is removed |

| \[edfi\].\{DisciplineActionDisciplineIncident\} | The 3.1 schema no longer allows discipline action records with students that are not associated with the discipline incident.  
Every record in \[edfi\].\{DisciplineActionDisciplineIncident\} must have a corresponding record in \{edfi\}.\{StudentDisciplineIncidentAssociation\} with the same \{StudentUSI\}, \{SchoolId\}, and \{IncidentIdentifier\}. |

| \[edfi\].\{RestraintEvent\} | Ensure that there are no duplicate \[RestraintEventIdentifier\] values for the same student at the same school |

| \[edfi\].\{OpenStaffPosition\} | Ensure that there are no two duplicate \[RequisitionNumber\] entries for the same education organization |

| \[edfi\].\{AccountCode\} | This table must be empty before upgrading. Due to a major schema change, data in this table cannot be preserved from 2.x to 3.x |

**Other Compatibility Conditions**
There are several other less common items not included above. The migration utility will check for these items automatically and provide guidance messages as needed. For additional technical details, please consult the Troubleshooting Guide below.

Step 2. Install Required Tools

- Ensure that you have an instance of the Ed-Fi ODS / API running locally that has been set up following the instructions in Getting Started - Installation Steps.

Step 3. Install the Ed-Fi ODS Migration Utility

- Install the Ed-Fi ODS Migration utility from a PowerShell prompt using the following command:

```
c:\>mkdir {YourInstallFolder}
c:\>dotnet tool install EdFi.Ods.Utilities.Migration --tool-path {YourInstallFolder}
```

Step 4. Back Up and Create a Working Copy of the 2.x Target Database

- Create a full backup of the target v2.x EdFi_Ods database.
- Restore this backup to your SQL Server instance as a copy, in a new location.
- Make note of the database connection string as it will be required for a later step.

Step 5. (Multi-Year ODS Only) Create a Calendar Configuration File

In most cases, you can skip this step if the ODS contains valid calendar data for only one school year. The migration tool will usually be able to resolve this item automatically for you.

The v3.x ODS upgrade includes enhancements to the calendar, including tracking of school year information for every calendar event and session. To ensure that your new calendar is accurate, the migration tool will need to know which school year to associate with every calendar item in your ODS.

Example calendar configuration files can be downloaded from the download panel on the right.

The following is an example of a calendar configuration file (CSV) for Grand Bend ISD:

```
Example 1: Grand-Bend calendar-config.csv
SchoolId,SchoolYear,StartDate,EndDate
```

Note that each line contains a SchoolId, SchoolYear, the first calendar day and the last calendar day of the school year.

- Create a new calendar configuration file (.CSV) by referencing the above examples. Your configuration file must match the following format (including header row):

```
Example 2: calendar-config.csv
SchoolId,SchoolYear,StartDate,EndDate
{Your_First_School_Id},20XX,{FirstDayOfSchoolYear},{LastDayOfSchoolYear}
{Your_Second_School_Id},20XX,{FirstDayOfSchoolYear},{LastDayOfSchoolYear}
...
```

- Store the configuration file in a location that is accessible by your SQL server

Step 6. (ODS with NO Extensions Only) Run the Migration Utility
Open a new command prompt and navigate to the Ed-Fi ODS Migration utility installation folder:

```
CD {YourInstallFolder}
```

Launch the console tool. Modify the below example parameters to match your environment. Replace the example "uri://ed-fi.org" with the Namespace Prefix to insert for your new descriptors:

```
```

Notes for upgrades performed on a remote database

- The calendar configuration file path provided during upgrade must be accessible by your SQL server for importing and validation.
- The XML files used to create v3.x descriptors are by default located in "{YourInstallFolder}\store\edfi.ods.utilities.migration\1.0.0\edfi.ods.utilities.migration\1.0.0\tools\netcoreapp2.2\any\Descriptors\Descriptors\3.1". This directory must also be accessible by your SQL server for importing. If you need to copy this directory to a new location, you may make use of the --DescriptorXMLDirectoryPath "C:\PATH\TO\YOUR\DESCRIPTOR\XML" parameter to point the migration tool to the new location of these items.
- Users who are running the migration tool on an extended ODS must use the --BypassExtensionValidationCheck option during upgrade in order to allow the tool to make changes when extension dependencies are present.
- Details on upgrading an extended ODS are included in the steps below.
- The console tool will check your ODS for compatibility, and then proceed to perform an in-place upgrade on the specified database.
- If you encounter any compatibility messages or errors during upgrade
  - During the upgrade process, your ODS will be checked for compatibility with the latest version. If changes are required, you may encounter a compatibility message, and the upgrade will stop.

**Example**

```
Action Required: [ [U] Unchecked - This action must have a [R]equired or [O]ptional value. This value may be found in [S]pecif\n
```
- After making the required changes (or writing custom scripts), simply launch the upgrade utility again. The upgrade will proceed where it left off and retry with the last script that failed.
- See the Troubleshooting Guide below for additional guidance
- Once the process runs to completion with no errors, your upgrade is complete
- For a summary of all available parameter options for the Ed-Fi ODS console-based migration tool, please see the included reference below.

**Step 7. (Extended/Modified ODS Only) - Perform a Test Migration Without Extensions**

- It is highly recommended that you first perform a test migration without your extension tables present in order to ensure that all core upgrade requirements have been met
  - On your ODS copy, temporarily drop all extension tables.
  - Complete the migration process exactly as detailed in Step 6, above.
  - Once the migration has completed successfully on the core Ed-Fi data, restore your v2.0 ODS working copy from backup, including all extension tables, and proceed with the next step

**Step 8. (Extended/Modified ODS Only) - Write Custom Migration Scripts for Extensions**
Step 8a. Locate the Ed-Fi Migration scripts and review the current upgrade conventions

- The edfi migration scripts are located by default in `{YourInstallFolder}\store\edfi.ods.utilities.migration\1.0.0\tools\netcoreapp2.2\any\Scripts`.
- All SQL scripts are run in numerical order, with version-specific upgrade further categories.
- The v2.0 to v3.1 upgrade passes though v2.5 on the way. It will execute scripts in the following version directories:
  - v24_to_v25
  - v25_to_v31

The directory structure appears as follows:

<table>
<thead>
<tr>
<th>Directory Name</th>
<th>What does this directory contain?</th>
<th>What scripts should you add here?</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 Setup</td>
<td>The first group of scripts that is run before data migration begins. Setup-type scripts, such as the creation of stored procedures used during upgrade are added here.</td>
<td>Optional: You may place custom scripts that you would like to execute before compatibility checks or schema updates begin here. Scripts in each version directory will be executed in numerical order.</td>
</tr>
<tr>
<td>01 Compatibility Check</td>
<td>Ed-Fi data compatibility preconditions are checked using the scripts in this directory.</td>
<td>None needed.</td>
</tr>
</tbody>
</table>
| 02 Upgrade | Contains the core upgrade scripts for each version. All SQL scripts are run in numerical order by path. Details have been included below this chart. The v2.0 to v3.1 upgrade will execute scripts in the following version directories:  
  - v24_to_v25  
  - v25_to_v31 | Optional:  
  - You may place custom scripts that you would like to execute on your v2.0 database in directory: 02 Upgrade\v24_to_v25\{step}.  
  - Scripts that you want to execute on a v2.5 database may be placed in directory: 02 Upgrade\v25_to_v31\{step}.  
  - Scripts in each version directory will be executed in numerical order.  
  - Details on version-specific upgrade steps are included below this table. |
| 03 Cleanup | Removes temporary data generated during upgrade | Optional: You may place custom scripts that you would like to execute after all schema updates have been performed here. Scripts in will be executed in numerical order order. |

Version-Specific Upgrade Step Details for v2.x to v3.4

The core upgrade script directory, 02 Upgrade, contains several upgrade steps for each version which are detailed below. You can insert custom scripts at any point in the process as needed for custom Extensions.

<table>
<thead>
<tr>
<th>Step/Directory Name</th>
<th>What is contained</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>Source Validation Check</td>
</tr>
<tr>
<td>2</td>
<td>Create Upgrade Tempdata</td>
</tr>
<tr>
<td>3</td>
<td>Drop Extended Properties</td>
</tr>
<tr>
<td>4</td>
<td>Drop Views</td>
</tr>
<tr>
<td>5</td>
<td>Drop Procedures, Types</td>
</tr>
<tr>
<td>6</td>
<td>Drop Constraints</td>
</tr>
<tr>
<td>7</td>
<td>Import Descriptors</td>
</tr>
<tr>
<td>8</td>
<td>Create New Tables</td>
</tr>
<tr>
<td>9</td>
<td>Update Existing Tables</td>
</tr>
<tr>
<td>10</td>
<td>Drop Old Tables</td>
</tr>
<tr>
<td>11</td>
<td>Create Constraints</td>
</tr>
<tr>
<td>12</td>
<td>Create User Roles</td>
</tr>
<tr>
<td>13</td>
<td>Create Views</td>
</tr>
<tr>
<td>14</td>
<td>Create Procedures</td>
</tr>
<tr>
<td>15</td>
<td>Create Extended Properties</td>
</tr>
<tr>
<td>16</td>
<td>Destination Validation Check</td>
</tr>
</tbody>
</table>

**Step 8b. Write scripts to drop ALL foreign keys on your extension tables that depend on the edfi schema**

- All edfi primary keys and indexes are dropped automatically during upgrade. The existence of any external constraint that depends on edfi data will result in an error state.
You may insert these scripts in the following directory
- Option 1: Insert with similar Ed-Fi scripts in directory: Scripts/02 Upgrade/{version}/## Drop Constraints
- Option 2: (To run before all upgrade tasks begin): Scripts/00 Setup/{version}

Scripts will be executed in numerical order. Number your scripts such that they run before the edfi scripts are executed. This will prevent a sql exception from occurring when dependent items on the edfi schema are dropped.

Tip: The following query can quickly identify all constraints that need to be dropped for migration to proceed. Use this as a guide when writing your custom migration scripts.

### Quickly show all external dependencies that will need to be dropped

```sql
SELECT DISTINCT
    parentSchema.name AS 'External Schema Name',
    parentObject.name AS 'Table Name',
    constraintObject.name AS 'Constraint Name',
    CONCAT('References ', referencedSchema.name, '.', referencedObject.name) AS [Conflict Reason],
    CONCAT('ALTER TABLE ', parentSchema.name, '.', parentObject.name, ' DROP CONSTRAINT ', constraintObject.name) AS [Example Code For Reference]
FROM sys.foreign_key_columns fk
INNER JOIN sys.objects parentObject
    ON fk.parent_object_id = parentObject.object_id
INNER JOIN sys.objects referencedObject
    ON fk.referenced_object_id = referencedObject.object_id
INNER JOIN sys.schemas parentSchema
    ON parentObject.schema_id = parentSchema.schema_id
INNER JOIN sys.schemas referencedSchema
    ON referencedObject.schema_id = referencedSchema.schema_id
INNER JOIN sys.objects constraintObject
    ON constraintObject.object_id = fk.constraint_object_id
WHERE referencedSchema.name = 'edfi'
AND parentSchema.name NOT IN ('edfi', 'migration_tempdata')
AND parentObject.type = 'U'
AND referencedObject.type = 'U'
```

### Step 8c. Create Other Migration Scripts as Needed for Custom Extensions

- After creating scripts that drop all dependent constraints, you can re-test your data migration to ensure that it will complete without error. Be sure to add the `--BypassExtensionValidation` check option at the command line.
- Create the remaining scripts needed to upgrade your extension tables for v3.x, using the chart provided above for reference.

### Step 8d. Create Changed Record Queries Migration Scripts for Custom Extensions (Required only if the feature was enabled)

- Changed Record Queries is an optional feature that was introduced in ODS 3.1. Changed Record Query artifacts for the core entities will be upgraded automatically by the migration utility if the feature was enabled in the source database. However, if you are using Changed Record Query feature in the source ODS and you have custom extensions, then the change scripts to update Changed Record Query artifacts for the extension tables will need to be added into the scripts folder located in Scripts/02 Upgrade/{Version}_Changes.

### Step 8e. Upgrade Your Extended ODS

- With all custom scripting in place, you may now proceed with the upgrade as detailed in Step 6. (ODS with NO Extensions Only) Run the Migration Tool, above.

### Step 9. Post-Upgrade: Review Data Changes

1. Review warnings/action items generated during upgrade
   - By default, these will be stored in "C:\ProgramData\Ed-Fi-ODS-Migration\Migration.log"
   - This message list will contain action items that may require your attention
2. Review your upgraded ODS.
A schema named [v2_to_v3_deprecated] will be created to store a copy of major objects that were dropped or altered significantly during upgrade, including v2 descriptor/type references. If desired, this entire schema is safe to delete once you are positive that the data contained will no longer be needed.

Some data elements that were part of the v2 model are either no longer a part of the v3 model, or may be altered to meet the upgraded schema requirements. The following chart summarizes a list of major items that will be altered or dropped during upgrade.

### Objects dropped during upgrade

<table>
<thead>
<tr>
<th>What</th>
<th>Where to find it after upgrade</th>
</tr>
</thead>
</table>
| **Dropped tables no longer part of the v3 model:**  
- Tables capturing Assessment Family data  
- Some PostSecondaryInstitution related references no longer used  
- V2 Ed-Fi Descriptors no longer part of the v3 model | **Tables will be copied to schema [v2_to_v3_deprecated] if they are not empty at upgrade time** |
| References to Ed-Fi Type tables that were contained in v2x descriptors. Types are no longer a part of the v3 model. | The table [v2_to_v3_deprecated].[TypeToDescriptorAssociationBackup] will contain a list of deprecated types ([CodeValue] only) that were associated with Descriptors at the time of upgrade. |
| Characteristic data that was stored directly on [edfi].[Student] incompatible with the new model because an [StudentEducationOrganizationAssociation] reference is now required beginning in v3. This includes economic disadvantaged status, school food services eligibility, and displacement status. | For all students that contain non-null data in the related columns on [edfi].[Student], an entry will be generated in [v2_to_v3_deprecated].[Student]. |

### Notes on key important items altered during upgrade

<table>
<thead>
<tr>
<th>Affected Objects</th>
<th>Change Description</th>
<th>Why</th>
</tr>
</thead>
</table>
| Types, Descriptors | - Namespace changed from v2 format (http://{edorg}/Descriptor/{DescriptorName}.xml) to v3 format (uri://{edorg}/ {DescriptorName})  
- Rare: In the event that duplicate descriptors associated with the same organization are encountered, an integer will be added to the [CodeValue] for uniqueness  
- New v3 descriptors inserted during upgrade with namespace specified at upgrade time  
- In the event that a Type table was modified from the default, these changed types will be converted into new descriptors as well | Significant type/descriptor changes have occurred between v2 and v3  
- Beginning in v3, type tables are no longer part of the model. These objects will be replaced with several Descriptors new to v3.  
- Namespaces are now required, and must be the new v3 format exactly. The upgrade utility will do the work to convert valid v2 namespaces to the new v3 format for you.  
- Rare: The schema requires that each combination of [CodeValue] + [Namespace] be unique for each Descriptor, which could result in potential duplicates in some cases where duplicate v2 [CodeValue] entries exist. |

Note: The state of the descriptor table with old namespaces is copied to [v2_to_v3_deprecated].[Descriptor] at upgrade time.
The v2 [UniqueSectionCode] is used to create the new v3 [SectionIdentifier]. For sections that are considered duplicates under the new v3 model, the new [SectionIdentifier] will contain the old [UniqueSectionCode] plus an integer that is automatically appended during upgrade.

In the older v2 model, the primary key on [edfi].[Section] permits a [UniqueSectionCode] to be associated with the same School/Session/Course multiple times.

As of v3, the key has been simplified. A [SectionIdentifier] can only be associated with the same School/Session/Course once.

In the older v2 model, the Session table used [TermDescriptorId] in the key. The v3 key replaces [TermDescriptorId] with [SessionName].

---

### Ed-Fi ODS Migration Tool Parameter Reference

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>--Database</td>
<td>Database Connection String</td>
<td>--Database &quot;Data Source=YOUR\SQLSERVER;Initial Catalog=Your_EdFi_Ods_Database ;Integrated Security=True&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td>--DescriptorNamespace</td>
<td>Descriptor Namespace prefix to be used for new and upgraded descriptors.</td>
<td>--DescriptorNamespace &quot;uri://ed-fi.org&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Namespace must be provided in 3.x format as follows: <code>uri://education_organization_here</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Valid characters for an education organization name: alphanumeric and $-_.+!*'(),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Script Usage: Provided string value will be escaped and substituted directly in applicable SQL where the <code>$DescriptorNamespace</code> variable is used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--CredentialNamespace</td>
<td>Namespace prefix to be used for all new Credential records.</td>
<td>--CredentialNamespace &quot;uri://ed-fi.org&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Namespace must be provided in 3.x format as follows: <code>uri://education_organization_here</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Valid characters for an education organization name: alphanumeric and $-_.+!*'(),</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Script Usage: Provided string value will be escaped and substituted directly in applicable SQL where the <code>$CredentialNamespace</code> variable is used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--CalendarConfigFilePath</td>
<td>Path to calendar configuration, which must be accessible from your sql server.</td>
<td>--CalendarConfigFilePath &quot;C:\PATH\TO\YOUR\CALENDAR_CONFIG.csv&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Script Usage: Provided string value will be substituted directly in dynamic SQL where the <code>$CalendarConfigFilePath$</code> variable is used. The <code>\</code> character is not permitted by the upgrade utility for this value.</td>
<td></td>
<td>Single-Year ODS: No (unless prompted by the upgrade tool) Multi-Year ODS: Yes</td>
</tr>
<tr>
<td>--DescriptorXMLDirectoryPath</td>
<td>Path to directory containing 3.1 descriptors for import, which must be accessible from your sql server</td>
<td>--DescriptorXMLDirectoryPath &quot;C:\PATH\TO\YOUR\DESCRIPTOR\XML&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Script Usage: Provided string value will be substituted directly in dynamic SQL where the <code>$DescriptorXMLDirectoryPath$</code> variable is used. The <code>\</code> character is not permitted by the upgrade utility for this value.</td>
<td></td>
<td>Local Upgrade: No (applicable to most cases) Remote Upgrade: Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Usage Troubleshooting Guide

The below section provides additional guidance for many common compatibility issues that can be encountered during the upgrade process.

<table>
<thead>
<tr>
<th>Error received during upgrade</th>
<th>Explanation</th>
<th>How to fix it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Required: Unable to proceed with migration because the BypassExtensionValidationCheck option is disabled ...</td>
<td>An external dependency on the edfi schema has been found. As a courtesy, the migration tool will not proceed with the upgrade process without your permission. Common Examples: - You have a table on the extension schema - An object on any schema other than edfi has a foreign key that depends directly on data on the edfi schema Why: This notification is intended to bring extension items to your attention that will require manual handling. All primary keys and indexes on the edfi schema are dropped during upgrade. The existence of any external constraint that depends on these objects will result in an unhandled sql exception. (For details, see the troubleshooting section: &quot;sqlException: The constraint '{CONSTRAINT_NAME_HERE}' is being referenced by table '{TABLE_NAME_HERE}', foreign key constraint '{FOREIGN_KEY_NAME_HERE}'&quot;)</td>
<td>After reviewing the data and dependencies on your extension tables, add the --BypassExtensionValidationCheck option at the command line. This will give the migration tool permission to proceed even if there are extension items present. Please review Step 8. Write Custom Migration Scripts for Extensions (above) before proceeding.</td>
</tr>
<tr>
<td>Action Required: edfi. StaffCredential ...</td>
<td>The column StateOfIssueStateAbbreviationTypeId must be non-null for all records. This value will become part of a new primary key on the 3.1 schema.</td>
<td>Add a [StateOfIssueStateAbbreviationTypeId] for all records in [edfi]. [StaffCredential]. This is the abbreviation for the name of the state (within the United States) or extra-state jurisdiction in which a license or credential was issued. The table is compatible for upgrade if the below query returns 0 results. Expand to see code</td>
</tr>
</tbody>
</table>

SELECT * FROM [edfi]. [StaffCredential] WHERE [StateOfIssueStateAbbreviationTypeId] IS NULL
### Action Required - An EducationOrganizationId must be resolvable for every student in the following table(s) for compatibility with the upgraded schema starting in version 3.0:

(Provided list of tables includes one or more of the following):

- edfi.StudentProgramParticipation
- edfi.StudentCharacteristic
- edfi.StudentIndicator
- edfi.StudentLearningStyle
- edfi.StudentAddress
- edfi.StudentIdentificationCode
- edfi.StudentElectronicMail
- edfi.StudentInternationalAddress
- edfi.StudentLanguage
- edfi.StudentRace
- edfi.StudentDisability
- edfi.StudentTelephone
- edfi.PostSecondaryEventPostSecondaryInstitution

The upgrade utility must be able to locate an [EducationOrganizationId] for every student with data in the listed tables to proceed.

Beginning in v3.1, the schema structure now requires that these student information items be defined separately for each associated EducationOrganization rather than simply linking them to a student.

The easiest way to meet this requirement is to ensure that every student has a corresponding record in [edfi].[StudentSchoolAssociation] or [edfi].[StudentEducationOrganizationAssociation].

The upgrade tool will use this information to handle the rest of the data conversion tasks for you.

### Action Required: edfi.Assessment ...

All assessments must have a [Namespace] set. (This data may be found in [edfi].[Assessment] or [edfi].[AssessmentFamily]). In v3.x, the schema required that this column be non-null.

Add a [Namespace] for all assessment records.

The table is compatible for upgrade if the below query returns 0 results:

**Expand to see code**

```
SELECT * FROM [edfi].[Assessment] a
LEFT JOIN [edfi].[AssessmentFamily] f
ON a.[AssessmentFamilyTitle] = f.[AssessmentFamilyTitle]
WHERE COALESCE(a.[Namespace], f.[Namespace]) IS NULL
```
| Action Required: edfi. OpenStaffPosition ... | There may be no two duplicate RequisitionNumber entries for the same education organization. This is uniqueness if required for the upgraded primary key on this table. | Update the RequisitionNumber values on edfi.OpenStaffPosition. Ensure that the same value is not used twice for the same education organization.

The table is compatible for upgrade if the below query returns 0 results.

**Expand to see code**

```
SELECT
    [EducationOrganizationId],
    [RequisitionNumber], COUNT ([RequisitionNumber]) AS [NumberOfMatchingRecords]
FROM [edfi].[OpenStaffPosition]
GROUP BY
    [EducationOrganizationId],
    [RequisitionNumber]
HAVING COUNT ([RequisitionNumber]) > 1
```

| Action Required: edfi. RestraintEvent ... | There may be no two duplicate RestraintEventIdentifier values for the same student at the same school. This is uniqueness if required for the upgraded primary key on this table. | Update the RestraintEventIdentifier values on edfi.RestraintEvent. Ensure that the same RestraintEventIdentifier is not reused for the same student at the same school.

The table is compatible for upgrade if the below query returns 0 results.

**Expand to see code**

```
SELECT
    [RestraintEventIdentifier] , [SchoolId],
    [StudentUSI], COUNT ([RestraintEventIdentifier]) AS [NumberOfDuplicateRecords]
FROM [edfi].[RestraintEvent]
GROUP BY
    [RestraintEventIdentifier],
    [SchoolId], [StudentUSI]
HAVING COUNT ([RestraintEventIdentifier]) > 1
```

Action Required: edfi.
GradingPeriod ...

There may be no two duplicate `PeriodSequence` values for the same school during the same grading period.

Additionally, if prompted by the upgrade tool, all `PeriodSequence` values must be non-null.

This compatibility requirement is a result of a primary key change between 2.x and v3:

- **Old v2.0 Primary Key:** GradingPeriodDescriptorId, SchoolId, BeginDate
- **New v3.x Primary Key:** GradingPeriodDescriptorId, SchoolId, PeriodSequence (new), SchoolYear (new). (BeginDate is removed.)

Ensure that there are no two records with the same SchoolId, GradingPeriodDescriptorId, PeriodSequence, and SchoolYear.

- **Note:** The edfi. GradingPeriod new SchoolYear column is derived from the old v2.0 BeginDate value. The SchoolYear from your calendar configuration will be used.
- **Tip:** Depending on the needs of your organization, you may decide to remove unneeded conflicting records, update the edfi. GradingPeriod.PeriodSequence values, or assign a different value to edfi. GradingPeriod. PeriodSequence. GradingPeriodDescriptorId.

**Tip:** You can be certain that the table is compatible for upgrade if the below query returns 0 results.

### Version 1 - Simplified Calendar:

Expand to see code

```
SELECT [SchoolId], [GradingPeriodDescriptorId], [PeriodSequence], COUNT ([PeriodSequence]) AS [NumberOfDuplicateRecords]
FROM [edfi]. [GradingPeriod]
GROUP BY [SchoolId], [GradingPeriodDescriptorId], [PeriodSequence]
HAVING COUNT ([PeriodSequence]) > 1
```

(Requirements may be less strict than noted in the above query for some multi-year Calendars. See the compatibility check script in the 01 Bootstrap directory for exact technical requirements.)
**Action Required:** edfi.

**DisciplineActionDisciplineIncident...**

Every record in `[edfi].[DisciplineActionDisciplineIncident]` must have a corresponding record in `[edfi].[StudentDisciplineIncidentAssociation]` with the same `[StudentUSI]`, `[SchoolId]`, and `[IncidentIdentifier]`.

The V3 schema no longer allowed discipline action records with students that are not associated with the discipline incident. A foreign key will be added to the new schema enforcing this.

- Remove any records in `[edfi].[DisciplineActionDisciplineIncident]` that involve a student not associated with the incident.
- For students that are involved in the incident, add a corresponding record to `[edfi].[StudentDisciplineIncidentAssociation]` with the same `[StudentUSI]`, `[SchoolId]`, and `[IncidentIdentifier]`.

The table is compatible for upgrade if the below query returns 0 results

**Expand to see code**

```sql
SELECT d.*
FROM [edfi].[DisciplineActionDisciplineIncident] d
LEFT JOIN [edfi].[StudentDisciplineIncidentAssociation] s
ON s.[StudentUSI] = d.[StudentUSI]
AND s.[SchoolId] = d.[SchoolId]
AND s.[IncidentIdentifier] = d.[IncidentIdentifier]
WHERE s.[StudentUSI] IS NULL
```
The calendar configuration file contains the start date and end date for each school year. To support the new calendar features in V3, the migration tool uses this configuration file to assign a SchoolYear to all CalendarDate related entries in the database.

There are several variations of this type of error which all have a similar meaning. The migration tool found date records in the specified table that could not be assigned a school year based on the BeginDate and EndDate information provided.

Either the calendar configuration file will need to be edited, or data in the specified table will need to be modified.

1. Make note of the list of School Ids and the Table name provided in the error message.
2. Open your calendar configuration file. Find the line corresponding to each School Id. Ensure that the BeginDate and EndDate for each SchoolYear is accurate.
   - If a School Id is missing from the calendar configuration file, be sure to add it.
3. Check the date information in the specified table for accuracy.
   - Carefully check the ODS records in table specified by the error message for the specified School Id.
   - Make sure there are no date entries that fall outside the school year information you have provided in the calendar configuration.
   - If the table already has SchoolYear data (such as edfi.Session or edfi.GradebookEntry), make sure the SchoolYear data is accurate.

Example:

Example Query: [edfi].[Calendar Date]

```
SELECT *
FROM [edfi].[CalendarDate]
WHERE [SchoolId] = {School_Id_from_error_message}
AND (
    [Date] < {BeginDate_from_calendar_configuration_file}
    OR [Date] > {EndDate_from_calendar_configuration_file}
)
```
**SqlException:**
The constraint '{CONSTRAINT_NAME_HERE}' is being referenced by table '{TABLE_NAME_HERE}', foreign key constraint '{FOREIGN_KEY_NAME_HERE}' or similar:

The object '{OBJECT_NAME_HERE}' is dependent on column '{COLUMN_NAME_HERE}'.

This type of unhandled SQL exception occurs when the migration process tries to alter an item that is being referenced by an external object, such as a foreign key on another table, or schema-bound view.

**Common causes**
- The `--BypassExtensionValidationCheck` option was enabled, and the EdFi Migration script tried to drop an index that is being referenced by a foreign key on another schema.
- There is a schema-bound view with a direct dependency on a table that is being modified.

Make sure that you have dropped ALL foreign keys and views from other schemas that have a dependency on the edfi schema. This must be done by adding custom migration scripts for your ODS, or by dropping these items by hand. Any external dependencies present will result in a SQL exception.

You do not need to drop any constraints on the edfi schema itself: This is handled automatically for you.

For tips on locating foreign key dependencies quickly, see the query in Step 8b. above.
A data validation failure was encountered on destination object \textit{(table name here)} ...

Data was modified in a location that was not expected to change. A validation check from script directory \\
\texttt{Scripts/02 Setup/\{version\}/## Source Validation Check} has failed.

This state is triggered if certain records are modified in the middle of migration that the upgrade utility expected would remain unchanged. The upgrade will be halted for you as a precaution to prevent unintended data loss.

**Common Causes:**

- Updates were made (either by hand or other means) in the middle of the migration process after the data compatibility checks passed.
- Custom migration scripts performed an insert/update/delete on an object that the migration utility expects to remain unchanged by default.

If you are testing potential updates to the 2.x ODS data by hand (data only / no custom scripts):

- For testing of data updates only: Simply restore the target 2.x ODS and make the desired inserts/updates/deletes before launching the utility.
  - It is recommended to create your own custom migration scripts if testing more complex operations
- You should ensure that all 2.x data updates have been applied before the data compatibility check passes.

If you are writing your own custom scripts:

- Check the output of the table mentioned in the error message carefully and ensure that the detected v3.1 change is as intended.
- If the error references a table /column that you know is intended to change during upgrade, you may optionally remove the data validation check.
  - These items can be found in directory \texttt{Scripts/02 Setup/\{version\}/## Source Validation Check}
  - \textit{It is not recommended to modify the edfi schema, as this may make upgrade tasks difficult in the future. By convention, custom tables are normally found in an outside location, such as the extension schema.}

If your v2.x Ed-Fi ODS schema is unmodified, and you are not making edits to the current scripting or data, this error should not occur during normal operation.

- Try the upgrade process again, and if the same error occurs, you may have encountered a bug.