DOCUMENTING DATABASE USAGES AND SCHEMA CONSTRAINTS IN DATABASE-CENTRIC APPLICATIONS

ISSTA'16

Mario Linares-Vásquez, Boyang Li,
Christopher Vendome, Denys Poshyvanyk

Database-Centric Applications (DCAs)

Source code

- High level features require different database operations

- SQL-style operations

- API calls using de-facto libraries or ORM frameworks

Database-Centric Applications (DCAs)

Source code

- High level features require different database operations
- SQL-style operations
- API calls using de-facto libraries or ORM frameworks

Database

- Tables, columns, types, constraints
- Underlying domain model of DCAs, including business rules and terms

Some challenges in DCAs

- DB schema and source code evolves collaterally (i.e., asynchronously)

- Schema changes have significant impact on DCAs' code

Understanding database schema evolution: A case study

Anthony Cleve[a,*], Maxime Gobert[a], Loup Meurice[a], Jerome Maes[a], Jens Weber[b]

[a] University of Namur, Belgium
[b] University of Victoria, Canada

Co-evolving Code-Related and Database-Related Changes in a Data-Intensive Software System

Mathieu Goeminne, Alexandre Decan, Tom Mens
Département Informatique, Université de Mons
7000 Mons, Belgique
firstname.lastname@umons.ac.be

Collateral Evolution of Applications and Databases

Dien-Yen Lin Iulian Neamtiu
University of California, Riverside
Riverside, CA 92521, USA
{dienyen,neamtiu}@cs.ucr.edu

An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications

Dong Qiu Bixin Li
Southeast University, China
{dongqiu, bx.li}@seu.edu.cn

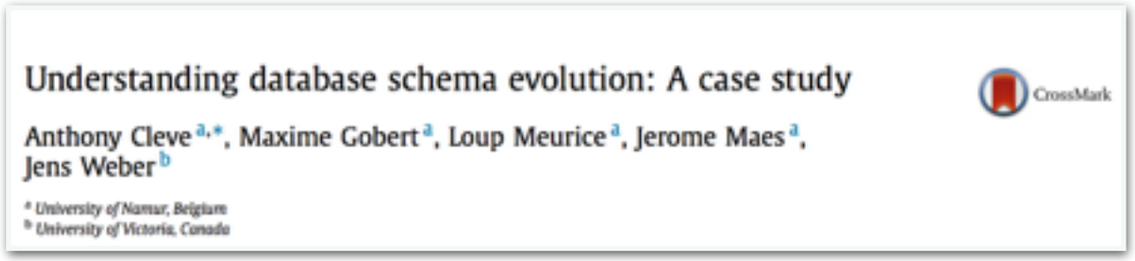
Zhendong Su
University of California, Davis, USA
su@cs.ucdavis.edu

Impact Analysis of Database Schema Changes

Andy Maule, Wolfgang Emmerich and David S. Rosenblum
London Software Systems
Dept. of Computer Science, University College London
Gower Street, London WC1E 6BT, UK
{a.maule|w.emmerich|d.rosenblum}@cs.ucl.ac.uk

Some challenges in DCAs

- Lack of usage of referential integrity in schemas impact the understanding of the schemas



Understanding database schema evolution: A case study

Anthony Cleve[a,*], Maxime Gobert[a], Loup Meurice[a], Jerome Maes[a], Jens Weber[b]

[a] University of Namur, Belgium
[b] University of Victoria, Canada

- Tracing DB schema constraints along source code method call-chains is a “moderate” or a “very hard” challenge

How Do Developers Document Database Usages in Source Code?

Mario Linares-Vásquez, Boyang Li, Christopher Vendome, and Denys Poshyvanyk
The College of William and Mary
Email: {mlinarev, boyang, cvendome, denys}@cs.wm.edu

Example: Xinco

- Document Management System
- In active development since 2004
- J2EE Architecture + Web Services
- 23 tables and 135 attributes
- Referential integrity

https://sourceforge.net/p/xinco/

Example: com.bluecubs.xinco.core.server.XincoCoreNodeServer.deleteFromDB

```
public void deleteFromDB(boolean delete_this, XincoDBManager DBM,int userID)
                throws XincoException {
      int i=0;
      try {
          Statement stmt;
          fillXincoCoreNodes(DBM);
          fillXincoCoreData(DBM);
          for (i=0;i<getXinco_core_nodes().size();i++) {
              ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i))
              .deleteFromDB(true, DBM,userID);
          }
          for (i=0;i<getXinco_core_data().size();i++) {
              XincoIndexer.removeXincoCoreData([...]);
              XincoCoreDataServer.removeFromDB([...]);
              [...]
          }
          if (delete_this) {
              XincoCoreAuditServer audit= new XincoCoreAuditServer();
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE
                                  xinco_core_node_id=" + getId());
              stmt.close();
              audit.updateAuditTrail("xinco_core_node",new String [] {"id ="+getId()},
                      DBM,"audit.general.delete",userID);
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
              stmt.close();
          }
          DBM.con.commit();
      } catch (Exception e) {
          [...]
      }
  }
```

Example: com.bluecubs.xinco.core.server.XincoCoreNodeServer.deleteFromDB

```
public void deleteFromDB(boolean delete_this, XincoDBManager DBM,int userID)
                throws XincoException {
      int i=0;
      try {
          Statement stmt;
          fillXincoCoreNodes(DBM);
          fillXincoCoreData(DBM);
          for (i=0;i<getXinco_core_nodes().size();i++) {
              ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i))
              .deleteFromDB(true, DBM,userID);
          }
          for (i=0;i<getXinco_core_data().size();i++) {
              XincoIndexer.removeXincoCoreData([...]);
              XincoCoreDataServer.removeFromDB([...]);
              [...]
          }
          if (delete_this) {
              XincoCoreAuditServer audit= new XincoCoreAuditServer();
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE
                                  xinco_core_node_id=" + getId());
              stmt.close();
              audit.updateAuditTrail("xinco_core_node",new String [] {"id ="+getId()},
                      DBM,"audit.general.delete",userID);
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
              stmt.close();
          }
          DBM.con.commit();
      } catch (Exception e) {
          [...]
      }
  }
```

2 database operations

2 tables

Example: com.bluecubs.xinco.core.server.XincoCoreNodeServer.deleteFromDB

```
public void deleteFromDB(boolean delete_this, XincoDBManager DBM,int userID)
                throws XincoException {
      int i=0;
      try {
          Statement stmt;
          fillXincoCoreNodes(DBM);
          fillXincoCoreData(DBM);
          for (i=0;i<getXinco_core_nodes().size();i++) {
              ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i))
              .deleteFromDB(true, DBM,userID);
          }
          for (i=0;i<getXinco_core_data().size();i++) {
              XincoIndexer.removeXincoCoreData([...]);
              XincoCoreDataServer.removeFromDB([...]);
              [...]
          }
          if (delete_this) {
              XincoCoreAuditServer audit= new XincoCoreAuditServer();
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE
                                  xinco_core_node_id=" + getId());
              stmt.close();
              audit.updateAuditTrail("xinco_core_node",new String [] {"id ="+getId()},
                      DBM,"audit.general.delete",userID);
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
              stmt.close();
          }
          DBM.con.commit();
      } catch (Exception e) {
          [...]
      }
  }
```

2 database operations

2 tables

---

14 database operations

2 UPDATE, 4 SELECT, 2 INSERT, 6 DELETE

Example: com.bluecubs.xinco.core.server.XincoCoreNodeServer.deleteFromDB

```
public void deleteFromDB(boolean delete_this, XincoDBManager DBM,int userID)
                throws XincoException {
      int i=0;
      try {
          Statement stmt;
          fillXincoCoreNodes(DBM);
          fillXincoCoreData(DBM);
          for (i=0;i<getXinco_core_nodes().size();i++) {
              ((XincoCoreNodeServer)getXinco_core_nodes().elementAt(i))
              .deleteFromDB(true, DBM,userID);
          }
          for (i=0;i<getXinco_core_data().size();i++) {
              XincoIndexer.removeXincoCoreData([...]);
              XincoCoreDataServer.removeFromDB([...]);
              [...]
          }
          if (delete_this) {
              XincoCoreAuditServer audit= new XincoCoreAuditServer();
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_ace WHERE
                                  xinco_core_node_id=" + getId());
              stmt.close();
              audit.updateAuditTrail("xinco_core_node",new String [] {"id ="+getId()},
                      DBM,"audit.general.delete",userID);
              stmt = DBM.con.createStatement();
              stmt.executeUpdate("DELETE FROM xinco_core_node WHERE id=" + getId());
              stmt.close();
          }
          DBM.con.commit();
      } catch (Exception e) {
          [...]
      }
  }
```

24 attributes with constraints

9 REFERENTIAL INTEGRITY

12 NOT NULL ATTRIBUTES

4 VARCHAR LIMITS

2 UNIQUE ATTRIBUTES

DBSCRIBE

*Image from: https://utopiaordystopia.com/2013/12/29/preparing-for-a-new-dark-age/monk-scribe/

Motivation

1. Understanding DB usages and schema constraints is crucial to understand how features are implemented

2. Manually documenting/investigating DB usages is time consuming

3. Less information about the DB can be inferred from source code methods at higher levels of the call-chains

DBScribe

1. **Updated documentation** at method-level

3. Textual description of **db-usages and related constraints**

3. **Local and delegated** db operations

4. **Automatic** documentation for different layers in a DCA

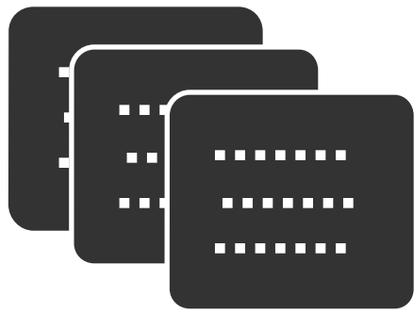
5. Combines **static-analysis** and **summarization techniques**

DBScribe

Source Code

1. SQL queries/ statements detection

2. Partial call graph extraction



Templates

DB

3. DB schema constraints extraction

4. DB usages and and constraints propagation

5. HTML description generation

1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```
public CourseSchedule(int offerID){
   try{
     Connection conn = Database.getConnection();
      String selectpart = "Select *”;
     String fromPart = " FROM courseschedule”
     String wherePart =" WHERE offerID= ?”;
     String scheduleSelect = select part + fromPart + wherePart;
     […]
     PreparedStatement statement = conn.prepareStatement(scheduleSelect);
     […]
     }catch(SQLException e){
          System.out.println("Error retrieving schedule”);
          System.out.println(e.getMessage());
          e.printStackTrace();
     }
}
```

VariablesMap

SQL-related calls

1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```
public CourseSchedule(int offerID){
   try{
      Connection conn = Database.getConnection();
      String selectpart = "Select *";
      String fromPart = " FROM courseschedule"
      String wherePart =" WHERE offerID= ?";
      String scheduleSelect = select part + fromPart + wherePart;
      […]
      PreparedStatement statement = conn.prepareStatement(scheduleSelect);
      […]
      }catch(SQLException e){
           System.out.println("Error retrieving schedule");
           System.out.println(e.getMessage());
           e.printStackTrace();
      }
}
```

VariablesMap

```
selectpart -> "Select *"
```

SQL-related calls

1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```
public CourseSchedule(int offerID){
   try{
      Connection conn = Database.getConnection();
      String selectpart = "Select *";
      String fromPart = " FROM courseschedule"
      String wherePart =" WHERE offerID= ?";
      String scheduleSelect = select part + fromPart + wherePart;
      […]
      PreparedStatement statement = conn.prepareStatement(scheduleSelect);
      […]
      }catch(SQLException e){
           System.out.println("Error retrieving schedule");
           System.out.println(e.getMessage());
           e.printStackTrace();
      }
}
```

VariablesMap

```
selectpart -> "Select *"

fromPart -> " FROM courseschedule"
```

SQL-related calls

1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```
public CourseSchedule(int offerID){
   try{
      Connection conn = Database.getConnection();
      String selectpart = "Select *";
      String fromPart = " FROM courseschedule"
      String wherePart =" WHERE offerID= ?";
      String scheduleSelect = select part + fromPart + wherePart;
      […]
      PreparedStatement statement = conn.prepareStatement(scheduleSelect);
      […]
      }catch(SQLException e){
           System.out.println("Error retrieving schedule");
           System.out.println(e.getMessage());
           e.printStackTrace();
      }
}
```

VariablesMap

```
selectpart -> "Select *"
fromPart -> " FROM courseschedule"
wherePart -> " WHERE offerID= ?"
```

SQL-related calls

1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```
public CourseSchedule(int offerID){
   try{
      Connection conn = Database.getConnection();
      String selectpart = "Select *";
      String fromPart = " FROM courseschedule"
      String wherePart =" WHERE offerID= ?";
      String scheduleSelect = select part + fromPart + wherePart;
      […]
      PreparedStatement statement = conn.prepareStatement(scheduleSelect);
      […]
      }catch(SQLException e){
           System.out.println("Error retrieving schedule");
           System.out.println(e.getMessage());
           e.printStackTrace();
      }
}
```

VariablesMap

```
selectpart -> "Select *"

fromPart -> " FROM courseschedule"

wherePart -> " WHERE offerID= ?"

scheduleSelect -> "Select * FROM
course schedule  WHERE offerID= ?"
```

SQL-related calls

1. SQL queries/statements detection

DBScribe finds API calls that execute SQL-statements, and the corresponding SQL literals

```
public CourseSchedule(int offerID){
   try{
      Connection conn = Database.getConnection();
      String selectpart = "Select *";
      String fromPart = " FROM courseschedule"
      String wherePart =" WHERE offerID= ?";
      String scheduleSelect = select part + fromPart + wherePart;
      […]
      PreparedStatement statement = conn.prepareStatement(scheduleSelect);
      […]
      }catch(SQLException e){
           System.out.println("Error retrieving schedule");
           System.out.println(e.getMessage());
           e.printStackTrace();
      }
}
```

VariablesMap

```
selectpart -> "Select *"

fromPart -> " FROM courseschedule"

wherePart -> " WHERE offerID= ?"

scheduleSelect -> "Select * FROM
course schedule  WHERE offerID= ?"
```

SQL-related calls

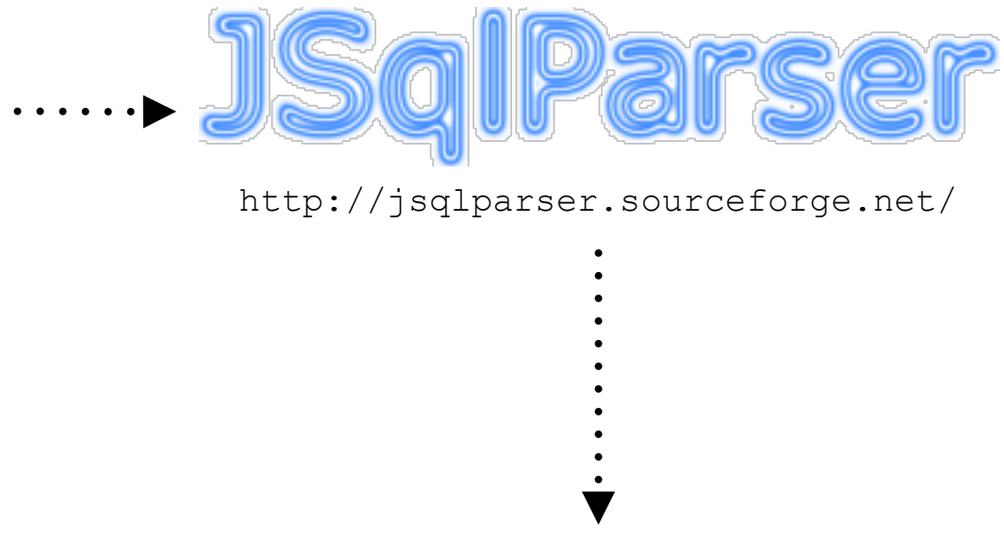
```
Select * FROM course schedule  WHERE
offerID= ?
```

1. SQL queries/statements detection

SQL-related calls in method *m*

```
Select * FROM course schedule  WHERE
offerID= ?


INSERT INTO investment VALUES
(ssn,capitalGains,capitalLosses,
stockDividend)
```



< literal$_1$, operation-type$_1$, tables$_1$, fields$_1$>

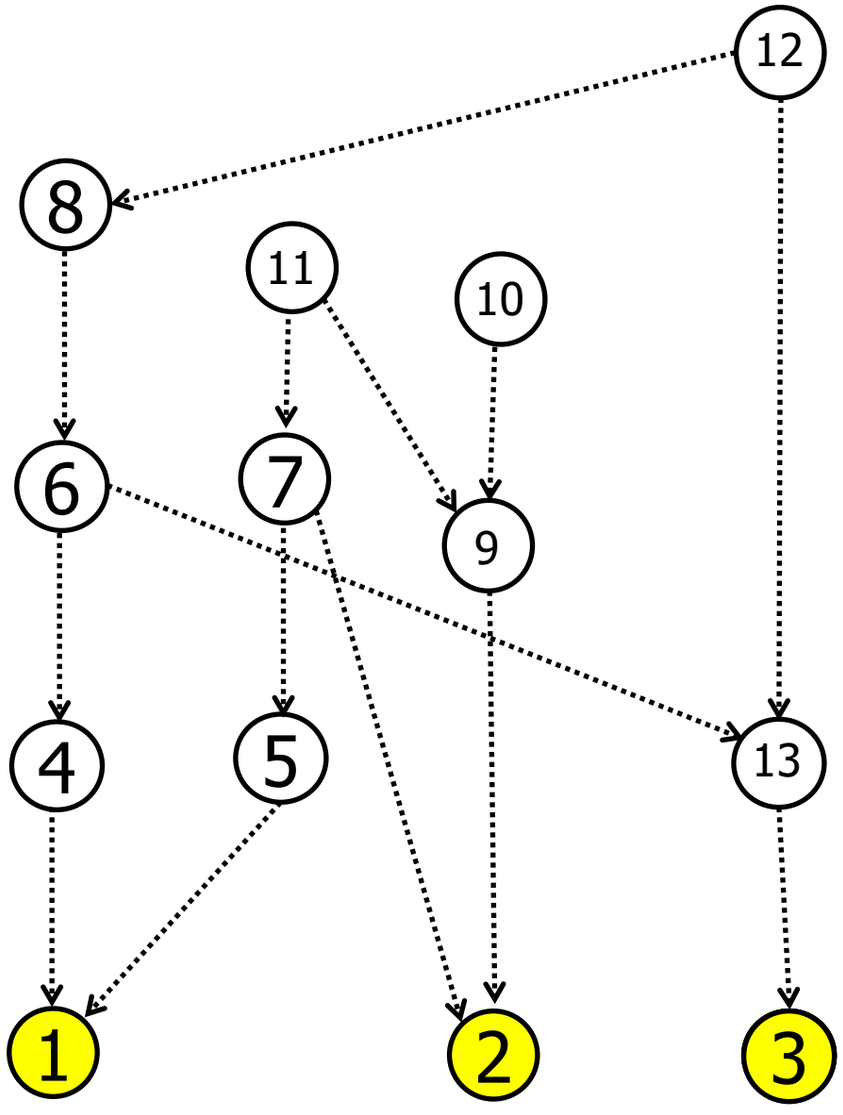
.

.

< literal$_n$, operation-type$_n$, tables$_n$, fields$_n$>

2. Partial call graph extraction

Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets



M={1,2,3}

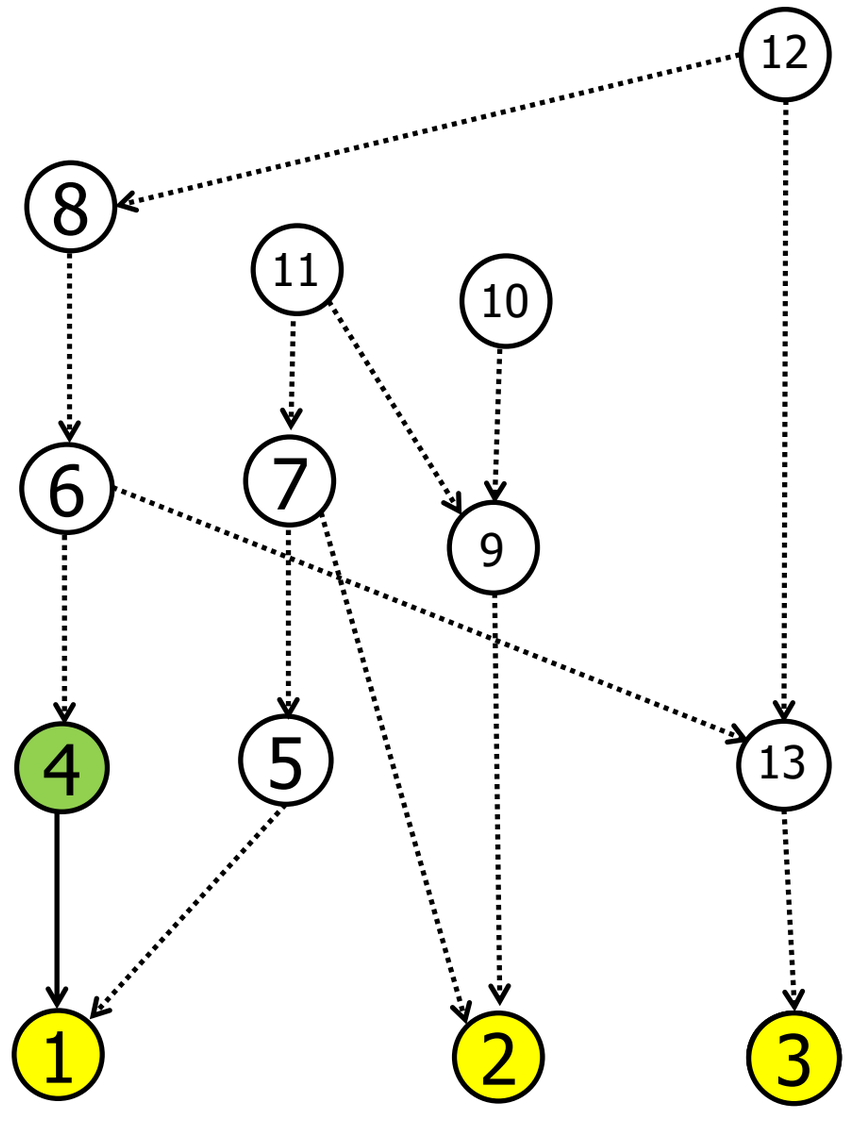
2. Partial call graph extraction

Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets

Call chains:

4 -> 1

M={1,2,3}



2. Partial call graph extraction

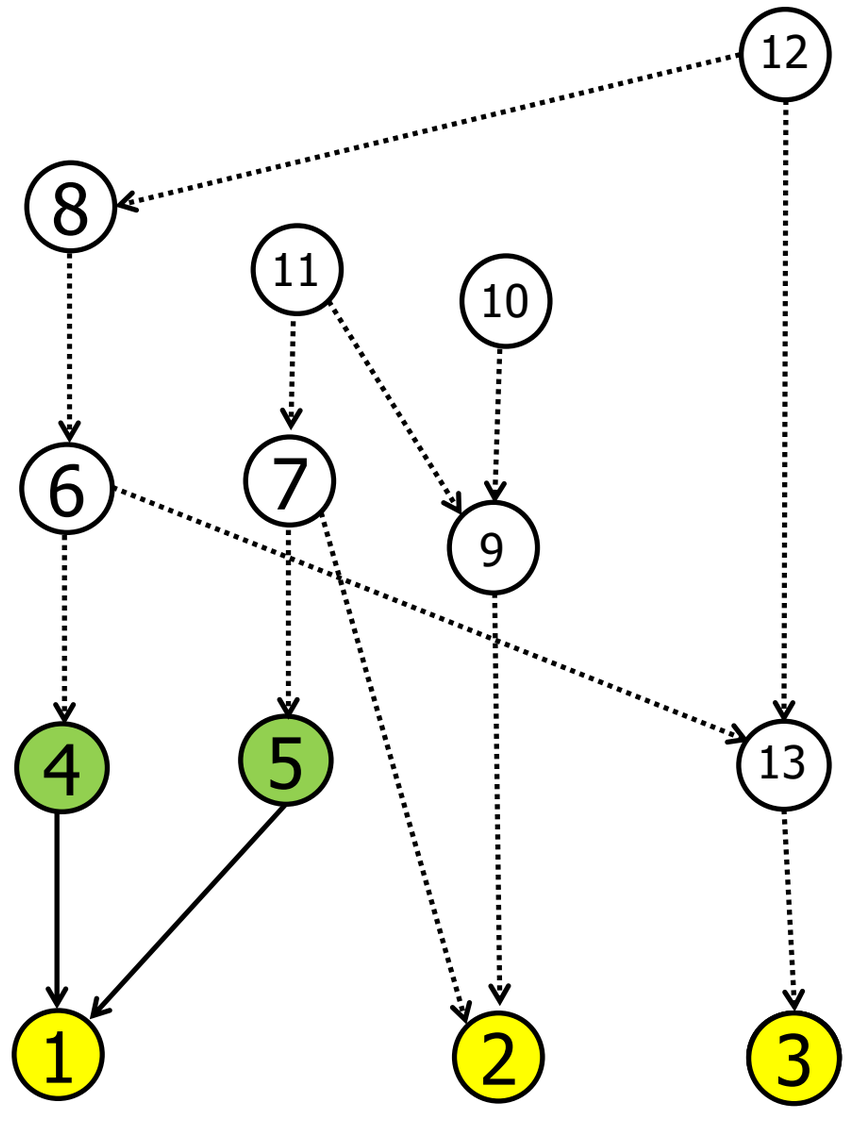
Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets

Call chains:

4 -> 1

5 -> 1

M={1,2,3}



2. Partial call graph extraction

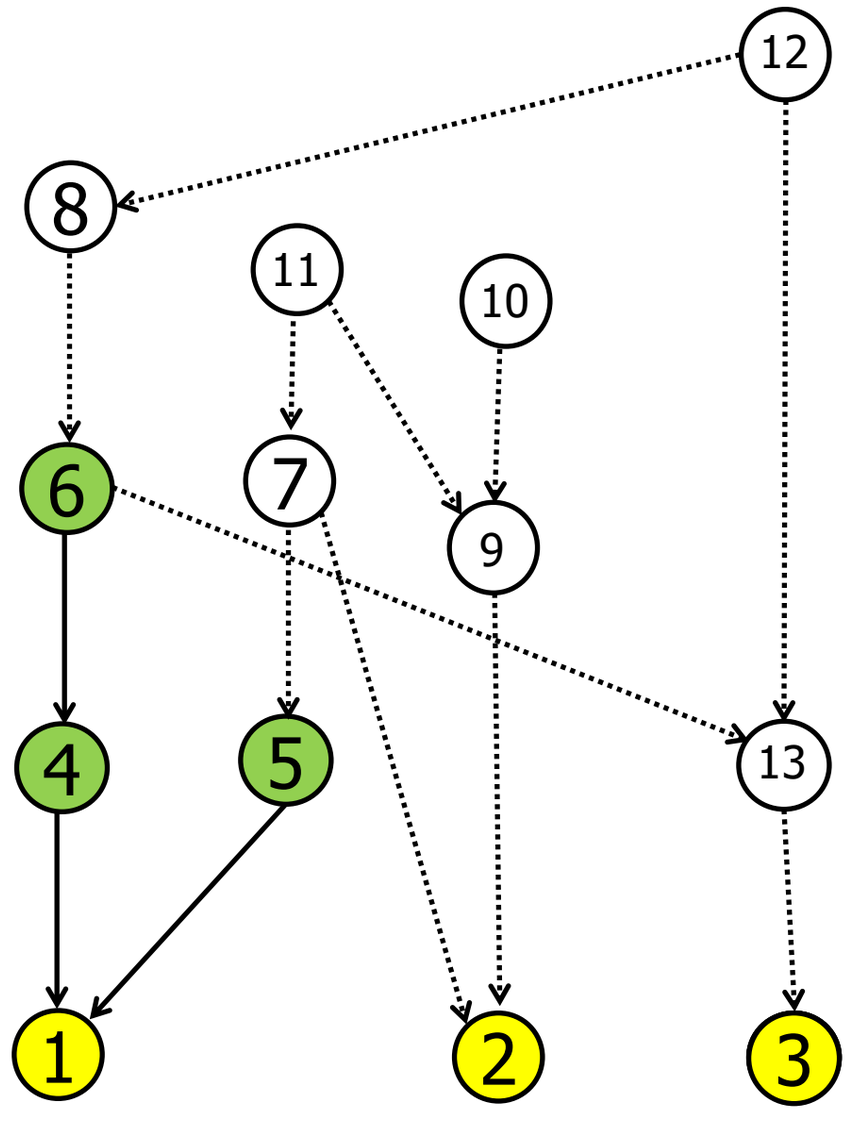
Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets

Call chains:

6 -> 4 -> 1

5 -> 1

M={1,2,3}



2. Partial call graph extraction

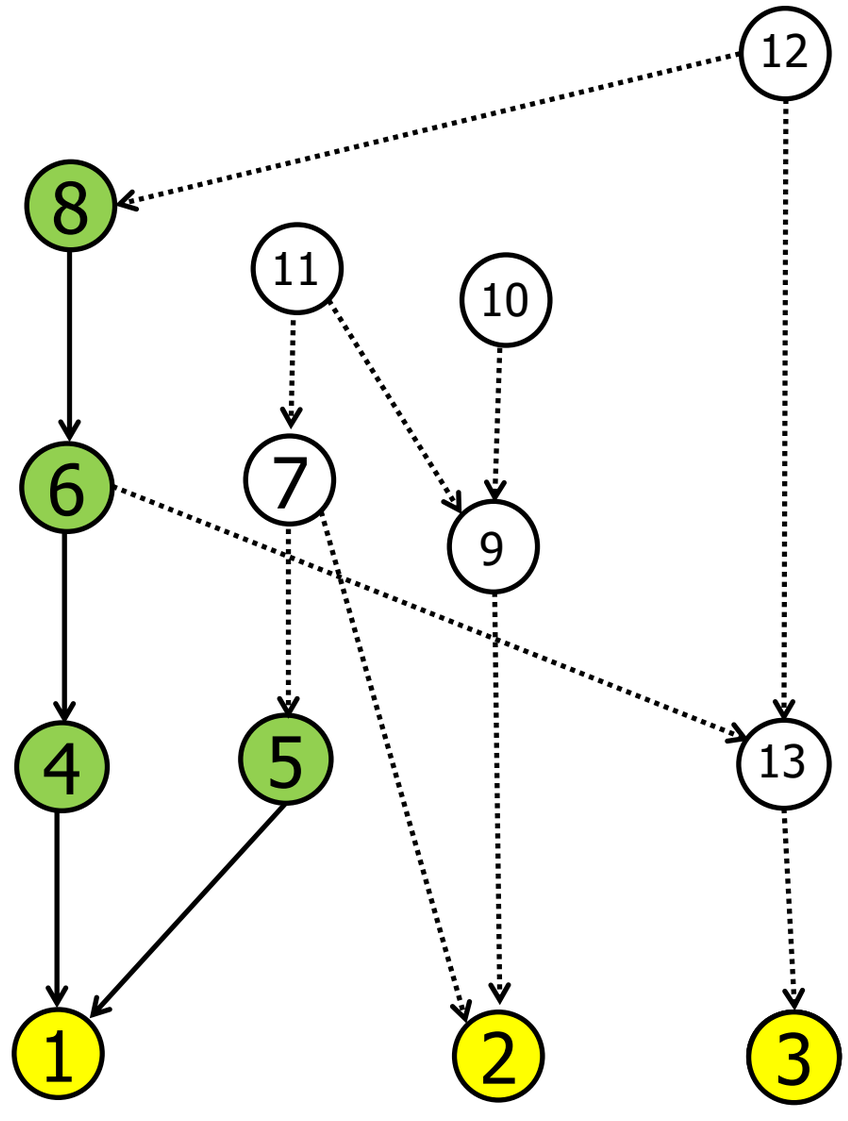
Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets

Call chains:

8-> 6 -> 4 -> 1

5 -> 1

M={1,2,3}



2. Partial call graph extraction

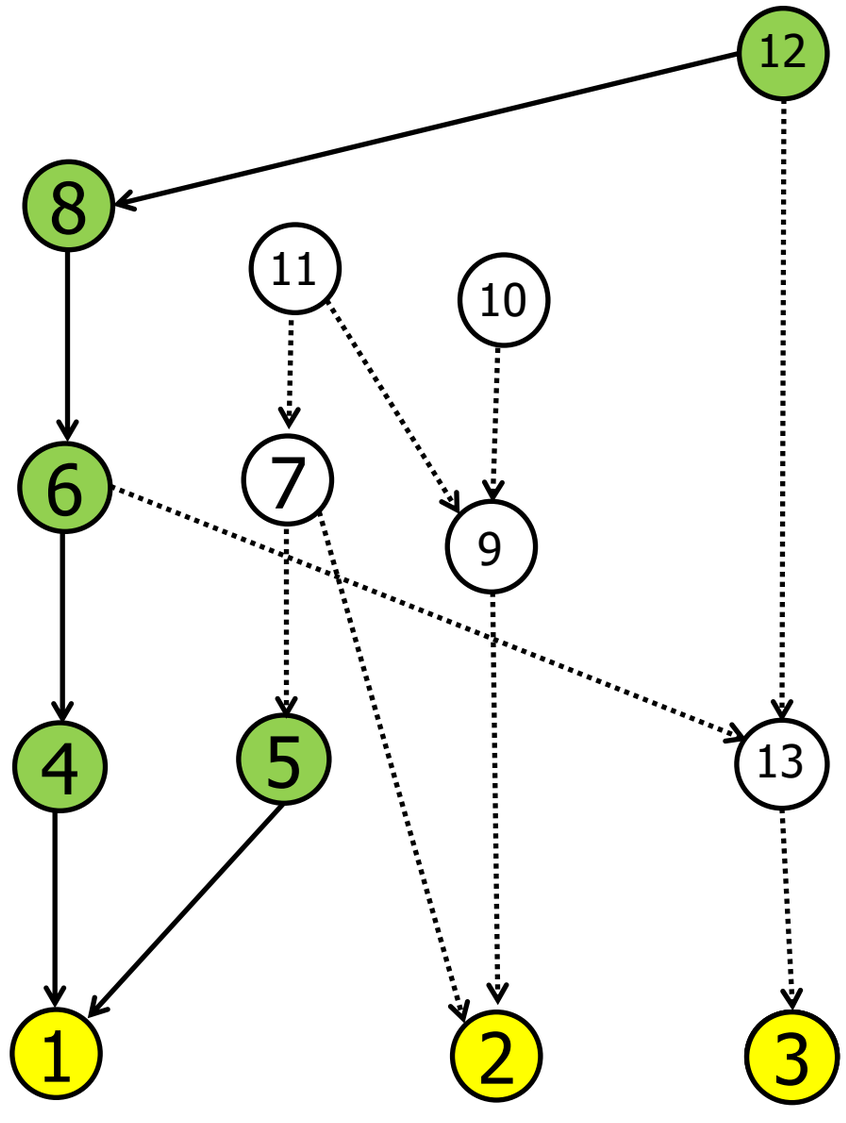
Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets

Call chains:

12 -> 8-> 6 -> 4 -> 1

5 -> 1

M={1,2,3}



2. Partial call graph extraction

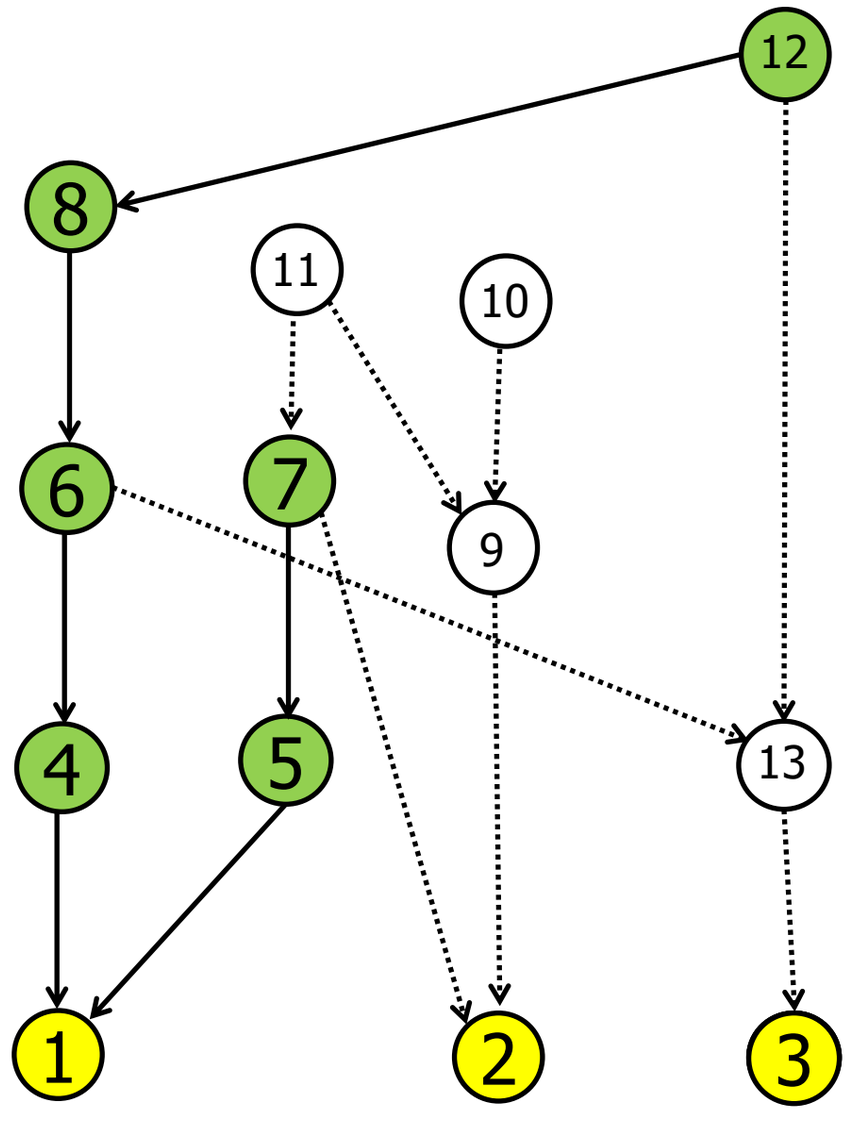
Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets

Call chains:

12 -> 8-> 6 -> 4 -> 1

7 -> 5 -> 1

M={1,2,3}



2. Partial call graph extraction

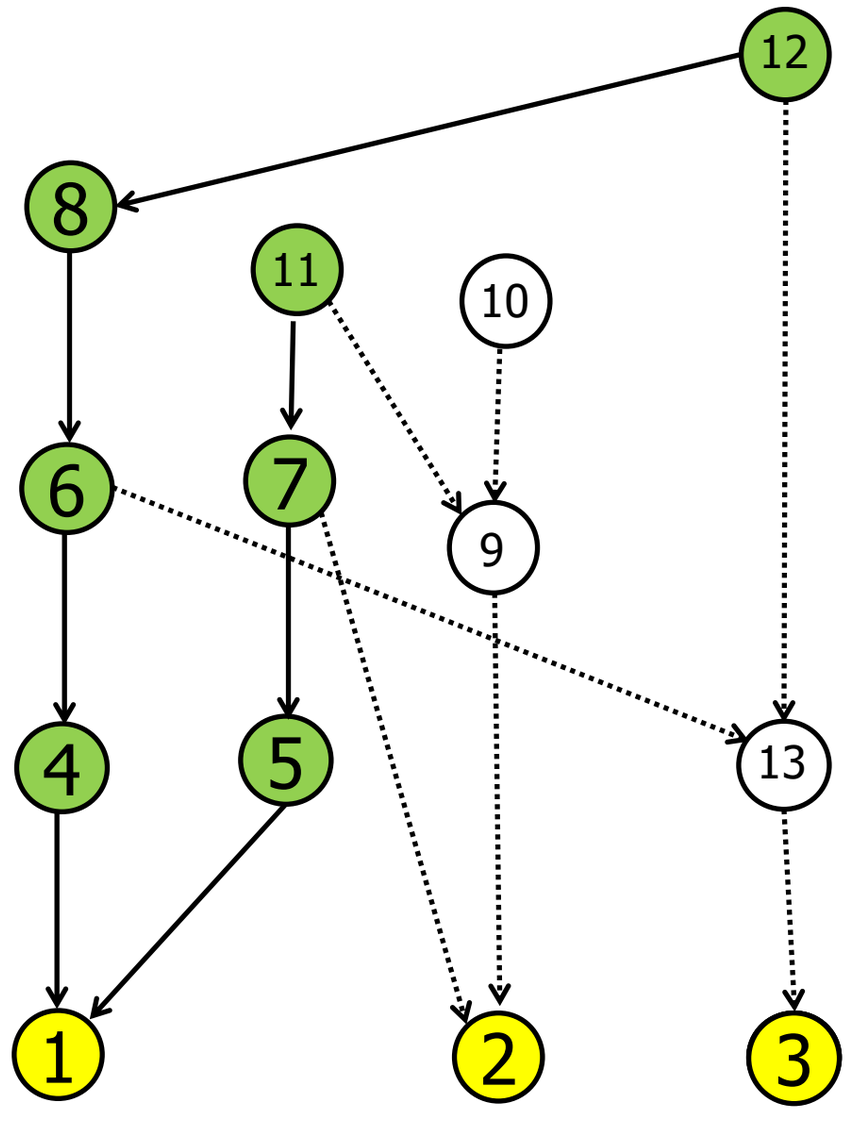
Given a set of methods ***M*** invoking SQL statements/ queries, DBScribe finds the set of call-chains that end at any method in ***M,*** based on the callers sets

Call chains:

12 -> 8-> 6 -> 4 -> 1

11 -> 7 -> 5 -> 1

M={1,2,3}



3. DB schema constraints extraction

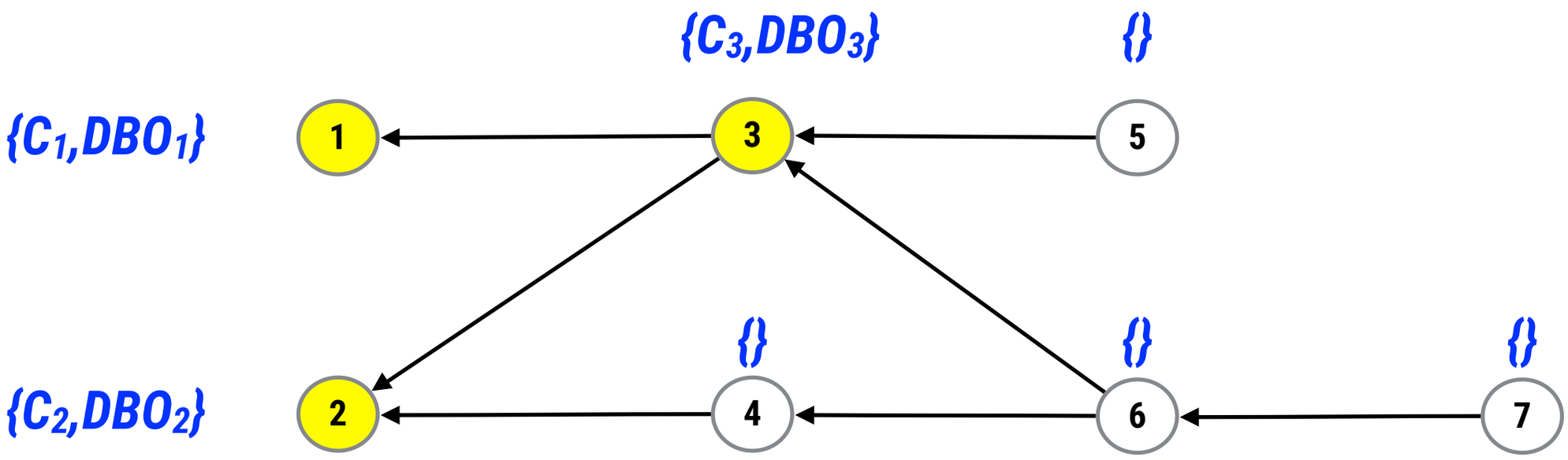
MASTER SCHEMA

$< table_1, attribute_1, constraint_1 >$
$< table_1, attribute_2, constraint_2 >$
.
$< table_i, attribute_j, constraint_k >$

- Auto-numeric columns
- Non-null columns
- Foreign keys
- Varchar limits
- Columns that should contain unique values

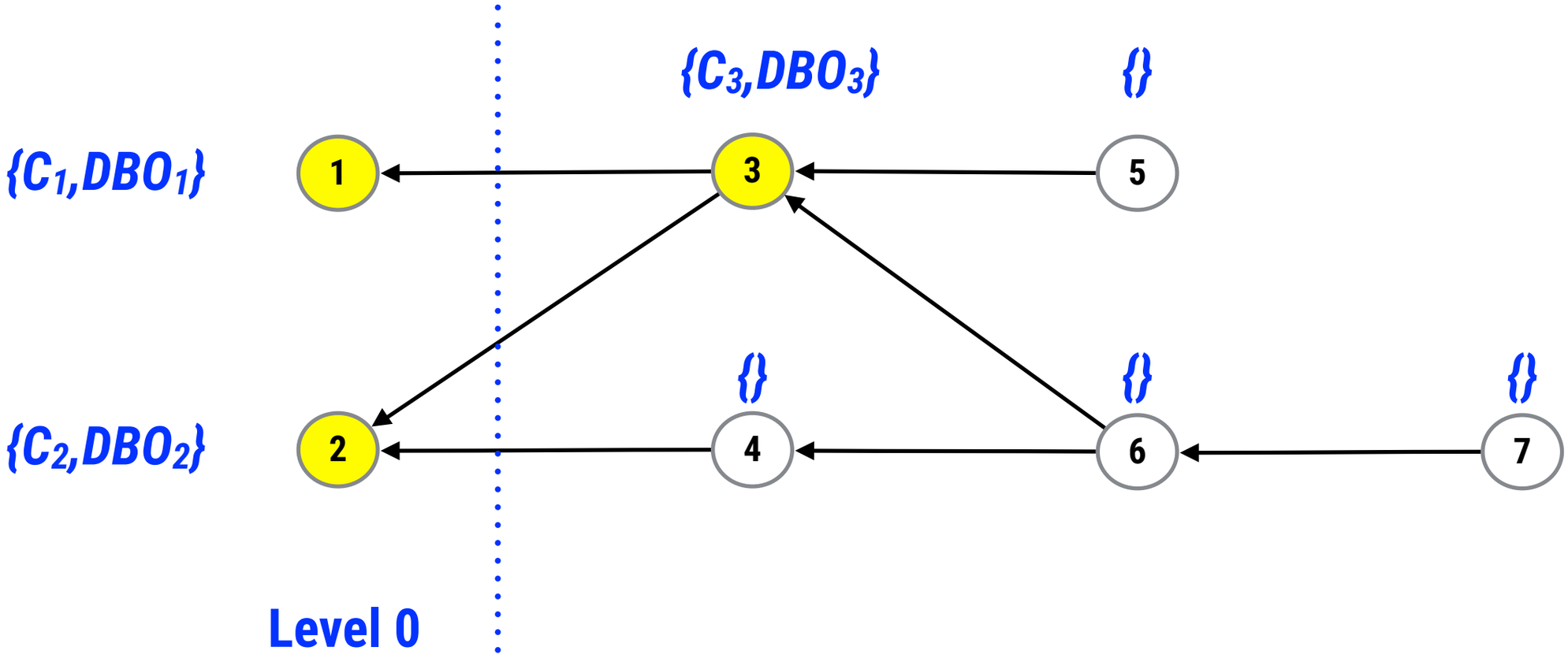
4. DB usages and constraints propagation

Given a set of ***constraints (C)*** and ***db operations (DBO)*** invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom



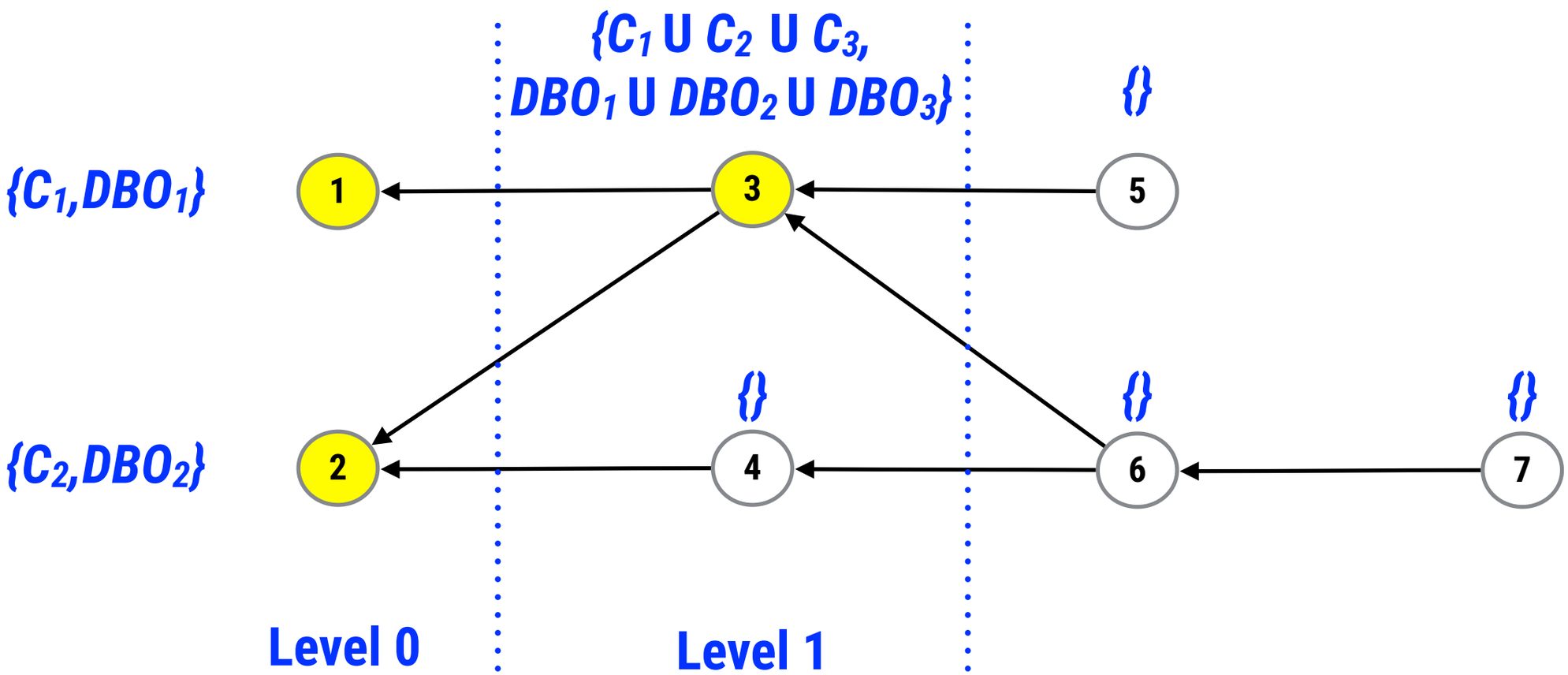
4. DB usages and constraints propagation

Given a set of ***constraints (C)*** and ***db operations (DBO)*** invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom



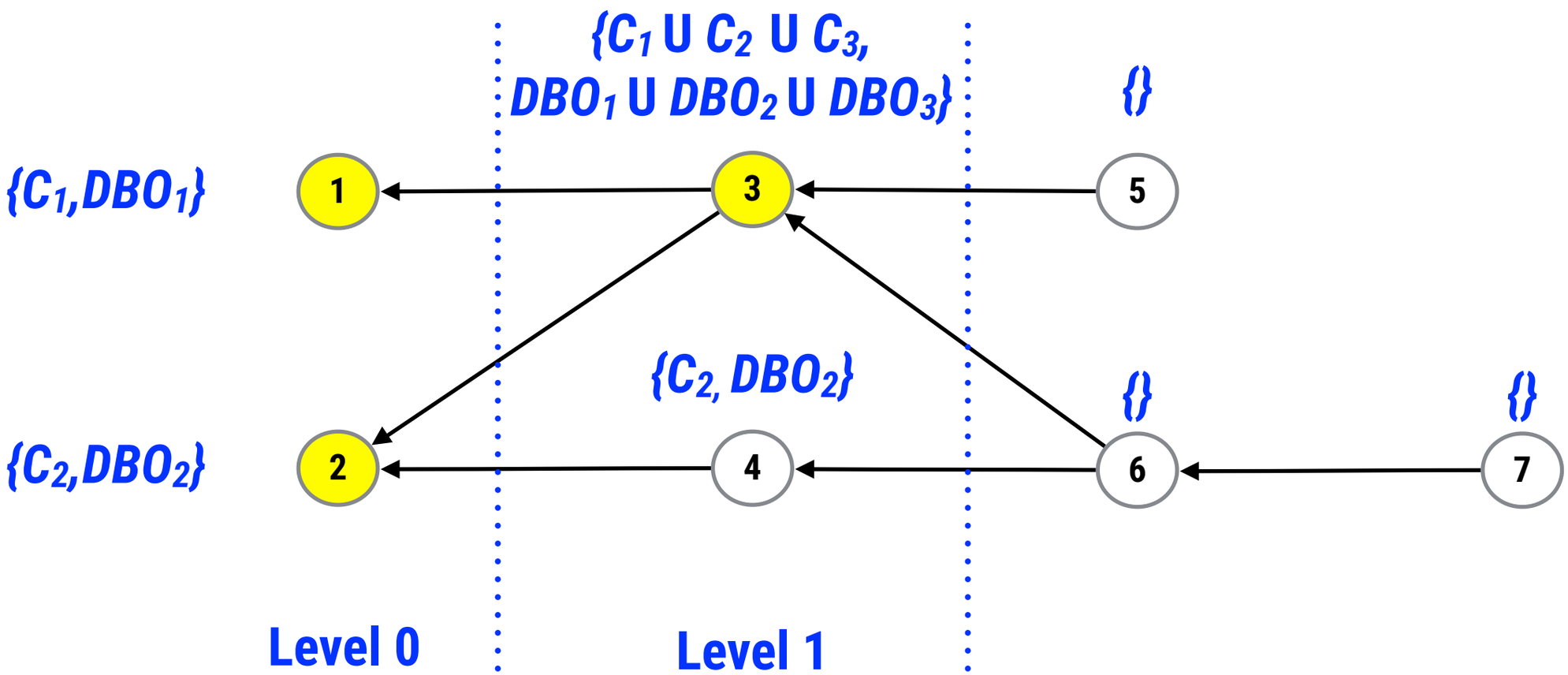
4. DB usages and constraints propagation

Given a set of ***constraints (C)*** and ***db operations (DBO)*** invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom



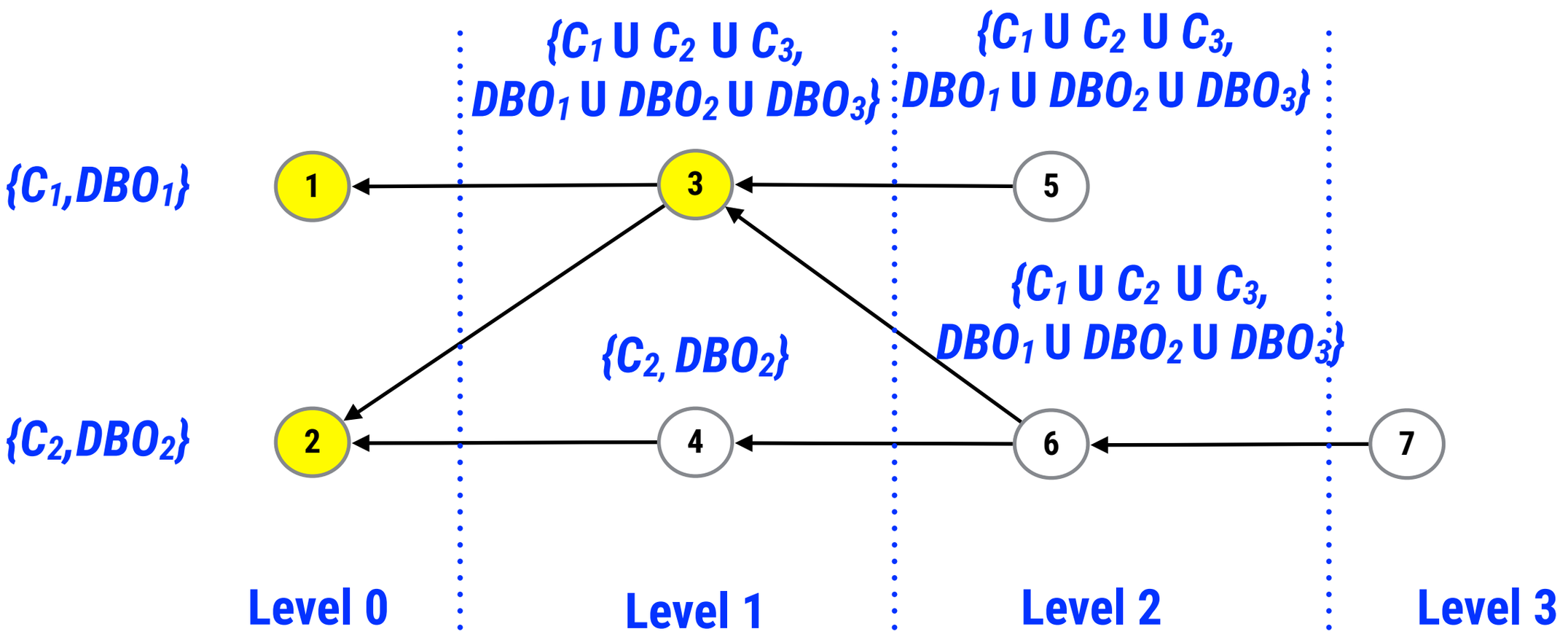
4. DB usages and constraints propagation

Given a set of ***constraints (C)*** and ***db operations (DBO)*** invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom

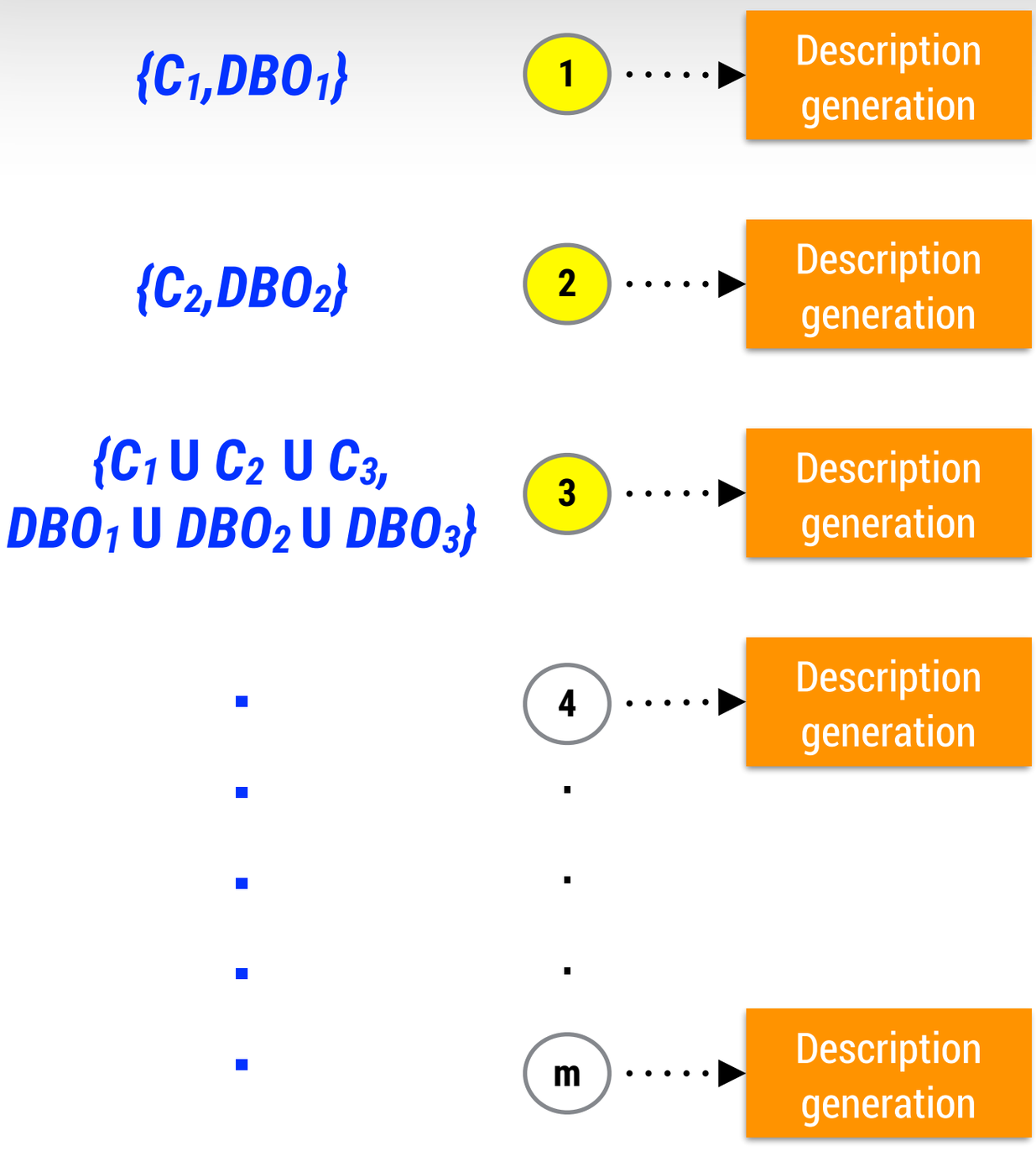


4. DB usages and constraints propagation

Given a set of ***constraints (C)*** and ***db operations (DBO)*** invoked by each method, DBScribe propagates (iteratively) them through the call chains, from the bottom



5. HTML descriptions generation



Example

com.umas.code.CourseOffered.addOneSeatFilledToCourseOffered()

This method implements the following db-related operations:

- It queries the table(s) *COURSESOFFERED*

- It updates the *SeatsFilled* attribute(s) in table *COURSESOFFERED*

This method invokes db-related operations via delegation:

- It queries the table(s) *SEMESTER* via *the chain-call* *com.umas.code.CourseOffered.checkIfCurrent* ➦ *com.umas.code.CourseOffered.getCurrentSemesterID*

Some constraints that should be taken into the account are the following:

- Make sure the values in *COURSESOFFERED.SeatsFilled* are not null

Example

com.umas.code.CourseOffered.addOneSeatFilledToCourseOffered()

This method implements the following db-related operations:

- It queries the table(s) *COURSESOFFERED*

- It updates the *SeatsFilled* attribute(s) in table *COURSESOFFERED*

This method invokes db-related operations via delegation:

- It queries the table(s) *SEMESTER* via *the chain-call* *com.umas.code.CourseOffered.checkIfCurrent* ➦ *com.umas.code.CourseOffered.getCurrentSemesterID*

Some constraints that should be taken into the account are the following:

- Make sure the values in *COURSESOFFERED.SeatsFilled* are not null

Example

com.umas.code.CourseOffered.addOneSeatFilledToCourseOffered()

This method implements the following db-related operations:

- It queries the table(s) *COURSESOFFERED*

- It updates the *SeatsFilled* attribute(s) in table *COURSESOFFERED*

This method invokes db-related operations via delegation:

- It queries the table(s) *SEMESTER* via *the chain-call* *com.umas.code.CourseOffered.checkIfCurrent* ➦ *com.umas.code.CourseOffered.getCurrentSemesterID*

Some constraints that should be taken into the account are the following:

- Make sure the values in *COURSESOFFERED.SeatsFilled* are not null

Example

com.umas.code.CourseOffered.addOneSeatFilledToCourseOffered()

This method implements the following db-related operations:

- It queries the table(s) *COURSESOFFERED*

- It updates the *SeatsFilled* attribute(s) in table *COURSESOFFERED*

This method invokes db-related operations via delegation:

- It queries the table(s) *SEMESTER* via *the chain-call* *com.umas.code.CourseOffered.checkIfCurrent* ➦ *com.umas.code.CourseOffered.getCurrentSemesterID*

Some constraints that should be taken into the account are the following:

- Make sure the values in *COURSESOFFERED.SeatsFilled* are not null

Example

com.umas.code.Admin.addAdmin(String, Department)

This method invokes db-related operations via delegation:

- It inserts the *UIN, Salary, OfficeAddress, OfficeHours* attributes into table *EMPLOYEE* via *a call to the com.umas.code.Employee.addEmployee method*

- It queries the table(s) *EMPLOYEE* via *the chain-call com.umas.code.Employee.addEmployee* ➚ *com.umas.code.Employee.addEmployeeCheck*

Some constraints that should be taken into the account are the following:

- Make sure the strings to be stored in *EMPLOYEE* do not overflow the varchar limits: *45 (OfficeAddress, OfficeHours)*

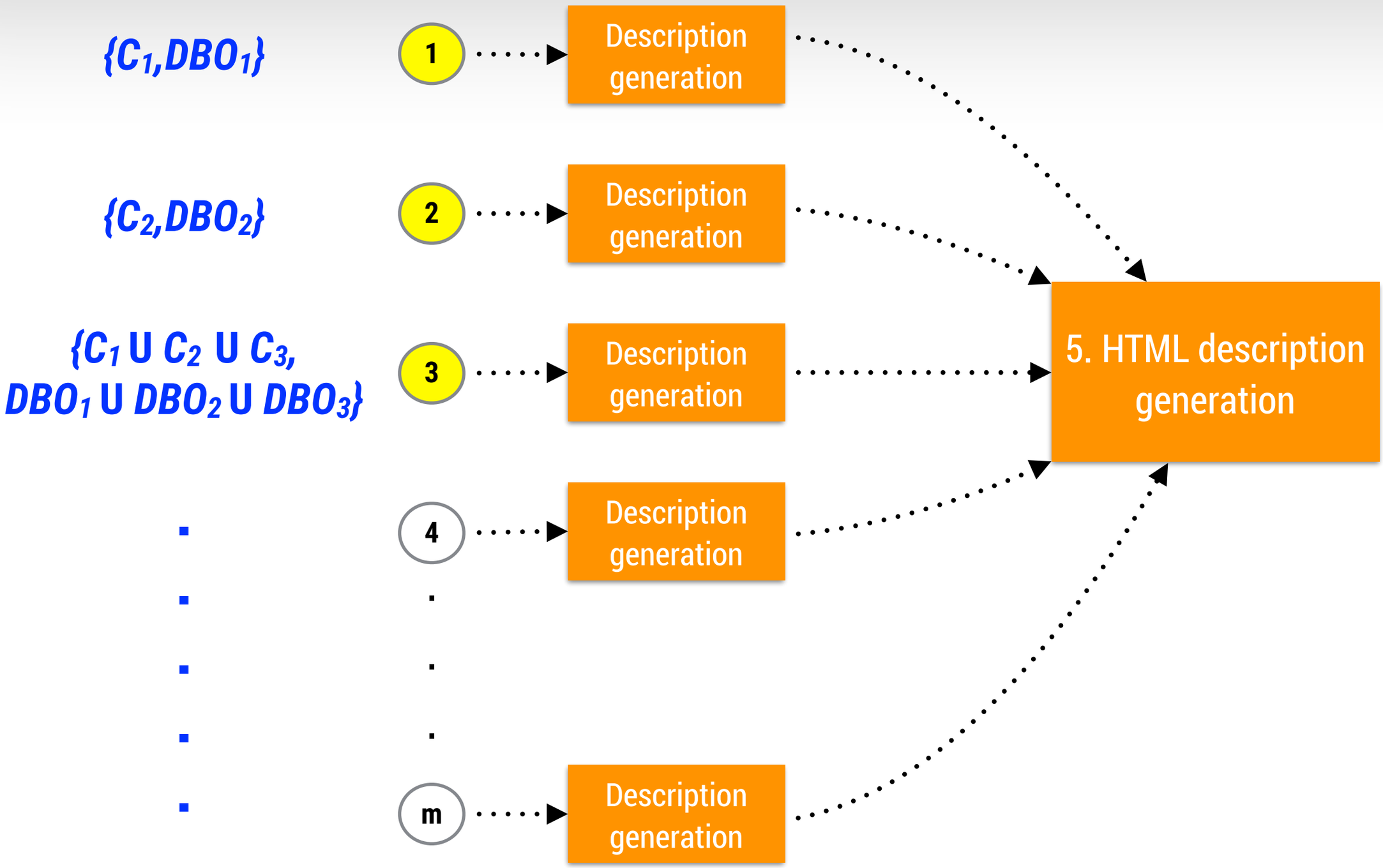
- Make sure the values in *EMPLOYEE.Salary* are not null

- Make sure the values in *EMPLOYEE.UIN* are not null

- Make sure the values of attribute *EMPLOYEE.UIN* are unique because there is a UNIQUENESS constraint

- When inserting into table *EMPLOYEE*, make sure the referential integrity imposed by attribute(s) *UIN* is accomplished. The foreign keys in the table are the following: *(UIN ❯ people.UIN)*

5. HTML descriptions generation



Example

DBScribe report

Summary: 26 methods with SQL local invocations, 21 methods mixing local and delegated SQL invocations, and 22 methods with only delegated SQL invocations.

Methods

com.bluecubs.xinco.core.server.XincoCoreLangu
com.bluecubs.xinco.core.server.XincoCoreLangu
com.bluecubs.xinco.core.server.XincoCoreLangu
com.bluecubs.xinco.core.server.XincoCoreLangu
com.bluecubs.xinco.core.server.XincoCoreLogSe
com.bluecubs.xinco.core.server.XincoCoreLogSe
com.bluecubs.xinco.core.server.XincoCoreLogSe
com.bluecubs.xinco.core.server.XincoCoreNodeS
com.bluecubs.xinco.core.server.XincoCoreNodeS
com.bluecubs.xinco.core.server.XincoCoreNodeS
com.bluecubs.xinco.core.server.XincoCoreNodeS
com.bluecubs.xinco.core.server.XincoCoreNodeS
com.bluecubs.xinco.core.server.XincoCoreNodeS
com.bluecubs.xinco.core.server.XincoCoreNodeS
com.bluecubs.xinco.core.server.XincoCoreUserS
com.bluecubs.xinco.core.server.XincoCoreUserS
com.bluecubs.xinco.core.server.XincoCoreUserS
com.bluecubs.xinco.core.server.XincoCoreUserS

Methods with local invocations:

com.bluecubs.xinco.add.server.XincoAddAttributeServer.XincoAddAttributeServer(int, int, XincoDBManager)

This method implements the following db-related operations:

- It queries the table(s) *XINCO_ADD_ATTRIBUTE*

com.bluecubs.xinco.add.server.XincoAddAttributeServer.getXincoAddAttributes(int, XincoDBManager)

This method implements the following db-related operations:

- It queries the table(s) *XINCO_ADD_ATTRIBUTE*

com.bluecubs.xinco.add.server.XincoAddAttributeServer.write2DB(XincoDBManager)

This method implements the following db-related operations:

- It inserts values for the first *8* columns into table *XINCO_ADD_ATTRIBUTE*

Some constraints that should be taken into the account are the following:

- Make sure the strings to be stored in *XINCO_ADD_ATTRIBUTE* do not overflow the varchar limits: *65535 (attrib_text), 255 (attrib_varchar)*

- When inserting into table *XINCO_ADD_ATTRIBUTE*, make sure the referential integrity imposed by attribute(s) *xinco_core_data_id* is accomplished. The foreign keys in the table are the following: *(xinco_core_data_id ❯ xinco_core_data.id)*

Limitations

- Current implementation supports **MySQL Server, JDBC, and Hibernate**

- Call graph extraction is **path insensitive** (over-approximation)

- **No inter-procedural analysis** for strings concatenation/ replacement in SQL literals

EVALUATION

*Image from https://en.wikipedia.org/wiki/Cuneiform_script

Empirical Study

5+2

Systems

52

Participants

2

Original developers

Empirical Study

5+2
Systems

52
Participants

2
Original developers

1. Quality of the descriptions: completeness, conciseness, expressiveness

2. Usefulness and user preferences

3. Industrial applicability

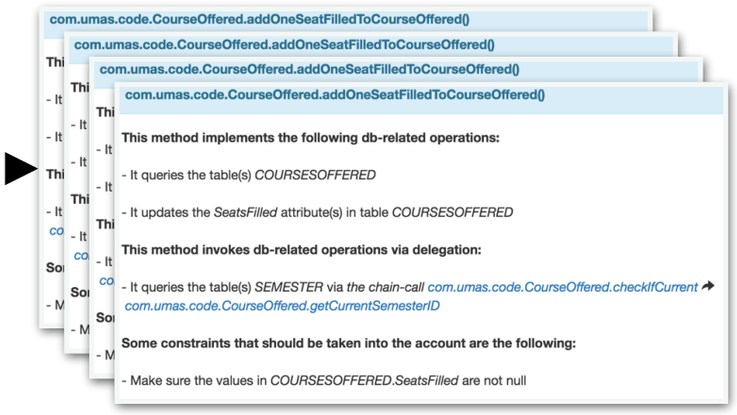
Quality of the descriptions

30 randomly selected **methods** and their **descriptions** (6 per system)

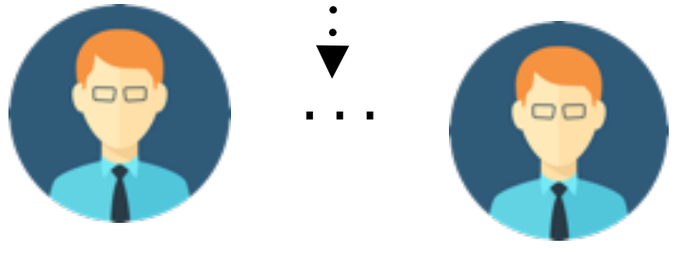
OS systems

DBScribe

Descriptions (method level)



6 **methods per participant**



Participants

Participant vs DBScribe

- Completeness
- Conciseness
- Expressiveness
- User preferences
- Usefulness

Quality of the descriptions

Completeness

66%
Does not miss any important info

29%
Misses some important info

5%
Misses most important info

Conciseness

71%
Contains no redundant info

25%
Contains some redundant info

4%
Contains a lot of redundant info

Quality of the descriptions

Expressiveness

77%

Is easy to read

19%

Is somewhat readable

4%

Is hard to read/understand

Usefulness

48/52 

Useful for understanding the database usages in source code methods

---

21 Incremental change

10 Maintenance

10 Bugs fixing

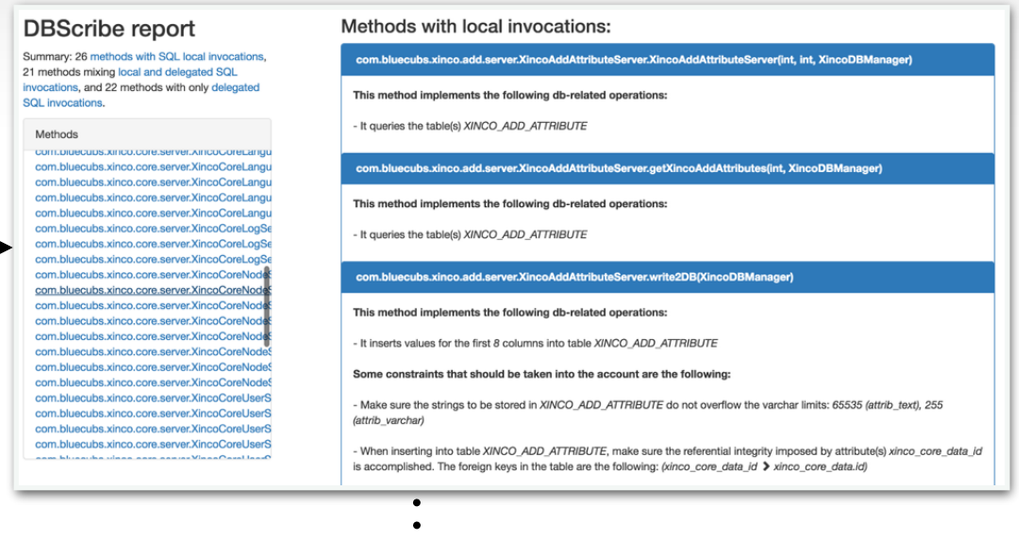
15 Others (e.g. test cases design)

Industrial applicability

2 industrial systems

DBScribe

HTML reports



- Completeness
- Conciseness
- Expressiveness
- User preferences
- Usefulness

Original developers/ maintainers

Industrial applicability

DBScribe is useful for incremental change and maintenance

“Based on the descriptions you can be aware all dependencies a table could have. It would let you estimate in a better way the impact due to future changes.”

“It helps you create a quick vision of the system with the basic method and code structure without looking at actual source code”

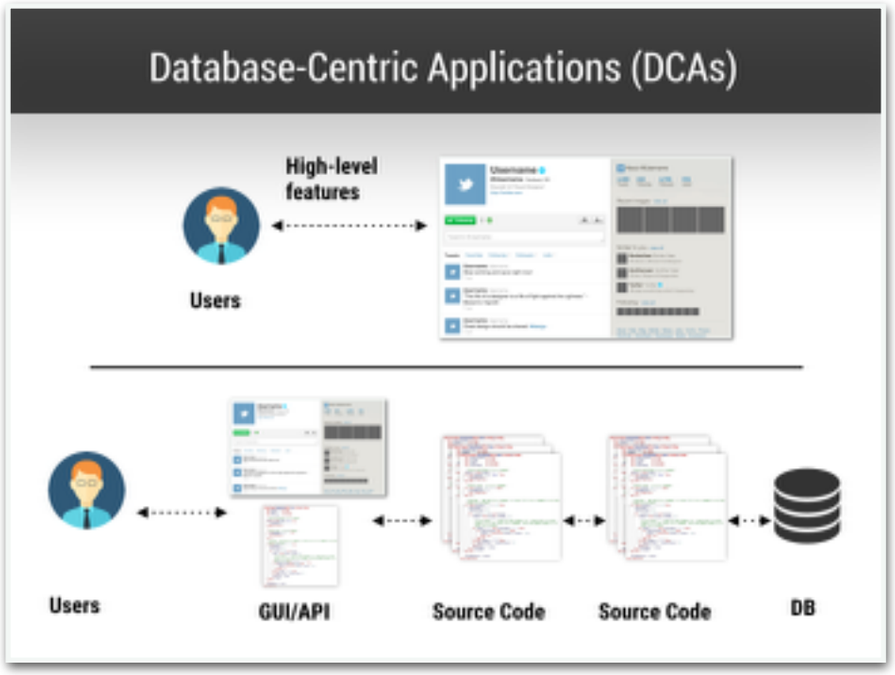
Industrial applicability

Feedback from developers

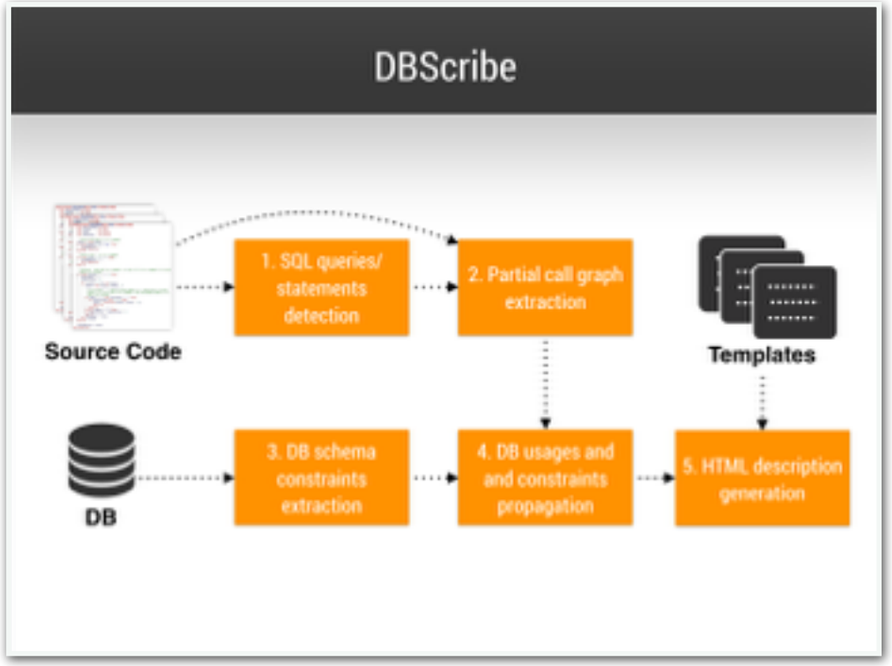
"The link system for call-chains works only in one way, one could get lost navigating a complex system […] . A navigation tree might be useful in this case."

"you should extend the approach to include JPA", "it would be better to have it in the IDE, something like right click->generate".

SUMMARY


Database-Centric Applications (DCAs)
High-level features
Users
Users
GUI/API
Source Code
Source Code
DB

Database-Centric Applications (DCAs)
High-level features
Users
Users
GUI/API
Source Code
Source Code
DB


DBScribe
Source Code
1. SQL queries/ statements detection
2. Partial call graph extraction
Templates
DB
3. DB schema constraints extraction
4. DB usages and and constraints propagation
5. HTML description generation

Database-Centric Applications (DCAs)

DBScribe

Empirical Study

5+2
Systems

52
Participants

2
Original devs.

1. Method description quality: completeness, conciseness, expressiveness

2. Usefulness and user preferences

3. Industrial applicability

Database-Centric Applications (DCAs)

DBScribe

Empirical Study

5+2
Systems

52
Participants

2
Original devs.

1. Method description quality: completeness, conciseness, expressiveness
2. Usefulness and user preferences
3. Industrial applicability

Summary

- ✓ **Mostly complete, concise, and readable**
- ✓ **Useful for software engineering tasks**
- ✓ **Can be applied in industry**

THANKS !!

http://www.cs.wm.edu/semeru/data/ISSTA16-DBScribe/

Table 3: Systems' statistics: Lines Of Code, TaBles in the DB schema, # of JDBC API calls involving SQL-Statements, # of SQL statements that *DB-Scribe* was Not able to Parse, # of Methods declaring SQL-statements Locally (ML), via Delegation (MD), Locally + Delegation (MLD), execution Time in sec.

System	LOC	TB	S	NP	ML	MD	MLD	T
UMAS [8]	32K	122	211	4	125	431	67	29.53
Riskit rev.96 [7]	12.7K	13	111	2	35	9	44	15.02
FINA 3.4.2 [2]	139.5K	52	710	26	312	118	99	130.78
Xinco rev.700 [9]	25.6K	23	76	15	26	22	21	31.41
OpenEmm 6.0 [5]	102.4K	68	200	110	73	12	1	104.78
System 1*	73.2K	53	398	27	262	660	24	71.07
System 2*	28.4K	24	164	8	106	247	44	40.13

Table 5: Answers to *"What software engineering tasks will you use this type of summary for?"*

Category	Subcategories
Incremental change (21)	Program comprehension (11), Add new features (4), Impact analysis (4), Concept location (1), Change database schema (1)
Bugs (10)	Debugging (6), Bug fixing (4)
Maintenance (10)	Maintenance (4), Refactoring (2), Re-modularization (2), Re-engineering (2)
Others (15)	Documentation (9), Change db-related code (3), Test cases design (2), Systems integration (1)