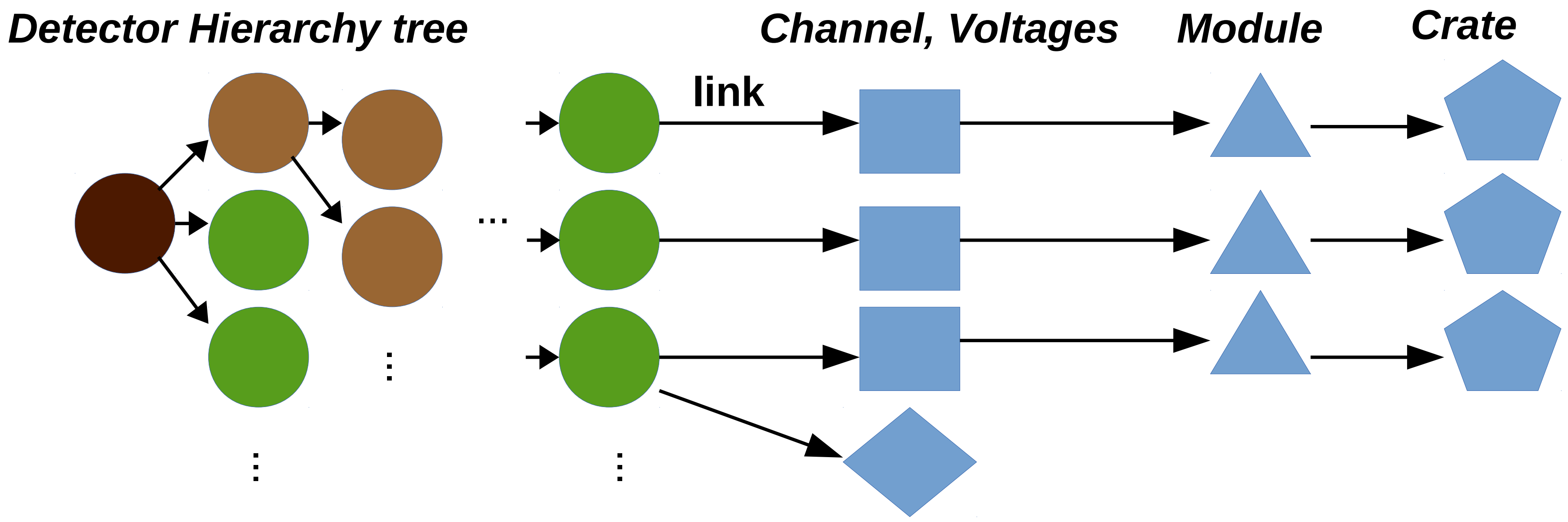
Merging data between data bases.

Orlando Soto

June 18, 2014

Introduction

- We have a table representing the detector hierarchy (Detector_Hierarchy, Nerses) and tables representing special characteristics of the instrumentation used in each detector (e.g. Crate, Module, Channel, Voltages).
- Multiple users can have a copy of these tables with his own information and his own scripts to fill data into the tables.
- With this scenario, an script to combine data bases data could be useful to save time making the scripts.
- To perform this task an script based on sqlalchemy was developed.



Script development

- SQLAlchemy has his own tool for merging data into data bases, but this tool is based on primary key rather than other information.
- If multiple users fill his own table copies, the primary keys could be different for the same element from different users.

  To work around this issue a set of python classes was written in a python module called 'DataBaseHandler.py'

DataBaseHandler module (1)

- This module contains the sqlalchemy declarative base classes to perform operations over the tables, and includes a special class called 'Node' which includes common methods for generic data base classes.
- All the table classes inherit from declarative_base sqlalchemy class and the Node class.

DataBaseHandler details (2)

```
class Crate(Base,Node):
    ''' Class used to handle rows of the Crate table in the data base.
        This class needs the sqlalchemy module.
        Parameters:
        Methods:
    '''
    __table__ = Base.metadata.tables['Crate']
    children = relationship("Module", cascade="all",  backref=backref("parent", uselist=False) )
    parent = None

class Module(Base,Node):
    ''' Class used to handle rows of the Module table in the data base.
        This class needs the sqlalchemy module.
    '''
    __table__ = Base.metadata.tables['Module']
    fk_name = 'crateid'
    key_name = 'slot'
    ForeignKeyConstraint([__table__.c.crateid],['Crate.crateid'])
    children = relationship("Channel", cascade="all",  backref=backref("parent", uselist=False) )

class Channel(Base,Node):
    ''' Class used to handle rows of the Channel table in the data base.
        This class needs the sqlalchemy module.
        Parameters:
        Methods:
    '''
    __table__ = Base.metadata.tables['Channel']
    fk_name = 'moduleid'
    key_name = 'channel'
    detector_hierarchy = relationship('Detector_Hierarchy', backref=backref("channel", uselist=False, lazy='joined'))
    ForeignKeyConstraint([__table__.c.moduleid],['Module.moduleid'])
    children = []
```

DataBaseHandler details (3)

```python
    class Detector_Hierarchy(Base,Node):
        ''' Class used to handle rows of the Detector_Hierarchy table in the data base.
            This class needs the sqlalchemy module.
        '''
        __table__ = Base.metadata.tables['Detector_Hierarchy']
        level = 0
        root_name = 'BCAL' # Root name must be define for tree structured tables only.
        tree_type = ('Detector','Discriminator','Side','Wedge','Number') # Types of tree must be define.
        fk_name = 'chanid'
        ForeignKeyConstraint([__table__.c.parent_id],['Detector_Hierarchy.id'])
        ForeignKeyConstraint([__table__.c.chanid],['Channel.chanid'])
#        ForeignKeyConstraint([__table__.c.volt_id],['Voltages.volt_id'])
        children = relationship("Detector_Hierarchy", cascade="all",  backref=backref("parent",remote_side=[__table__.c.id]) )

        global time_format
        def fill(self,session,data,fk): # node refers to the starting point node.
            starting_name = self.name
            start_index = data.index(starting_name)+1 if starting_name else 0
            node = self
            for i in range(start_index,len(data)-1):
                node_dict = dict( zip( node.col_names,( data[i],node.tree_type[i],node.id,datetime.now().strftime(time_format) ) ) )
                node = node.insert(session,node_dict) # The commit function at the end fill the id column.
            i = len(data)-1;
            node_dict = dict( zip( node.col_names,( data[i],node.tree_type[i],node.id,datetime.now().strftime(time_format) ) ) )
            node.insert(session,node_dict,fk)
            return node



        def get_leaves(self):
            leaves = []
            for item in self.children:
                if not item.children:
                    leaves.append(item)
                else:
                    le = item.get_leaves()
                    leaves.append(le)
            return flatten(leaves)
```

merge_db.py script

The script merges the data based on Detector_Hierarchy table and creates the links to other tables if is specified.
The names on Detector_Hierarchy must be given in the format:
root_name: ... :branch_i e.g. BCAL:lv:U

The help of the script shows the following:

```
 ./merge_db.py [options] source_file destiny_file

The script will merge data from 'source_file' data base into 'destiny_file' data base creating a new file called 'destiny_file.out',
preserving the information in source and destiny data bases.
Must be used with SQLite3 data base files.

options:
[ -s or --starting-point' starting_node ]  : String representing the starting node to proceed with the merging.
                                              E.g. 'BCAL:DISC' will paste BCAL:DISC and all his branches at 'BCAL' in the destiny data base.
[ -l or --link-key table_name]             : Name of the table to be linked through the leaves of the data base tree (detector hierarchy).
[ -f or --force ]                          : If the merge starting point exists in the destiny data base, it will be overwritten.
[ -i --source-file file_name ]             : Use 'file_name' as source data base.
[ -o or --dest-file   file_name ]          : Use 'file_name' as destiny data base.
[ -u or --update col_n | --all | -a ]      : Update using the column names specified. 'col_n' must be a list with comma separated, e.g. 'type,chanid'.
                                             Alternatively you can put --all or -a instead, and all column except primary and foreign keys will be updated.
[ --dest-schema ]                          : Use destiny data base schema to read data bases.
[ -h or --help ]                           : Show this help.

Example:
./merge_db.py -s BCAL:DISC -l 'channel' tt.db tt_dest.db

This command will create 'tt_dest.db.out' containing the data of tt.db (BCAL:DISC) merged into tt_dest.db.
```

Usage examples and considerations (1)

- The source and destiny data bases must contain the tables defined in DataBaseHandler.py
- Let's suppose two data bases source.db and destiny.db.
  - At least one of the Detector_Hierarchy schemes (set of columns) must contain the other.
  - The user must consider the table schemes in the data bases, it should use the simplest (fewer columns). Could be modified to do this automatically.

    Columns of destiny.db : id, parent_id, name, type, chanid and mtime

    Columns of source.db : id, parent_id, name, type, chanid, mtime and volt_id

    The script must be executed with the option '--dest-schema'.
  - The link key ('--link-key') must match the name of a relationship collection in the Detector_Hierarchy definition in DataBaseHandler.py

    ```
    detector_hierarchy = relationship('Detector_Hierarchy',
    backref=backref("channel" ...
    ```

    option ./merge_db **-l channel** ...

Usage examples and considerations (2)

- Let's suppose the source.db Detector_Hierarchy leaves have one more link to another table and destiny.db has neither the table definition nor the column in Detector_Hierarchy.

  Steps to follow:

  - Create the extra column (volt_id) in Detector_Hierarchy on destiny.db
  - Create the extra table (Voltages) on destiny.db (E.g. execute .schema on source.db, copy and paste in destiny.db)
  - Modify DataBaseHandler.db
    - **Create extra table definition.**
    - **Set the foreign key on Detector_Hierarchy definition.**
  - Run the script to update the destiny data base.

    ./merge_db.py -l voltage -u -a source.db destiny.db
  - Your result will be in 'destiny.db.out' data base

```
class Voltages(Base,Node):
    """ Class used to handle rows of the Detector_Hierarchy table in the data base.
        This class needs the sqlalchemy module.
        Parameters:
        Methods:
    """
    __table__ = Base.metadata.tables['Voltages']
    detector_hierarchy = relationship('Detector_Hierarchy', backref=backref("voltages", uselist=False, lazy='joined'))
    parent = None
    children = []
```

```
class Detector_Hierarchy(Base,Node):
.
.
    ForeignKeyConstraint([__table__.c.volt_id],['Voltages.volt_id'])
```

Future work and improvements

- The common columns on source.db and destiny.db could be obtained in the script.
- Multiple link at once.
- Include update on linked tables.
- Find bugs.

Questions and comments?