MYSQL Wrapper

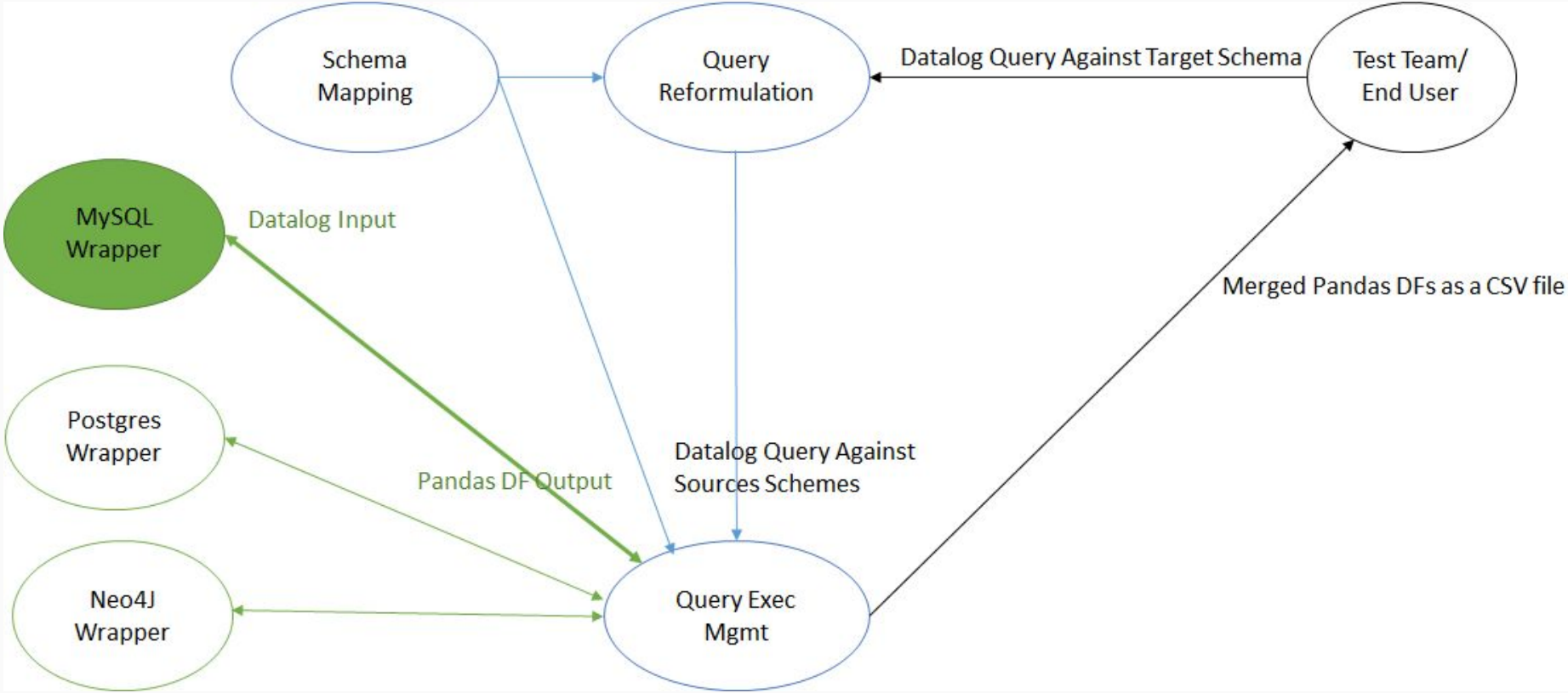
DSE203 - Final Project

Alex Egg, Deepthi Nagaraj, Peyman Hesami

Agenda

- ➔ Introduction and Project goals
- ➔ Ideal implementation pipeline
- ➔ Alternate implementation - Interface design
- ➔ Steps taken, Tools used and Syntax assumed
- ➔ Parser details
- ➔ Testing and Final Deliverable
- ➔ Future work
- ➔ Demo

End to End System Architecture Layout



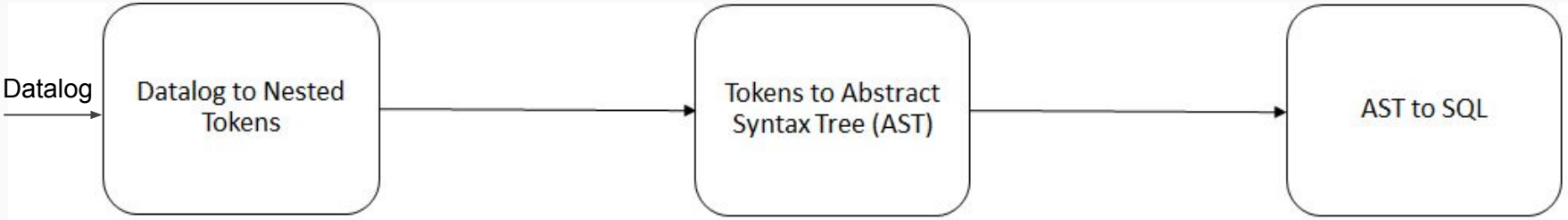
Project goals

The goal of this project is to create a wrapper around the MySQL database which hosts the GTD data.

The wrapper would be used to:

1. Parse a Datalog Query as input
2. Generate the respective SQL Query
3. Execute the SQL Query on the MySql database
4. Return the query result as type: pandas.DataFrame

Ideal implementation pipeline



Map datalog to rule, a fact, an assertion rule or an assertion fact based on datalog grammar

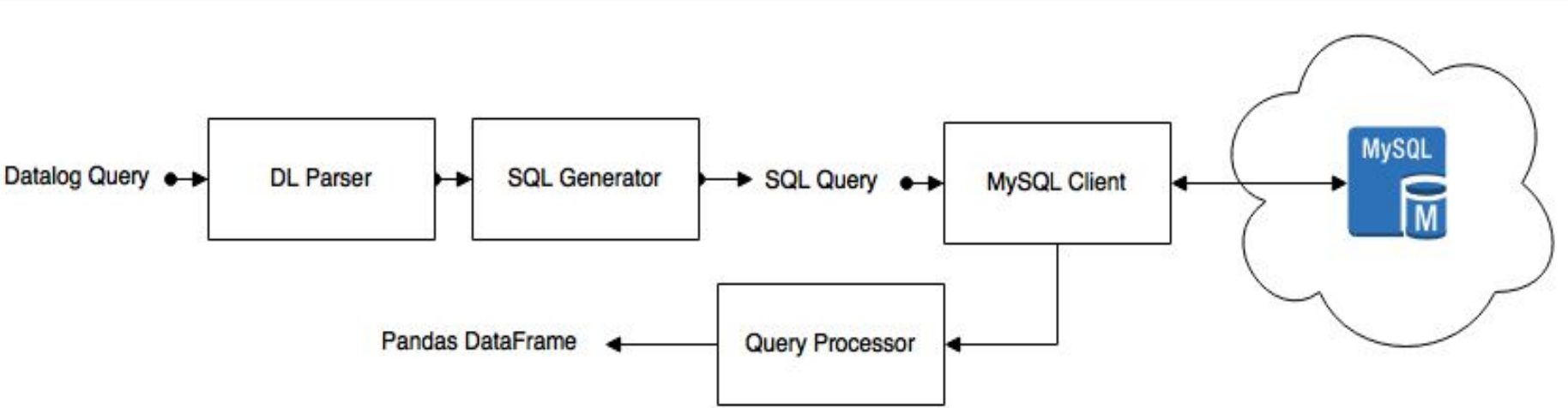
Breaks down into its more basic forms according to the Datalog Syntax, such as numbers, constants, variables, ..

SQL based on the type of query (i.e. conjunctive, disjunctive, or assert query)

Python YADI Package

MYSQL interface details (Alternative approach)

- Simple python MySQL client: PyMySQL



Steps taken

1. Finalize the syntax for datalog by working with other teams
2. Parse the datalog query using python regex statements and convert into sql query
3. Run the sql query on the mysql database and return the results as a Pandas dataframe
4. Verify the functionality of the wrapper through testing

Tools Used

Native Python Datalog Parser (no 3rd party lib)

- Python
- Regex
- Pandas
- NumPy
- PyMySql

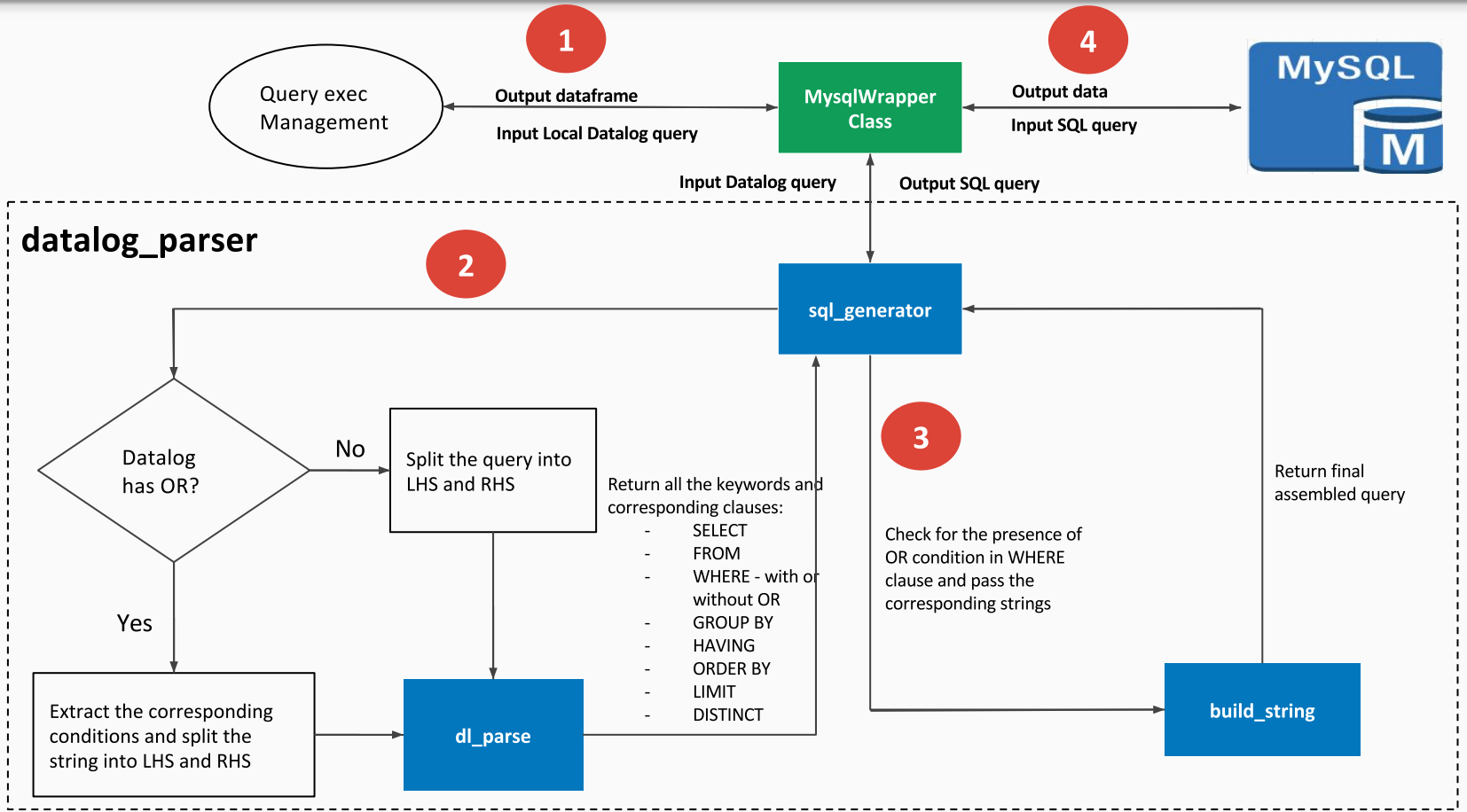
Datalog Syntax

```
Result(e, c,p) :=   relation(a, ‘string’, _, c),
                    d+f >= 2,
                    l < 50,
                    GROUP_BY ([c], p = MAX(e)),
                    p > 40,
                    SORT_BY (p, 'DESC'),
                    LIMIT (10),
                    DISTINCT;
```

OR operation is denoted by two datalog queries with the same head:
Result(a):= relation(c, a), c>20, d<30
Result(a):= relation(c, a,), c <50

AND operation is denoted by one query and one head:
Result(a):= relation1(c, a), relation2(d, a, _), c>20, d<10
Note: relation1 and relation2 are not applicable to MySQL wrapper as there is only one relation

Parser Flow Chart



Testing

Testing was carried out in two steps:

- **Local testing using custom queries:**
  - We created some specific queries to test out the end-to-end working on our end
- **Testing using Testing team queries:**
  - Used the queries created by Testing team to test the overall package

Final Deliverable

- Python package: mysql_wrapper
  https://github.com/DSE220-Final-Project/DSE203-final-project

```
from mysql_wrapper import mysql_wrapper
query= "DB1(s1) :- gtd(_, 2013, 12, _, _, _, _,'Iraq', _, _, …).”

wrapper = mysql_wrapper.MysqlWrapper()
dataframe = wrapper.execute(query)

SELECT summary AS s1 FROM  gtd WHERE imonth = 12 AND country_txt = 'Iraq' AND iyear = 2013
                                                    s1
0    12/01/2013: A suicide bomber detonated an expl...
1    12/01/2013: An explosive device targeted a pol...
2    12/01/2013: An explosive device detonated near...
```

Other possible improvements (Future work)

- ➔ Implement group_by/sort_by/limit/distinct as functions (in both wrapper and mediator)
- ➔ Implement An Abstract Syntax Tree (AST) as an entity between Wrappers and Execution entity:
  - ◆ Pass the AST tree (instead of datalog) to the wrappers to avoid duplicate datalog parsing across wrappers.

Demo

Questions?

Appendix

Parser details

- ➔ Splitting the datalog based on the head to extract any possible UNION
- ➔ Splitting each individual datalog based on the separator (:=)
  - ◆ The left side will give the projected columns (SELECT Clause) and their aliases
  - ◆ The right side will be parsed based on the following rules:
    - Extract a table in the form of relation (a,_,2,d)
      - Extract the table name and add it to the FROM clause
      - Extract the strings and numbers indices and extract the actual column name from a hash table.
      - Extract the string and numbers and add them as equality to the WHERE clause
      - Extract the variable (non-string/non-numbers) and use them in combination of the predicates for non-equalities in the WHERE clause

Parser details- cont’d

- Extract GROUP_BY([d], c=func(a))
  - Extract GROUP_BY (using regex) and its two arguments
  - Extract the first argument and add it to GROUP BY clause
  - Extract the second argument, map the variable inside func to the actual column name and add it to SELECT clause as: SELECT func(col_name(c)) AS c

- Extract SORT_BY ([c], sort_type)
  - Extract SORT_BY using (regex) and its two arguments
  - Extract the first argument, map it to the actual column name.
  - Extract the second argument and alongside the first argument add it to the ORDER BY clause as : ORDER BY col_name(c) sort_type

Parser details- cont’d

- Extract LIMIT (limit_num)
  - Extract string LIMIT (using regex) and its argument and add it to the LIMIT clause as LIMIT limit_num

- Extract the rest of the right hand side as the predicates
  - Extract the rest of the expressions, separated by comma, as predicates.
  - For each comma separated expression, split the expression by allowed comparisons (>,<,=,!=,>=,<=).
    - Extract the comparison operator (operator)
    - Extract the left hand side and map it to the actual column name (col_name(var))
    - Extract the right hand side as the value of comparison (val)
  - Add this to WHERE clause as col_name(var) operator var