LabKey User Conference 2018

Methods for Loading Data Into LabKey

Tony Galuhn and Marty Pradere

11/5/18

LabKey
PARTNERS IN SCIENCE

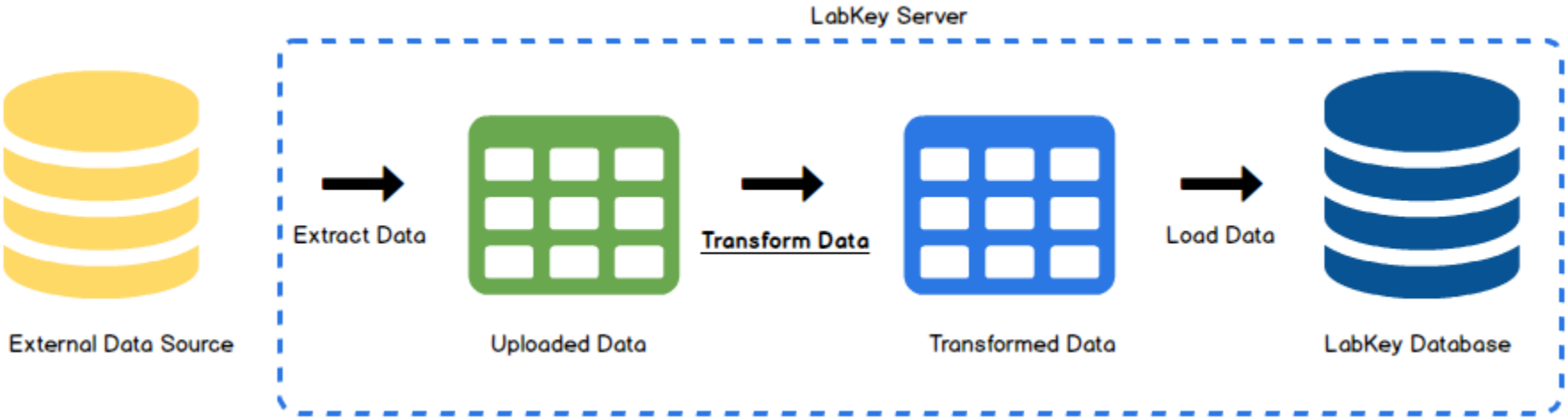
Overview

- Options for Community Edition
  - Transform Mechanisms
    - Assay Transforms
    - Trigger Scripts
- Options for Premium Edition
  - ETL
    - Transform Mechanisms
    - Usage Scenarios

Options For Loading Data

- Community Edition
  - “Insert Row” - Data Entry Forms
  - Bulk Import
    - “Import Data” - TSVs
    - Assay Upload
    - Study/Folder/List Import

Transform Step



- Transform, validate and cascade operations
- Programmatically apply custom expert rules and business logic

Transform Scenarios

- Align source column names with target column names

source.collection_date → **dataset.date**

- You want the input data to go through calculations or operations that change the values before inserting into the target table.
  - g ➡ kg
  - “10” ➡ “10 mL”

Transform Scenarios (cont'd)

- Filter the incoming data or change the numbers of columns.

Sequence
acggt … atgtg

→

SNP 1	SNP 2	SNP 3
g	c	g

- Transform input data based on data already stored in LabKey
  - Import:

Weight (kg)
80

+ Query DB:

Height (cm)
180

→

BMI
18.5

Validation Scenarios

- Validation beyond what is provided by built in LabKey validation providers (range, required, regex)

Pregnant	Gender
true	male
true	female

Validation Error

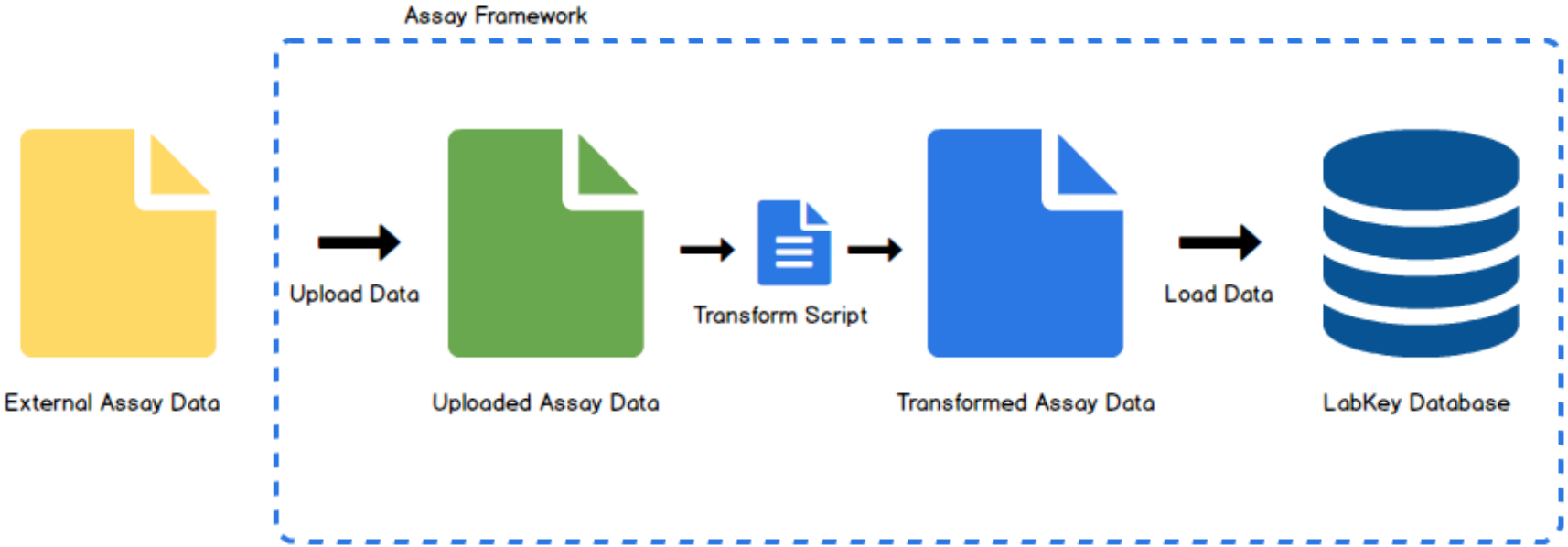
- Validation based on data existing in other tables
  - Primate EHR looks at several values (weight, body condition, blood lab results) when importing blood draw volumes to ensure blood draws have not exceeded blood draw limits

Cascading Operations Scenarios

- Perform other operations in addition to importing data into the target table
- Insert records into multiple tables

- Based on validation or some other condition, send email notifications.

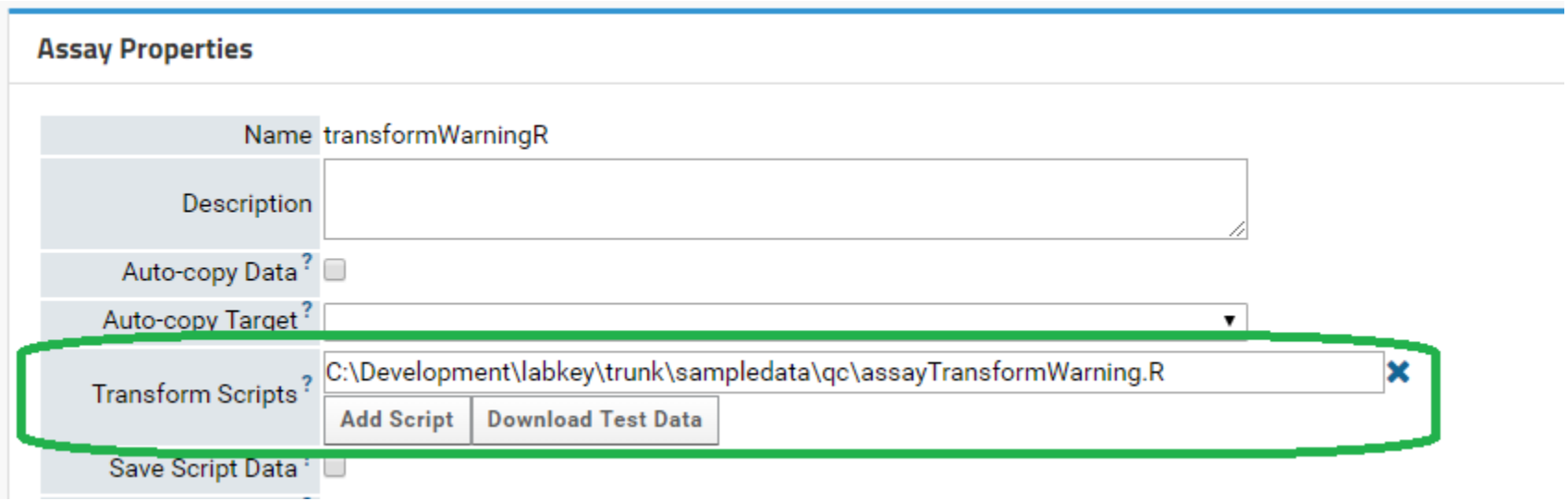
Assay Transform Scripts



Assay Transform Scripts

- Inspect, validate and transform data uploaded to an assay.
  - Perl, Python, R, Java
  - Runnable from command line
  - Part of assay design

Assay Transform Scripts



Assay Transform Script – Execution

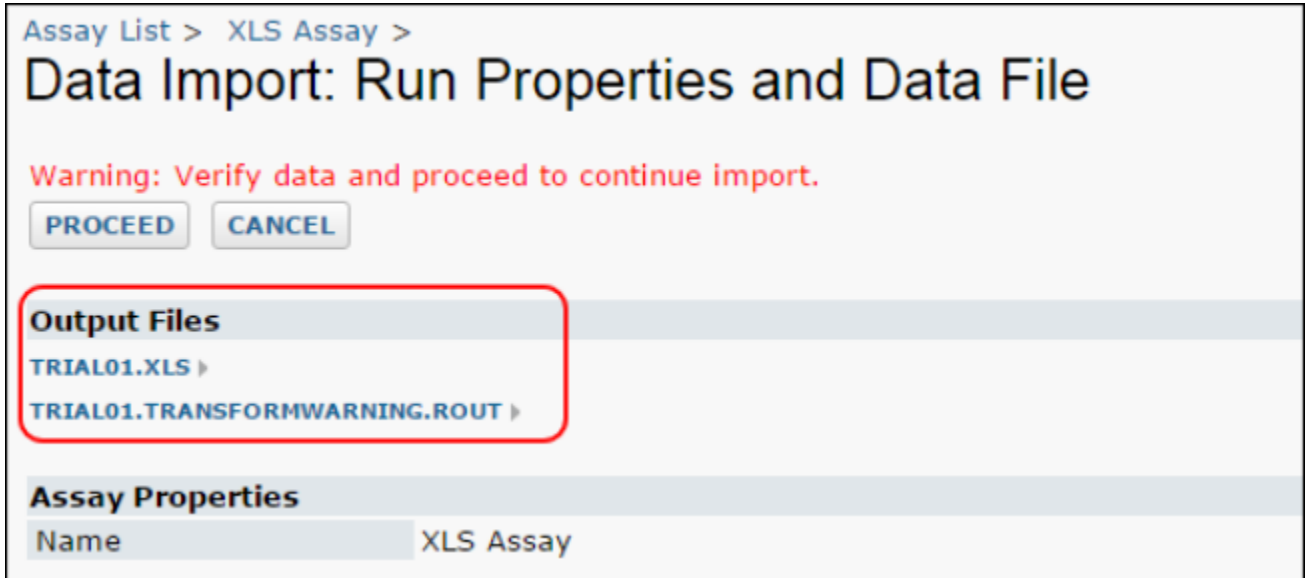
- Entire data file uploaded and available to the script.
- Script can validate and transform the data, outputting the data to a TSV file to be uploaded.
- Run and batch properties can be transformed as well

Assay Transform Script – Validation

- Errors and warnings can be written to error files which will display the warnings and errors on the assay upload page.

Assay Transform Script – Warnings

- Warnings provide a confirmation step for the assay uploader and provides the transformed files for review



Assay Transform Script – Warnings

- HTML messages can also be displayed as warnings

Assay List > XLS Assay >

Data Import: Run Properties and Data File

Abnormal Values

Column	Row	Value	Comment
2	25	10	Value out of range
2	103	11	Value out of range

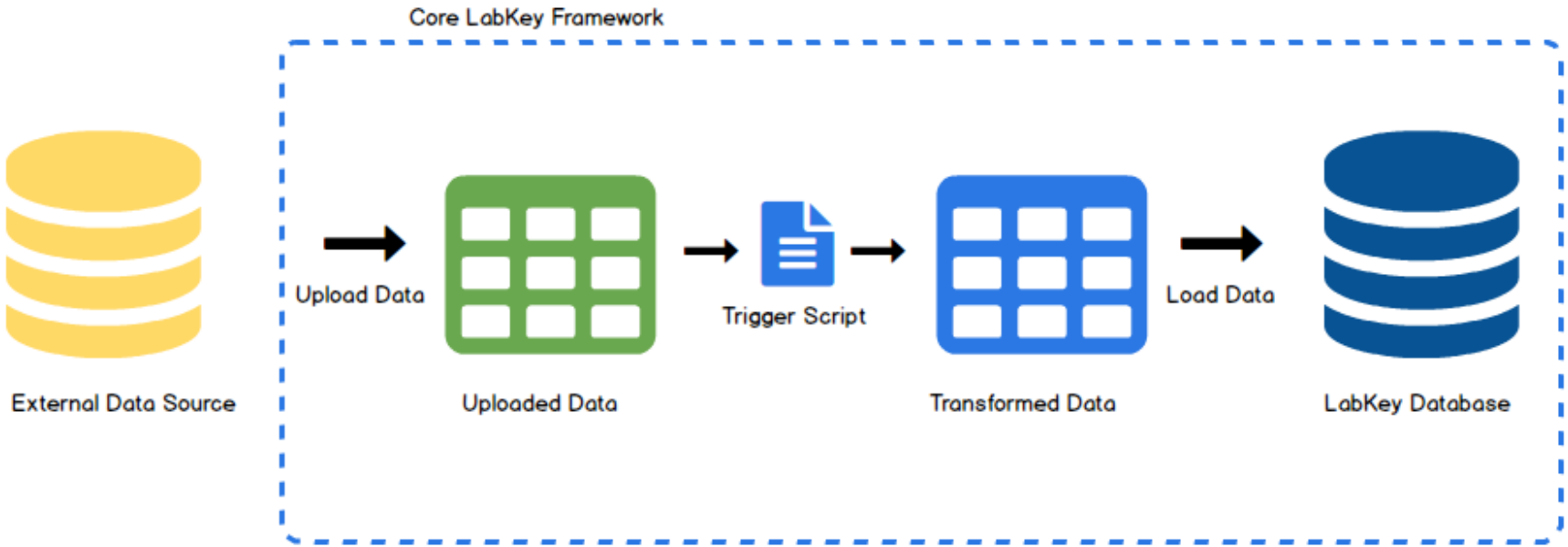
PROCEED CANCEL

Output Files

TRIAL01.XLS

TRIAL01.TRANSFORMWARNING.ROUT

Trigger Scripts



Trigger Scripts – Availability

	Insert New Row	Import Data (Data Grid)	Import via Client API	Import via Folder/ Study/List/ XAR	ETL
Lists	Yes	Yes	Yes	Yes	Yes
Datasets	Yes	No	Yes	No	Yes
Assays	No	No	No	No	No
Module/ External Schemas	Yes	Yes	Yes	N/A	Yes

Trigger Scripts

- What are trigger scripts?
  - Scripts that are triggered by a LabKey data event such as insert/update/delete.
  - Useful for both data integration as well as data entry.
  - Another way to apply your own business logic or expert rules.

Trigger Scripts – Functions

- Trigger Functions by Type
  - Batch
    - Executed once for a given batch. For example if inserting 100 rows in one batch, these are executed once.
    - init and complete functions
  - Row
    - Executed for each row.
    - beforeInsert, beforeUpdate, beforeDelete, afterInsert, afterUpdate, afterDelete functions

Trigger Scripts – Example

- {myModule}/resources/queries/{schema}/{table}.js

```
function init(event, errors) {
    ...
}
function beforeInsert(row, errors) {
    ...
}
function afterInsert(row, errors) {
    ...
}
function complete(event, errors) {
    ...
}
```

Trigger Scripts – JavaScript

- JavaScript triggers
  - Server side JavaScript (Rhino)
  - Useful for file-based modules
  - No compilation of Java

Trigger Scripts – Java

- Java Triggers
  - More performant
  - Easier to debug
  - Requires Java compilation

Trigger Script – Validate Data

- Errors can be thrown three ways
  - Field level error. This will return the error message and identify the field that caused the error.
  - Row level error. This will return the error but not identify the field.
  - Return false. This will cancel the insert/update/delete with a generic error message

Trigger Scripts – LabKey APIs

- For cascading other operations or more complex validation and transforms, APIs can be used
- Java Triggers
  - Use Java APIs
- JavaScript Triggers
  - Non-web browser, server side APIs available
  - Synchronous

EHR Trigger Scripts (Premium)

- Primate EHR has an additional layer of trigger scripts built on top of the core LabKey trigger scripts.
- Contains business logic common to the Primate Research Centers
- Adds additional row functions beforeUpsert, afterUpsert, onBecomePublic and afterBecomePublic

Options For Loading Data

- Premium Features
  - Driven by customer need
  - File Watchers
    - Premium Study Reload
  - Database Data Sources - ETL!

ETLs – DataIntegration Module

- Extract – Transform – Load
- In 18.3, becoming Premium functionality
  - Reflecting LabKey's investment in improving this feature area
  - Authoring
  - Administration
  - Performance
  - Documentation

ETL Overview

- XML Definition
  - One or more steps
  - Run on
    - Polling interval
    - Set Schedule
    - Manual trigger
  - May be a full data copy, or an incremental transfer

Sample ETL

```
<etl xmlns="http://labkey.org/etl/xml">
  <name>My Sample ETL</name>
  <description>append rows from source to target</description>
  <transforms>
    <transform id="step1" type="org.labkey.di.pipeline.TransformTask">
      <description>Copy to target</description>
      <source schemaName="mySourceSchema" queryName="mySourceQuery"/>
      <destination schemaName="myTargetSchema" queryName="myTargetQuery"/>
    </transform>
  </transforms>
</etl>
```

ETL Step Types, Examples

- Source Query To Target
  - Source and/or Target may be in local LabKey data source, or exposed via external schemas
- Remote LabKey Source via web API
- Execute Stored Procedures
- Run Java Tasks

ETL Authoring

- Previously only available as module resources
  - Required creating custom module
  - Permissions to deploy a module
  - Iteration difficult, as author and deployer may not be the same person
  - Based on feedback for a more robust experience…
- 18.2 ETL Definition Editor!

ETL Definition Editor

Edit Custom ETL Definition

ETL Test

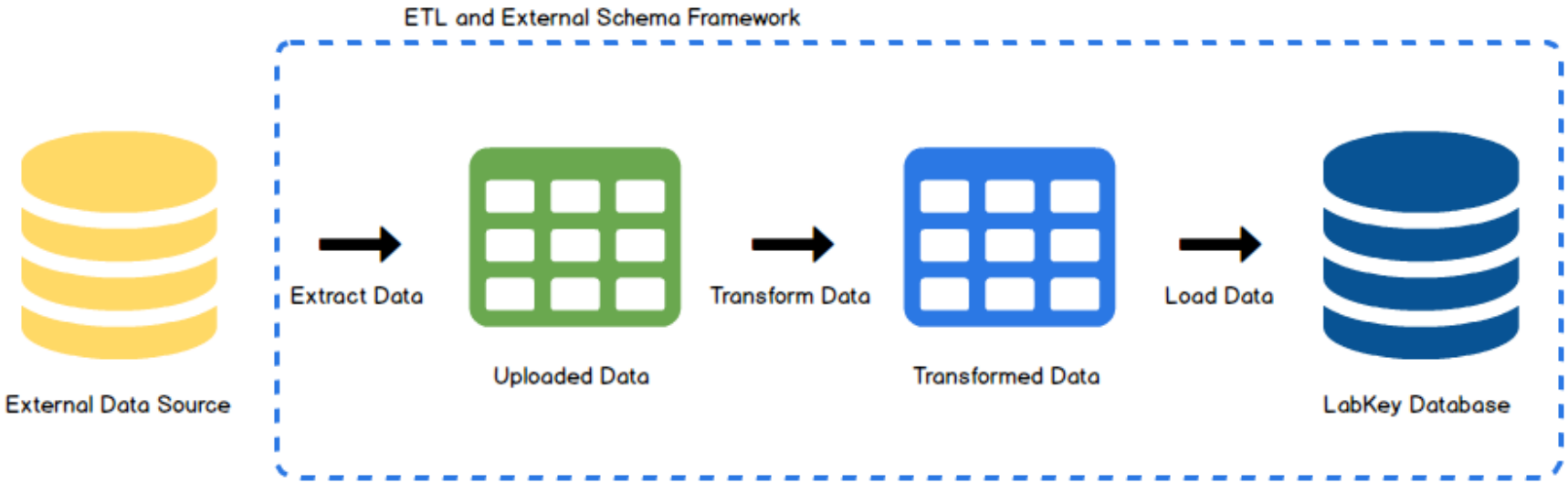
New Definition

Copy From Existing

```
1  <etl xmlns="http://labkey.org/etl/xml">
2    <name>My Sample ETL</name>
3    <description>append rows from source to target</description>
4    <transforms>
5      <transform id="step1" type="org.labkey.di.pipeline.TransformTask">
6        <description>Copy to target</description>
7        <source schemaName="mySourceSchema" queryName="mySourceQuery"/>
8        <destination schemaName="myTargetSchema" queryName="myTargetQuery"/>
9      </transform>
10     <
11   </
12 </et
13
```

<transform
</transforms>

ETL Transformations



ETL Transformations – Source Queries

- Source query transformations
  - Rename columns
  - Simple transformations like unit conversions and string manipulations

```
SELECT
collection_date AS date,
weight_in_g / 1000 AS weight_in_kg,
volume || ' mL' AS volume
```

ETL Transformations – Column Transforms

- Column Mapping
  - Map source column names to target column names
  - Define using a columnTransform in the ETL XML definition

```
<columnTransforms>
    <column source="collection_date" target="date"/>
</columnTransforms>
```

ETL Transformations – Column Transforms

- Java Transform Classes
  - Java class to provide custom business logic to ETL import
  - Allows complex transformations and validation
  - Aligned with a column in the ETL XML using a columnTransform

```
<columnTransforms>
        <column source="createdBy" target="createdBy"
        transformClass="org.labkey.snprc_ehr.columnTransforms.UserTransform"/>
</columnTransforms>
```

ETL Transformations – Column Transforms

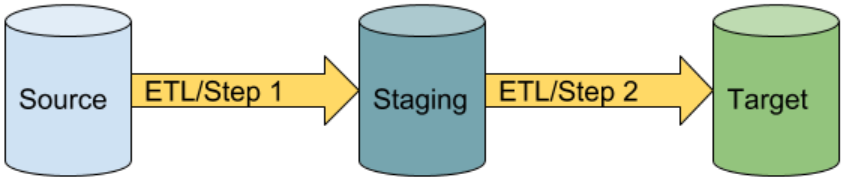
- Java transform class example
  - Problem: Need to import createdBy and modifiedBy columns but source data only has email address and target table requires a LabKey user id.
  - Solution: Java transform class for these columns to lookup user ids in LabKey for the imported email addresses. If user does not exist create user in LabKey. doTransform function takes user email address as parameter and returns user ID.

ETL Strategies/Practices

- Group Steps, or Multiple ETLs?
  - Example: Study Datasets
    - Do changes to the source affect multiple datasets at once?
      - Multiple steps in one ETL definition
    - Do upstream changes impact a single dataset?
      - Multiple ETLs
  - Are the target queries relational?
    - Multiple steps in one ETL definition
  - Advanced scenarios
    - ETL definitions can wrap other ETL definitions
    - ETL definitions can have a step to queue another ETL definition job

ETL Strategies/Practices, continued

- Staging Tables
  - In many cases, a direct transfer from source to target is appropriate
  - Other situations may require more advanced transformation or validation



Incremental Strategies

- Get all rows - SelectAllFilterStrategy
- Time based - ModifiedSinceFilterStrategy
  - Most common
  - Look for new/modified rows since date/timestamp of previous job run

Incremental Strategies, continued

- Batch based – RunFilterStrategy
  - Relational data
  - Multistage transfer pipeline
  - An earlier upstream process is writing a batch of parent-child data to the source, and can label with a batch/runId
  - Essential that all child records are transferred at same time as parent

Questions?

Appendix

ETL Transformations – Column Transforms

- Java Transform Classes
  - Extend ColumnTransform class
  - Override doTransform function
    - Called for every input row for the given column.
    - Takes the source value as a parameter and returns the value to be saved.
  - ColumnTransformException for errors to be displayed in the pipeline log

Assay Transform Scripts – Assay Upload

- Extracting Data and Setup
  - Uploads entire assay data file to LabKey
  - Creates a TSV file with information about the upload (run properties). Used to pass information between the script and LabKey.

Assay Transform Scripts – Execution

- Entire file is uploaded to the server
- Transform script execution
  - Run properties file can be accessed from the scripts using a token
    - ${runInfo}
  - Run properties file then contains the paths to the input and output data files as well as run and batch configuration parameters

Assay Transform Script – Transform Data

- Access the input file uploaded onto LabKey using the path supplied in the run properties
- Transform data
  - Renaming and reshaping data
  - Performing calculations based on uploaded or previously saved data
  - Use LabKey APIs
- Write data to output file at path provided in run properties

Assay Transform Script – Loading Data

- Once the transform script is finished executing, the assay framework will look for errors written by the script to the errors files.
- If no errors, the assay framework will look for data in the transformed data file and if that exists, load into the database.

Trigger Script – Transform Data

- Transform data by updating the row parameter passed into the function

```
function beforeInsert(row, errors) {
    if(row.comment) {
        row.comment = "Updated comment";
    } else {
        row.comment = "New comment";
    }
}
```

Trigger Script – Validation Examples

```
function beforeInsert(row, errors) {
    if(row.isFather == true && row.gender == "female") {
        errors[null] = "Incorrect gender for father."        // Row level error
    }

    if(row.age > 99) {
        errors.age = "Age out of range.";                    // Field level error
    }

    if(row.id == null) {
        return false;                                        // Generic error
    }
}
```

Trigger Scripts – Caching

- Trigger life cycle
  - A new trigger instance is created with each batch
  - JS global variables or Java member variables will be valid for the scope of the batch
- Caching
  - Doing database queries or other expensive operations in the row functions can be a significant performance cost
  - If possible, perform the database queries in the init function and cache the results in a global or member variable

Trigger Scripts – Caching Example

- Example: When ETLing data into a dataset, need to verify the participant Id is valid.
- Don't: Query the database in beforeInsert to check if the participant Id exists.
- Do: Query the database for all Ids in the init function and cache the values in a global or member variable. Verify data against the cache in beforeInsert