Writing Loaders for the Data Warehouse – a User Guide

Version 1.1. Sept 2024

Paul *Watson, Newcastle University, paul.watson@ncl.ac.uk*

1 Introduction

The e-Science Central Data Warehouse was designed to enable healthcare study data to be stored stared and analysed. This document describes how loaders can be used to populate the warehouse. It requires that the data to be stored has been converted into a Python Dictionary of type:

```
DataToLoad = Dict[str, Union[Value, List['DataToLoad'], List[str]]]
where
Value = Union[int, float, str, DateTime]
```

The dictionary key is the field name.

The value can then be of type:

- "Value", e.g., an integer, or
- a list of dictionaries, e.g., data on each family member, or
- a list of strings, to represent a set of options, e.g., ['sticks', 'wheelchair', 'roller']

The loading of the data from an instance of DataToLoad is performed by a call to load_data.load_data:

```
def load_data(data_warehouse_handle,
              data: DataToLoad,
              data_name: str,
              mapper: Dict[str, Loader],
              study: Study,
              bounds: Bounds = None,
              trial: Optional[Trial] = None,
              participant: Optional[Participant] = None,
              source: Optional[Source] = None,
              cursor=None)
           -> Tuple[bool, List[MeasurementGroupInstance],
List[str]]:
"""
Load one item of data into the datawarehouse.
:param data_warehouse_handle: handle to access the data warehouse
:param data: the data item containing the fields to be loaded
:param data_name: the name of the type of data: this is used to find
the correct loader
:param mapper: a dictionary mapping from data_name to the loader
function
:param study: study id in the data warehouse
:param bounds: tuple holding bounds used to check data
:param trial: optional trial id
:param participant: optional participant id (must already be in the
warehouse)
:param source: optional source id
```

```
:param cursor: an optional cursor for accessing the datawarehouse
:return: Success?, list of measurement group instance ids for the
measurement groups inserted, error messages
"""
```

The optional bounds parameter holds:

```
Bounds = Tuple[IntBounds, RealBounds, DateTimeBounds, CategoryIds,
CategoryValues]
```

This parameter is used to ensure that all data inserted into the data warehouse conforms to a set of criteria:

- bounded values (bounded ints, reals and datetimes) must lie between the minimum and maximum limits set for them (and stored in the appropriate bounded limits tables). E.g., if a bounded int can lie between 1 and 3, then a value of 5 will be rejected.
- Categories must have ids and values that are stored in the category table. e.g., if declared categories are (1, 'bungalow'), (2, 'flat'), (3, 'multi-storey house') then an id of 4, or a category value of 'apartment' would be rejected.

It can be passed into the `load_data` function, but if it is missing then its value is automatically generated from the data warehouse contents.

The `data_name` is a unique string that identifies the type of the data. This is used as the key into the mapper parameter – a dictionary that maps from the `data_name` to a loader function that picks out the values of the fields in the data so that they can be loaded into the warehouse.

2 Loader Function

A set of library functions must be provided to load each of the different types of data into the warehouse.

The signature of a loader is:

```
data: DataToLoad, bounds: Bounds -> LoaderResult
```

The type of `LoaderResult` is:

```
Tuple[
      List[Tuple[MeasurementGroup, List[LoadHelperResult]]],
      Optional[DateTime],
      Optional[Trial],
      Optional[Participant],
      Optional[Source]]
```

The `MeasurementGroup` parameter is the id of the Measurement Group to be loaded.

If the `DateTIme`, `Trial`, `Participant` and Source value are present then they will be associated with each measurement inserted into the Data Warehouse.

`LoadHelperResult` is explained in the next section.

3 Loading a Field

Each loader function is responsible for loading in each field that makes up an instance of a measurement group. This done by a set of field loaders that are in `import_with_checks`.

Each field loader is of type:

```
MeasurementType, DataToLoad, field: str
      -> LoadHelperResult
```

Or (for fields where the Bounds need to be checked):

```
MeasurementType, DataToLoad, field: str, Bounds
      -> LoadHelperResult
```

where

`LoadHelperResult is of type: Tuple[bool, List[ValueTriple], str]`
where

```
ValueTriple = Tuple[MeasurementType, ValType, Value]
```

The three fields in the Tuple are:

- A Boolean that is true if loading of the field was successful
- A List of ValueTriples. Each triple represents a value to be loaded into the data warehouse.
- A string that holds an error message if the first field of the tuple is false

A wide range of Field Loaders are provided, covering all the types of data that the data warehouse can store:

```
load_int
load_optional_int
load_real
load_optional_real
load_string
load_optional_string
load_datetime
load_optional_datetime
load_boolean
load_optional_boolean
load_nominal_from_id
load_optional_nominal_from_id
load_ordinal_from_id
load_optional_ordinal_from_id
load_categorical_from_id_in_string
load_nominal_from_id_in_string
load_optional_nominal_from_id_in_string
load_ordinal_from_id_in_string
load_optional_ordinal_from_id_in_string
load_categorical_from_value
load_nominal_from_value
```

```
load_optional_nominal_from_value
load_ordinal_from_value
load_optional_ordinal_from_value
load_bounded_int
load_optional_bounded_int
load_bounded_real
load_optional_bounded_real
load_bounded_datetime
load_optional_bounded_datetime
load_external
load_optional_external
```

Note that each loader has two versions – one if the field is optional, and the other if it is mandatory.

The loader returns a tuple of type LoaderResult which is defined as:

```
LoaderResult =
    Tuple[
         List[Tuple[MeasurementGroup, List[LoadHelperResult]]],
         Optional[DateTime],
         Optional[Trial],
         Optional[Participant],
         Optional[Source]]
```

The fields in `LoaderResult` are:

- a list of tuples holding the measurement group id, and the results of reading in each measurement type from the dictionary. We will explain in the loading lists section below why a list of tuples can be returned
- An optional value for the DateTime field stored with each measurement in the measurement group instance. If it is not provided then the `time` value stored with each measurement in the data warehouse will be set to the time that the measurement group instance was inserted into the data warehouse.
- An optional value for the `Trial` id to be stored with each measurement in the measurement group instance. If this is not provided then the value of the `Trial` will be given a null value for each measurement stored in the data warehouse.
- An optional value for the Participant id to be stored with each measurement in the measurement group instance. If this is not provided, then the value of `Participant` will be set to a null value for each measurement stored in the data warehouse.
- An optional value for the Source id to be stored with each measurement in the measurement group instance. If this is not provided, then the value of `Source` it will be set to a null value for each measurement stored in the data warehouse.

3.1 Categorial Field Loaders

There are three variants of each categorical loader:

```
load_*_from_id : load the integer id of the category
load_*_from_id_in_string : load the integer id of the category when
it is encoded in a string
```

```
load_*_from_value : load the value of the category; this is a string
```

(where `*` is “`nominal`” or “`ordinal`”)

For example, here is a loader for a measurement group (with an id of 40) holding information on a drug taken by a participant. This group contains two measurement types: a string that holds the name of the drug (measurement type 400), and an integer holding the dose (measurement type 401):

```
def drugs_loader(data: DataToLoad, bounds: Bounds) -> LoaderResult:
    drug_mg_id: MeasurementGroup = 40
    drug_mgi = [(drug_mg_id,
                 [iwc.load_string(400, data, 'drug'),
                  iwc.load_int(401, data, 'dose')]
                 )]
    return drug_mgi, None, None, None, None
```

3.2 Loading Sets

A set is where the field being loaded holds a set of elements. As the data warehouse has no concept of sets, each possible element is represented by a different `measurementtype`, with `valtype` set to `Boolean`. Therefore, the `load_a_set` loaders have the following types:

```
load_set(
           measurement_types: List[MeasurementType],
           data: DataToLoad,
           field: str,
           value_list: List[str]) -> LoadHelperResult

load_optional_set(measurement_types: List[MeasurementType], data:
DataToLoad, field: str, value_list: List[str]) -> LoadHelperResult
```

The `measurement_types` field gives a list of the ids of the measurement types, while the `value_list` gives a list of the elements corresponding to each `measurement_type`. The list retrieved from the field in the dictionary must be a subset of the `value_list`.

e.g., if the `measurement_types` = [100,101,102] and `value_list` is: [“red”, “green”, “blue”] and the field in the dictionary holds [“red”, “blue”] then:

the value in the instance of `measurement_type` 100 will be set to `True`

the value in the instance of `measurement_type` 101 will be set to `False`

the value in the instance of `measurement_type` 102 will be set to `True`

3.3 Loading Lists

A field in the dictionary to be loaded into the data warehouse may contain a list of elements. The following functions load lists:

```
load_list(
      data: DataToLoad,
      field: str,
      loader: Callable[[DataToLoad], LoaderResult],
      mg_id: MeasurementGroup,
      bounds: Bounds) ->
            List[Tuple[MeasurementGroup, List[LoadHelperResult]]]

load_optional_list(
      data: DataToLoad,
      field: str,
      loader: Callable[[DataToLoad], LoaderResult],
      mg_id: MeasurementGroup,
      bounds: Bounds) ->
            List[Tuple[MeasurementGroup, List[LoadHelperResult]]]
```

In each list loader, as well as the usual data, field and bounds parameters, there are also:

- `loader`: a loader function to load a measurement group instance from each element of the list (e.g., `drugs_loader`)
- `mg_id`: the measurement group id of the current measurement group instance

For example, below is a walking_and_drugs_loader function that reads in an instance of a measurement group (with an id of 39) holding information on some walking features measured for a participant. The "`data`" dictionary also has a field ("`drugs`") that holds a list of information to be loaded on the drugs taken by the participant. Each of the elements of this list is read in by the `drugs_loader` function described earlier, and this results in a new instance of the measurement group 40 being loaded into the data warehouse for each element in the list.

The result of the `walking_and_drugs_loader` function is therefore a set of measurement group instances ready to be loaded into the data warehouse (assuming there are no errors). This set contains one instance of measurement group 39, and multiple instances of measurement group 40 - one instance for each drug the patient is taking.

```
def walking_and_drugs_loader(data: DataToLoad, bounds: Bounds)
-> LoaderResult:
    turn_group: MeasurementGroup = 39
    turn_group_instance: List[Tuple[MeasurementGroup,
List[LoadHelperResult]]] = \
        [(turn_group,
          [iwc.load_datetime(370, data, 'visit-date'),
           iwc.load_string(371, data, 'visit-code'),
           iwc.load_string(1839, data, 'wb_id'),
           iwc.load_int(1843, data, 'Turn_Id'),
           iwc.load_real(1844, data, 'Turn_Start_SO'),
           iwc.load_real(1845, data, 'Turn_End_SO'),
           iwc.load_real(1846, data, 'Turn_Duration_SO'),
           iwc.load_real(1847, data,
'Turn_PeakAngularVelocity_SO')])]
    drug_group_instances: List[Tuple[MeasurementGroup,
List[LoadHelperResult]]] = \
        iwc.load_list(data, 'drugs', drugs_loader, turn_group,
bounds)
    return turn_group_instance+drug_group_instances, None, None,
None, None
```