Using the Changed Record Queries

The Ed-Fi ODS / API platform contains data that gets updated frequently. The platform tracks inserts, updates, and deletes, and surfaces those changes to client systems through a feature called changed record queries, or “change queries.” Change queries allow client systems to narrow requests for data to only data that has changed since a specified point in time. This allows client systems to stay in sync with the ODS / API without having to pull a complete dataset.

Change queries is an optional feature, so you’ll need to check with your target platform host to see if it’s enabled.

About Change Queries

The change queries feature was designed to have a simple architecture, and to integrate with the core API client authorization, and to be simple to use. This ensures that the system is performant, secure, and easy to maintain. This approach results in the following properties:

- The solution provides a reference to records that have changed since a previous point. It does not directly provide the changed data itself. This allows client systems to optimize the application of changes in the most efficient means for the particulars of their system.
- The reference provided by the solution will only be the highest (i.e., the most recent) change version. This is different from, say, a change data capture system that provides a log of every change.
- The solution does not target immediate consistency, but rather provides a reliable means of eventual consistency.
- The solution will work for most use-cases, but absolute consistency cannot be guaranteed. A periodic re-synchronization may be required for some uses.
- The technical article Changed Record Queries has implementation details which may be of interest to some client system developers.

Overview of Change Query Endpoints

The simple design means that the core operations are basic. This section provides an overview.

Available Change Versions Resource

The ODS / API uses a change version (as opposed to, say, a date) in the form of a sequential long integer. A global Available Change Versions API resource provides information on the current change version. This resource allows clients to request a reference to changed records they have not already requested or processed.

GET /changeQueries/v1/availableChangeVersions

Minimum and Maximum Change Version Parameters

The Minimum Change Version and Maximum Change Version parameters allow clients to request the latest representation of all resources that were modified within the given change version window. These parameters are available on every data resource, both as part of the Ed-Fi Data Standard and in extension models. The parameters are also compatible with the existing parameters to support paging using the offset and limit parameters. Using paging parameters plus change version parameters, all records can be retrieved over multiple calls.

GET /data/v3/ed-fi/students?
minChangeVersion=234378&maxChangeVersion=234974&offset=100&limit=100

Deletes Route

The Deletes route allows clients to get deleted records for any resource. This route also supports the existing paging parameters of offset and limit.

GET /data/v3/ed-fi/students/deletes?
minChangeVersion=234378&maxChangeVersion=234974&offset=100&limit=100

Synchronization Using the Change Query Endpoints

The primary purpose of the change queries feature is to support periodic synchronization of data. This section covers the basics.

Simple Daily Synchronization Example
The following example shows the logical flow for a daily synchronization process that only looks at Student records.

- **Initial sync to get all records:**
  - GET /changeQueries/v1/availableChangeVersions. For this example, assume we get 100 as a response.
  - GET /data/v3/ed-fi/students?maxChangeVersion=100. Returns Student records up to sequential change 100. Note that the minChangeVersion is not required.
  - The results will not include deleted Student records, so for an initial synch you won’t need special handling.
  - It is strongly recommended to run an incremental synchronization immediately after the initial synchronization due to the time required to transfer all the data. Additional data may be coming in while doing the initial synchronization that can cause potential referential integrity issues due to the eventually consistent nature of the feature, an incremental sync will retrieve that data.
  - As with any large data retrieval process, it is recommended to perform the initial synchronization process during a period of low activity on the API, to reduce contention for resources.

- **Subsequent sync (e.g., 1 day later) to get updates:**
  - GET /changeQueries/v1/availableChangeVersions. Assume we get 250 as a response.
  - GET /data/v3/ed-fi/students?minChangeVersion=100&maxChangeVersion=250. Returns any created or updated Student records through change 250. Note that the Minimum Change Version is the previous maximum.
  - GET /data/v3/ed-fi/students/deletes?minChangeVersion=100&maxChangeVersion=250. This API call is, of course, optional if your system does not need to be aware of deleted records.

**Usage Notes**

A few things to keep in mind when developing your synchronization:

- **Always get and specify the current Maximum Change Version, even on initial sync.** Since data is volatile, data can change during your initial processing, so this ensures you won't miss a change that occurs after your initial sync.
- **An open Minimum Change Version is okay since 0 represents a static point.** However, for incremental updates, you’ll want to specify a Minimum.
- **Keep dependency order in mind when pulling updates and deletes.** For example, if you have a system that enforces referential integrity, you’ll need to pull data in reverse dependency order to ensure that valid relationships can be established.
- **Span your change version window across two (or more) change windows if the data is heavily updated during your synchronization time.** Heavy updates during the time the client is synchronizing can cause errors in loading some resources, due to the eventually consistent nature of the design. This can be most easily resolved by spanning over multiple change windows, which ensures all records are retrieved.

As an example, assuming a daily synchronization schedule:

<table>
<thead>
<tr>
<th>Synchronization</th>
<th>When Performed</th>
<th>AvailableChangeVersions Result</th>
<th>MinChangeVersion Used</th>
<th>MaxChangeVersion Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Start</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Incremental #1</td>
<td>Immediately after completion of start</td>
<td>200</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>Incremental #2</td>
<td>1 day after start</td>
<td>300</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>Incremental #3</td>
<td>2 days after start</td>
<td>400</td>
<td>200</td>
<td>400</td>
</tr>
</tbody>
</table>