SkiRaff an ETL Testing Framework for pygrametl

June 16, 2016

Alexander Branborg abran13@student.aau.dk
Arash Michael Aami Kjær ams13@student.aau.dk
Mathias Claus Jensen mcje13@student.aau.dk
Mikael Vind Mikkelsen mvmi12@student.aau.dk

Department of Computer Science
Aalborg University
Denmark

AALBORG UNIVERSITY
DENMARK

Agenda

Predicates

- Why are they useful?
- Usage/Implementation
- Alternative Implementation

Predicates
Agenda

- Why are they useful?
- Usage/Implementation
- Alternative Forms of Implementation

Predicates

Why are they useful?

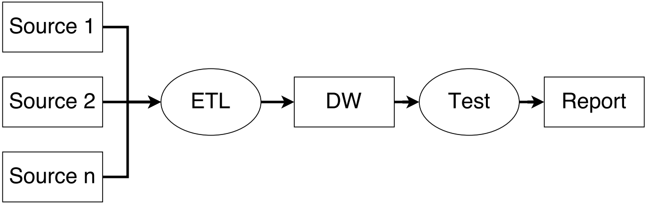
- Systems level testing
  - Data loss

Predicates

Why are they useful?

Alexander Branborg,
Arash Michael Aami Kjær,
Mathias Claus Jensen,
Mikael Vind Mikkelsen

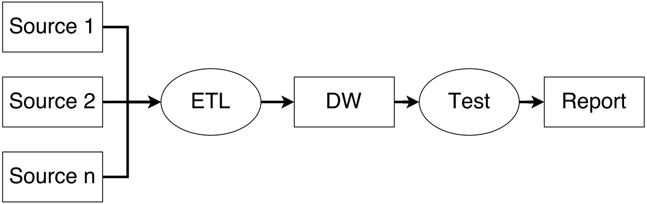
- Systems level testing
  - Data loss
- Source to target test
-



Predicates

Why are they useful?

- Systems level testing
  - Data loss
- Source to target test



-
- Regression testing
- Business Rules

Predicates
Why are they useful?

Predicates available in SKiRaff

- RowCountPredicate
- ColumnNotNullPredicate
- ReferentialIntegrityPredicate
- FunctionalDependencyPredicate
- SCDVersionPredicate
- CompareTablePredicate
- RuleRowPredicate
- RuleColumnPredicate

Predicates available in SKiRaff

- RowCountPredicate
- ColumnNotNullPredicate
- **ReferentialIntegrityPredicate**
- **FunctionalDependencyPredicate**
- SCDVersionPredicate
- CompareTablePredicate
- **RuleRowPredicate**
- RuleColumnPredicate

Functional Dependency - Why is it useful?

- A, B –> C

6

Functional Dependency - Why is it useful?

- A, B –> C
- DW holds certain hierarchical properties

Predicates

Usage - Functional Dependency

Predicates

Why are they useful?

Usage/Implementation

Alternative Implementation

Setup:

```
FunctionalDependencyPredicate(table_name=['CountryDim','AuthorDim'],alpha='city',beta='country')
```

SQL querie:

```
SELECT DISTINCT t1.country, t2.city
FROM countrydim NATURAL JOIN authordim AS t1, countrydim NATURAL JOIN authordim AS t2
WHERE t1.city = t2.city
AND t1.country <> t2.country
```

Predicates

Implementation - Functional Dependency

```
# Creates part of select statement to get keys
select_alpha = ["t1." + str(a) for a in self.alpha]
select_beta = ["t2." + str(b) for b in self.beta]
select_sql = select_alpha + select_beta

# SQL setup for the left side of the dependency in WHERE−
     clause
alpha_sql_generator = ("␣t1.{}␣=␣t2.{}␣".format(a, a)
                       for a in self.alpha)
and_alpha = ’␣AND␣’.join(alpha_sql_generator)

# SQL setup for the right side of the dependency in WHERE−
     clause
beta_sql_generator = ("␣(t1.{}␣<>␣t2.{})␣".format(b, b)
                      for b in self.beta)
or_beta = ’␣OR␣’.join(beta_sql_generator)
```

Predicates

Implementation - Functional Dependency

Predicates

Why are they useful?

Usage/Implementation 9

Alternative Implementation

```
# Final setup of the entire SQL command
lookup_sql = "SELECT␣DISTINCT␣" + ','.join(select_sql) + \
             "␣FROM␣" + \
             "␣(␣" + "␣NATURAL␣JOIN␣".join(self.table_name
                 ) + "␣)␣" + \
             "␣AS␣t1␣," + \
             "␣(␣" + "␣NATURAL␣JOIN␣".join(self.table_name
                 ) + "␣)␣" + \
             "␣AS␣t2␣" + \
             "␣WHERE␣" + and_alpha + "␣AND␣" + or_beta
```

SQL querie:

```
SELECT DISTINCT t1.country, t2.city
FROM countrydim NATURAL JOIN authordim AS t1, countrydim
    NATURAL JOIN authordim AS t2
WHERE t1.city = t2.city
AND t1.country <> t2.country
```

Predicates

Implementation - Functional Dependency

Predicates
Why are they useful?
Usage/Implementation 10
Alternative Implementation

```python
cursor = dw_rep.connection.cursor()
cursor.execute(lookup_sql)
query_result = cursor.fetchall()
cursor.close()

# Create dict, so that attributes have names
names = [t[0] for t in cursor.description]
dict_result = []
for row in query_result:
    dict_result.append(dict(zip(names, row)))

# If any rows were fetched. Assertion fails
if not dict_result:
    self.__result__ = True
```

Predicates

Usage - Referential Integrity

Referential Integrity - Why is it useful?

- Most DBMS's have various referential integrity rules

Referential Integrity - Why is it useful?

- Most DBMS's have various referential integrity rules
- Not removing the correct data from all tables

Predicates

Usage - Referential Integrity

Setup:

```
ReferentialIntegrityPredicate(
    refs={’FactTable’: (’BookDim’, ’AuthorDim’),
          ’AuthorDim’: (’CountryDim’)},
    points_to_all=True,
    all_pointed_to=True
)
```

SQL querie:

```
SELECT *
FROM facttable
WHERE NOT EXISTS(
    SELECT NULL FROM author_dim
    WHERE facttable.aid = author_dim.aid
    )
```

Predicates

Implementation - Referential Integrity

```
missing_keys = []

    # Maps table names to table_representations
    refs = {}
    for alpha, beta in self.refs.items():
        b = []
        if isinstance(alpha, str):
                a = dw_rep.get_data_representation(alpha)
        else:
            raise ValueError('Expected␣string␣in␣refs,␣got
                :␣' +
                                    str(type(x)))
        if isinstance(beta, str):
            b.append(dw_rep.get_data_representation(beta))
```

Predicates

Implementation - Referential Integrity

```
        else:
            for x in beta:
                if isinstance(x, str):
                    b.append(dw_rep.
                              get_data_representation(x))
                else:
                    raise ValueError('Expected␣string' + '␣in␣refs,␣got:␣' + str(type(x)))
        refs[a] = tuple(b)
    self.refs = refs
```

Predicates

Implementation - Referential Integrity

```python
# If references not given. We check refs between all tables.
if not self.refs:
    self.refs = dw_rep.refs

# Performs check for each pair of main table and foreign key table.
for table, dims in self.refs.items():
    for dim in dims:
        key = dim.key

        # Check that each entry in main table has match
        if self.points_to_all:
            query_result = referential_check(table, dim, key, dw_rep)

            if query_result:
                for row in query_result:
                    msg = '{}:␣{}␣in␣{}␣not␣found␣in␣{}' \
                        .format(key, row[0], table.name, dim.name)
                    missing_keys.append(msg)
```

Predicates

Implementation - Referential Integrity

```
        # Check that each entry in foreign key table has match
        if self.all_pointed_to:
            query_result = referential_check(dim, table, key, dw_rep)

            if query_result:
                for row in query_result:
                    msg = '{}:␣{}␣in␣{}␣not␣found␣in␣{}' \
                        .format(key, row[0], dim.name, table.name)
                    missing_keys.append(msg)

if not missing_keys:
    self.__result__ = True
```

RuleRowPredicate - Why is it useful?

- Gives the user freedom to check for things our other predicate can't
- But with an easy setup

RuleRowPredicate - Why is it useful?

- Gives the user freedom to check for things our other predicate can't
- But with an easy setup
- However slower than others due to the lack of SQL implementation

Predicates

Usage - RuleRowPredicate

Setup:

```python
def no_autobios(name, title):
    return not name == title

RuleRowPredicate(table_name=['AuthorDim','FactTable','BookDim']
                 constraint_function=no_autobios,
                 column_names=['name', 'title'],
                 constraint_args=[],
                 column_names_exclude=False)
```

Predicates

Implementation - RuleRowPredicate

```
# Gets the attribute names for columns needed for test
column_arg_names = self.setup_columns(dw_rep, self.table_name, self.column_names, self.column_names_exclude)

func_args = inspect.getargspec(self.constraint_function).args
if len(func_args) != len(column_arg_names) + len(self.constraint_args):
    raise ValueError("""Number of columns and number of arguments do not match""")
```

Predicates

Implementation - RuleRowPredicate

```
# Iterates over each row, calling the constraint function upon it
for row in dw_rep.iter_join(self.table_name):

    # Finds parameters. First attributes then additional params.
    arguments = []
    for name in column_arg_names:
        arguments.append(row[name])

    if self.constraint_args:
        arguments.append(*self.constraint_args)

    # Runs function on parameters
    if not self.constraint_function(*arguments):
        wrong_rows.append(row)

if not wrong_rows:
    self.__result__ = True
```

Now: SQL queries

```
    def run(self, dw_rep):
        pred_sql = \
            "␣SELECT␣COUNT(*)␣" + \
            "␣FROM␣" + "NATURAL␣JOIN␣".join(self.table_name)

        cursor = dw_rep.connection.cursor()
        cursor.execute(pred_sql)
        query_result = cursor.fetchall()
        cursor.close()

        if query_result[0] == self.number_of_rows:
            self.__result__ = True
```

Predicates

Alternative Implementation - row_count_predicate

Alexander Branborg,
Arash Michael Aami Kjær,
Mathias Claus Jensen,
Mikael Vind Mikkelsen

Alternative: Representation objects in python

```
    def run(self, dw_rep):
        self.row_number = 0
        self.table = []

        for row in dw_rep.get_data_representation(self.table_name):
            self.table.append(row)
            self.row_number += 1

        if len(self.table) == self.number_of_rows:
            self.__result__ = True
        else:
            self.__result__ = False
```

Thank you for using this theme!

AALBORG UNIVERSITY
DENMARK