OHDSI Collaborator Meeting: Unit & Regression Testing of your Common Data Model

20-FEB-2018

Erica Voss

Clair Blacketer / Ajit Londhe / Jamie Weaver
Today’s Discussion

• High-level Testing Terminology

• Life Cycle of Testing a CDM
  – White Rabbit / Rabbit In a Hat
  – Testing Framework
  – How to Execute Testing Process

• Janssen Specific Examples
Testing Terms

• **Unit Testing**
  – individual aspects of your ETL requirements are tested

• **Regression Testing**
  – Ensuring that previously developed and tested aspects of a ETL continue to work
  – Building up a series of unit tests allow you to regression test
Unit Testing Example

• ETL States:
  Person born in the future should be excluded from the CDM

• Unit Test:
  Person 1234 with date of birth year 2099

• CDM Builder Behavior:
  Expect no Person 1234 in the CDM
Regression Testing Examples

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person with two genders is excluded.</td>
<td></td>
</tr>
<tr>
<td>Person with two birth years &gt;2 yrs apart is excluded.</td>
<td></td>
</tr>
<tr>
<td>Person with two birth years &lt;2 yrs apart is kept with last birth year selected.</td>
<td></td>
</tr>
<tr>
<td>Person born before 1900 is excluded.</td>
<td></td>
</tr>
<tr>
<td>Person born in 2099 is excluded.</td>
<td></td>
</tr>
<tr>
<td>Person born in 2014 but enrolled in 2012 is excluded.</td>
<td></td>
</tr>
<tr>
<td>Person born in 2013 but enrolled in 2012 is kept, latest birth year taken.</td>
<td></td>
</tr>
<tr>
<td>Person with two enrollment_detail records has one person record.</td>
<td></td>
</tr>
<tr>
<td>Person with sex=3 is excluded.</td>
<td></td>
</tr>
<tr>
<td>Person has record with sex=3 but last record has sex=1, person is kept.</td>
<td></td>
</tr>
<tr>
<td>Person born the same year as enrolled, use first enrollment month to impute month of birth and day of birth.</td>
<td></td>
</tr>
<tr>
<td>Person with Rx benefits is kept.</td>
<td></td>
</tr>
<tr>
<td>Person without Rx benefits is excluded.</td>
<td></td>
</tr>
<tr>
<td>Person with last enrollment_detail record that has egeoloc=11 gets associated to NJ.</td>
<td></td>
</tr>
</tbody>
</table>
Life Cycle of Testing a CDM

1. New DB
2. White Rabbit Scan
3. Rabbit In a Hat
   - TestingFramework.R
   - CDM Test Package
   - ETL Document
   - Program CDM Builder
   - CDM BUILDER

   - All Pass?
      - NO
      - YES

   - ETL Documentation

New DB → White Rabbit Scan → Rabbit In a Hat → TestingFramework.R → CDM Test Package → ETL Document → Program CDM Builder → CDM BUILDER

CDM Builder Change Required

When CDM Builder Change Required, the process goes back to New DB.
White Rabbit Scan

- Scans source data providing detailed information on the tables, fields, and values that appear in a field.

- Example form NHANES:

After connecting on the location tab the scan tab generates the scan report.
**Example form NHANES:**

<table>
<thead>
<tr>
<th>RIDAGEYR</th>
<th>Frequency</th>
<th>RIDAGEMN</th>
<th>Frequency</th>
<th>RIDRETH1</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>396</td>
<td>3.0</td>
<td>9276</td>
<td>3066</td>
<td></td>
</tr>
<tr>
<td>8.0</td>
<td>376.8</td>
<td>4.0</td>
<td>464</td>
<td>2129</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>293.4</td>
<td>1.0</td>
<td>41</td>
<td>1921</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>291.1</td>
<td>5.0</td>
<td>354</td>
<td>1547</td>
<td></td>
</tr>
<tr>
<td>7.0</td>
<td>238.2</td>
<td>2.0</td>
<td>352</td>
<td>1308</td>
<td></td>
</tr>
<tr>
<td>8.0</td>
<td>233.9</td>
<td>1.0</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0</td>
<td>222.15</td>
<td>0.0</td>
<td>33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RIDAGERY = age in years

RIDAGEMN = Age in Month (for <25 months of age)

RIDRETH1 = race-ethnicity

• Using the scan report you can automatically generate functions to help you build test cases.

Open the scan report created in WhiteRabbit and then click ‘Generate ETL Test Framework’
Rabbit in the Hat

• Source the TestingFramework.R in RStudio to make the test functions available

<table>
<thead>
<tr>
<th>Global Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
</tr>
<tr>
<td>frameworkContext</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_demo_i</td>
<td>Large function (663.9 Kb)</td>
</tr>
<tr>
<td>declareTest</td>
<td>function (id, description)</td>
</tr>
<tr>
<td>expect_attribute_definition</td>
<td>function (attribute_definition_id, attribute)</td>
</tr>
<tr>
<td>expect_cohort</td>
<td>function (cohort_definition_id, subject_id, cohort_id)</td>
</tr>
<tr>
<td>expect_cohort_attribute</td>
<td>function (cohort_definition_id, subject_id, cohort_id)</td>
</tr>
<tr>
<td>expect_cohort_definition</td>
<td>function (cohort_definition_id, cohort_definition_id)</td>
</tr>
<tr>
<td>expect_condition_era</td>
<td>function (condition_era_id, person_id, condition_era)</td>
</tr>
</tbody>
</table>

ADD_ functions allow you to add data to raw tables to generate your test cases

EXPECT_ functions allow you to define what you expect in the CDM
CDM Test Package

- R package that stitches all your tests together using functions from TestingFramework.R
CDM Test Package: Example Test Case 1

```sql
declareTest(id = patient$person_id, "Person born in 2099 is excluded. Id is PERSON_ID.")
add_enrollment_detail(enrolid=patient$enrolid, dobyr="2099")
expect_no_person(person_id = patient$person_id)
```

- Example, ETL describes that if a person is born in the future they should be excluded from the CDM

- **ADD_** data to raw data tables to mimic this behavior, this adds to INSERT.SQL

- **EXPECT_** to state what you expect to occur in CDM, this adds to TEST.SQL
Example Test Case 1

```sql
declareTest(id = patient$person_id, "Person born in 2099 is excluded. Id is PERSON_ID.")
add_enrollment_detail(enrollid=patient$enrollid, dobyr="2099")
extpect_no_person(person_id = patient$person_id)
```

- **Example** Test Case 1 describes that if a person is born in the future they should be excluded from the CDM.
- **ADD_** data to raw data tables to mimic this behavior, this adds to INSERT.SQL
- **EXPECT_** to state what you expect to occur in CDM, this adds to TEST.SQL

**Rabbit in a Hat Genius!**
Where are all the other values the table needs? The scan report just populates them with the most common value.
Example, ETL describes that a CPT 90686—"influenza virus vaccine" that the OMOP Vocabulary associates with a concept in the drug domain.

ADD_ data to raw data tables to mimic this behavior, this adds to INSERT.SQL

EXPECT_ to state what you expect to occur in CDM, this adds to TEST.SQL
Execute Testing

1. INSERT.SQL populates raw DB

2. Run CDM Builder

3. TEST.SQL tests new CDM

4. Review test results

5. Augment test cases or CDM Builder till all tests pass
Execute Testing:
1) R populates raw DB

---

Test Case 106: Person born in 2099 is excluded. Id is PERSON_ID.

```
INSERT INTO enrollment_detail (boe, cap, dobyr, drugcovg, dtend, dtstart, enrold, mas, medicare, memdays, mhsacovg, plantyp, sex, stdrace, version, year)
VALUES ('4', '1', '2099', '1', '2016-06-30', '2016-06-01', '106', '9', '0', '31', '0', '2', '2', '1', '10', '2016');
```

- Test Case 1, we wanted to test a birth year of 2099

- The White Rabbit and the TestingFramework have filled in the rest of the values with the most common value from the scan
  - Sometimes you may additionally need to set multiple values in the test case (i.e. set the dates so things align correctly)
  - You can override the most common value with defaults in the TestingFramework

```r
set_defaults_enrollment_detail(drugcovg = '1')
```
Execute Test Cases:
2) Run CDM Builder

- **NATIVE** (a database with all the raw tables has been set up and populated with your test cases in previous step)

- Run CDM Builder writing to CDM tables in **TESTING** (a database with a blank CDM schema for the builder to insert the resulting data)
Execute Testing:
3) R tests new CDM

- The EXPECT_ functions have written tests similar to the above

- If we find PERSON_ID 109 in the CDM who was born in 2099 the test will fail

```sql
-- 106: Person born in 2099 is excluded. Id is PERSON_ID.
INSERT INTO test_results
SELECT 106 AS id,
    'Person born in 2099 is excluded. Id is PERSON_ID.' AS description,
    'Expect person' AS test,
    CASE
        WHEN (SELECT COUNT(*) FROM person WHERE person_id = '106') != 0 THEN 'FAIL'
        ELSE 'PASS'
    END AS status;
```
Execute Testing:
4) Review test results

- All TEST.SQL to write out to a TEST_RESULTS table in your CDM, review the failures

- Failures may indicate either a bug or a poorly written test case
Execute Test Cases:
5) Augment Test Cases/Builder

- If changes are required to your Builder having some sort of issue tracking will help keep you organized and also help track what changes are within each CDM Builder release
Some Janssen R&D Specific Ideas
Janssen ETLs and Testing are Open Source
Janssen Bug Fix Sprint Process

1. Select what we can tackle in the month
2. Developers make change / update test cases
3. Test updated Builder
4. Update and test until all test cases pass
5. Run full CDM Build
6. Run ACHILLES, review HEEL
7. Connect to ATLAS, test “Dummy Cohorts”
8. Bless or Reject CDM
Vocabulary Compare

- Your ETL can be working perfectly by adopting a new Vocabulary can bring change

- We try to quantify this change but:
  - Characterizing the differences between two versions of the Vocabulary
  - Understand what change have the biggest impact on our data
Janssen Best Practices & Comments

• Write out test cases before programming
  – The list will keep you focus
  – Walk through the ETL document to generate the list

• Generating “dummy test data” has allowed us to test edge cases that may never occur in the data

• Try to test one item at a time

• Have your testing environment set and ready to go

• Some database are not as performant as others with the INSERT.SQL, a non parallel database is preferred (e.g. MS SQL Server)

• This presentation primarily demos how Janssen R&D tests, testing processes could and might look slightly different for your organization
Without Martijn Schuemie’s contribution of OHDSI tools White Rabbit and Rabbit in a Hat, testing your CDM would be painful. Thank you!
Resources

- White Rabbit GitHub (inclusive of Rabbit In a Hat)
- White Rabbit Wiki
- Rabbit in a Hat Wiki
- Rabbit in a Hat Testing Framework Wiki
- Janssen Truven Test Cases