Extraction of Embedded Queries via Static Analysis of Host Code

Petros Manousis, Apostolos Zarras, Panos Vassiliadis
University of Ioannina, Ioannina, Greece

George Papastefanatos
Research Center “Athena” \ IMIS, Athens, Greece

Embedded queries in data-intensive information systems

String-based

```
1 $result = db_query('SELECT source, alias FROM {url_alias} WHERE source in
      (:system) AND language = :language_none ORDER BY pid asc;', $args);
```

Object-based

```
1  function _profile_get_fields($category,$register=FALSE) {
2   $query = db_select('profile_field');
3   if ($register) {
4    $query->condition('register',1);
5   }
6   else {
7    $query->condition('category',db_like($category),'LIKE');
8   }
9   while (!user_access('administer users')) {
10   $query->condition('visibility',PROFILE_HIDDEN,'<>');
11  }
12  return $query->fields('profile_field')->orderBy('category','ASC')
13    ->orderBy('weight','ASC')->execute();
14 }
```

Why bother?

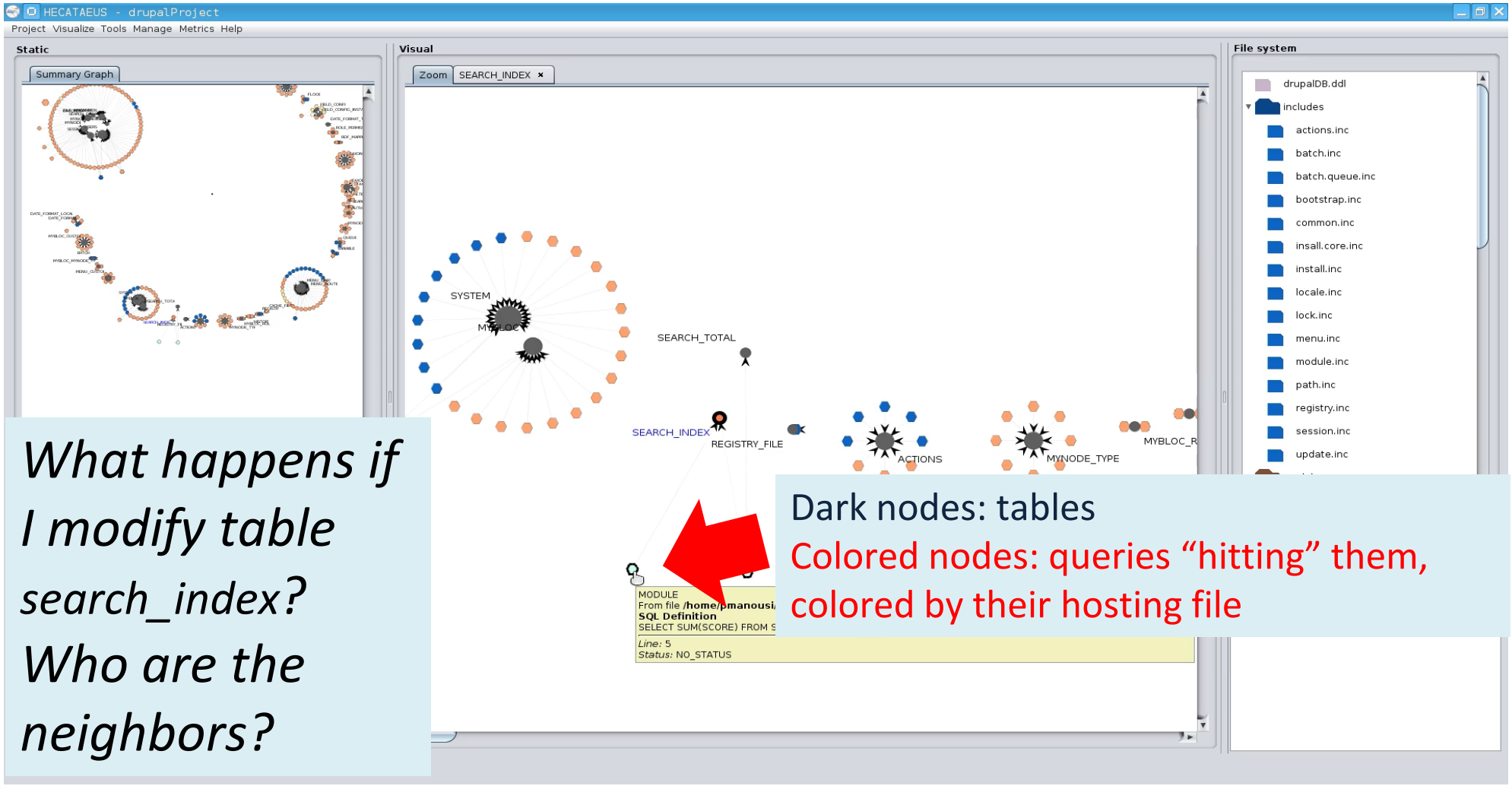
- It is really important to locate embedded queries in the host application of a data intensive information system
- We need to be able to locate, inspect, and visualize data-related code for

  - understanding how data and code inter-relate

  - determining evolution's possible impacts

  - migrating the application to another development language
- Yet, identifying the location and semantics of these queries is really hard, as already shown

Hecataeus tool for charting and impact prediction



What happens if I modify table search_index? Who are the neighbors?

Dark nodes: tables

Colored nodes: queries "hitting" them, colored by their hosting file

Hecataeus tool: **http://www.cs.uoi.gr/~pvassil/projects/hecataeus/**

State of the art

	Host languages	Query type	Variants
Christensen et al. (Static Anal. Symp. 2003)	Java	String-based	
Gould et al. (ICSE 2004)	Java	String-based	
Cleve et al. (WCRE 2006)	Java	String-based	Partial
Van den Brink et al. (SCAM 2007)	Pl/SQL, COBOL, V. Basic	String-based	
Ngo and Tan (IST 2008)	PHP	String-based	
Maule et al. (ICSE 2008)	C#	String-based	
Annamaa et al. (Asian Symp. Prog. Lang. & Syst. 2010)	Java	String-based	ASTs

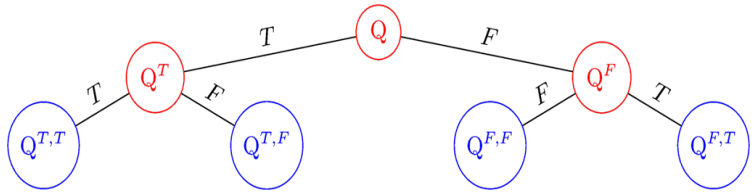
Research goals

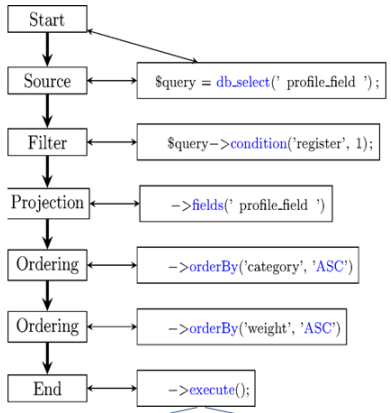
1. Be able to **extract embedded queries** with a simple, generic and understandable method regardless of the host language.
2. Provide understandable intermediate results (e.g., due to loop and branch statements of the source code).
3. Move from code dependent (string or object based constructed query) to a universal code-independent query representation.
4. Be able to output the queries in more than one “concrete” query language, to facilitate
    a. testing the correctness of the extraction
    b. migration from one system to another

Roadmap

1. Overview
2. **A method for Embedded Query Extraction**
3. Experiments
4. Discussion

```php
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {
        $query->condition('register', 1);
  }
  else {
        $query->condition('category', db_like($category), 'LIKE');
  }
  if    (!user_access('administer users')) {
        $query->condition('visibility', PROFILE_HIDDEN, '<>');
  }
  return $query->fields('profile_field')
            ->orderBy('category', 'ASC')
            ->orderBy('weight', 'ASC')
            ->execute();
}
```

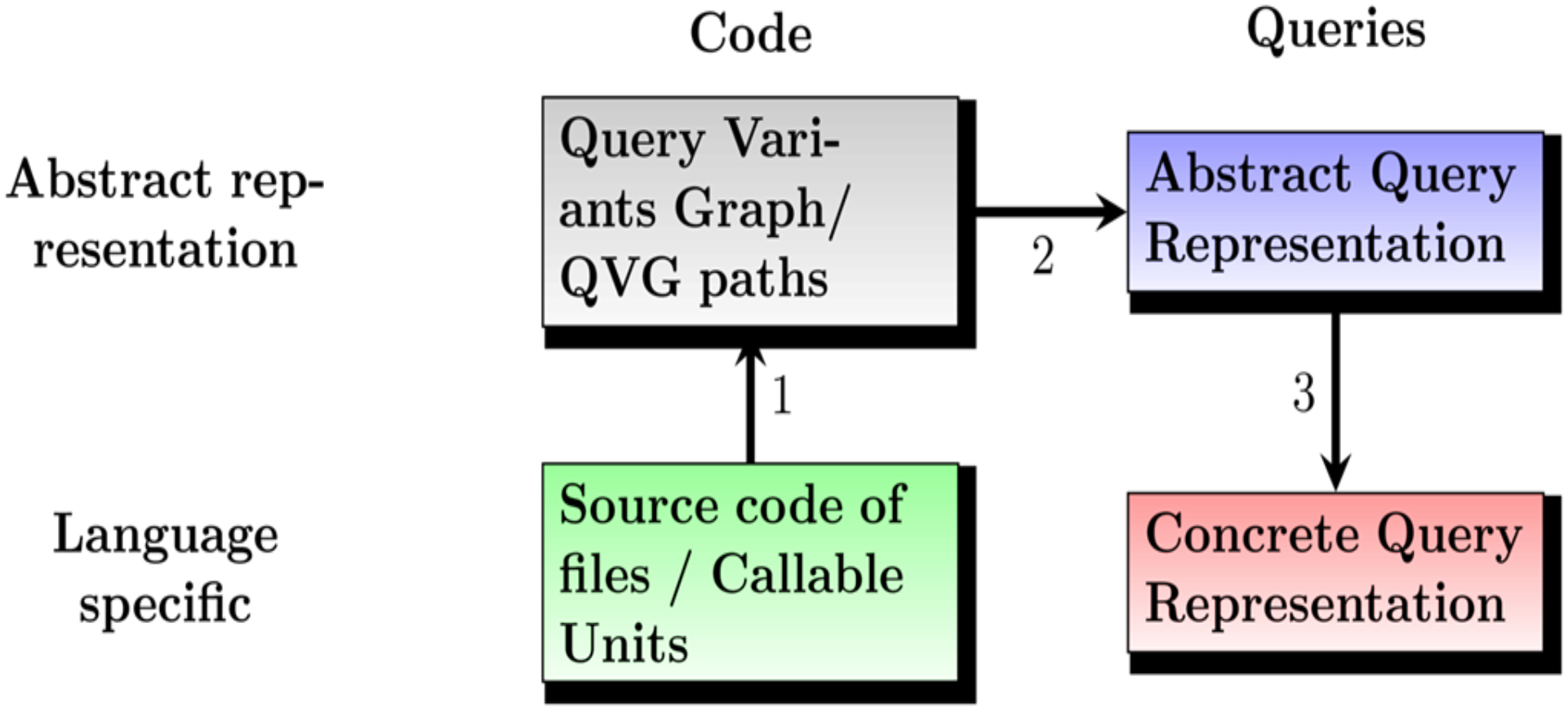





```sql
SELECT profile_field.*
FROM profile_field
WHERE register = 1
ORDER BY category ASC, weight ASC
```

```
db.profile_field.aggregate
([
  { $match: {register: { $eq: 1 } } },
  { $sort: { category: 1, weight: 1 } },
  { profile_field.*: 1 }
])
```

Overview of solution



Roadmap

Input: Source files + keywords

Concrete source code representation

Abstract source code representation

Abstract Query Representation

Output: Concrete, target query representation

Input: Project's folder pf, Files to exclude $excl$, Beginners $begins$, API functions $apiF$, Language specifics $lspecs$
Output: The Abstract Query Representations of the project.

```
1  paths = ∅;
2  files2search = Recursively search the project's folder;
3  foreach file f : files2search do
4  |   if f ∈ excl then
5  |   |   files2search− = f;                    ▷ Remove files w/o queries
   |   end
   end
6  CallableUnits = split file into Callable Units;
7  foreach CallableUnit ∈ CallableUnits do
8  |   QVGs += create a Query Variants Graph (QVG) per Callable Unit;
   end
9  foreach QVG ∈ QVGs do
10 |   paths+ = to transform each QVG to a set of QVG paths;
   end
11 foreach path ∈ paths do
12 |   AQRs+ = extract the embedded queries into an Abstract Query
   |   Representation (AQR);
   end
13 foreach AQR ∈ AQRs do
14 |   export AQR to a target language;
   end
```

Roadmap

Input: Source files + keywords

Concrete source code representation

Abstract source code representation

Abstract Query Representation

Output: Concrete, target query representation

Input: Project's folder pf, Files to exclude $excl$, Beginners $begins$, API functions $apiF$, Language specifics $lspecs$
Output: The Abstract Query Representations of the project.

```
1  paths = ∅;
2  files2search = Recursively search the project's folder;
3  foreach file f : files2search do
4  |   if f ∈ excl then
5  |   |   files2search− = f;                    ▷ Remove files w/o queries
   |   end
   end
6  CallableUnits = split file into Callable Units;
7  foreach CallableUnit ∈ CallableUnits do
8  |   QVGs += create a Query Variants Graph (QVG) per Callable Unit;
   end
9  foreach QVG ∈ QVGs do
10 |   paths+ = to transform each QVG to a set of QVG paths;
   end
11 foreach path ∈ paths do
12 |   AQRs+ = extract the embedded queries into an Abstract Query
   |   Representation (AQR);
   end
13 foreach AQR ∈ AQRs do
14 |   export AQR to a target language;
   end
```

List of db-related functions

Example of source code file

```
/**
 * Process variables for profile-wrapper.tpl.php.
 */
function template_preprocess_profile_wrapper(&$variables) {
  $variables['current_field'] = '';
  if ($field = arg(1)) {
          $variables['current_field'] = $field;
          $variables['theme_hook_suggestions'][] = 'profile_wrapper__' . $field;
  }
}

function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  while (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')->orderBy('category', 'ASC')->orderBy('weight', 'ASC')-
>execute();
}
```

Steps
1. Remove comments
2. Remove files w/o db access

Example of source code file without comments

```
function template_preprocess_profile_wrapper(&$variables) {
  $variables['current_field'] = '';
  if ($field = arg(1)) {
          $variables['current_field'] = $field;
          $variables['theme_hook_suggestions'][] = 'profile_wrapper__' . $field;
  }
}

function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  while (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')->orderBy('category', 'ASC')->orderBy('weight', 'ASC')-
>execute();
}
```

Steps
1. Remove comments
2. Remove files w/o db access

Has the source code of the file any DB connection?

```
function template_preprocess_profile_wrapper(&$variables) {
  $variables['current_field'] = '';
  if ($field = arg(1)) {
          $variables['current_field'] = $field;
          $variables['theme_hook_suggestions'][] = 'profile_wrapper__' . $field;
  }
}

function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  while (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')->orderBy('category', 'ASC')->orderBy('weight', 'ASC')-
>execute();
}
```

Steps
1. Remove comments
2. Remove files w/o db access

Roadmap

Input: Project's folder pf, Files to exclude $excl$, Beginners $begins$, API functions $apiF$, Language specifics $lspecs$
Output: The Abstract Query Representations of the project.

```
1  paths = ∅;
2  files2search = Recursively search the project's folder;
3  foreach file f : files2search do
4  |  if f ∈ excl then
5  |  |  files2search− = f;                    ▷ Remove files w/o queries
   |  end
   end
6  CallableUnits = split file into Callable Units;
7  foreach CallableUnit ∈ CallableUnits do
8  |  QVGs += create a Query Variants Graph (QVG) per Callable Unit;
   end
9  foreach QVG ∈ QVGs do
10 |  paths+ = to transform each QVG to a set of QVG paths;
   end
11 foreach path ∈ paths do
12 |  AQRs+ = extract the embedded queries into an Abstract Query
   |  Representation (AQR);
   end
13 foreach AQR ∈ AQRs do
14 |  export AQR to a target language;
   end
```

Split file to **Callable Units** (methods/functions/...)

```php
function template_preprocess_profile_wrapper(&$variables) {
  $variables['current_field'] = '';
  if ($field = arg(1)) {
          $variables['current_field'] = $field;
          $variables['theme_hook_suggestions'][] = 'profile_wrapper__' . $field;
  }
}
```

01

```php
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  while (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')->orderBy('category', 'ASC')->orderBy('weight', 'ASC')->execute();
}
```

02

Steps
1. Split code in Callable Units (Cus)
2. Remove CUs w/o db access

Keep only DB related functions

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  while (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')
               ->orderBy('category', 'ASC')
               ->orderBy('weight', 'ASC')
               ->execute();
}
```

Steps
1. Split code in Callable Units (Cus)
2. Remove CUs w/o db access

Roadmap

Input: Source files + keywords

Concrete source code representation

Abstract source code representation

Abstract Query Representation

Output: Concrete, target query representation

Input: Project's folder pf, Files to exclude $excl$, Beginners $begins$, API functions $apiF$, Language specifics $lspecs$
Output: The Abstract Query Representations of the project.

```
1  paths = ∅;
2  files2search = Recursively search the project's folder;
3  foreach file f : files2search do
4  |  if f ∈ excl then
5  |  |  files2search− = f;                    ▷ Remove files w/o queries
   |  end
   end
6  CallableUnits = split file into Callable Units;
7  foreach CallableUnit ∈ CallableUnits do
8  |  QVGs += create a Query Variants Graph (QVG) per Callable Unit;
   end
9  foreach QVG ∈ QVGs do
10 |  paths+ = to transform each QVG to a set of QVG paths;
   end
11 foreach path ∈ paths do
12 |  AQRs+ = extract the embedded queries into an Abstract Query
   |  Representation (AQR);
   end
13 foreach AQR ∈ AQRs do
14 |  export AQR to a target language;
   end
```

Roadmap

Input: Project's folder pf, Files to exclude $excl$, Beginners $begins$, API functions $apiF$, Language specifics $lspecs$
Output: The Abstract Query Representations of the project.

```
1  paths = ∅;
2  files2search = Recursively search the project's folder;
3  foreach file f : files2search do
4  |  if f ∈ excl then
5  |  |  files2search− = f;                    ▷ Remove files w/o queries
   |  end
   end
6  CallableUnits = split file into Callable Units;
7  foreach CallableUnit ∈ CallableUnits do
8  |  QVGs += create a Query Variants Graph (QVG) per Callable Unit;
   end
9  foreach QVG ∈ QVGs do
10 |  paths+ = to transform each QVG to a set of QVG paths;
   end
11 foreach path ∈ paths do
12 |  AQRs+ = extract the embedded queries into an Abstract Query
   |  Representation (AQR);
   end
13 foreach AQR ∈ AQRs do
                                          et language;
```

<u>Steps</u>
For each CU
build a tree of possible execution paths

```
1  function _profile_get_fields($category,$register=FALSE) {
2   $query = db_select('profile_field');
3   if ($register) {
4    $query->condition('register',1);
5   }
6   else {
7    $query->condition('category',db_like($category),'LIKE');
8   }
9   while (!user_access('administer users')) {
10   $query->condition('visibility',PROFILE_HIDDEN,'<>');
11  }
12  return $query->fields('profile_field')->orderBy('category','ASC')
13    ->orderBy('weight','ASC')->execute();
14 }
```

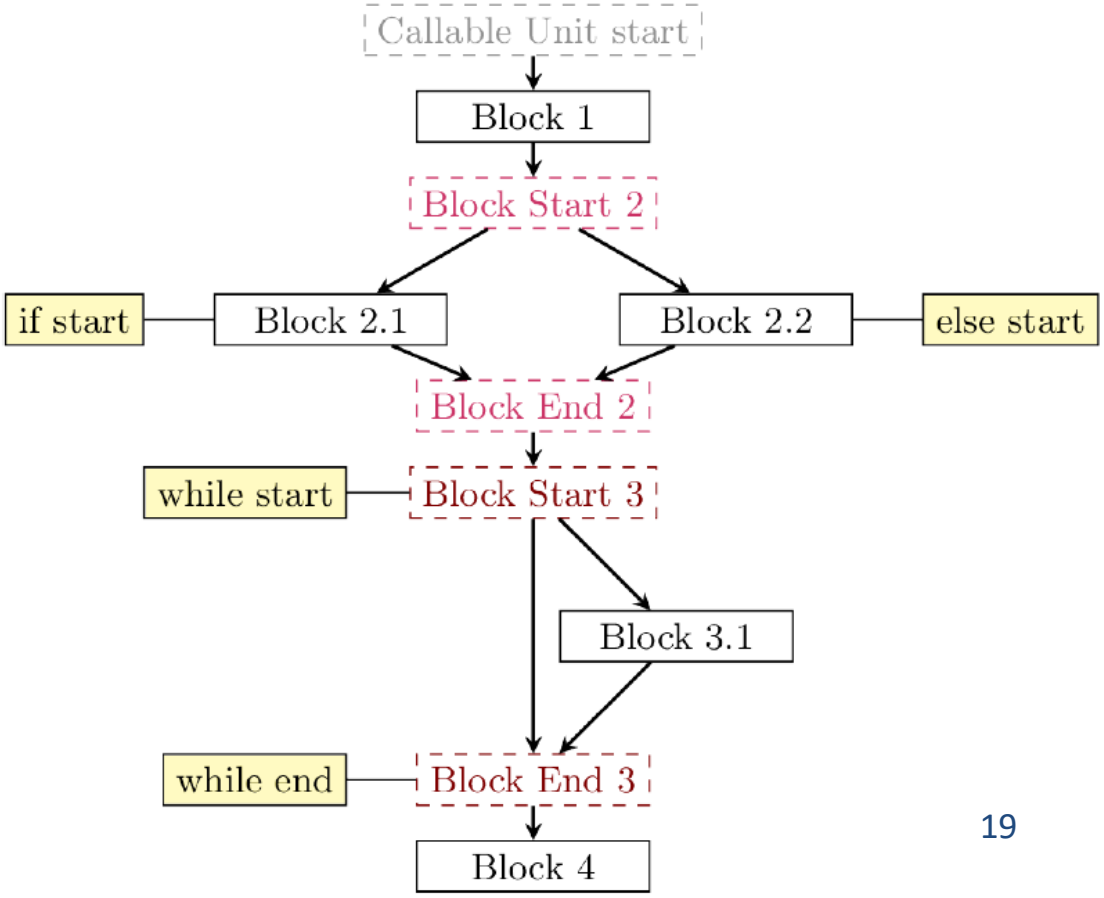
1 2.1 2.2 2 3.1 3 4 root

Query Variants Graph

The **Query Variants Graph** is a "tree-like" graph with

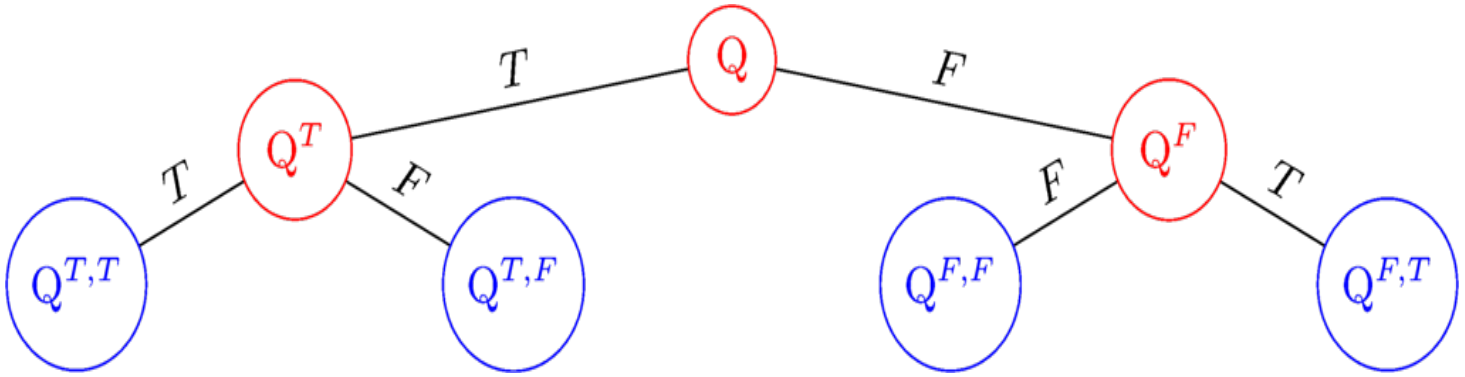
- blocks as its white nodes
- loops and conditionals as its red nodes
- alternatives of the control flow as its edges

Each node carries the respective src code with it



QVG serves an interim means to decide:

- all the alternative query generation control flow paths
- which query-generating parts of the CU pertain to each path



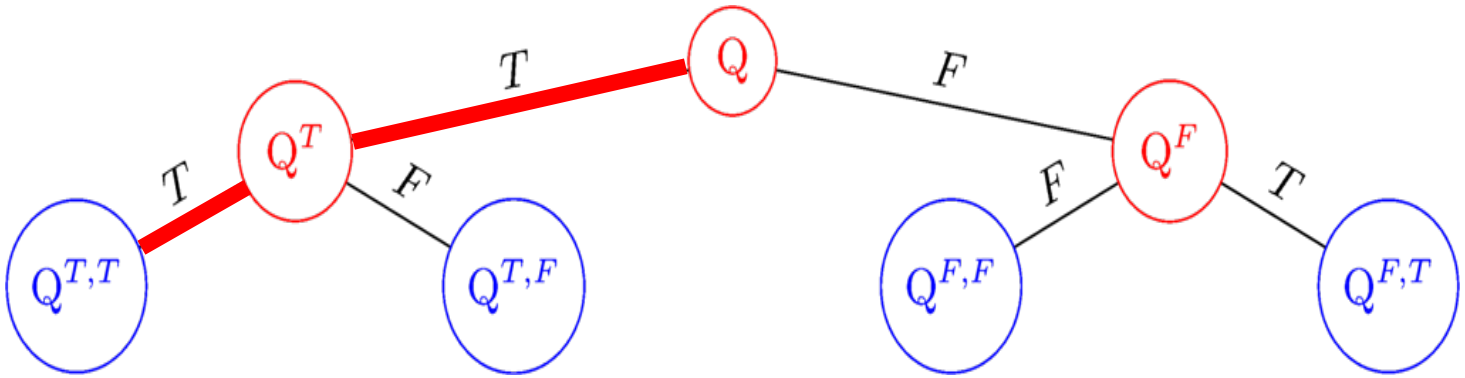
In Q we have not taken any decision for the branches of the query.

In Q^T and Q^F we have taken a decision for the first branch (Q^T has the code that will be executed if the condition is True, and in Q^F the code that will be executed if the condition is False).

Likewise, in $Q^{T,T}$, $Q^{T,F}$, $Q^{F,F}$ and $Q^{F,T}$ we have taken decisions for all the branches of the query Q.
$Q^{T,T}$ is when in both branches the conditions are true.
$Q^{T,F}$ is when the first condition is true while the second is false.
$Q^{F,F}$ is when in both branches the conditions are false.
$Q^{T,F}$ is when the first condition is false while the second is true.

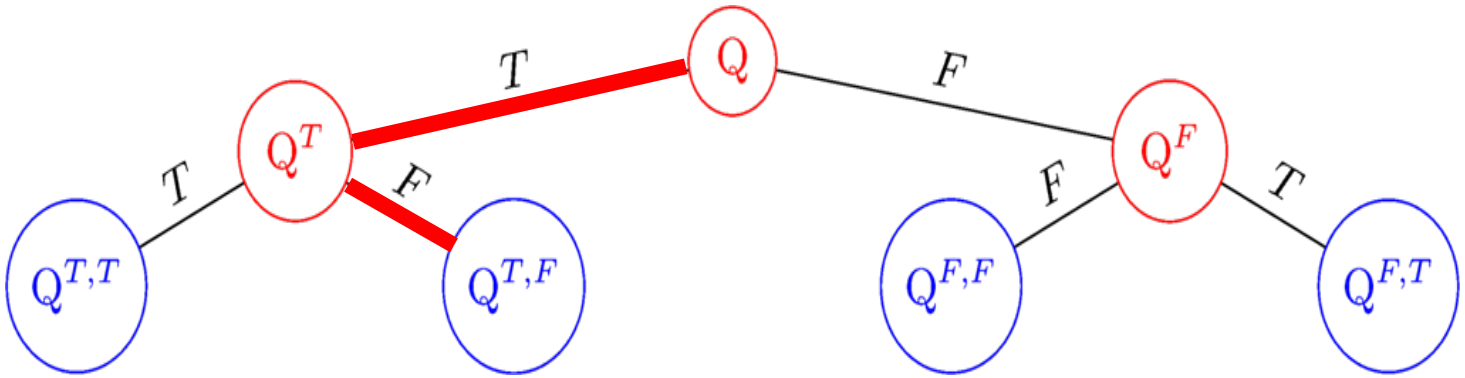
$Q^{T,T}$ execution plan

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  if    (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')
               ->orderBy('category', 'ASC')
               ->orderBy('weight', 'ASC')
               ->execute();
}
```



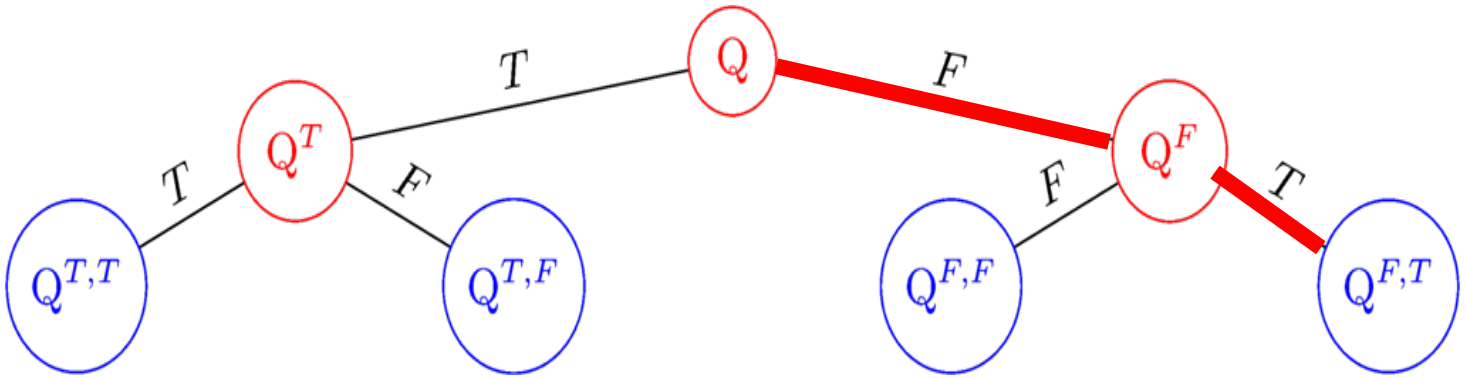
$Q^{T,F}$ execution plan

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  if (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')
               ->orderBy('category', 'ASC')
               ->orderBy('weight', 'ASC')
               ->execute();
}
```



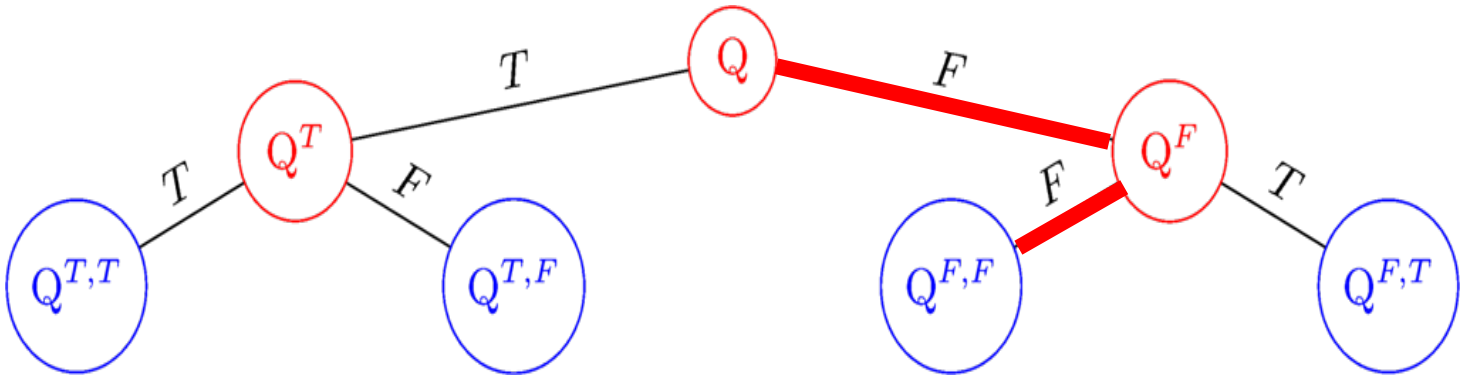
$Q^{F,T}$ execution plans

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  if    (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')
               ->orderBy('category', 'ASC')
               ->orderBy('weight', 'ASC')
               ->execute();
}
```



$Q^{F,F}$ execution plan

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  if   (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')
               ->orderBy('category', 'ASC')
               ->orderBy('weight', 'ASC')
               ->execute();
}
```



On the side: loops converted to branches

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  if (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')
               ->orderBy('category', 'ASC')
               ->orderBy('weight', 'ASC')
               ->execute();
}
```

This `if` used to be a `while`.

Yet: for abstracting the query structure, there is no difference!

Roadmap

Input: Source files + keywords

Concrete source code representation

Abstract source code representation

Abstract Query Representation

Output: Concrete, target query representation

Input: Project's folder pf, Files to exclude $excl$, Beginners $begins$, API functions $apiF$, Language specifics $lspecs$
Output: The Abstract Query Representations of the project.

```
1  paths = ∅;
2  files2search = Recursively search the project's folder;
3  foreach file f : files2search do
4  |  if f ∈ excl then
5  |  |  files2search− = f;                    ▷ Remove files w/o queries
   |  end
   end
6  CallableUnits = split file into Callable Units;
7  foreach CallableUnit ∈ CallableUnits do
8  |  QVGs += create a Query Variants Graph (QVG) per Callable Unit;
   end
9  foreach QVG ∈ QVGs do
10 |  paths+ = to transform each QVG to a set of QVG paths;
   end
11 foreach path ∈ paths do
12 |  AQRs+ = extract the embedded queries into an Abstract Query
   |  Representation (AQR);
   end
13 foreach AQR ∈ AQRs do
14 |  export AQR to a target language;
   end
```

How to abstract each QVG path?

… At this point, we have at our disposal all the linear paths , each with its own sequence of db-related, query-generating, host-language statements

…Each QVG path creates one of the possible queries embedded in the code via its list of host language statements

Still, it is in host-language format. How to abstract?

Ingredients
- A family of Abstract Data Manipulation Operators
- A mapping of host language “method calls” to these operators
- An Abstract Representation of Queries (AQR)
- An algorithm to walk the path and create an AQR

Abstract Data Manipulation Operators

- an extensible palette of operators for data manipulation
- currently with a minimal set of operators

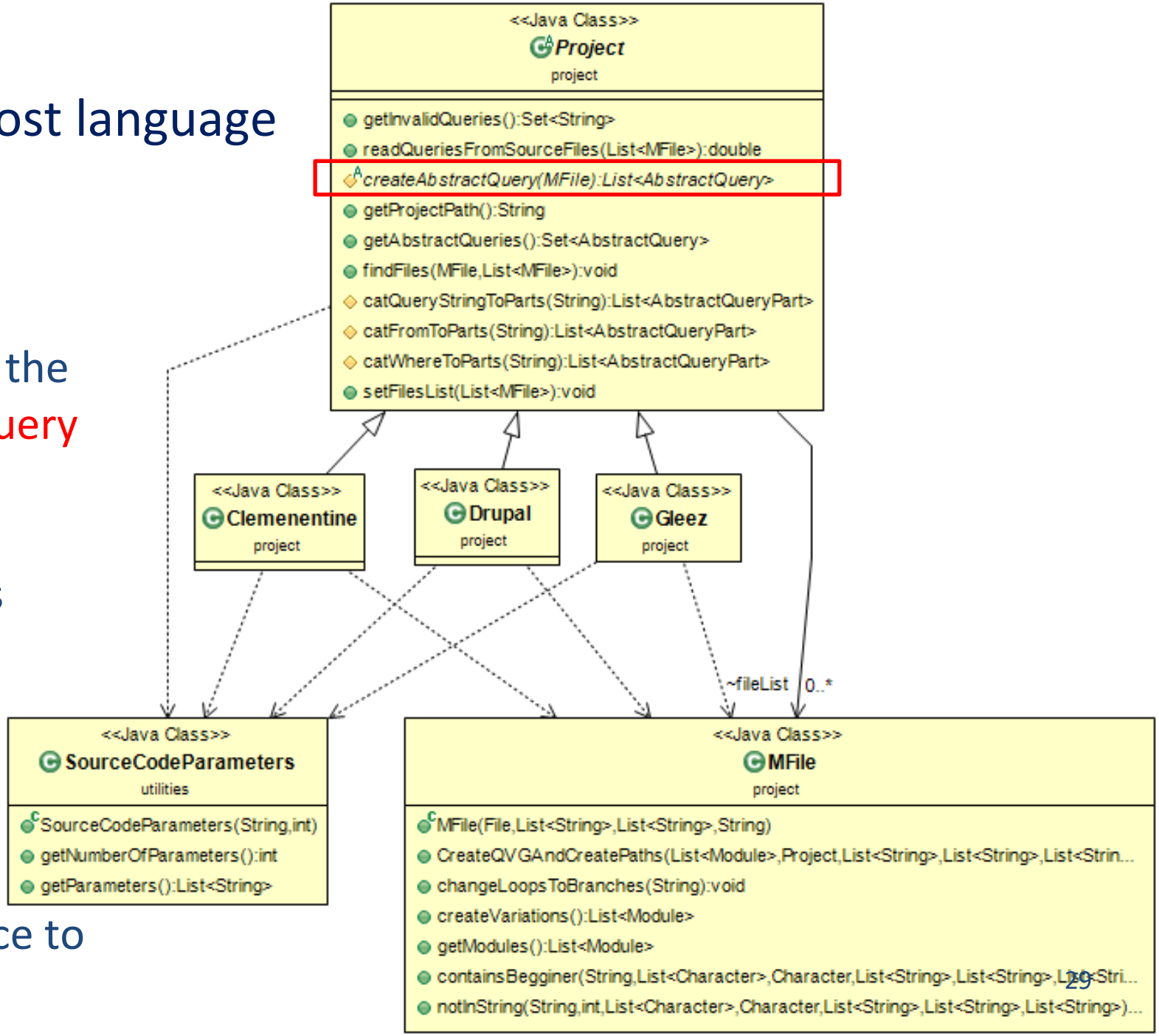
Source	Describes a provider of information in a query (e.g., a table in SQL).
Projector	Describes an output attribute (e.g., the SELECT attributes in SQL).
Comparator	Describes a filter that the output of the query should fulfil (e.g., the conditions of the WHERE clause in SQL).
Grouper	Used for summarizing of the output (used for grouping the incoming data in groups, each group identified by a unique combination of grouper values, e.g., the attributes of the GROUP BY clause in SQL).
Ordering	Used for sorting of the output (e.g., the attributes of the ORDER BY clause in SQL).
Limiter	Used for restricting the size of the output (e.g., the TOP/LIMIT clauses of an SQL query)
Aggregator	Used for applying an aggregate function to a input attributes (e.g., the MIN, MAX, COUNT, SUM, AVG functions in SQL)

Mapping of host language to ADMO

Per project:
Must materialize the CreateAbstractQuery method!

Creation includes cases of diff. #params, string manipulation, …

See the exact price to pay later…



Algorithm to transform host language statements to ADMO

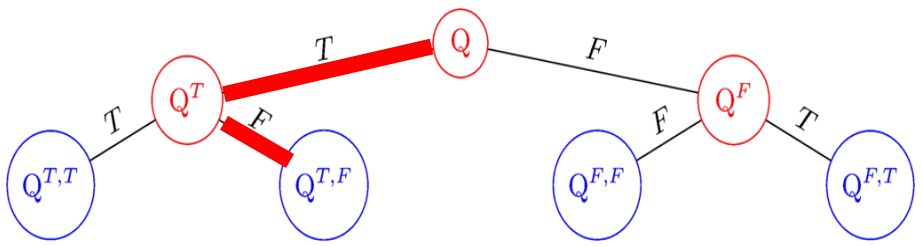
Input: A QVG path of a Callable Unit (P), a mapping (M) of the API functions to ADMOs
Output: The Abstract Query Representation of P.

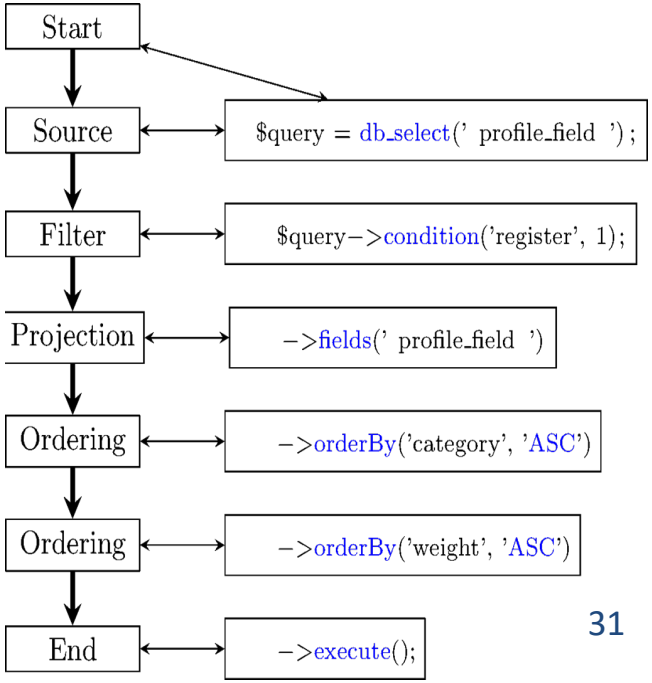
```
1 Add Start node for AQR ;
2 foreach QVGNode N ∈ P.nodes do
3     functionsOfNode = split contents of N to its functions;
4     foreach F ∈ functionsOfNode do
5         FAMDOs = M(F);            ▷ Find the ADMO nodes for function F
6         foreach fadmo ∈ FAMDOs do
7             Set function's F parameters to fadmo's ADMO parameters;
8             Add fadmo to AQR ;
          end
      end
  end
9 Add End node for Abstract Query Representation ;
```

Abstract Query Representation of a host path

(a sequence of ADMO operators to which individual host method calls are translated)

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {$query->condition('register', 1);}
  else {$query->condition('category', db_like($category), 'LIKE');}
  if (!user_access('administer users')) {$query->condition('visibility', PROFILE_HIDDEN, '<>');}
  return $query->fields('profile_field')
               ->orderBy('category', 'ASC')
               ->orderBy('weight', 'ASC')
               ->execute();
}
```





Roadmap

Input: Source files + keywords

Concrete source code representation

Abstract source code representation

Abstract Query Representation

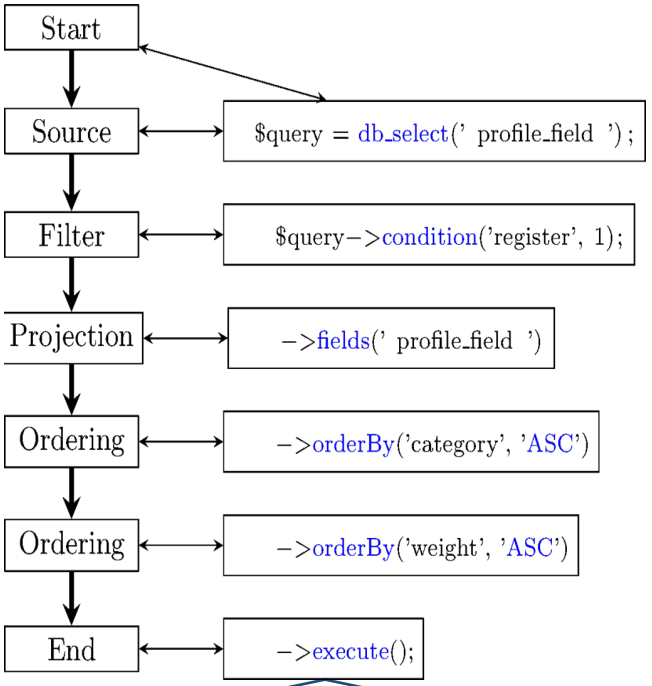
Output: Concrete, target query representation

Input: Project's folder pf, Files to exclude $excl$, Beginners $begins$, API functions $apiF$, Language specifics $lspecs$
Output: The Abstract Query Representations of the project.

```
1  paths = ∅;
2  files2search = Recursively search the project's folder;
3  foreach file f : files2search do
4  |  if f ∈ excl then
5  |  |  files2search− = f;                    ▷ Remove files w/o queries
   |  end
   end
6  CallableUnits = split file into Callable Units;
7  foreach CallableUnit ∈ CallableUnits do
8  |  QVGs += create a Query Variants Graph (QVG) per Callable Unit;
   end
9  foreach QVG ∈ QVGs do
10 |  paths+ = to transform each QVG to a set of QVG paths;
   end
11 foreach path ∈ paths do
12 |  AQRs+ = extract the embedded queries into an Abstract Query
   |  Representation (AQR);
   end
13 foreach AQR ∈ AQRs do
14 |  export AQR to a target language;
   end
```

AQR to target

Translate each ADMO operator to a part of a "query statement" depending on the target language



SQL

```
SELECT profile_field.*
FROM profile_field
WHERE register = 1
ORDER BY category ASC, weight ASC
;
```

MongoDB

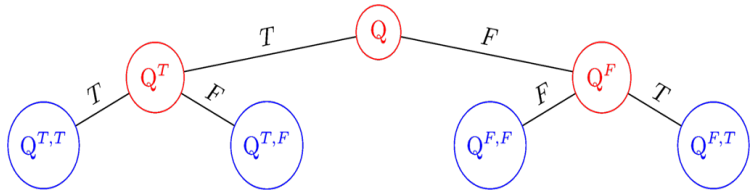
```
db.profile_field.aggregate
([
  { $match: {register: { $eg: 1 } } },
  { $sort: { category: 1, weight: 1 } },
  { profile_field.*: 1 }
])
```

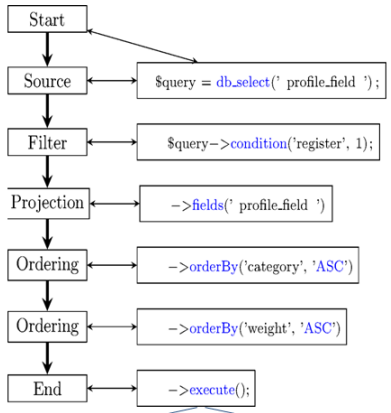

http://www.cs.uoi.gr/~pmanousi/publications/queryExtraction/

Roadmap

1. Overview
2. A method for Embedded Query Extraction
3. **Experiments**
4. Discussion

```php
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {
        $query->condition('register', 1);
  }
  else {
        $query->condition('category', db_like($category), 'LIKE');
  }
  if    (!user_access('administer users')) {
        $query->condition('visibility', PROFILE_HIDDEN, '<>');
  }
  return $query->fields('profile_field')
            ->orderBy('category', 'ASC')
            ->orderBy('weight', 'ASC')
            ->execute();
}
```






```sql
SELECT profile_field.*
FROM profile_field
WHERE register = 1
ORDER BY category ASC, weight ASC
```

```
db.profile_field.aggregate
([
  { $match: {register: { $eq: 1 } } },
  { $sort: { category: 1, weight: 1 } },
  { profile_field.*: 1 }
])
```

Experimental setup

Two projects analyzed:

- Clementine (a music player written in C++)
- Drupal (a CMS written in PHP)

Metrics:

Project	Lines of code	Files	Sub-folders	Variant queries	Fixed queries	Total
Clementine	210053	3072	159	10	14	24
Drupal	325421	1096	137	10	84	94

Recall: the fraction of the retrieved queries of each le over the actually existing ones.

Correctness: the fraction of the correctly reconstructed queries over the retrieved ones. “Correctly” = (a) retrieving *all* its structural parts + (b) assembling them as originally intended

Recall

Ideally, to assess **recall**, we need to manually verify the percentage of queries that our method extracts with respect to the queries that actually exist in the code.

Due to the vastness of the task, we have sampled the 10% of the database-related files.

Also, we performed automated searches based on the prescription of the project's manual, on how queries are authored in the code.

- **We manually inspected the code of the evaluated files and we were unable to find any other query, besides the ones that our tool reported.**
- **Similarly, all automated searches failed to produce any misses**

We claim **100% recall** based on the above.

Correctness

Used a classification of queries wrt their structure

1. *Fixed structure:* query structure not altered by host variables

   (1a) *All parts fixed: queries that have no variable at all*

   (1b) *Variable values in filtering:* queries that contain a variable that gets its value at execution time but does not intervene with the query structure. Typically, in a selection predicate.

2. *Variable structure:* query structure is defined at runtime via host variables. Typically occurs at FROM clause (!)

Correctness

Achieved

	Query class	Drupal-7.39	Clementine 1.2.3
Valid:	*all* parts fixed	28/94 (*29.7%*)	5/24 (*20.8%*)
Valid:	variable *values*	61/94 (*64.9%*)	14/24 (*58.3%*)
	overall	89/94 (**95.6%**)	19/24 (**79.1%**)
Invalid:	variable *structure*	05/94 (*04.4%*)	05/24 (*20.9%*)

100% correctness

Currently, we fail to handle them

Percentages of queries per category for the two data sets

User effort

There is a preprocessing step to translate the projects' API database-related functions to Abstract Data Manipulation Operators.

Project	API func./LOC	Host lang. (func./LOC)	Method fixed input
Clementine (C++)	4/59	9/341	11
Drupal (PHP)	11/251	9/347	11

Effort is measured with

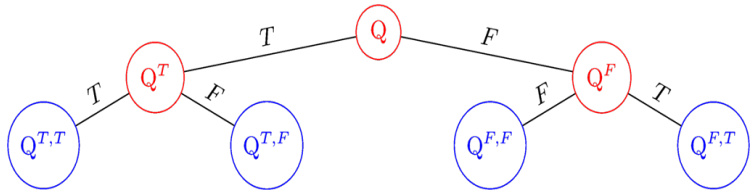
(a) the number of functions that needed translation from the project's API,

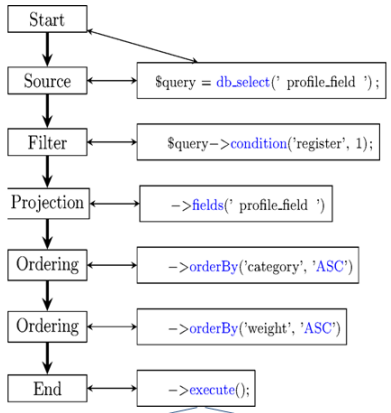
(b) the lines of code that were written for the translation of those API functions to Abstract Data Manipulation Operators.

Roadmap

1. Overview
2. A method for Embedded Query Extraction
3. Experiments
4. **Discussion**

```php
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {
        $query->condition('register', 1);
  }
  else {
        $query->condition('category', db_like($category), 'LIKE');
  }
  if    (!user_access('administer users')) {
        $query->condition('visibility', PROFILE_HIDDEN, '<>');
  }
  return $query->fields('profile_field')
            ->orderBy('category', 'ASC')
            ->orderBy('weight', 'ASC')
            ->execute();
}
```






```sql
SELECT profile_field.*
FROM profile_field
WHERE register = 1
ORDER BY category ASC, weight ASC
```

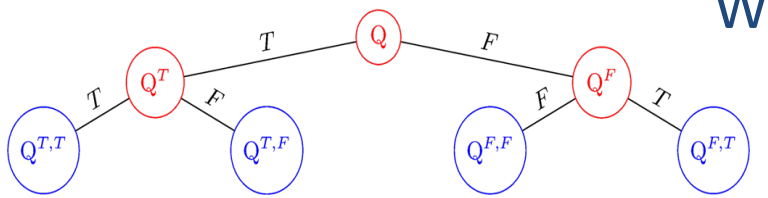
```
db.profile_field.aggregate
([
  { $match: {register: { $eq: 1 } } },
  { $sort: { category: 1, weight: 1 } },
  { profile_field.*: 1 }
])
```

We present a method to extract embedded queries from host code!

1

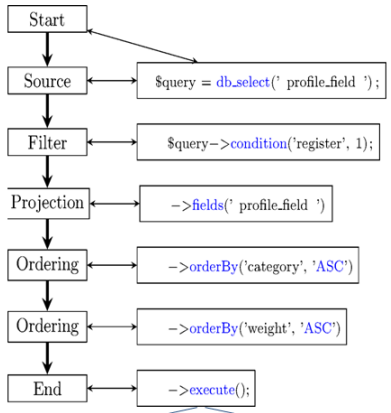
… and in a way that is simple and understandable

```
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {
        $query->condition('register', 1);
  }
  else {
        $query->condition('category', db_like($category), 'LIKE');
  }
  if    (!user_access('administer users')) {
        $query->condition('visibility', PROFILE_HIDDEN, '<>');
  }
  return $query->fields('profile_field')
            ->orderBy('category', 'ASC')
            ->orderBy('weight', 'ASC')
            ->execute();
}
```



2

We can also **represent interim structures** in a principled & meaningful way!




```
SELECT profile_field.*
FROM profile_field
WHERE register = 1
ORDER BY category ASC, weight ASC
```

```
db.profile_field.aggregate
([
  { $match: {register: { $eq: 1 } } },
  { $sort: { category: 1, weight: 1 } },
  { profile_field.*: 1 }
])
```

3

We can envision a generic, all-encompassing, language independent **representation of data retrieval** that treats queries as **workflows of data transformation Lego blocks**

Open Issues

Work more on ADMO

Broaden the support for more host languages

Test with more systems

Broaden the query class to incorporate more flexible query structures

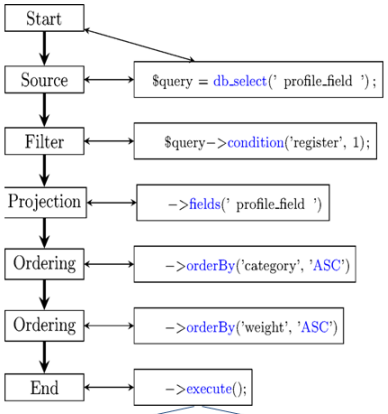
Improve the efficiency of the algorithms to gain memory (!) and time

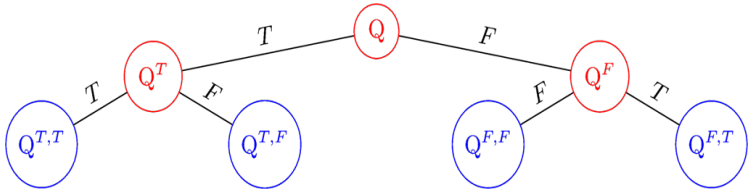
Danke schön!
Thank you!

```php
function _profile_get_fields($category, $register = FALSE) {
  $query = db_select('profile_field');
  if ($register) {
        $query->condition('register', 1);
  }
  else {
        $query->condition('category', db_like($category), 'LIKE');
  }
  if    (!user_access('administer users')) {
        $query->condition('visibility', PROFILE_HIDDEN, '<>');
  }
  return $query->fields('profile_field')
            ->orderBy('category', 'ASC')
            ->orderBy('weight', 'ASC')
            ->execute();
}
```

We present a method to extract embedded queries from host code!

… and in a way that is simple and understandable





We can also **represent interim structures** in a principled & meaningful way!

```sql
SELECT profile_field.*
FROM profile_field
WHERE register = 1
ORDER BY category ASC, weight ASC
```

```
db.profile_field.aggregate
([
  { $match: {register: { $eq: 1 } } },
  { $sort: { category: 1, weight: 1 } },
  { profile_field.*: 1 }
])
```

We can envision a generic, all-encompassing, language independent **representation of data retrieval** that treats queries as **workflows of data transformation Lego blocks**

http://www.cs.uoi.gr/~pmanousi/publications/2017_CAiSE/