SkiRaff an ETL Testing Framework for pygrametl

June 16, 2016

Alexander Branborg abran13@student.aau.dk
Arash Michael Sami 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

Problem

Hvad vil vi?

- Vi vil lave et framework som kan hjælpe ETL programmører med at teste deres systemer

Problem

Det nuværende marked

Problem

Det nuværende marked

- Table comparisons
  - e.g. AnyDBTest
  - Pro: Folk kan lave assertions omkring stort set alt
  - Con: Kræver meget kodning, hvor man nemt kan lave fejl

Problem

Det nuværende marked

- Table comparisons
  - e.g. AnyDBTest
  - Pro: Folk kan lave assertions omkring stort set alt
  - Con: Kræver meget kodning, hvor man nemt kan lave fejl
- GUI baseret testing
  - e.g. QuerySurge
  - Pro: Kræver ikke meget kode
  - Con: GUI baseret og kan hurtigt blive kompleks.

Problem

Kriterier til vores framework

Problem

Kriterier til vores framework

- Frameworket skal kunne bruges til automation af tests
  - Da agilt er vejen frem og automation af tests er en hjørne sten deri

Problem

Kriterier til vores framework

- Frameworket skal kunne bruges til automation af tests
  - Da agilt er vejen frem og automation af tests er en hjørne sten deri
- Frameworket skal mindske det krævede kode som skal skrives for at udføre ens tests
  - Mindre test kode leder som udgangspunkt til mindre bugs i ens tests
    - Nuværende test software kræver typisk meget kode i form af at sætte tables op

Problem

Kriterier til vores framework

- Frameworket skal kunne bruges til automation af tests
  - Da agilt er vejen frem og automation af tests er en hjørne sten deri
- Frameworket skal mindske det krævede kode som skal skrives for at udføre ens tests
  - Mindre test kode leder som udgangspunkt til mindre bugs i ens tests
    - Nuværende test software kræver typisk meget kode i form af at sætte tables op
- Det skal være kode orienteret
  - Samme filosofi som pygrametl

SkiRaff

SkiRaff

- Et framework til at teste ETL programmer

SkiRaff

- Et framework til at teste ETL programmer
- Man laver assertions om ens populated DW ved hjælp af Predicates
  - Disse Predicates modelere typiske ting som man vil teste for og kan tilpasses til ens DW

SkiRaff

- Et framework til at teste ETL programmer
- Man laver assertions om ens populated DW ved hjælp af Predicates
  - Disse Predicates modelere typiske ting som man vil teste for og kan tilpasses til ens DW
- Kan lave funktionelle tests på et system niveau
  - Pro: Vi tester systemet som en helhed, og kan fange fejl som er skyldet af at flere komponeneter interagere med hinanden
  - Con: Gør at det er svært at finde ud af præcis hvor fejl opstår

SkiRaff

- Et framework til at teste ETL programmer
- Man laver assertions om ens populated DW ved hjælp af Predicates
  - Disse Predicates modelere typiske ting som man vil teste for og kan tilpasses til ens DW
- Kan lave funktionelle tests på et system niveau
  - Pro: Vi tester systemet som en helhed, og kan fange fejl som er skyldet af at flere komponeneter interagere med hinanden
  - Con: Gør at det er svært at finde ud af præcis hvor fejl opstår
- Funktionalitet til at man kan udskifte data kilder til test data kilder
  - Hvis man bruger pygrametl

SkiRaff

- Et framework til at teste ETL programmer
- Man laver assertions om ens populated DW ved hjælp af Predicates
  - Disse Predicates modelere typiske ting som man vil teste for og kan tilpasses til ens DW
- Kan lave funktionelle tests på et system niveau
  - Pro: Vi tester systemet som en helhed, og kan fange fejl som er skyldet af at flere komponeneter interagere med hinanden
  - Con: Gør at det er svært at finde ud af præcis hvor fejl opstår
- Funktionalitet til at man kan udskifte data kilder til test data kilder
  - Hvis man bruger pygrametl
- Bygget til at kunne samarbejde med pygrametl
  - Kan dog sagtens bruges uden

SkiRaff

SkiRaff

- Et framework til at teste ETL programmer
- Man laver assertions om ens populated DW ved hjælp af Predicates
  - Disse Predicates modelere typiske ting som man vil teste for og kan tilpasses til ens DW
- Kan lave funktionelle tests på et system niveau
  - Pro: Vi tester systemet som en helhed, og kan fange fejl som er skyldet af at flere komponeneter interagere med hinanden
  - Con: Gør at det er svært at finde ud af præcis hvor fejl opstår
- Funktionalitet til at man kan udskifte data kilder til test data kilder
  - Hvis man bruger pygrametl
- Bygget til at kunne samarbejde med pygrametl
  - Kan dog sagtens bruges uden
- Kan bruges sammen med PEP249 compatible DBMS'er

SkiRaff

Overview af frameworkets komponenter [Lav en fin graf her!]

Demo

Demo Af SkiRaff

Arash

- Arash 1

Arash

Arash

- Arash 1
- Arash 2

Arash

Arash

- Arash 1
- Arash 2
- Arash 3

Arash

Arash

- Arash 1
- Arash 2
- Arash 3
- Arash 4

Predicates
Agenda

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

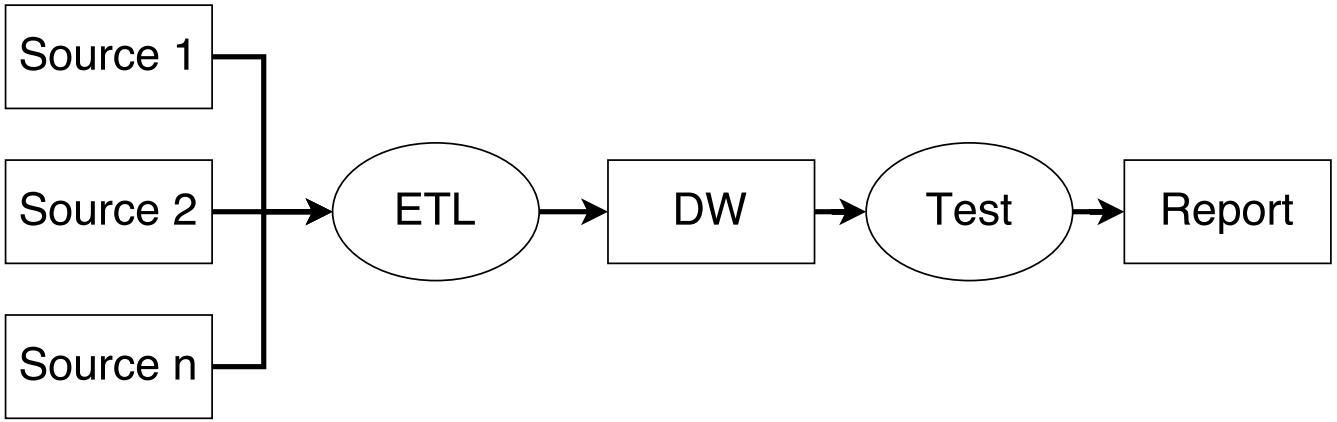
Predicates
Why are they useful?

- Systems level testing
  - Data loss

Predicates

Why are they useful?

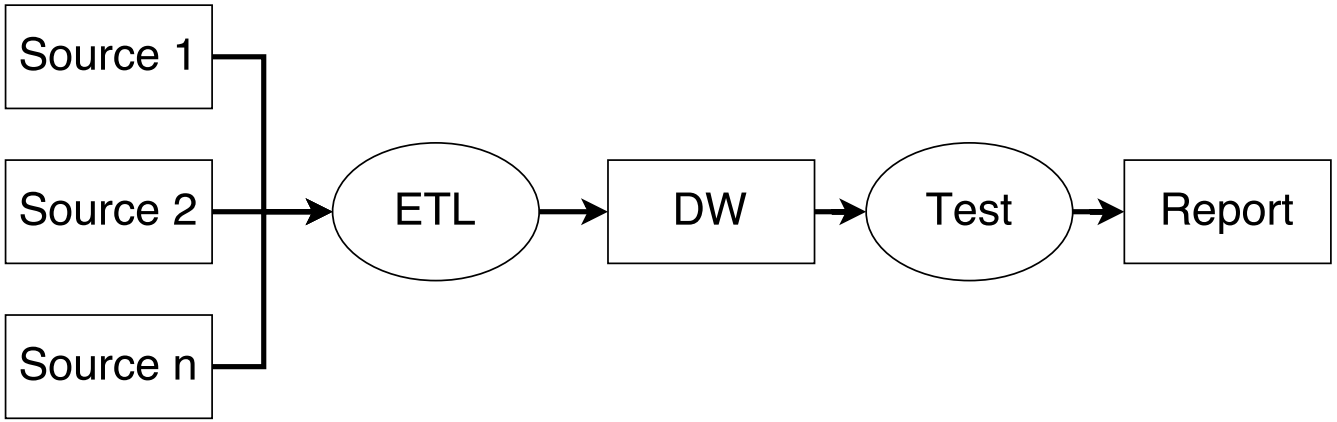
- Systems level testing
  - Data loss
- Source to target test



Predicates

Why are they useful?

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



- Regression testing
- Business Rules

Predicates available in SKiRaff

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

Predicates available in SKiRaff

- RowCountPredicate
- ColumnNotNullPredicate
- **ReferentialIntegrityPredicate**
  - **Advanceret predicat**
- **FunctionalDependencyPredicate**
  - **Ligner mange af vores predicater.**
- SCDVersionPredicate
- CompareTablePredicate
- **RuleRowPredicate**
  - **Bruger ikke SQL men representation**
- RuleColumnPredicate

Functional Dependency - Why is it useful?

- A, B –> C

Functional Dependency - Why is it useful?

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

Setup:

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

SQL querie:

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

Predicates
Implementation - Functional Dependency


```
1  # Creates part of select statement to get keys
2  select_alpha = ["t1." + str(a) for a in self.alpha]
3  select_beta = ["t2." + str(b) for b in self.beta]
4  select_sql = select_alpha + select_beta
5
6  # SQL setup for the left side of the dependency in WHERE-
       clause
7  alpha_sql_generator = ("␣t1.{}␣=␣t2.{}␣".format(a, a)
8                         for a in self.alpha)
9  and_alpha = '␣AND␣'.join(alpha_sql_generator)
10
11 # SQL setup for the right side of the dependency in WHERE-
       clause
12 beta_sql_generator = ("␣(t1.{}␣<>␣t2.{})␣".format(b, b)
13                        for b in self.beta)
14 or_beta = '␣OR␣'.join(beta_sql_generator)
```

Predicates
Implementation - Functional Dependency


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

SQL querie:

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

Predicates
Implementation - Functional Dependency


```
1  cursor = dw_rep.connection.cursor()
2  cursor.execute(lookup_sql)
3  query_result = cursor.fetchall()
4  cursor.close()
5
6  # Create dict, so that attributes have names
7  names = [t[0] for t in cursor.description]
8  dict_result = []
9  for row in query_result:
10     dict_result.append(dict(zip(names, row)))
11
12 # If any rows were fetched. Assertion fails
13 if not dict_result:
14     self.__result__ = True
```

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:

```
1 ReferentialIntegrityPredicate(
2     refs={'FactTable': ('BookDim', 'AuthorDim'),
3           'AuthorDim': ('CountryDim')},
4     points_to_all=True,
5     all_pointed_to=True
6 )
```

SQL querie:

```
1 SELECT *
2 FROM facttable
3 WHERE NOT EXISTS(
4     SELECT NULL FROM author_dim
5     WHERE facttable.aid = author_dim.aid
6     )
```

Predicates

Implementation - Referential Integrity

```
1  missing_keys = []
2
3      # Maps table names to table_representations
4      refs = {}
5      for alpha, beta in self.refs.items():
6          b = []
7          if isinstance(alpha, str):
8                  a = dw_rep.get_data_representation(alpha)
9          else:
10             raise ValueError('Expected string in refs, got
                   : ' +
11                                  str(type(x)))
12         if isinstance(beta, str):
13             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


```
1  # If references not given. We check refs between all
       tables.
2  if not self.refs:
3      self.refs = dw_rep.refs
4
5  # Performs check for each pair of main table and foreign
       key table.
6  for table, dims in self.refs.items():
7      for dim in dims:
8          key = dim.key
9
10         # Check that each entry in main table has match
11         if self.points_to_all:
12             query_result = referential_check(table, dim,
                   key, dw_rep)
13
14             if query_result:
15                 for row in query_result:
16                     msg = '{}: {} in {} not found in {}' \
17                         .format(key, row[0], table.name,
                               dim.name)
18                     missing_keys.append(msg)
```


Department of Computer Science
Aalborg University
Denmark

Predicates
Implementation - Referential Integrity


```
1          # Check that each entry in foreign key table has
               match
2          if self.all_pointed_to:
3              query_result = referential_check(dim, table,
                   key, dw_rep)
4
5              if query_result:
6                  for row in query_result:
7                      msg = '{}: {} in {} not found in {}' \
8                          .format(key, row[0], dim.name,
                               table.name)
9                      missing_keys.append(msg)
10
11 if not missing_keys:
12     self.__result__ = True
```


Department of Computer Science
Aalborg University
Denmark

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

Setup:

```
1 def no_autobios(name, title):
2     return not name == title
3
4 RuleRowPredicate(table_name=['AuthorDim','FactTable','
      BookDim']
5                  constraint_function=no_autobios,
6                  column_names=['name', 'title'],
7                  constraint_args=[],
8                  column_names_exclude=False)
```

Predicates
Implementation - RuleRowPredicate


```
1 # Gets the attribute names for columns needed for test
2 column_arg_names = self.setup_columns(dw_rep, self.
      table_name, self.column_names, self.
      column_names_exclude)
3
4 func_args = inspect.getargspec(self.constraint_function).
      args
5 if len(func_args) != len(column_arg_names) + len(self.
      constraint_args):
6     raise ValueError("""Number of columns and number of
          arguments do not match""")
```

Predicates

Implementation - RuleRowPredicate

```
1  # Iterates over each row, calling the constraint function
       upon it
2  for row in dw_rep.iter_join(self.table_name):
3
4      # Finds parameters. First attributes then additional
           params.
5      arguments = []
6      for name in column_arg_names:
7          arguments.append(row[name])
8
9      if self.constraint_args:
10         arguments.append(*self.constraint_args)
11
12     # Runs function on parameters
13     if not self.constraint_function(*arguments):
14         wrong_rows.append(row)
15
16 if not wrong_rows:
17     self.__result__ = True
```

Now: SQL queries

```
25      def run(self, dw_rep):
26          pred_sql = \
27              "␣SELECT␣COUNT(*)␣" + \
28              "␣FROM␣" + "NATURAL␣JOIN␣".join(self.
                    table_name)
29
30          cursor = dw_rep.connection.cursor()
31          cursor.execute(pred_sql)
32          query_result = cursor.fetchall()
33          cursor.close()
34
35          if query_result[0] == self.number_of_rows:
36              self.__result__ = True
```

Alternative: Representation objects in python

```
21      def run(self, dw_rep):
22          self.row_number = 0
23          self.table = []
24
25          for row in dw_rep.get_data_representation(self.
                table_name):
26              self.table.append(row)
27              self.row_number += 1
28
29          if len(self.table) == self.number_of_rows:
30              self.__result__ = True
31          else:
32              self.__result__ = False
```

Hvordan evaluerede vi SkiRaff?

- SkiRaff vs. Manual

Hvordan evaluerede vi SkiRaff?

- SkiRaff vs. Manual
- Metrikker: Statements & Runtime

Hvordan evaluerede vi SkiRaff?

- SkiRaff vs. Manual
- Metrikker: Statements & Runtime
- ETL program: Håndhæver ikke data integritet

Hvordan evaluerede vi SkiRaff?

- SkiRaff vs. Manual
- Metrikker: Statements & Runtime
- ETL program: Håndhæver ikke data integritet
- Test plan: Dækker alle SkiRaff predicates

Hvordan evaluerede vi SkiRaff?

	SkiRaff	Manual
Number of statements	11 stmt	110 stmt
Execution Time Setup	79.52 sec	79.44 sec
Execution Time Test Cases	18.02 sec	18.23 sec
Execution Time Total	97.52 sec	97.67 sec

Figure: Results af evaluering med 10000 rækker i hver tabel udover CountryDim

Metrikker

Statiske

- **Statements**

Metrikker

Statiske

- **Statements**
- Fog index

Metrikker

Statiske

- **Statements**
- Fog index
- Cyclomatic complexity

Metrikker

Statiske

- **Statements**
- Fog index
- Cyclomatic complexity

Dynamiske

- **Runtime**

Metrikker

Statiske

- **Statements**
- Fog index
- Cyclomatic complexity

Dynamiske

- **Runtime**
- Bug Count

Usability Testing

Udførsel

- Opskriv flere realistiske test planer

Usability Testing

Udførsel

- Opskriv flere realistiske test planer
- Få ekspert brugere til at implementere planer med forskellige værktøjer:
  - SkiRaff
  - Manuel
  - QuerySurge
  - AnyDBTest

Usability Testing

Udførsel

- Opskriv flere realistiske test planer
- Få ekspert brugere til at implementere planer med forskellige værktøjer:
  - SkiRaff
  - Manuel
  - QuerySurge
  - AnyDBTest
- Fokuser på implementations hastighed og udsagn

Usability Testing

Udførsel

- Opskriv flere realistiske test planer
- Få ekspert brugere til at implementere planer med forskellige værktøjer:
  - SkiRaff
  - Manuel
  - QuerySurge
  - AnyDBTest
- Fokuser på implementations hastighed og udsagn

Negativer

- Praktisk organisering

Usability Testing

Udførsel

- Opskriv flere realistiske test planer
- Få ekspert brugere til at implementere planer med forskellige værktøjer:
  - SkiRaff
  - Manuel
  - QuerySurge
  - AnyDBTest
- Fokuser på implementations hastighed og udsagn

Negativer

- Praktisk organisering
- Kvalitativ data kan også være svær at evaluere

Usability Testing

Udførsel

- Opskriv flere realistiske test planer
- Få ekspert brugere til at implementere planer med forskellige værktøjer:
  - SkiRaff
  - Manuel
  - QuerySurge
  - AnyDBTest
- Fokuser på implementations hastighed og udsagn

Negativer

- Praktisk organisering
- Kvalitativ data kan også være svær at evaluere
- Store mængder data skal behandles

Konklusion

Hvad har vi lavet

- SkiRaff: Et framework til test af pygrametl programmer

Konklusion

Hvad har vi lavet

- SkiRaff: Et framework til test af pygrametl programmer
- Dækker mange forskellige test cases med predicate klasserne

Konklusion

Hvad har vi lavet

- SkiRaff: Et framework til test af pygrametl programmer
- Dækker mange forskellige test cases med predicate klasserne
- Tests behøver færre linjer, men udføres med samme hastighed ift. manuel test

Konklusion

Hvad har vi lavet

- SkiRaff: Et framework til test af pygrametl programmer
- Dækker mange forskellige test cases med predicate klasserne
- Tests behøver færre linjer, men udføres med samme hastighed ift. manuel test

Perspektiv

- Business Intelligence i moderne sammenhæng

Konklusion

Hvad har vi lavet

- SkiRaff: Et framework til test af pygrametl programmer
- Dækker mange forskellige test cases med predicate klasserne
- Tests behøver færre linjer, men udføres med samme hastighed ift. manuel test

Perspektiv

- Business Intelligence i moderne sammenhæng
- SkiRaff og ETL udvikling

Thank you for listening

AALBORG UNIVERSITY
DENMARK