

Analyzing the Evolution of Data-Intensive Software Systems *in support to software maintenance*

Anthony Cleve
PReCISE Research Center
University of Namur, Belgium



Credits

This talk is based on joint work with ...

i.e., I shamelessly reused some slides by...

Credits

This talk is based on joint work with ...

i.e., I shamelessly reused some slides by...

At University of Namur

Maxime Gobert (MSc. student)

Jérôme Maes (Msc student)

Nesrine Noughi (PhD student)

Loup Meurice (PhD student)



Credits

This talk is based on joint work with ...

i.e., I shamelessly reused some slides by...

At University of Namur

Maxime Gobert (MSc. student)

Jérôme Maes (Msc student)

Nesrine Noughi (PhD student)

Loup Meurice (PhD student)



At University of Szeged, Hungary

Dr. Csaba Nagy (visiting Post-doc)



Credits

This talk is based on joint work with ...

i.e., I shamelessly reused some slides by...

At University of Namur

Maxime Gobert (MSc. student)

Jérôme Maes (Msc student)

Nesrine Noughi (PhD student)

Loup Meurice (PhD student)



At University of Szeged, Hungary

Dr. Csaba Nagy (visiting Post-doc)



At University of Victoria, Canada

Prof. Dr. Jens Weber (collaborator)

Credits

This talk is based on joint work with ...

i.e., I shamelessly reused some slides by...

At University of Namur

Maxime Gobert (MSc. student)

Jérôme Maes (Msc student)

Nesrine Noughi (PhD student)

Loup Meurice (PhD student)



At University of Szeged, Hungary

Dr. Csaba Nagy (visiting Post-doc)



At University of Victoria, Canada

Prof. Dr. Jens Weber (collaborator)



At University of Murcia, Spain

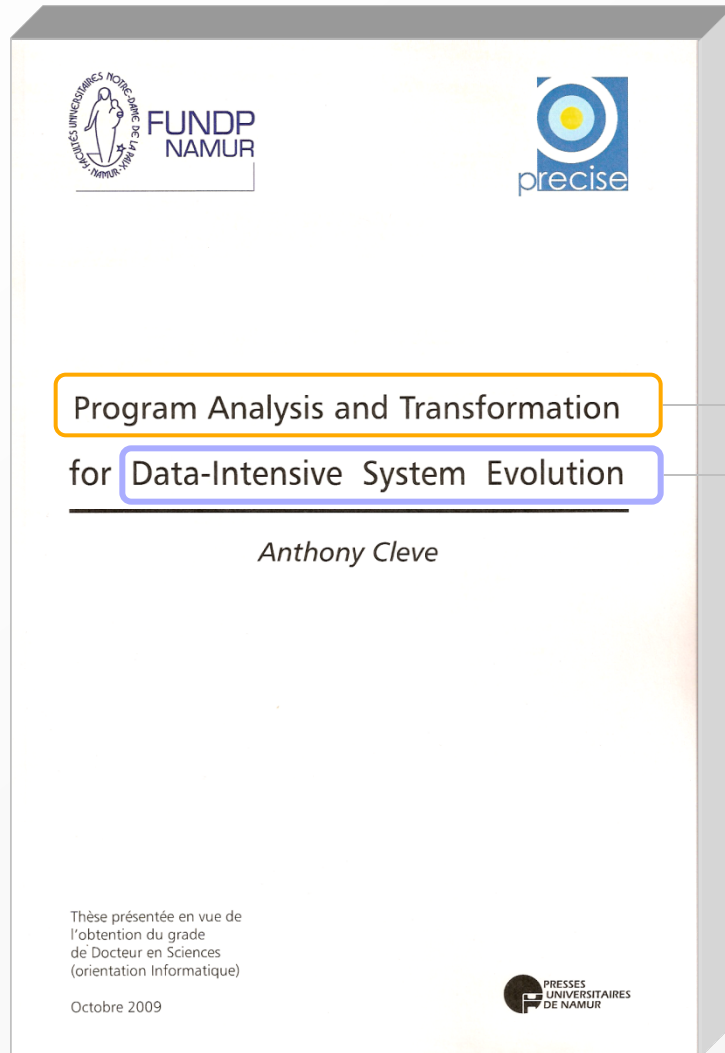
Fco Javier Bermudez (visiting PhD student)



Introduction

Our research field

Once upon a time... (in 2009)



Program Analysis and Transformation

for Data-Intensive System Evolution

Anthony Cleve

Thèse présentée en vue de
l'obtention du grade
de Docteur en Sciences
(orientation Informatique)

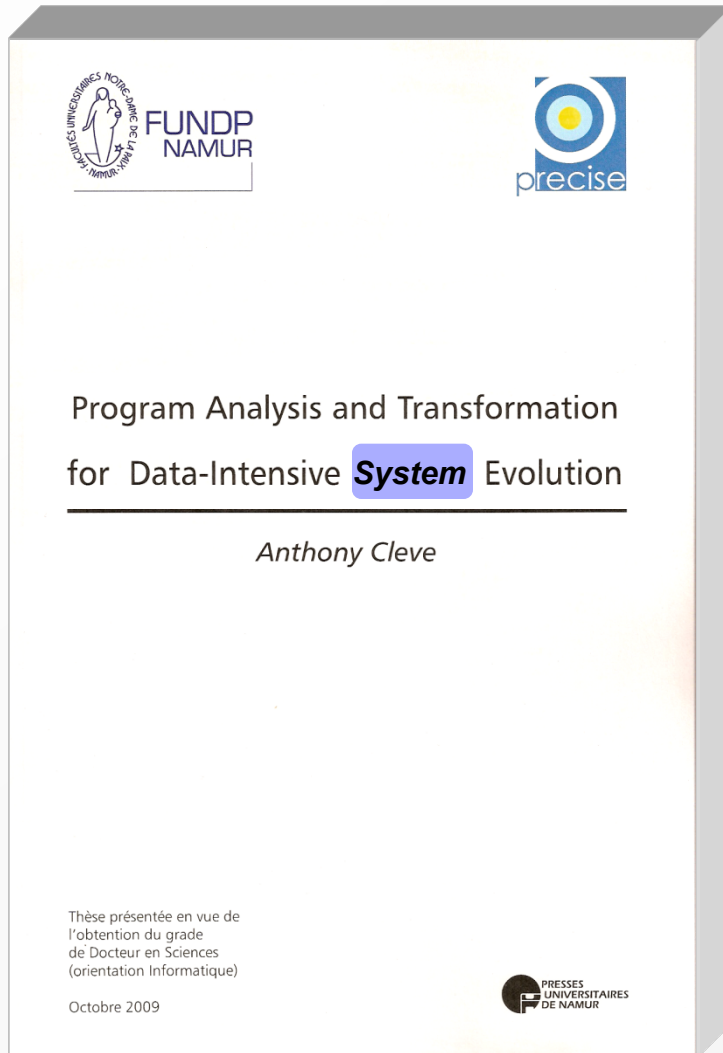
Octobre 2009

PRESSES
UNIVERSITAIRES
DE NAMUR

Proposed solutions

General problem

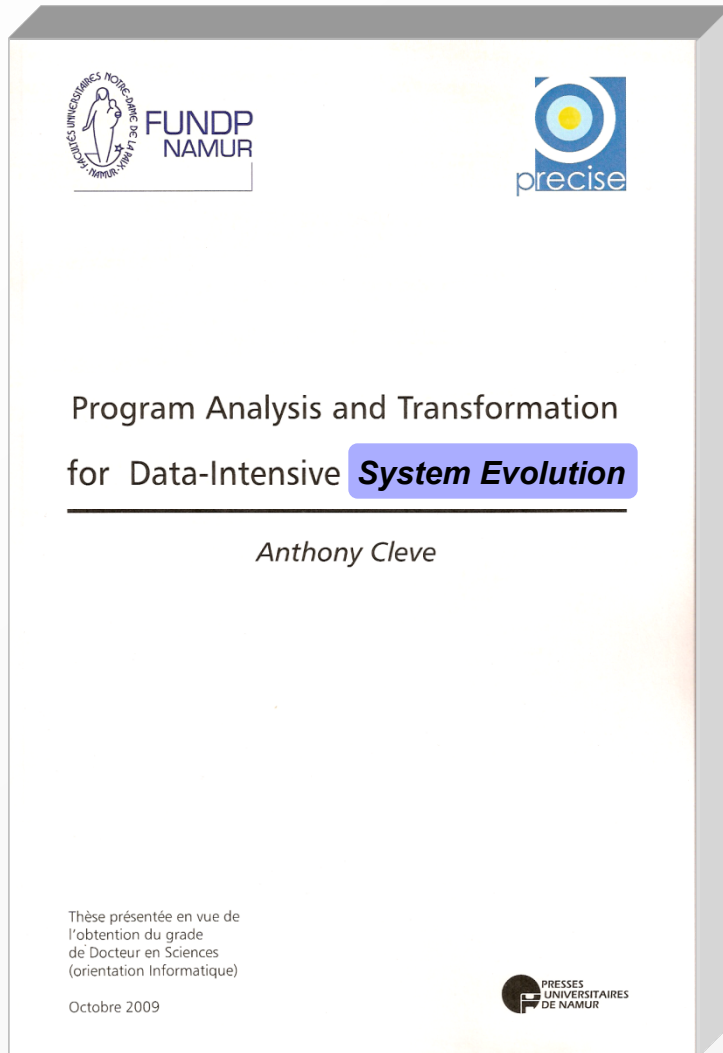
Problem statement



System = software systems

- play a major role for most organizations
- often large, heterogeneous and complex
- made of various ***inter-dependent artefacts***

Problem statement



System evolution = *inevitable* phenomenon

- Software systems are constantly evolving
 - business *pull*
 - IT *push*
 - error correction (repair)
- complex, expensive and highly risky process
- *consistency* between artefacts to be preserved

Why is software evolution so important?

System maintenance and evolution

responsible for **up to 90%** of total system costs !



Why is software evolution so important?

System maintenance and evolution

from **2x to 100x** more costly than initial system development



Why is software evolution so important?

System maintenance and evolution

up to **80%** of the maintenance time spent in trying to **understand**
... the **current** version of the system



Work distribution of system developers

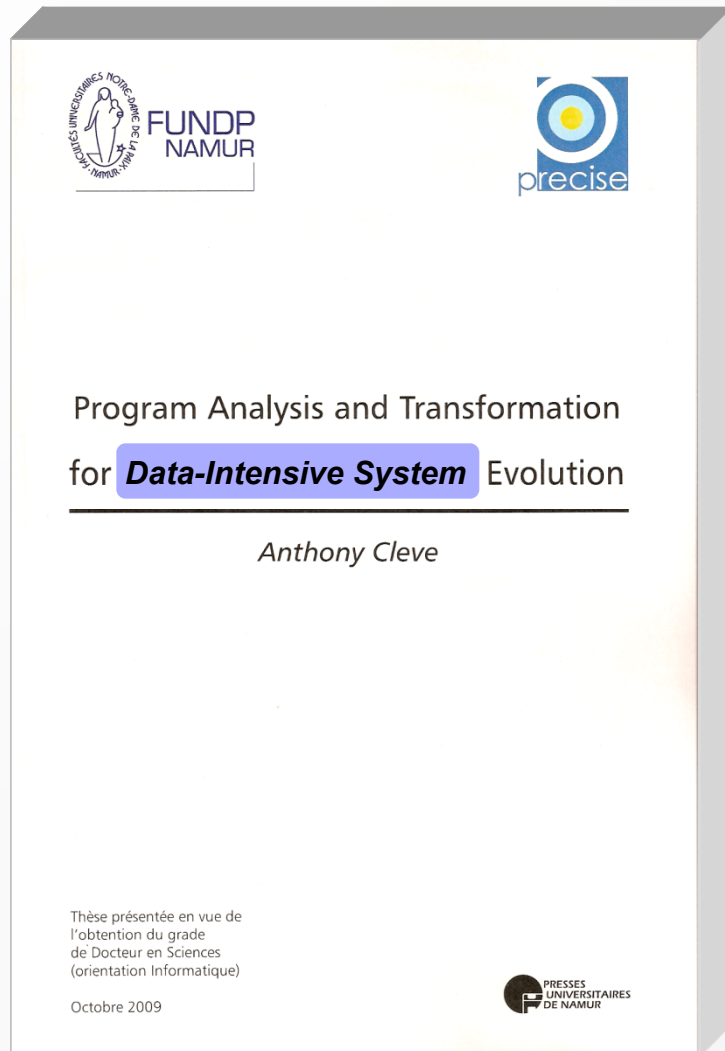
Years	New projects	Enhancements	Repairs	Total
1950	90	3	7	100
1960	8 500	500	1000	10 000
1970	65 000	15 000	20 000	100 000
1980	1 200 000	600 000	200 000	2 000 000
1990	3 000 000	3 000 000	1 000 000	7 000 000
2000	4 000 000	4 500 000	1 500 000	10 000 000
2010	5 000 000	7 000 000	2 000 000	14 000 000
2020	7 000 000	11 000 000	3 000 000	21 000 000

Today: 65% of all developers work on system maintenance and evolution

In 2020: only 30 % of them will work on new projects !

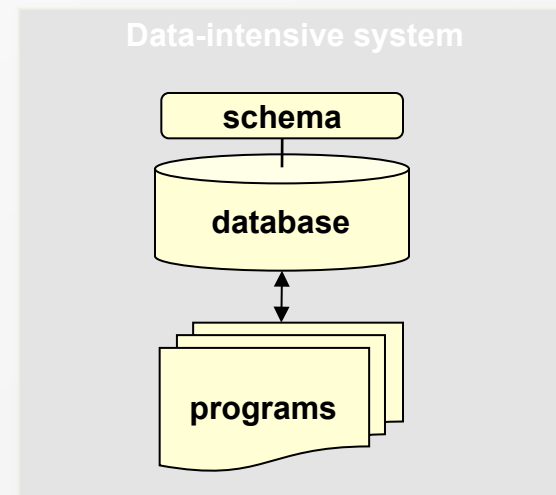
Source: Capers Jones, Software Productivity Research
(via Paul Klint. *The Software Evolution Volcano*, 2011.)

Focus on data-intensive systems



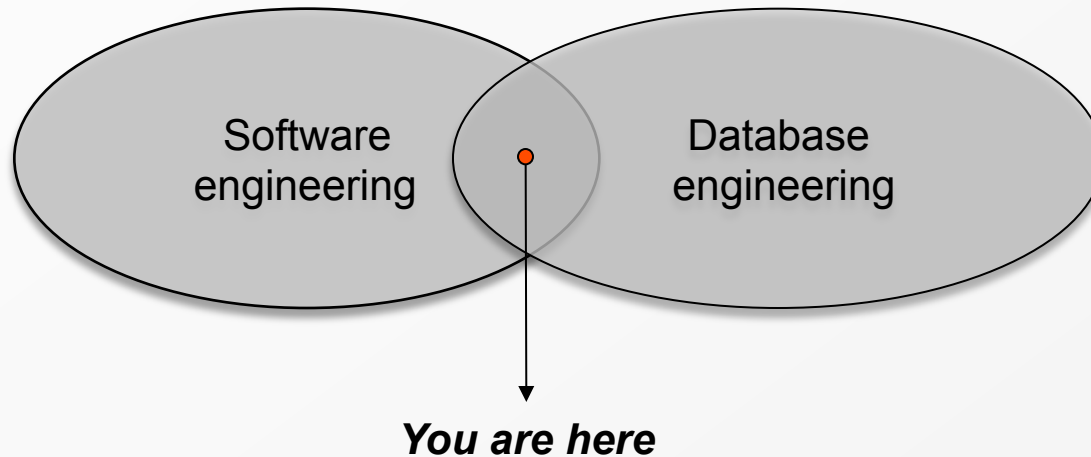
Data-intensive = intensive use of data

- a *database* containing mission-critical data
- a set of *programs* read and update this database
- *queries* expressed on top of the database *schema*

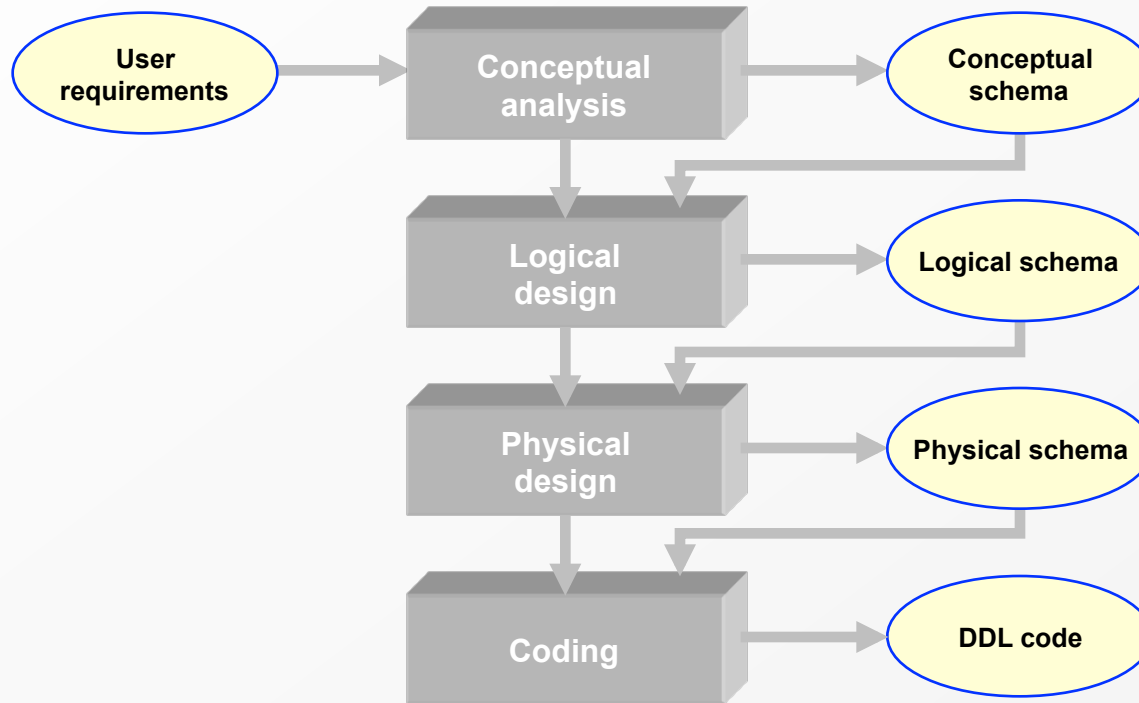


Data-intensive system evolution

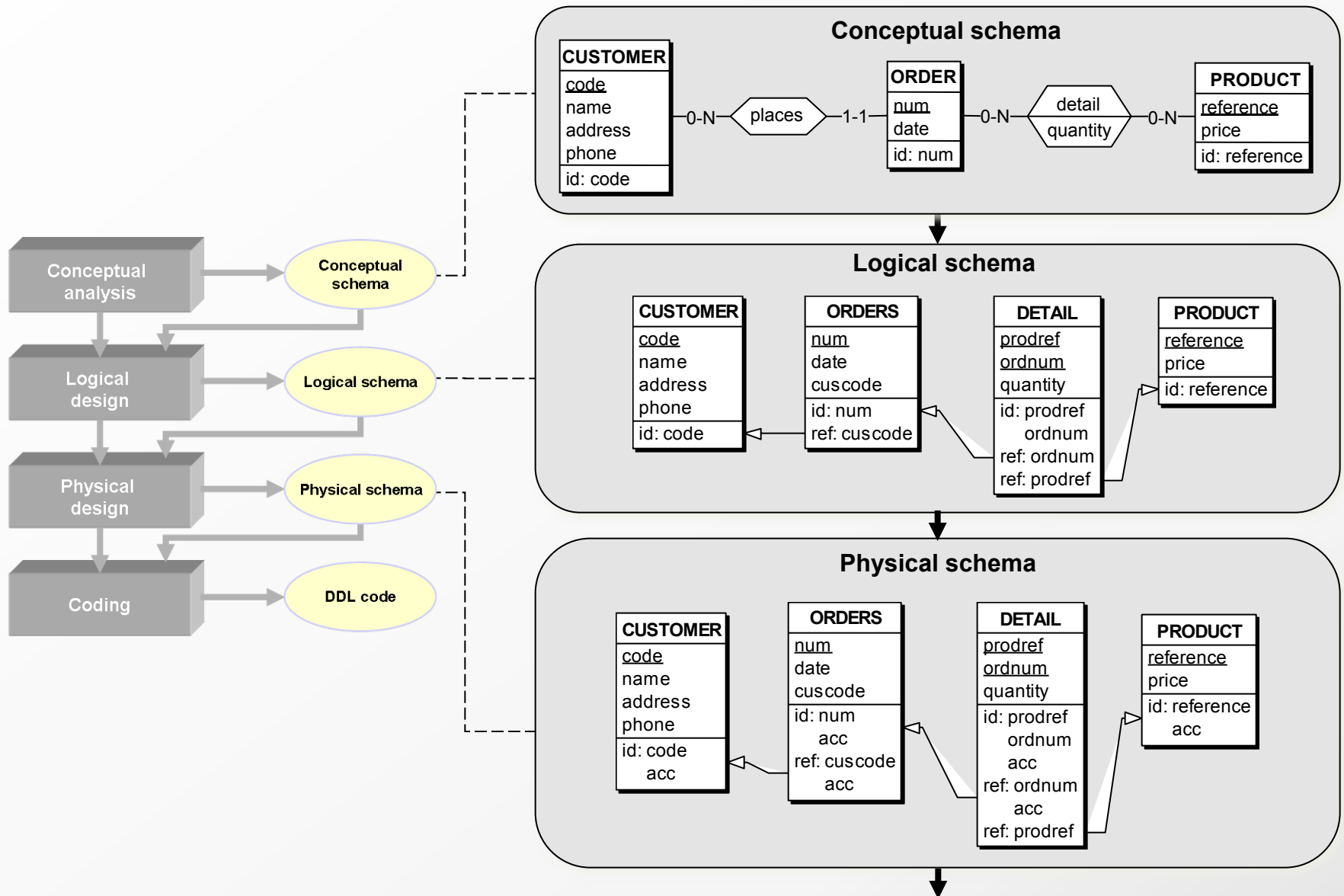
intersection (or union) of two distinct research communities



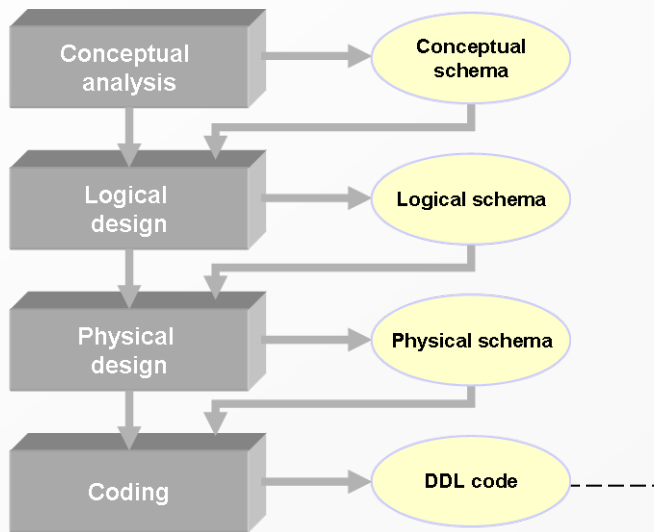
Database engineering (ideal view)



Database engineering (ideal view)



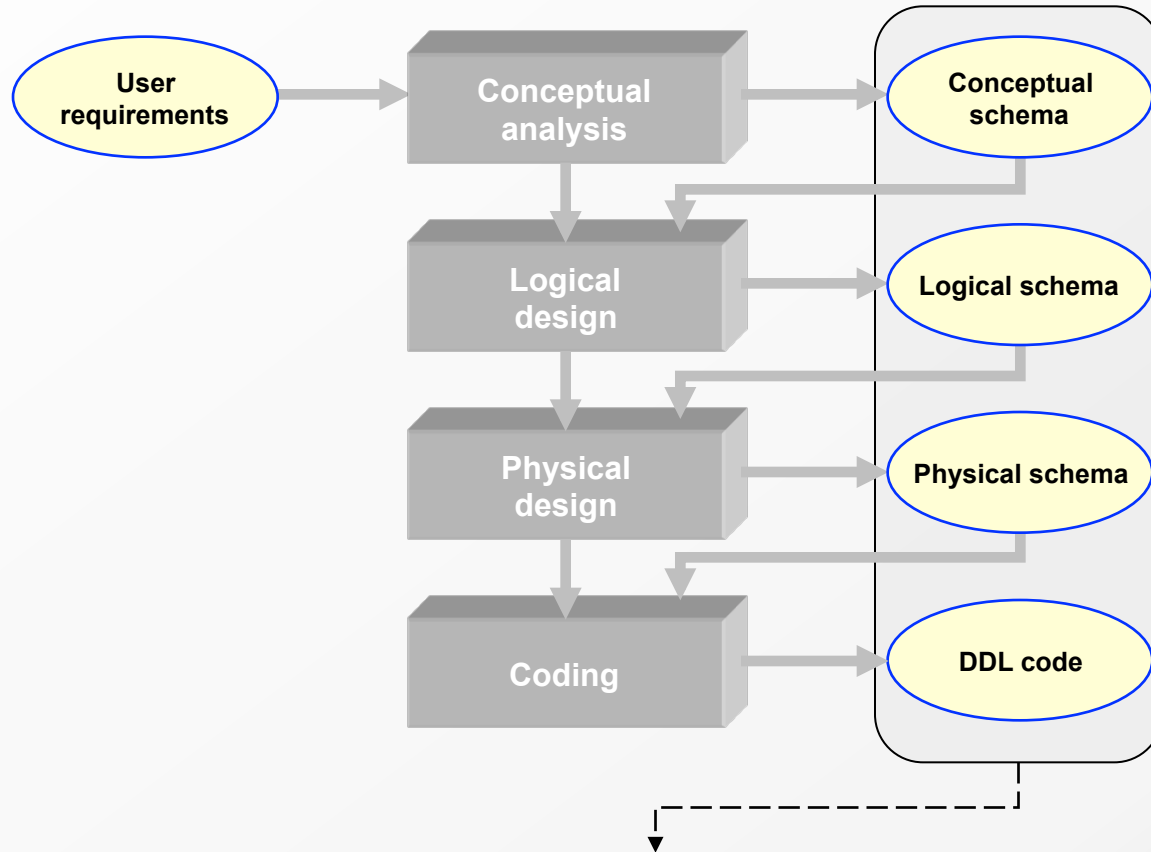
Database engineering (ideal view)



DDL code

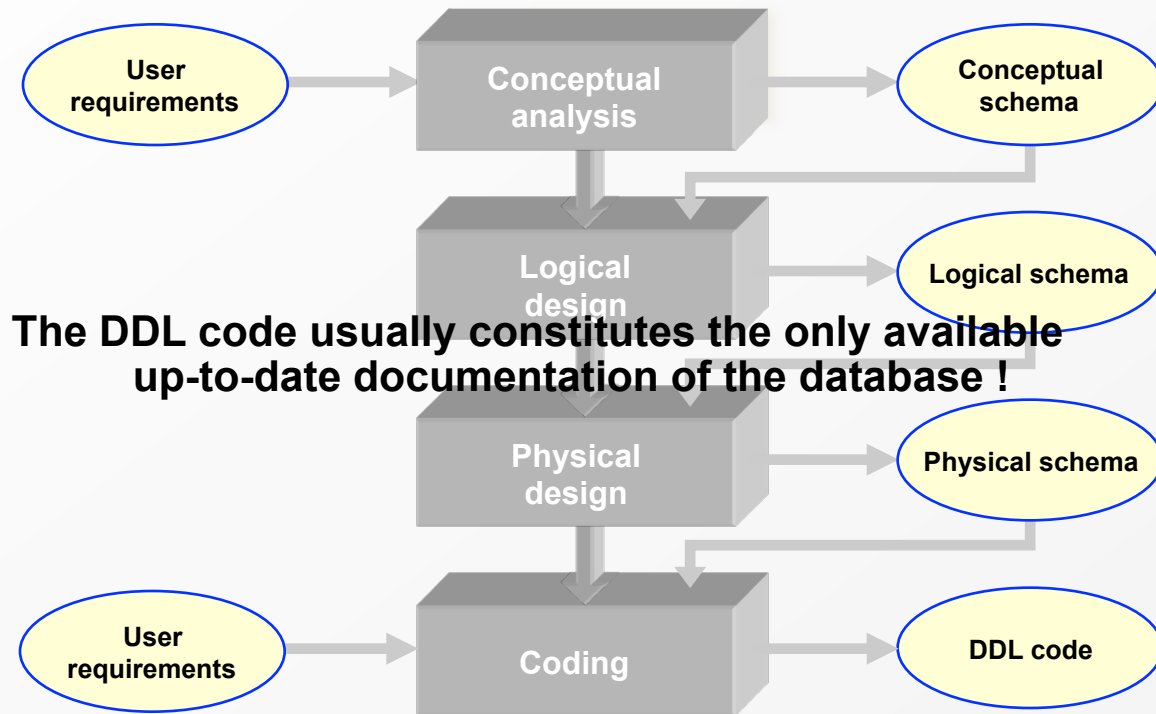
```
create table CUSTOMER (  
    code char(6) not null,  
    name char(20) not null,  
    address char(40) not null,  
    phone numeric(12) not null,  
    constraint ID_CUSTOMER primary key (code));  
  
create table DETAIL (  
    prodref char(6) not null,  
    ordnum char(6) not null,  
    quantity numeric(6) not null,  
    constraint ID_DETAIL primary key (prodref, ordnum));  
  
create table ORDERS (  
    num char(6) not null,  
    date date not null,  
    cuscode char(6) not null,  
    constraint ID_ORDERS primary key (num));  
  
create table PRODUCT (  
    reference char(6) not null,  
    price numeric(6,2) not null,  
    constraint ID_PRODUCT primary key (reference));  
  
alter table DETAIL add constraint REF_DET_ORD_FK  
    foreign key (ordnum) references ORDERS;  
  
alter table DETAIL add constraint REF_DET_PRO  
    foreign key (prodref) references PRODUCT;  
  
alter table ORDERS add constraint REF_ORD_CUS_FK  
    foreign key (cuscode) references CUSTOMER;  
  
create unique index CUSTOMER_IND on CUSTOMER (code);  
create unique index DET_IND on DETAIL (prodref, ordnum);  
create index DET_ORD_IND on DETAIL (ordnum);  
create unique index ORD_IND on ORDERS (num);  
create index ORD_CUS_IND on ORDERS (cuscode);  
create unique index PRODUCT_IND on PRODUCT (reference);
```

Database engineering (ideal view)

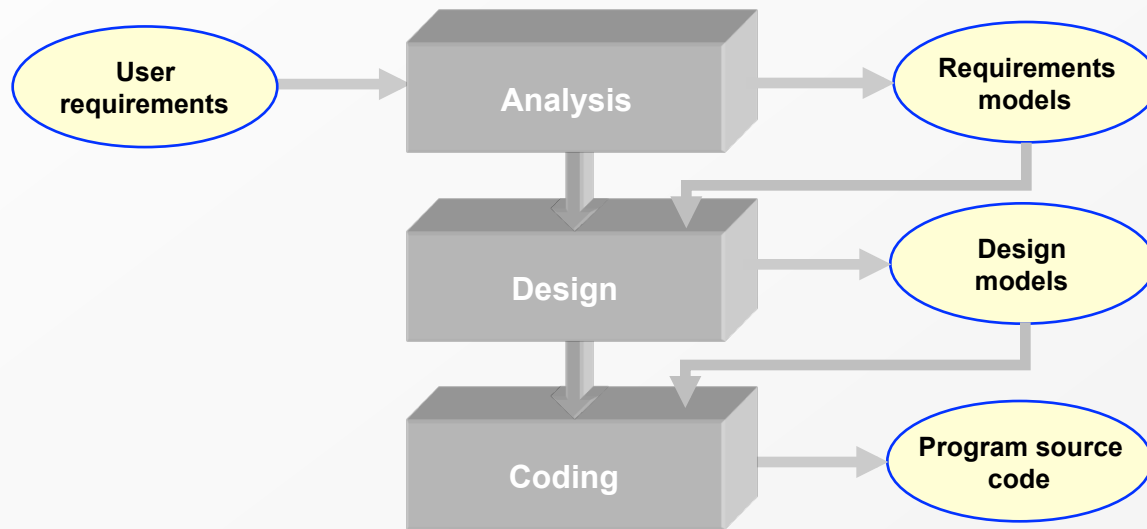


***Should form a complete and up-to-date
documentation of the database***

Database engineering (in practice)

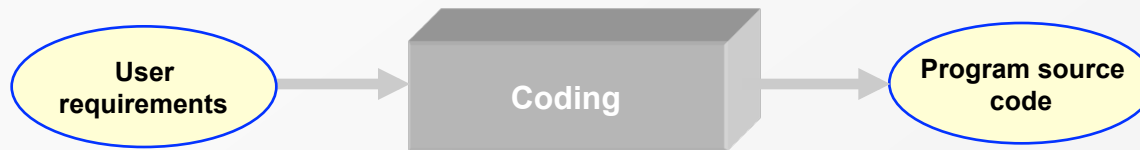


How about the programs?

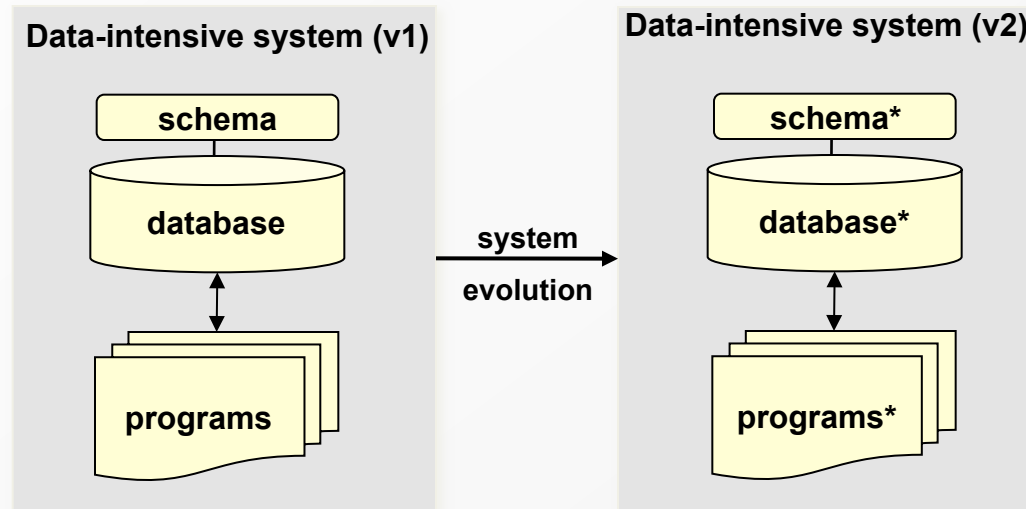


How about the programs ?

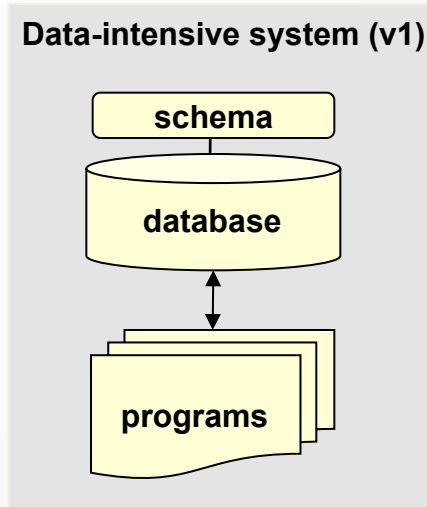
The source code often constitutes the only available up-to-date documentation of the programs !



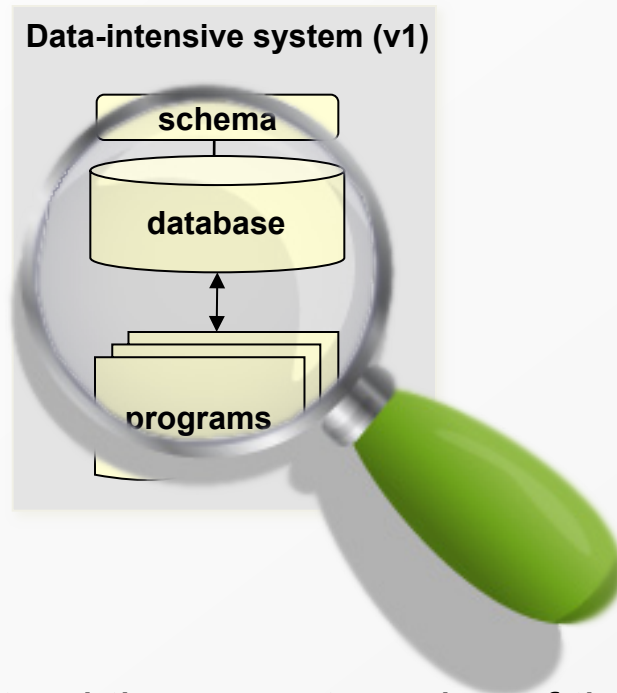
Data-intensive systems evolution



Data-intensive systems evolution



Data-intensive systems evolution

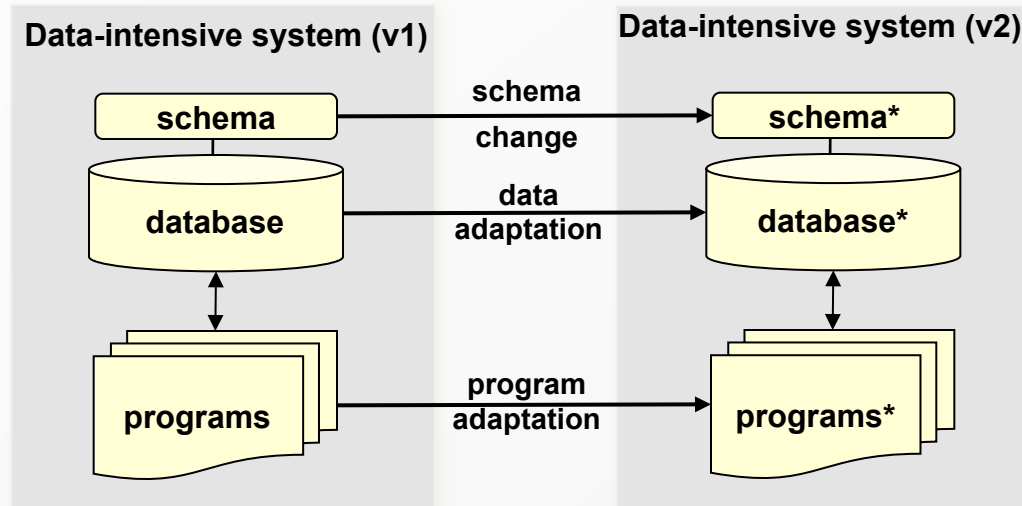


PHASE 1: Understand the current version of the system

= reverse-engineering process

1. Redocument the database schema (structure and constraints)
2. Redocument the programs (structure and behavior)

Data-intensive systems evolution



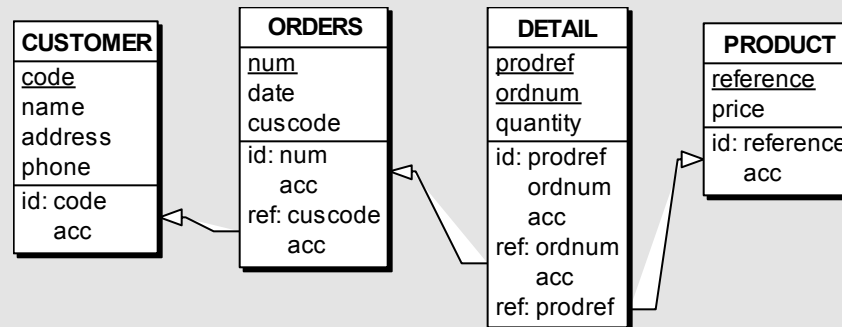
PHASE 2: Evolve the system towards a new version

= co-evolution process

1. Change the database schema
2. Adapt the database contents
3. Adapt the programs

When size does matter

DB schema used as illustration in my 1st database course

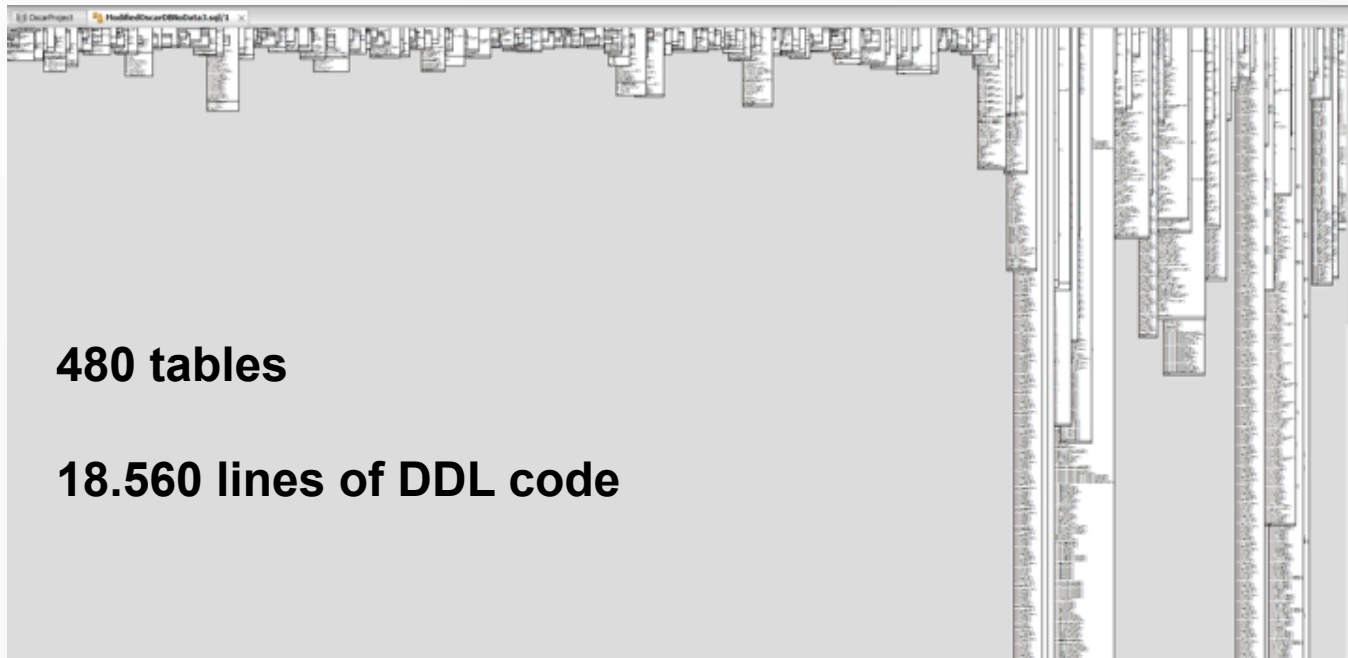


4 tables

< 40 lines of DDL code

When size does matter

DB schema of OSCAR, an healthcare system used in Canada



When size does matter

SQL query used as illustration in my 1st database course

```
select NCLI, NAME  
from CUSTOMER  
where CITY = 'Namur'
```

When size does matter

SQL query used in OSCAR, an healthcare system used in Canada

```
select appointment0_.appointment_no as appointm1_89_0_, demographi1_.demographic_no as demograp1_27_1_,
appointment0_.appointment_date as appointm2_89_0_, appointment0_.billing as billing89_0_, appointment0_.bookingSource as
bookingS4_89_0_, appointment0_.createdatetime as createda5_89_0_, appointment0_.creator as creator89_0_,
appointment0_.creatorSecurityId as creatorS7_89_0_, appointment0_.demographic_no as demograp8_89_0_, appointment0_.end_time as
end9_89_0_, appointment0_.imported_status as imported10_89_0_, appointment0_.lastupdateuser as lastupd11_89_0_,
appointment0_.location as location89_0_, appointment0_.name as name89_0_, appointment0_.notes as notes89_0_,
appointment0_.program_id as program15_89_0_, appointment0_.provider_no as provider16_89_0_, appointment0_.reason as reason89_0_,
appointment0_.reasonCode as reasonCode89_0_, appointment0_.remarks as remarks89_0_, appointment0_.resources as resources89_0_,
appointment0_.start_time as start21_89_0_, appointment0_.status as status89_0_, appointment0_.style as style89_0_, appointment0_.type as
type89_0_, appointment0_.updatedatetime as updated25_89_0_, appointment0_.urgency as urgency89_0_, demographi1_.title as
title27_1_, demographi1_.first_name as first3_27_1_, demographi1_.last_name as last4_27_1_, demographi1_.sex as sex27_1_,
demographi1_.month_of_birth as month6_27_1_, demographi1_.date_of_birth as date7_27_1_, demographi1_.year_of_birth as
year8_27_1_, demographi1_.address as address27_1_, demographi1_.city as city27_1_, demographi1_.province as province27_1_,
demographi1_.postal as postal27_1_, demographi1_.email as email27_1_, demographi1_.phone as phone27_1_, demographi1_.phone2
as phone15_27_1_, demographi1_.myOscarUserName as myOscar16_27_1_, demographi1_.hin as hin27_1_, demographi1_.ver as
ver27_1_, demographi1_.hc_type as hc19_27_1_, demographi1_.hc_renew_date as hc20_27_1_, demographi1_.roster_status as
roster21_27_1_, demographi1_.patient_status as patient22_27_1_, demographi1_.patient_status_date as patient23_27_1_,
demographi1_.date_joined as date24_27_1_, demographi1_.chart_no as chart25_27_1_, demographi1_.provider_no as provider26_27_1_,
demographi1_.end_date as end27_27_1_, demographi1_.eff_date as eff28_27_1_, demographi1_.roster_date as roster29_27_1_,
demographi1_.roster_termination_date as roster30_27_1_, demographi1_.roster_termination_reason as roster31_27_1_,
demographi1_.pcn_indicator as pcn32_27_1_, demographi1_.family_doctor as family33_27_1_, demographi1_.alias as alias27_1_,
demographi1_.previousAddress as previou35_27_1_, demographi1_.children as children27_1_, demographi1_.sourceOfIncome as
sourceO37_27_1_, demographi1_.citizenship as citizen38_27_1_, demographi1_.sin as sin27_1_, demographi1_.anonymous as
anonymous27_1_, demographi1_.spoken_lang as spoken41_27_1_, demographi1_.official_lang as official42_27_1_,
demographi1_.lastUpdateUser as lastUpd43_27_1_, demographi1_.lastUpdateDate as lastUpd44_27_1_, demographi1_.newsletter as
newsletter27_1_, demographi1_.country_of_origin as country46_27_1_, (select lst.description from lst_gender lst where
lst.code=demographi1_.sex) as formula21_1_, (select d.merged_to from demographic_merged d where d.deleted = 0 and
d.demographic_no = demographi1_.demographic_no) as formula22_1_, (select count(*) from admission a where
a.client_id=demographi1_.demographic_no and a.admission_status='current' and a.program_id in (select p.id from program p where
p.type='Bed' )) as formula23_1_, (select count(*) from health_safety h where h.demographic_no=demographi1_.demographic_no) as
formula24_1_ from appointment appointment0_, demographic demographi1_ where
appointment0_.demographic_no=demographi1_.demographic_no and demographi1_.hin<>' ' and
appointment0_.appointment_date>='2014-10-23' and appointment0_.appointment_date<='2014-10-23' and
(upper(demographi1_.province)='ONTARIO' or demographi1_.province='ON') group by demographi1_.demographic_no order by
demographi1_.last_name;
```

Episod I

The Origins

Episod I – The Origins

Once upon a time (in 2012-2013)...

- The OSCAR system
 - written in Java
 - > 2 millions lines of code
 - MySQL database



Episod I – The Origins

Once upon a time (in 2012-2013)...

- The OSCAR system
 - written in Java
 - > 2 millions lines of code
 - MySQL database
- Evolution goal
 - data migration towards NoSQL



Episod I – The Origins

Once upon a time (in 2012-2013)...

- The OSCAR system
 - written in Java
 - > 2 millions lines of code
 - MySQL database
- Evolution goal
 - data migration towards NoSQL
- Problem
 - lack of documentation (unsurprisingly)



Episod I – The Origins

Once upon a time (in 2012-2013)...

- The OSCAR system

written in Java

> 2 millions lines of code

MySQL database



- Evolution goal

data migration towards NoSQL

- Problem

lack of documentation (unsurprisingly)



Database reverse engineering (DBRE)
via a Master's thesis project

Standard approach to DBRE

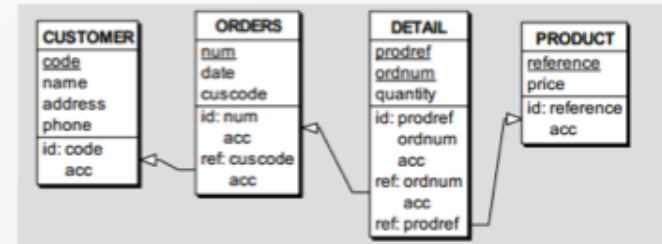
Step I: physical extraction

DDL code

```
create table CUSTOMER (  
  code char(6) not null,  
  name char(20) not null,  
  address char(40) not null,  
  phone numeric(12) not null,  
  constraint ID_CUSTOMER primary key (code));  
  
create table DETAIL (  
  prodref char(6) not null,  
  ordnum char(6) not null,  
  quantity numeric(6) not null,  
  constraint ID_DETAIL primary key (prodref, ordnum));  
  
create table ORDERS (  
  num char(6) not null,  
  date date not null,  
  cuscode char(6) not null,  
  constraint ID_ORDERS primary key (num));  
  
create table PRODUCT (  
  reference char(6) not null,  
  price numeric(6,2) not null,  
  constraint ID_PRODUCT primary key (reference));  
  
alter table DETAIL add constraint REF_DET_ORD_FK  
  foreign key (ordnum) references ORDERS;  
  
alter table DETAIL add constraint REF_DET_PRO  
  foreign key (prodref) references PRODUCT;  
  
alter table ORDERS add constraint REF_ORD_CUS_FK  
  foreign key (cuscode) references CUSTOMER;  
  
create unique index CUSTOMER_IND on CUSTOMER (code);  
create unique index DET_IND on DETAIL (prodref, ordnum);  
create index DET_ORD_IND on DETAIL (ordnum);
```



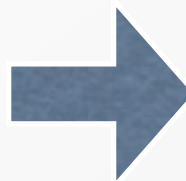
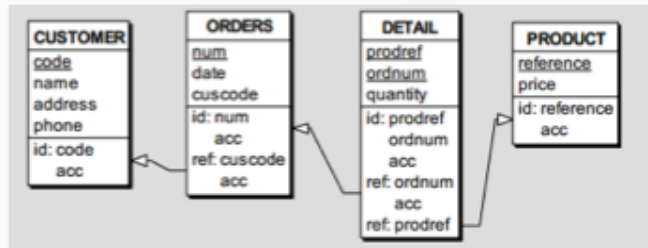
Physical schema



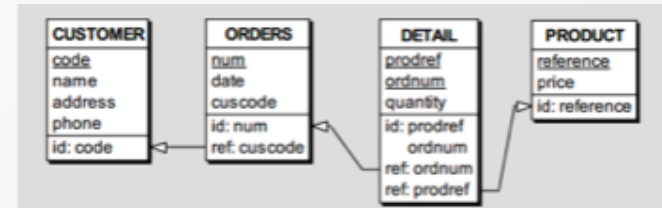
Standard approach to DBRE

Step II: logical refinement

Physical schema



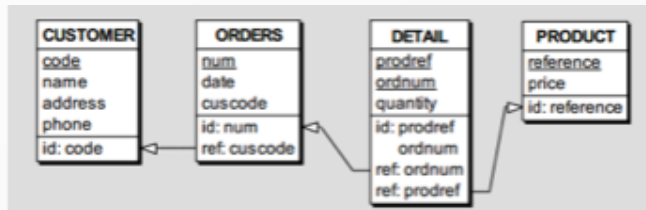
Logical schema



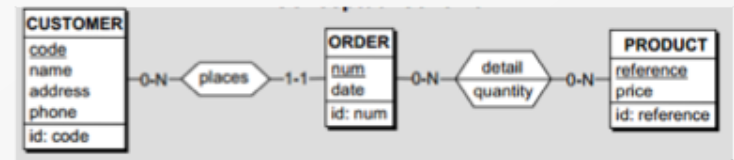
Standard approach to DBRE

Step III: conceptualization

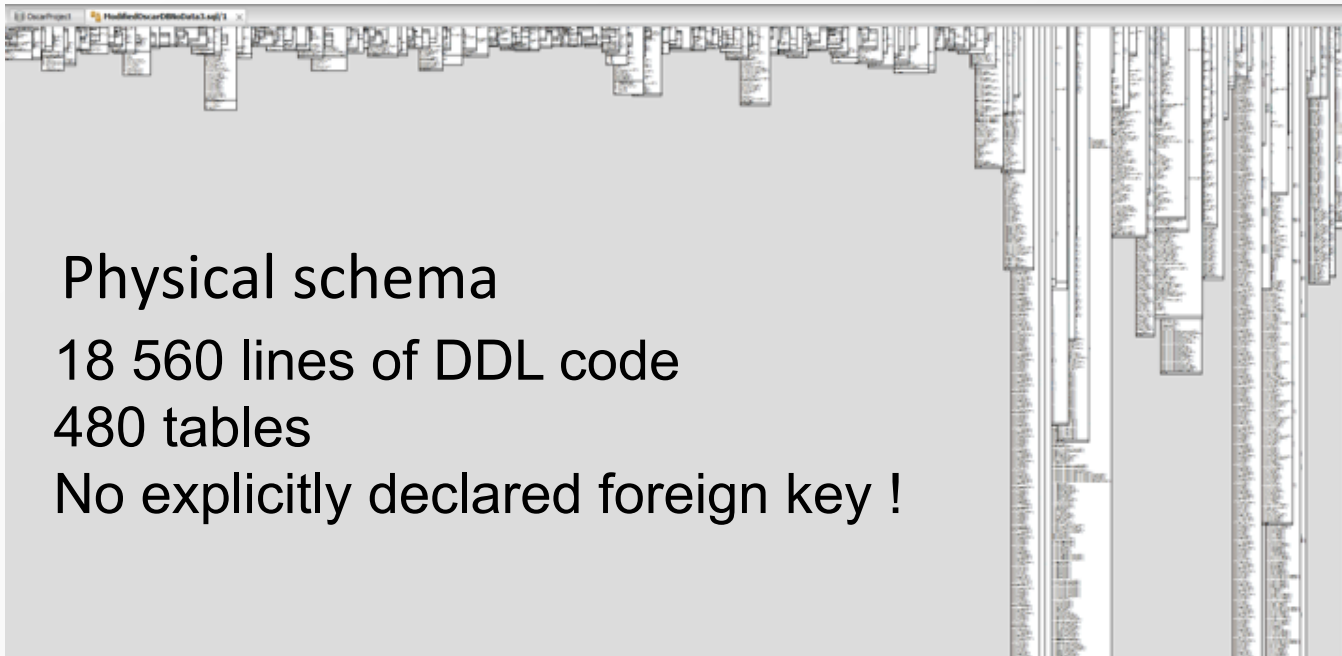
Logical schema



Conceptual schema



When applied to OSCAR...



Physical schema
18 560 lines of DDL code
480 tables
No explicitly declared foreign key !

The image shows a screenshot of a database physical schema diagram. The diagram is extremely dense and complex, with hundreds of small rectangular boxes representing tables and a vast network of lines representing relationships between them. The layout is chaotic, with many overlapping and overlapping lines, making it difficult to discern individual components. The text overlay on the left side of the image provides key statistics about the schema: 18,560 lines of DDL code, 480 tables, and the absence of explicitly declared foreign keys.

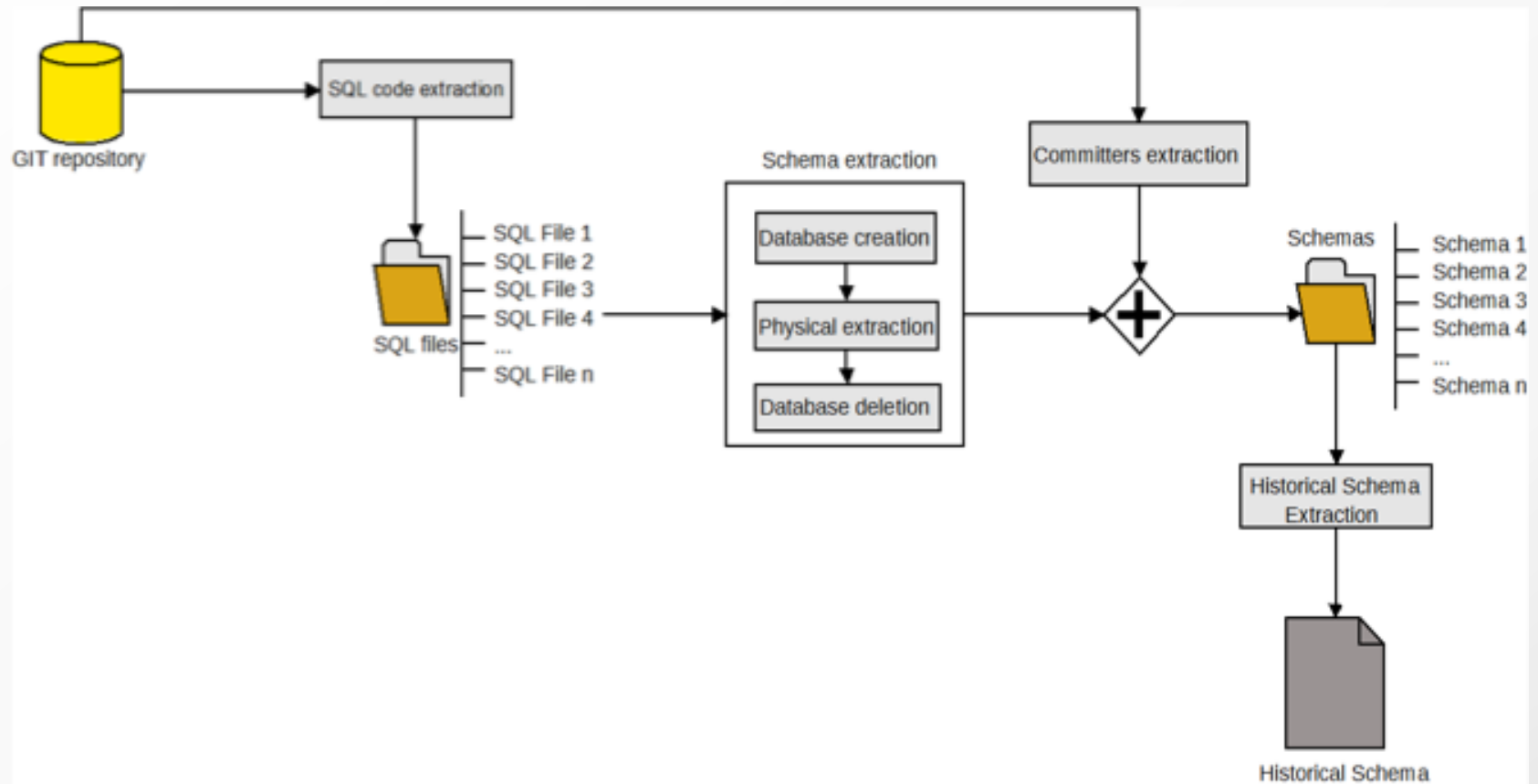
Crazy idea...

- History analysis techniques have been successfully used to support program analysis, understanding and evolution
- Analyzing the system history may provide additional insights about the current system version, and inform future evolutions
- So, let's follow the very same approach for databases !

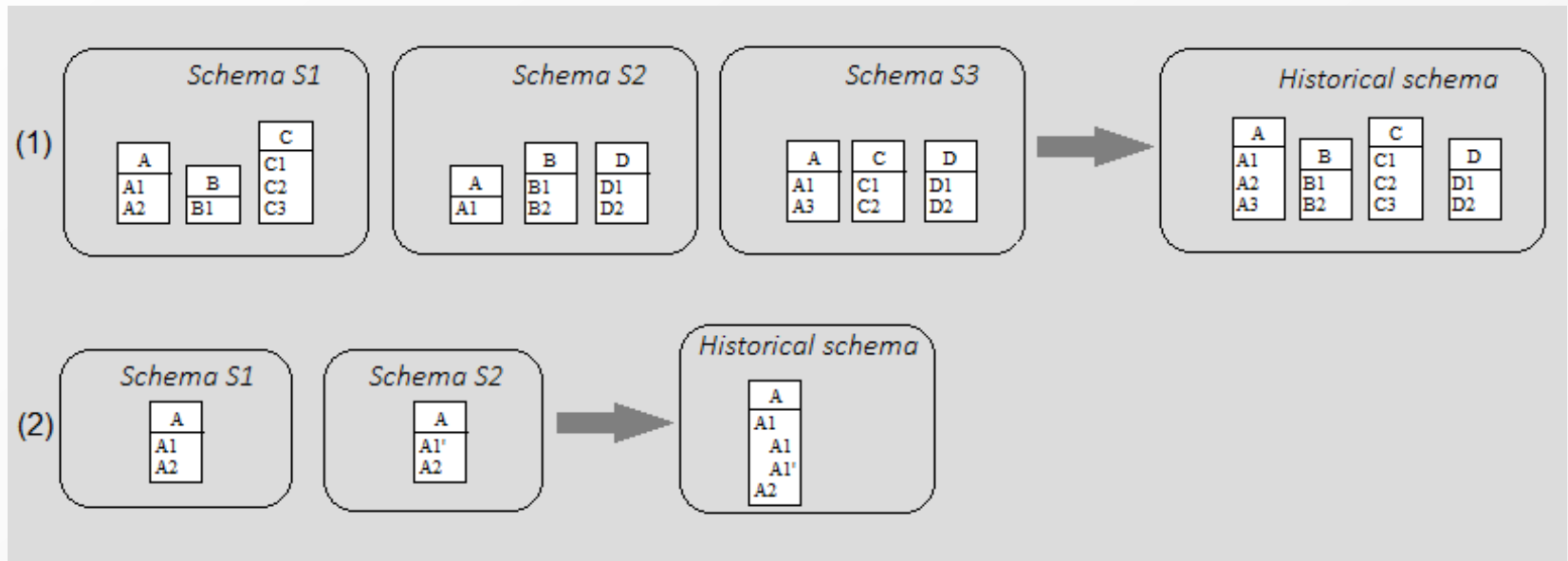
Research question

How can we extract, represent and exploit the evolution history of a database schema?

(initial) Approach



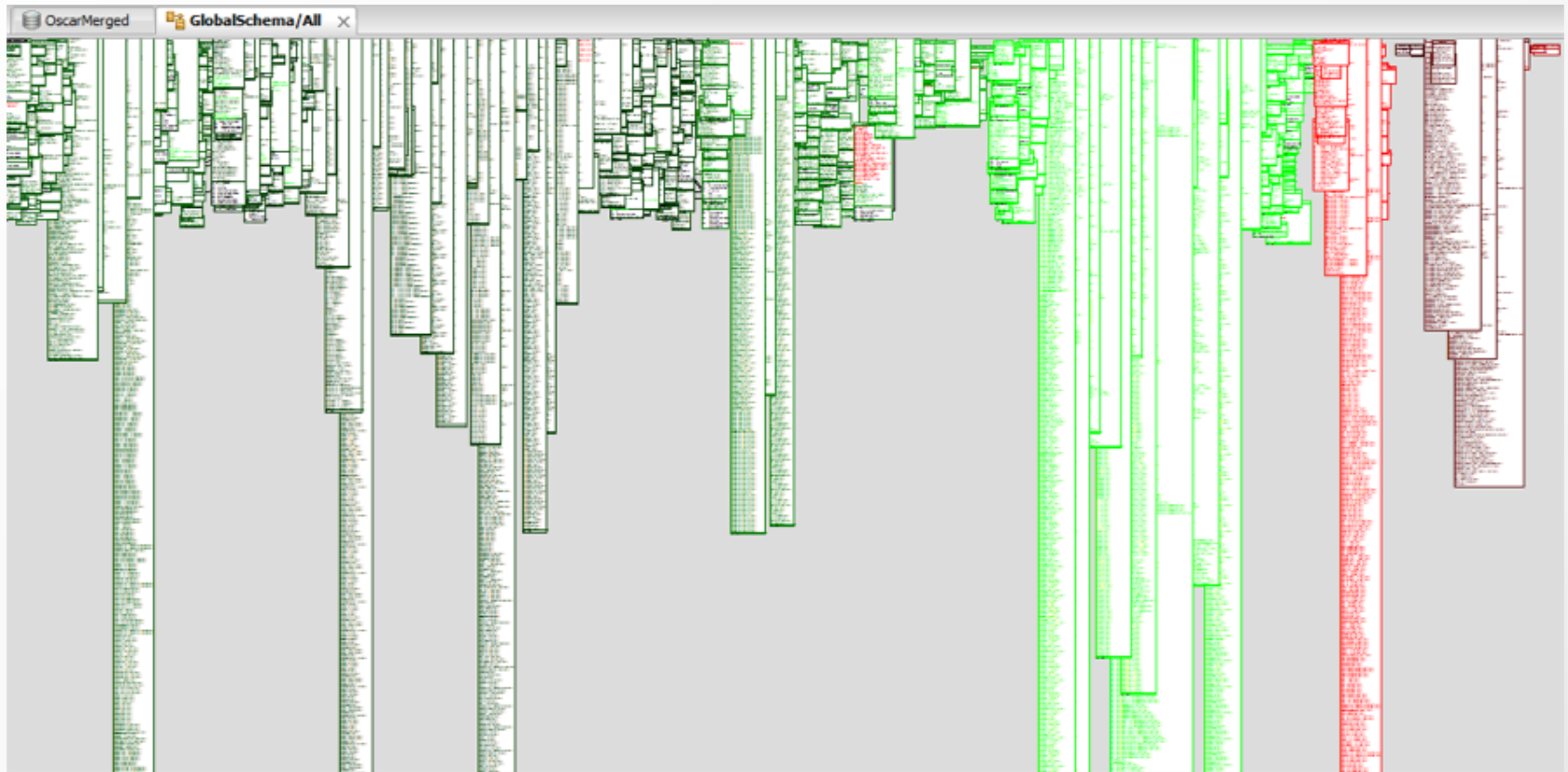
Historical schema



Historical schema

viewed within DB-MAIN

Historical schema of OSCAR (22/07/2003-27/06/2013, 670 schema versions)



Historical schema

viewed within DB-MAIN

The screenshot displays a complex historical schema diagram in DB-MAIN. The diagram consists of numerous vertical bars representing tables and their relationships, color-coded in green and red. A black circle highlights a specific table in the red section, with an arrow pointing to a detailed view of that table.

integratorconsent

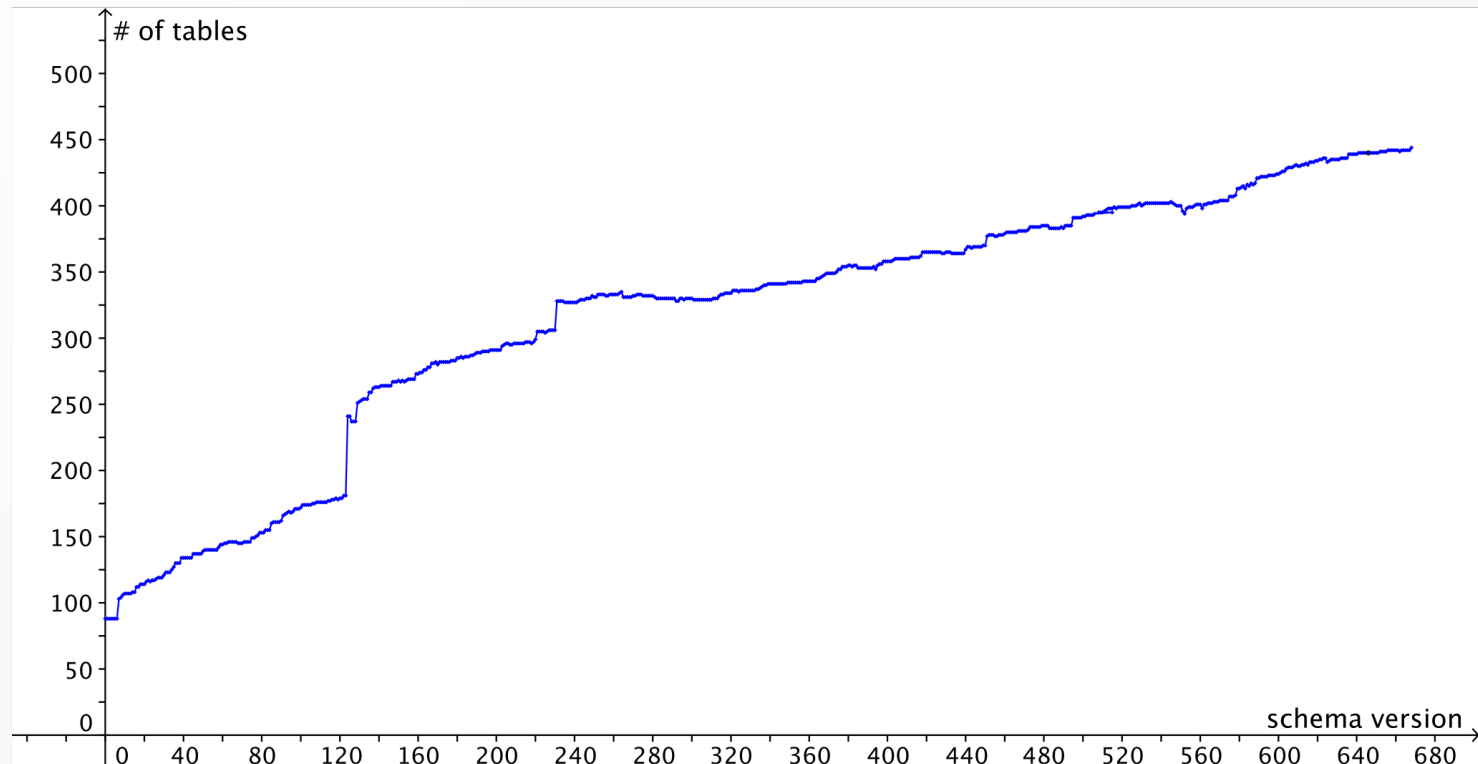
- consentLevel
- lastUpdate
- provider_no
- consentToPhoto
- consentToPreventions
- consentToNotes
- consentToIssues
- consentToHealthCardId
- consentToBasicPersonalId
- consentToStatistics
- consentToBasicPersonalData
- refusedToSign
- printedFormLocation[0-1]
- formVersion[0-1]
- consentToHealthNumberRegistry
- consentToMentalHealthData
- consentToAllNonDomainData
- consentToSearches
- restrictConsentToHic
- consentToShareData
- integratorFacilityId
- id[0-1]
- facilityId
- demographicId
- providerNo
- createdDate
- excludeMentalHealthData
- clientConsentStatus
- signatureStatus
- expiry[0-1]
- digitalSignatureId[0-1]
- id: id
- ref: digitalSignatureId
- ref: providerNo
- ref: demographicId
- ref: facilityId
- acc: createdDate

Property box
attribute: integratorconsent.consentToStatistics

Prop	User prop	Sem	Tech
nbCount	6		
isDead	<input checked="" type="checkbox"/>		
stopCounting	<input checked="" type="checkbox"/>		
creationSchema	oscar2008-06-21.sql		
creation_date	2008-06-21		
lastAppearanceSchema	oscar2008-12-21.sql		
severalLives	<input type="checkbox"/>		
isOpen	<input type="checkbox"/>		
lastAppearance_date	2008-12-21		
Default value			
MappingOID	1190377		
physLen	0		
physType			
Stereotype			
Value constraint			
line			

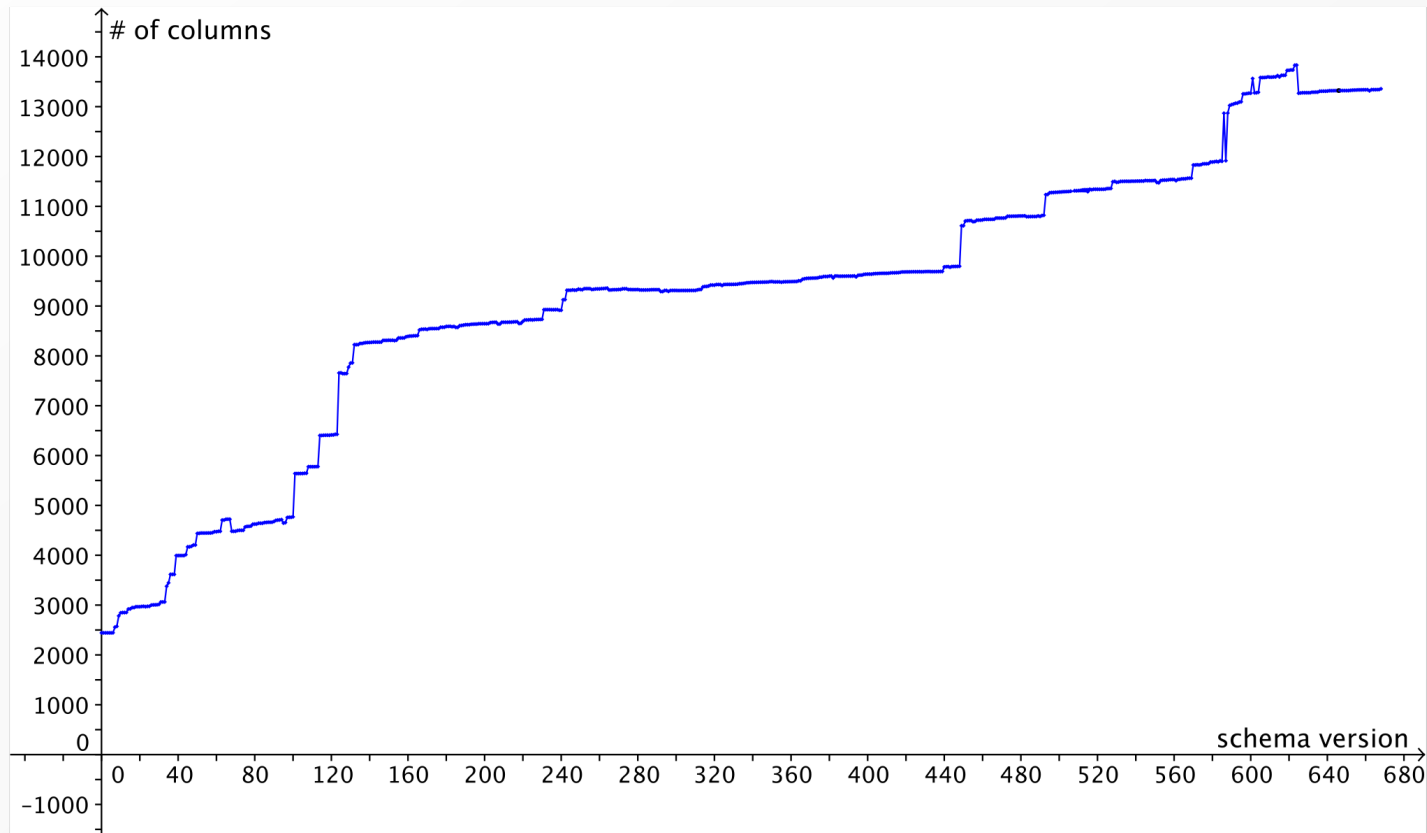
Historical schema analysis

evolution of the # of tables



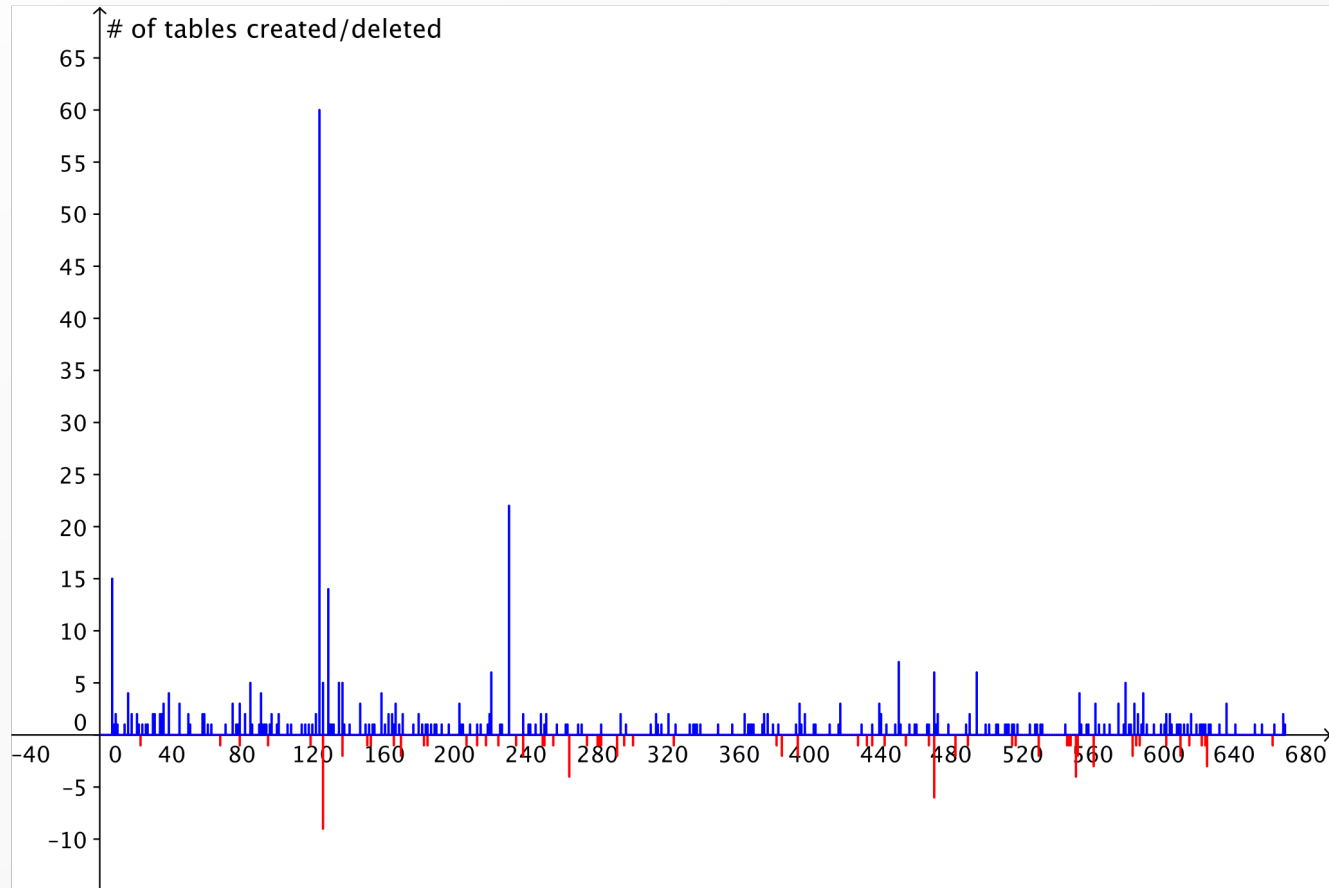
Historical schema analysis

evolution of the # of columns



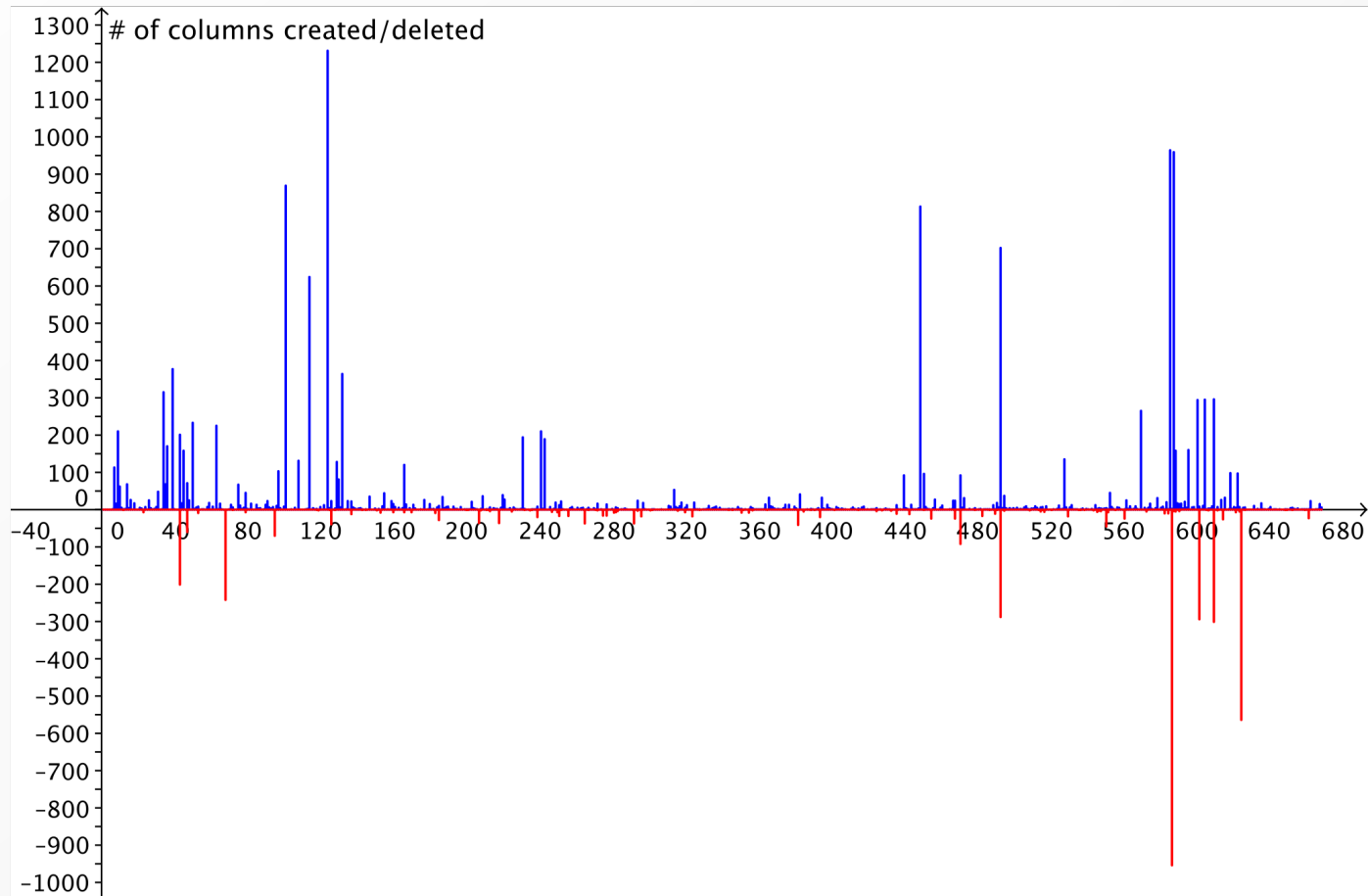
Historical schema analysis

creation/deletion of tables



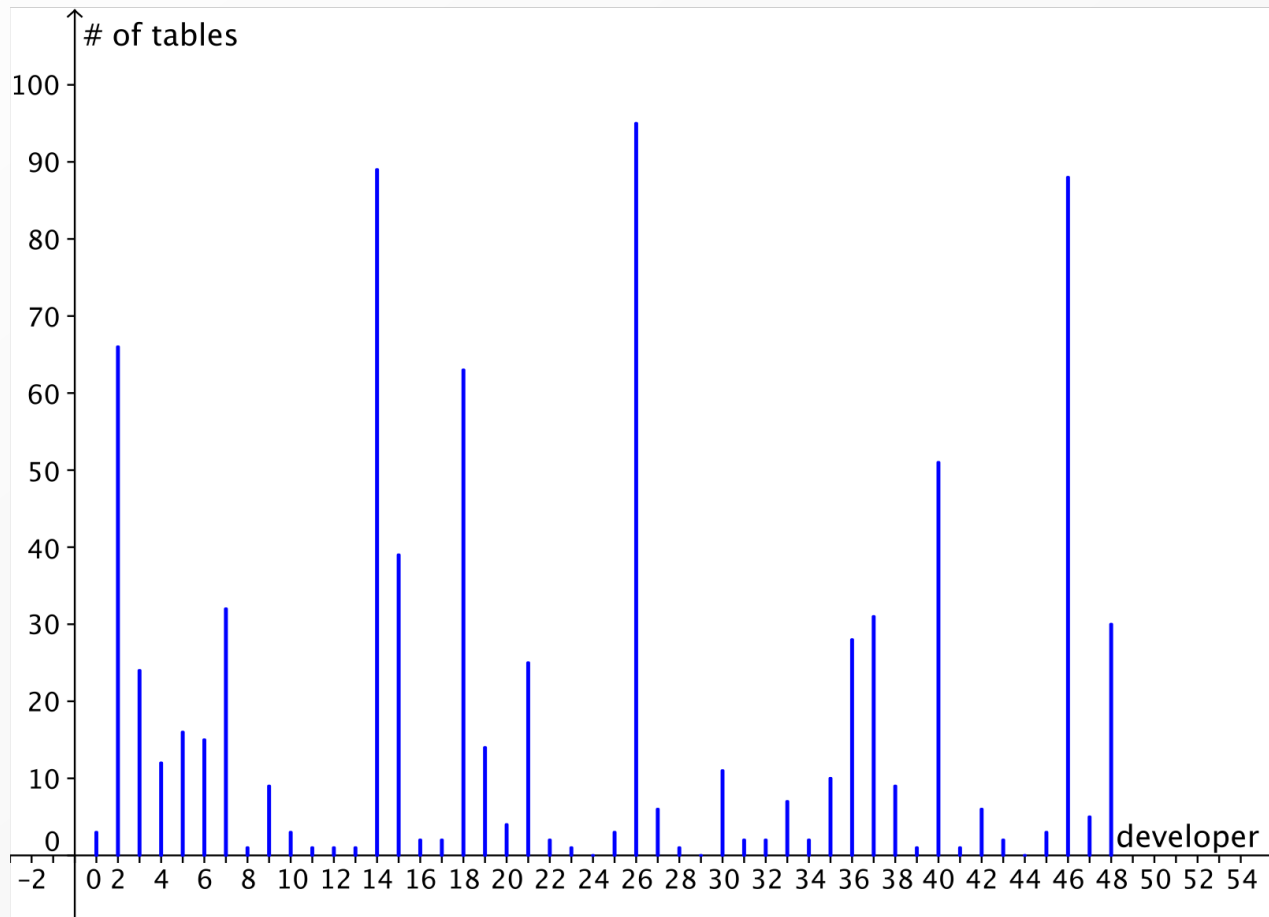
Historical schema analysis

creation/deletion of columns



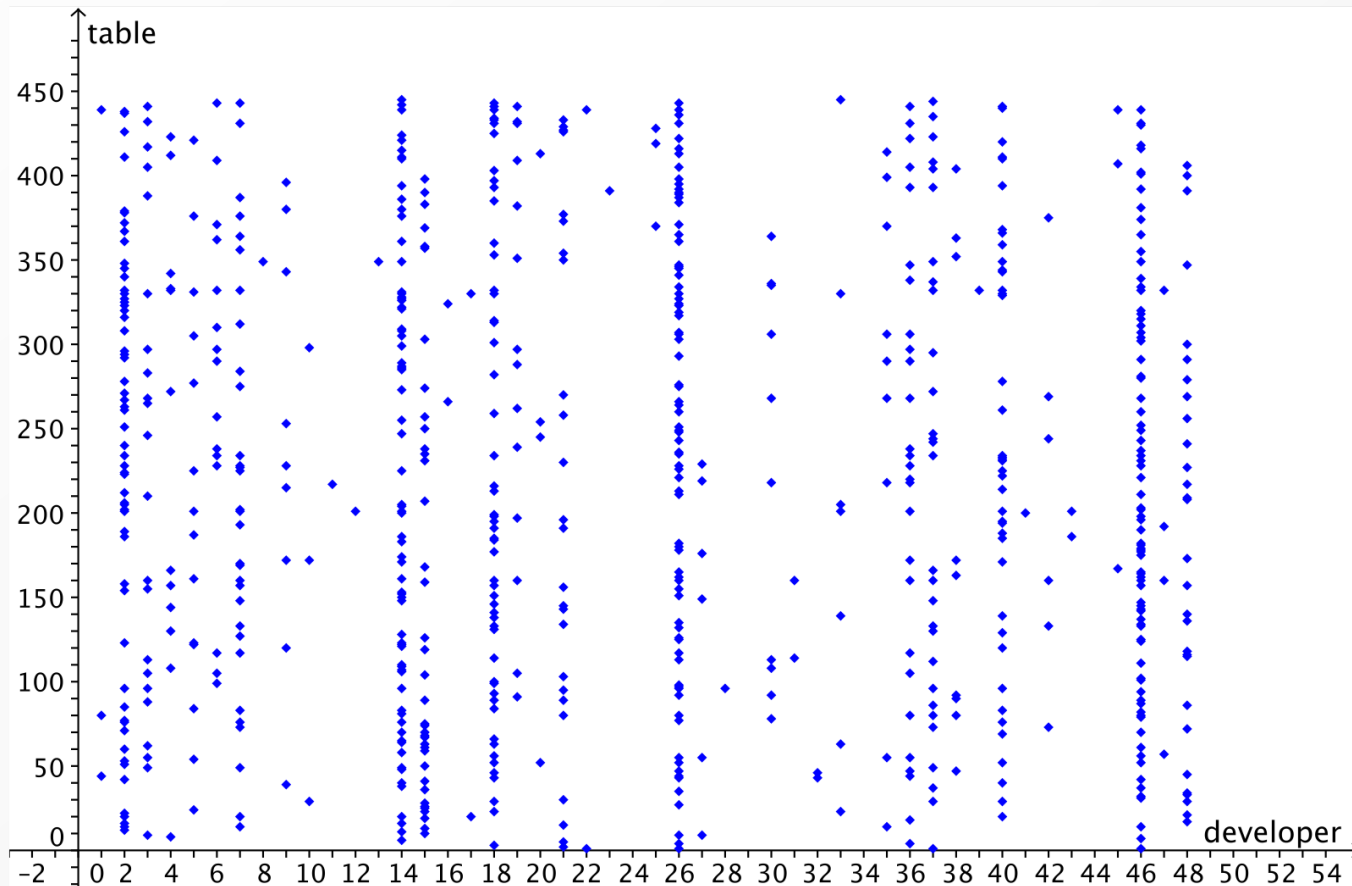
Historical schema analysis

how many tables did each developer touch?
(incl. creation, deletion, change)



Historical schema analysis

which table did each developer touch?



Conclusions of Episod I

Promising achievements at this stage

- Mining database schema history (prototype)
- Global historical schema extraction
- Basic 2D visualization within DB-MAIN

Expected improvements (from 09/2013)

- Improve/extend the proof-of-concept prototype into a more complete, robust tool suite
- Find better, more scalable visualizations
- Analyze a larger set of data-intensive systems

Episod II

DAHLIA

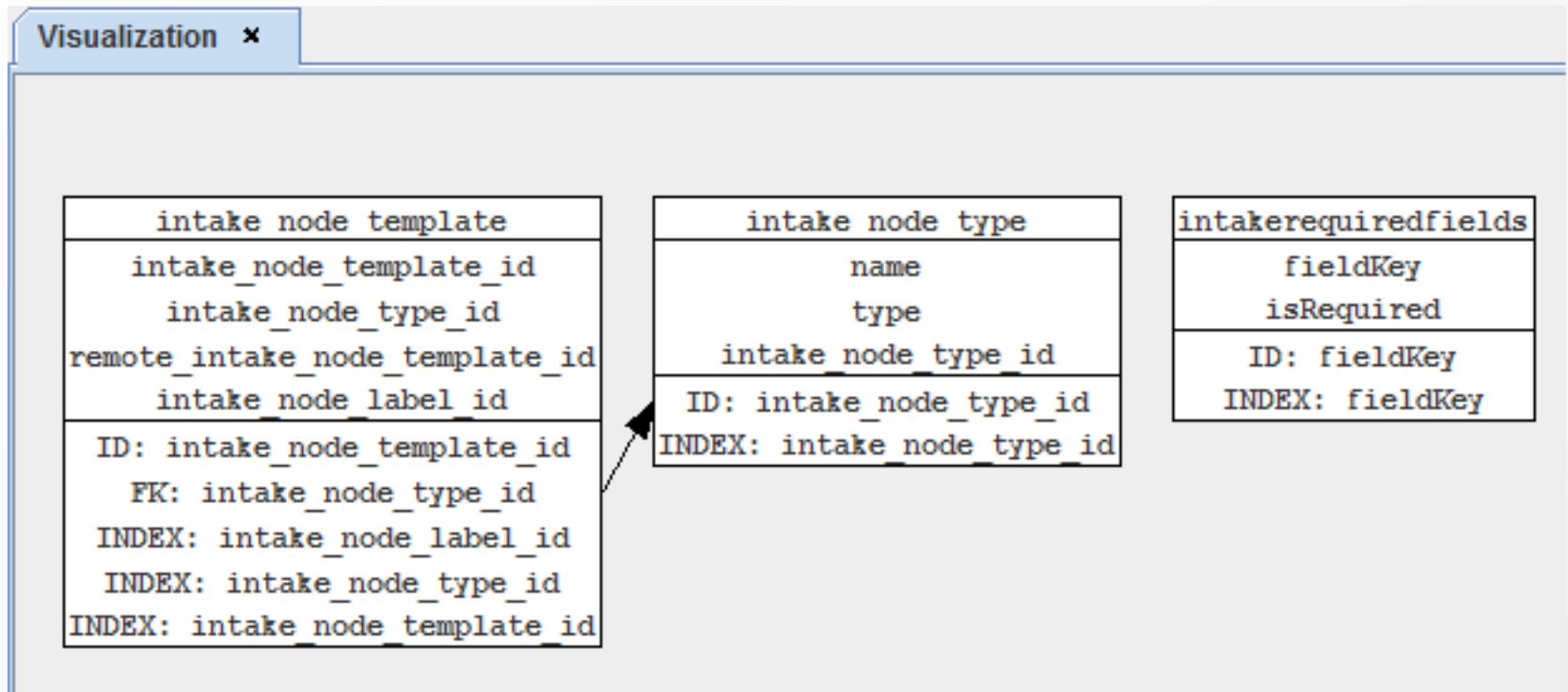
Episod II – DAHLIA(*)

(*) Database ScHema EvoLution Analysis
... in Highly Dynamic and Heterogeneous Systems
(like OSCAR)

DAHLIA = an interactive, visual analyzer of
database schema evolution

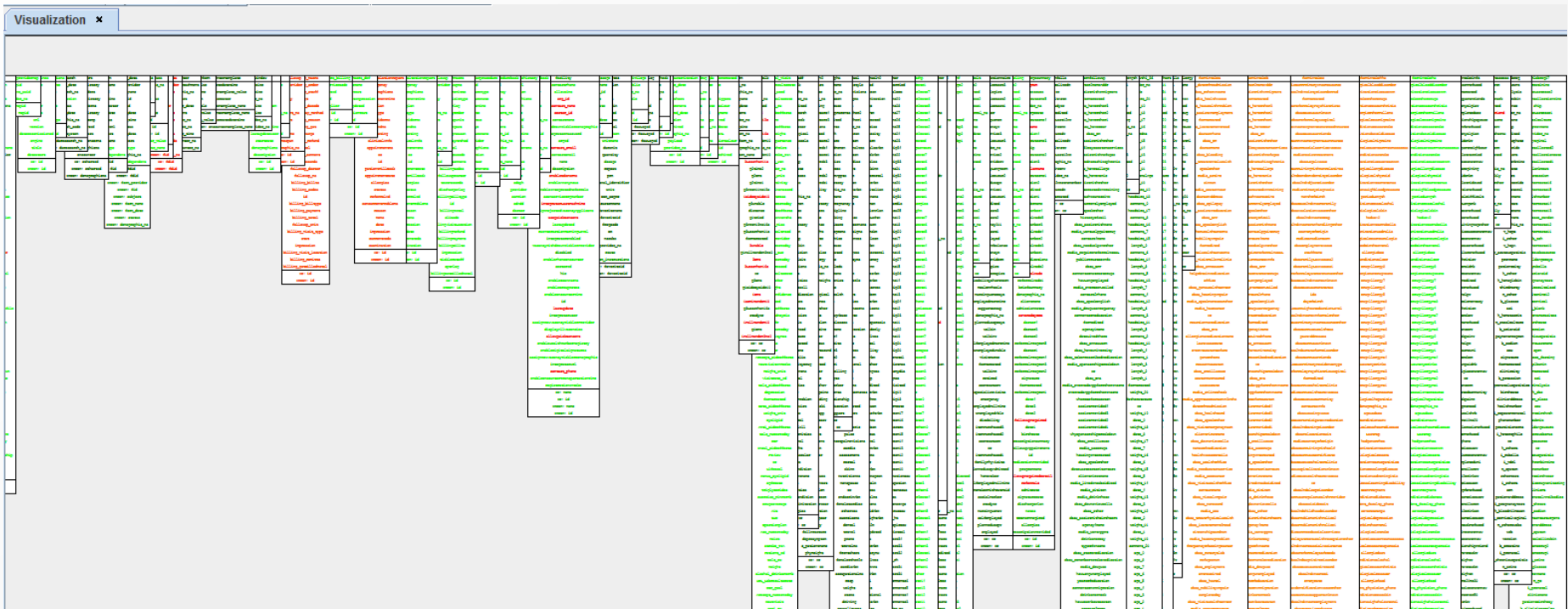
DAHLIA

visualizing a particular schema version in 2D



DAHLIA

visualizing an historical schema in 2D



DAHLIA

zoom on historical schema in 2D

secuserrole
role_no
role_name
activeyn
provider_no
orgcd
id
ID: role_name
provider_no
ID: id
INDEX: role_name
provider_no
INDEX: id

validations
name
id
maxLength
minLength
isDate
isNumeric
isTrue
regularExp
maxValue1
maxValue
minValue
ID: id
INDEX: id

waitinglist
position
listID
is_history
demographic_no
note
onListSince
id
ID: id
INDEX: listID
INDEX: id

vacancy_template
TEMPLATE_ID
WL_PROGRAM_ID
NAME
PROGRAM_ID
ACTIVE
ID: TEMPLATE_ID
INDEX: TEMPLATE_ID

waitinglistname
is_history
name
group_no
create_date
provider_no
ID
ID: ID
INDEX: ID

DAHLIA

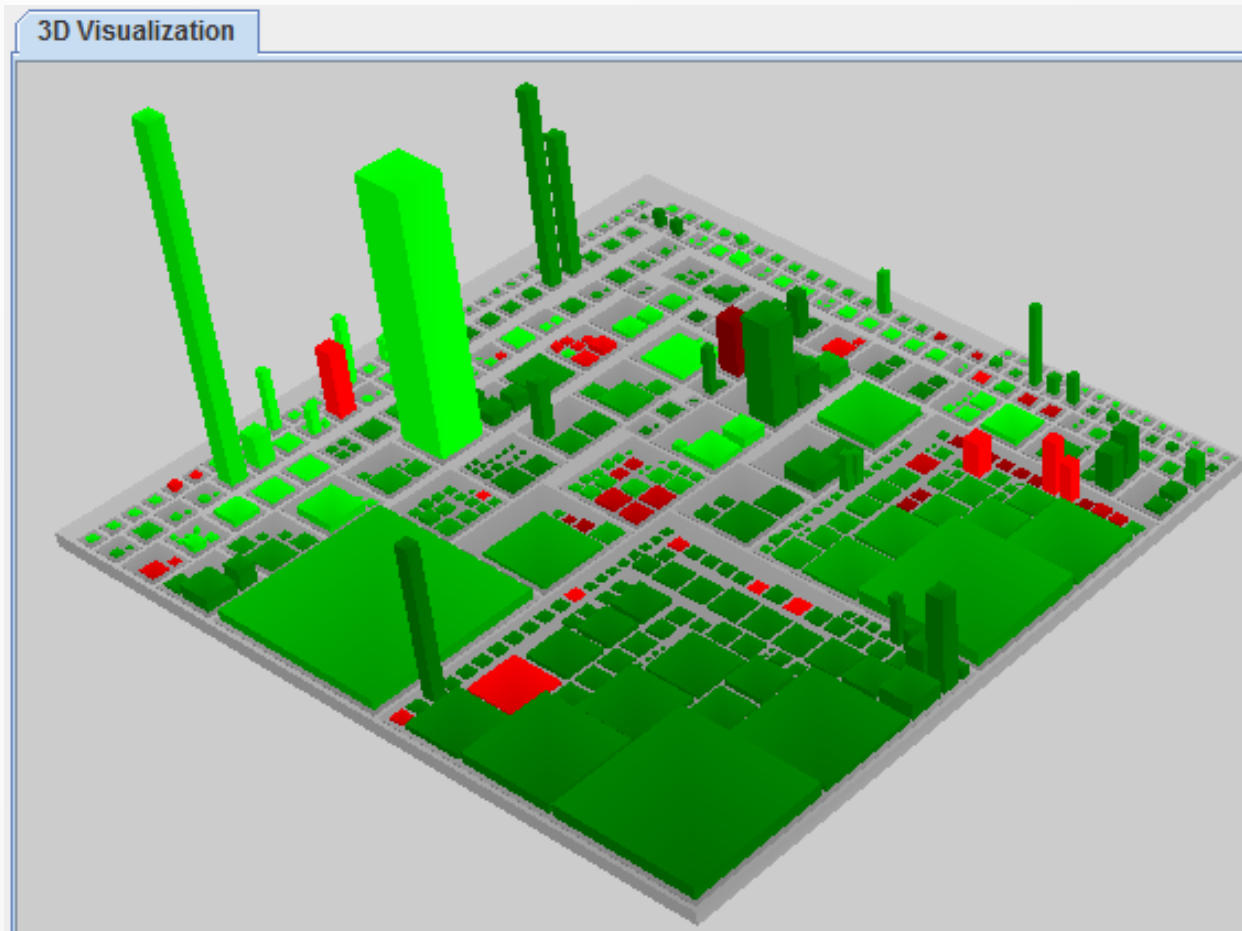
history of a particular schema object

secuserrole	validations	waitinglist	vacancy_template	waitinglistname
role_no	name	position	TEMPLATE_ID	is_history
role_name	id	listID	WL_PROGRAM_ID	name
activeyn	maxLength	is_history	NAME	group_no
provider_no	minLength	demographic_no	PROGRAM_ID	create_date

History: vacancy_template[PROGRAM_ID]		
Dates	Event	Committer
2012-04-29(13h35m09s)	CREATION	matthew.ma20110628@gmail.com
2013-02-15(23h13m22s)	DELETION	anniezhou91@gmail.com
2013-02-15(23h58m47s)	REAPPEARANCE	marc@mdumontier.com
2013-02-25(09h21m32s)	DELETION	marc@mdumontier.com

DAHLIA

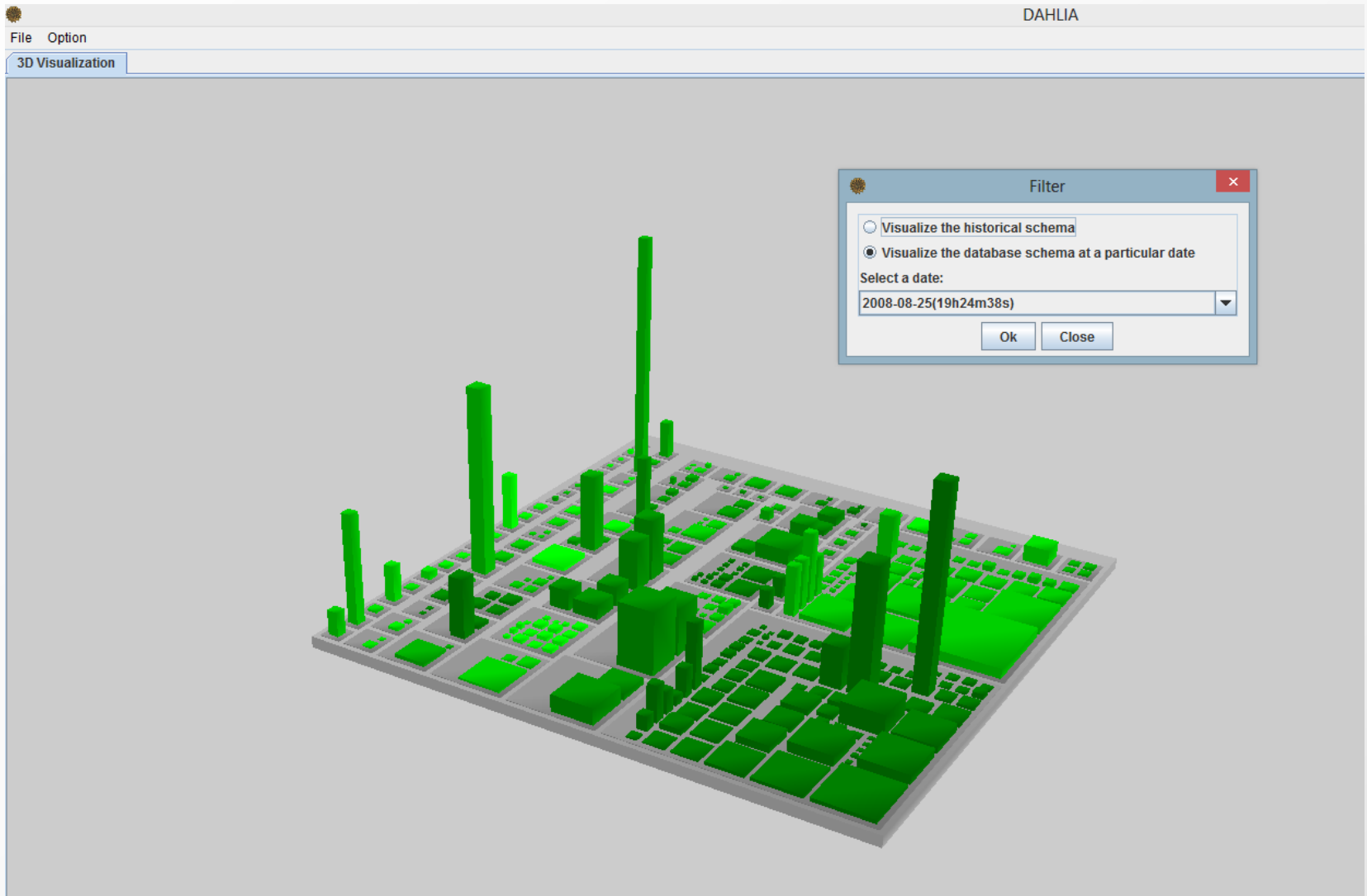
An historical schema in 3D (*)



(*) inspired by CodeCity (Wettel et al.)

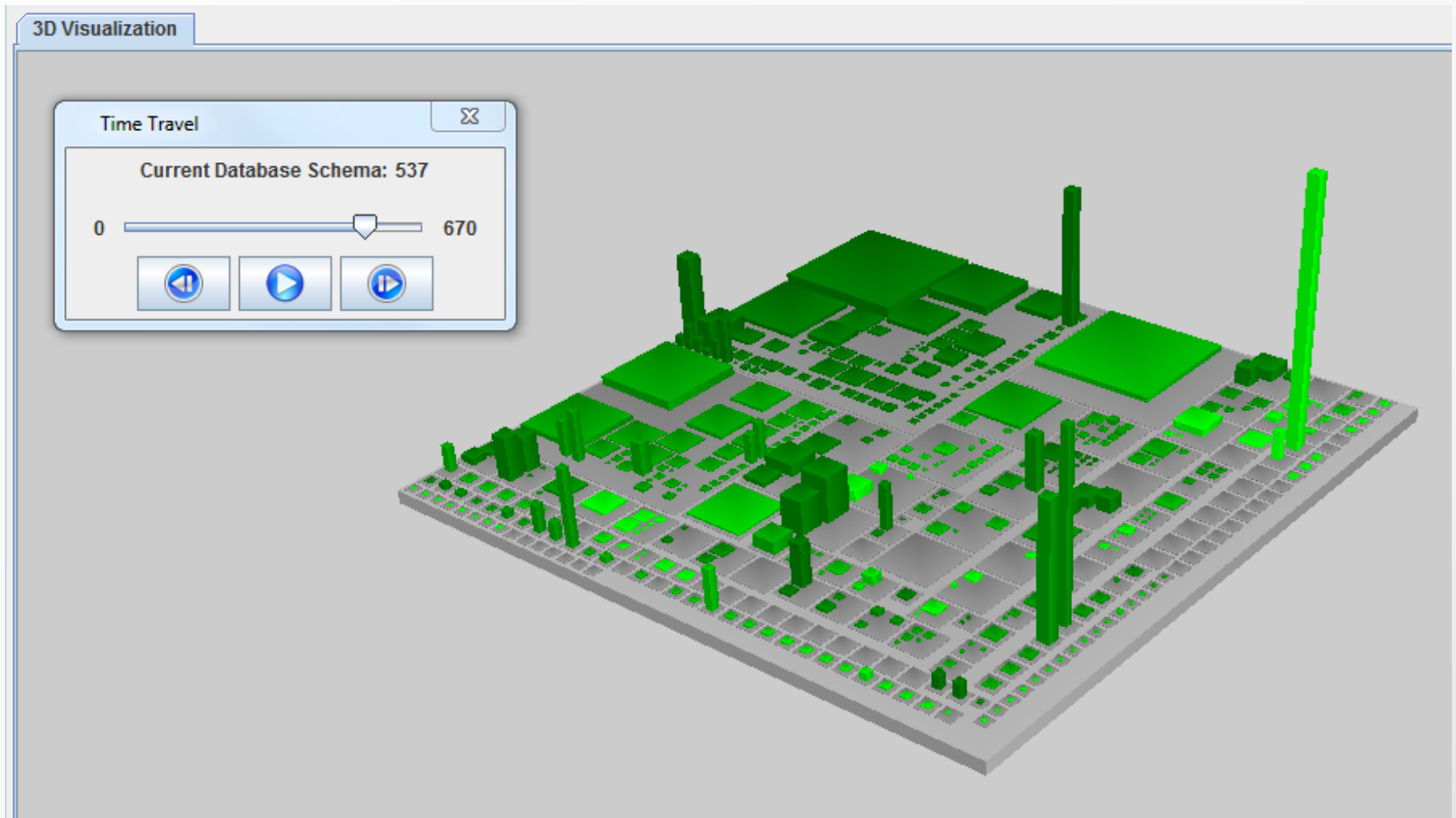
DAHLIA

visualizing a particular schema version in 3D



DAHLIA

travelling in time (back to the future)



DAHLIA

comparing two (non-)successive schema versions

DAHLIA

File Option

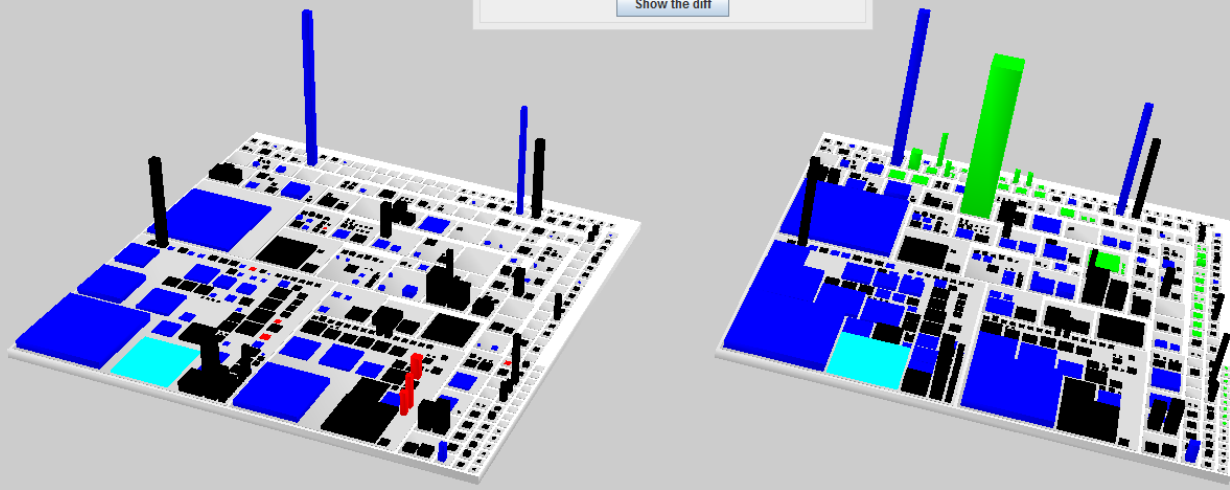
3D Visualization

Schema-Diff

Select the first date:
2011-08-03(21h45m14s)

Select the second date:
2013-06-27(15h38m29s)

Show the diff



document

docfilename	docfilename
number_of_pages	number_of_pages
doccreator	doccreator
responsible	responsible
document_no	document_no
appointment_no	appointment_no
publicl	publicl
reviewer	reviewer
docdesc	docdesc
contenttype	contenttype
updatedatetime	updatedatetime
reviewdatetime	reviewdatetime
docxml	docxml
program_id	program_id
source	source
status	status
docSubClass	docSubClass
observationdate	observationdate
docClass	docClass
doctype	doctype
ID: document_no	sourceFacility
INDEX: document_no	contentdatetime

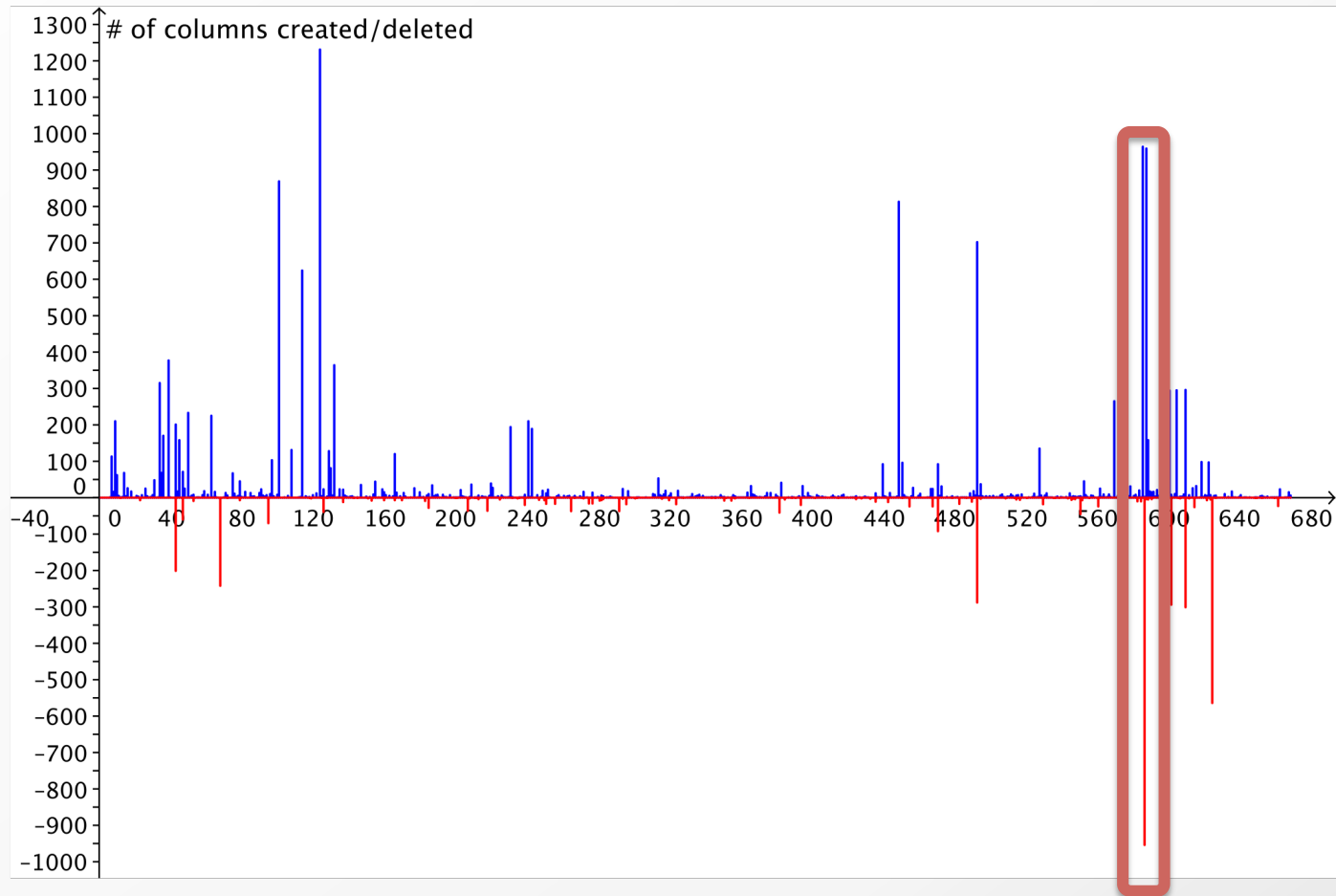
ID: document_no
INDEX: document_no

ID: document_no
INDEX: document_no

Online demo

DAHLIA

let's go back to a previous slide...



DAHLIA

table renaming or table deletion?

deleted on
24/03/2011

specshis
od_add
od_cyl
type
os_sph
os_add
os_cyl
appointment_no
update_time
od_prism
id
os_prism
demographic_no
os_axis
status
od_axis
date
provider
doctor
od_sph

eyeformspecshistory
odcyl
odprism
demographicno
ossph
type
osadd
oscyl
updatetime
odaxis
appointmentno
id
status
osaxis
date
odsph
provider
osprism
odadd
doctor

created on
24/03/2011

DAHLIA

Identifying table/column renamings

Visualization x Renaming detection: All tables[1] x

Add... Remove... Apply the renaming...

Renamed Table	Renaming Table	Date
team_bed	room_bed_historical	126
intake_instance	intake	137
intake_label	intake_node_template	137
system_message	systemmessage	283
eyeformconsulationrep...	eyeformconsultationre...	468
eyeform_followup	eyeformfollowup	471
eyeform_macro	eyeformmacro	471
ocularprocedurehis	eyeformocularprocedure	471
procedurebook	eyeformtestbook	471
specshis	eyeformspecshistory	471
testbookrecord	eyeformprocedurebook	471
remotedataretrievallog	remotedatalog	515
formintakehhx	formintakehx	610

DAHLIA

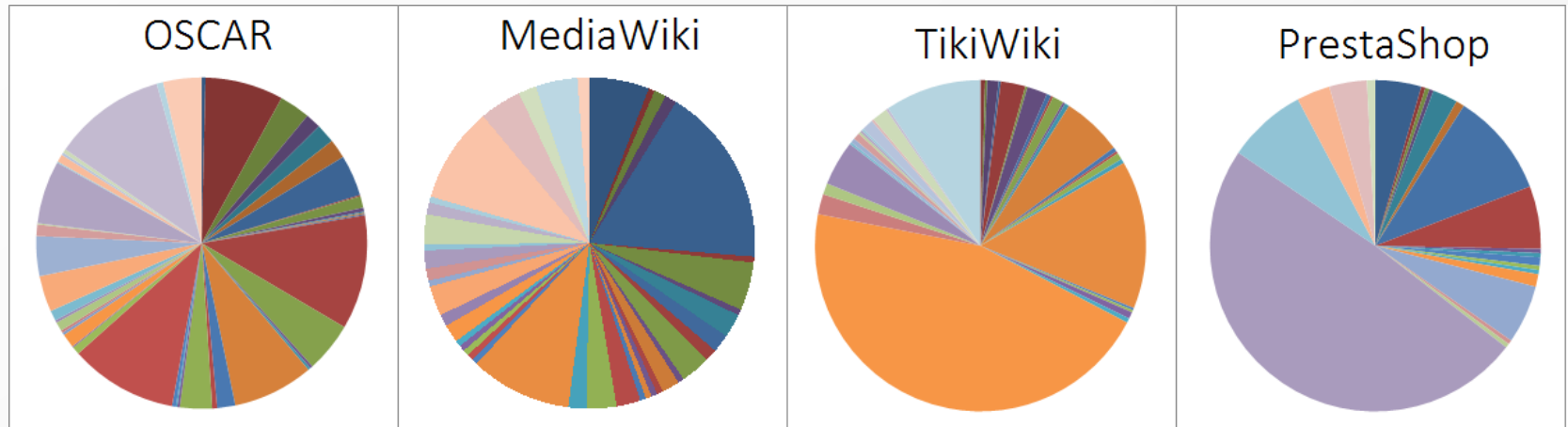
Identifying the most frequent schema changes

Project	Studied Period	#Tables	#Columns	#Versions
OSCAR	07/2003 → 06/2013	88 → 445	2443 → 13364	670
MediaWiki	05/2003 → 08/2013	17 → 50	100 → 337	359
TikiWiki	12/2006 → 07/2013	206 → 248	1525 → 1974	623
PrestaShop	12/2008 → 09/2012	113 → 157	564 → 890	229

Change type (%)	Oscar	MediaWiki	TikiWiki	PrestaShop
Adding table	9.7	9.6	19.4	19.6
Dropping table	1.5	3.9	3.4	1.7
Adding column	28.7	15.7	14.7	14.9
Dropping column	3.5	5.4	2.5	2.6
Adding ID	0.8	2.5	1.6	2.7
Dropping ID	0.3	0.9	1.4	0.7
Adding FK	0.05	0	0	0
Dropping FK	0.2	0	0	0
Adding index	2.3	12.5	5.4	14.7
Dropping index	0.4	4.3	2.3	2.3
Changing column datatype	41.6	44.1	48.8	39.6
Renaming table	0.2	0.11	0.1	0.1
Renaming column	10.6	0.9	2.5	1

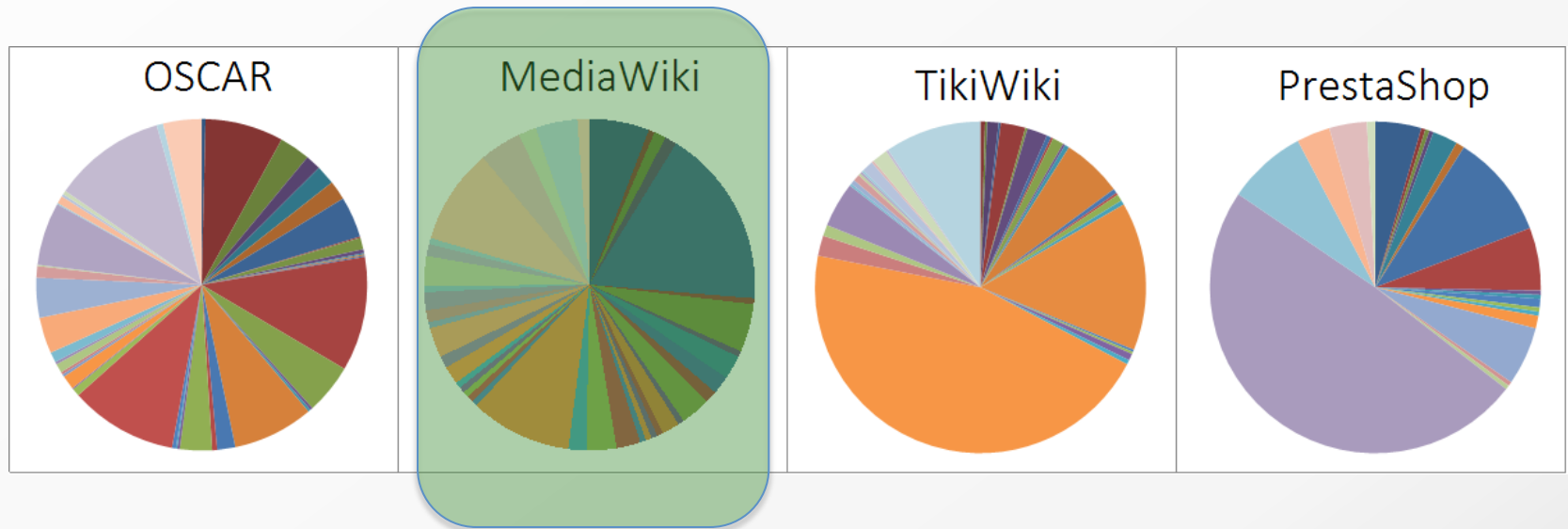
DAHLIA

Identifying database schema experts among the developers



DAHLIA

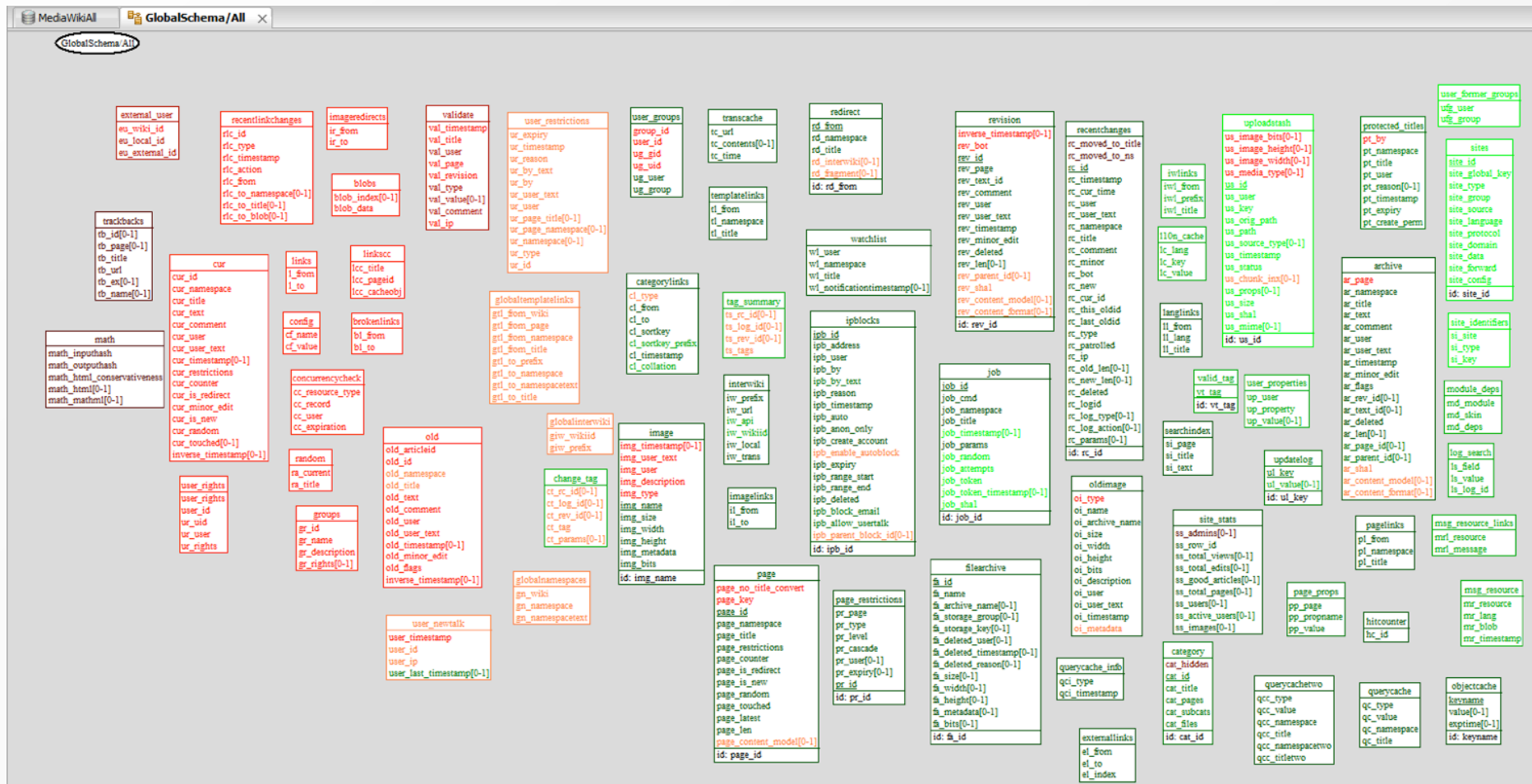
Identifying database schema experts among the developers



Historical schema

viewed within DB-MAIN

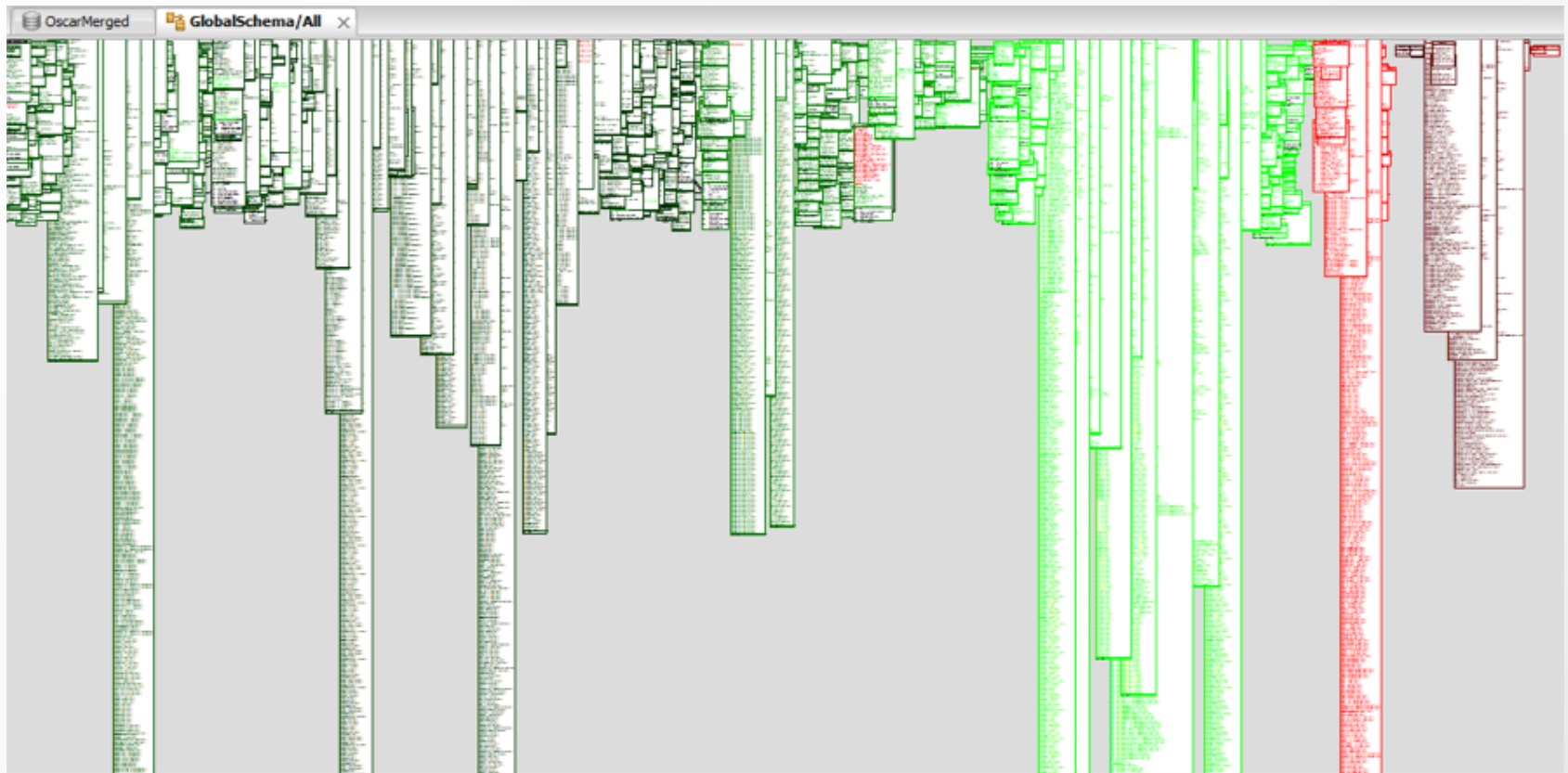
Historical schema of MediaWiki – the Wikipedia database (2003-2013)



Historical schema

viewed within DB-MAIN

Historical schema of OSCAR (2003-2013)



Conclusions of Episod II

Analyzing database schema evolution history

- Mining database schema history with DAHLIA
- More advanced visualization and interaction
- Interesting statistics for a few systems, beyond OSCAR

Expected future improvements (from 08/2014)

- Analyzing a larger set of data-intensive systems
- Analyzing database usage (i.e., database queries in programs)

Episod III

DAHLIA+

Episod III – DAHLIA+

Analyzing Database Usage

... in Highly Dynamic and Heterogeneous Java Systems
(like OSCAR)

Goals:

Extract the database queries (SQL) occurring in the source code of the programs (Java)

Analyze those queries to derive useful information, such as accessed tables and columns

Which tables are accessed in this query?

```
SELECT appointment.date, patient.firstname, patient.lastname  
FROM appointment  
JOIN patient ON appointment.patientid = patient.id  
WHERE appointment.date = '2016-05-11'
```


Which columns are accessed in this query?

```
SELECT appointment.date, patient.firstname, patient.lastname  
FROM appointment  
JOIN patient ON appointment.patientid = patient.id  
WHERE appointment.date = '2016-05-11'
```

... and in this one?

```
select billingser0_.billingservice_no as billings1_373_,
billingser0_.anaesthesia as anaesthe2_373_,
billingser0_.billingservice_date as billings3_373_, billingser0_.description
as descript4_373_, billingser0_.displaystyle as displays5_373_,
billingser0_.gstFlag as gstFlag373_, billingser0_.percentage as
percentage373_, billingser0_.region as region373_,
billingser0_.service_code as service9_373_,
billingser0_.service_compositecode as service10_373_,
billingser0_.sliFlag as sliFlag373_, billingser0_.specialty as specialty373_,
billingser0_.termination_date as termina13_373_, billingser0_.value as
value373_ from billingservice billingser0_ where
billingser0_.service_code='A001A' and
billingser0_.billingservice_date=(select
MAX(billingser1_.billingservice_date) from billingservice billingser1_
where billingser1_.billingservice_date<='2014-10-28' and
billingser1_.service_code='A001A');
```

... and in this one?

```
select appointmen0_.appointment_no as appointm1_89_0_, demographi1_.demographic_no as demograp1_27_1_,
appointmen0_.appointment_date as appointm2_89_0_, appointmen0_.billing as billing89_0_, appointmen0_.bookingSource as
bookingS4_89_0_, appointmen0_.createdatetime as createda5_89_0_, appointmen0_.creator as creator89_0_,
appointmen0_.creatorSecurityId as creatorS7_89_0_, appointmen0_.demographic_no as demograp8_89_0_, appointmen0_.end_time as
end9_89_0_, appointmen0_.imported_status as imported10_89_0_, appointmen0_.lastupdateuser as lastupd11_89_0_,
appointmen0_.location as location89_0_, appointmen0_.name as name89_0_, appointmen0_.notes as notes89_0_,
appointmen0_.program_id as program15_89_0_, appointmen0_.provider_no as provider16_89_0_, appointmen0_.reason as reason89_0_,
appointmen0_.reasonCode as reasonCode89_0_, appointmen0_.remarks as remarks89_0_, appointmen0_.resources as resources89_0_,
appointmen0_.start_time as start21_89_0_, appointmen0_.status as status89_0_, appointmen0_.style as style89_0_, appointmen0_.type as
type89_0_, appointmen0_.updatedatetime as updated25_89_0_, appointmen0_.urgency as urgency89_0_, demographi1_.title as title27_1_,
demographi1_.first_name as first3_27_1_, demographi1_.last_name as last4_27_1_, demographi1_.sex as sex27_1_,
demographi1_.month_of_birth as month6_27_1_, demographi1_.date_of_birth as date7_27_1_, demographi1_.year_of_birth as year8_27_1_,
demographi1_.address as address27_1_, demographi1_.city as city27_1_, demographi1_.province as province27_1_, demographi1_.postal
as postal27_1_, demographi1_.email as email27_1_, demographi1_.phone as phone27_1_, demographi1_.phone2 as phone15_27_1_,
demographi1_.myOscarUserName as myOscar16_27_1_, demographi1_.hin as hin27_1_, demographi1_.ver as ver27_1_,
demographi1_.hc_type as hc19_27_1_, demographi1_.hc_renew_date as hc20_27_1_, demographi1_.roster_status as roster21_27_1_,
demographi1_.patient_status as patient22_27_1_, demographi1_.patient_status_date as patient23_27_1_, demographi1_.date_joined as
date24_27_1_, demographi1_.chart_no as chart25_27_1_, demographi1_.provider_no as provider26_27_1_, demographi1_.end_date as
end27_27_1_, demographi1_.eff_date as eff28_27_1_, demographi1_.roster_date as roster29_27_1_, demographi1_.roster_termination_date
as roster30_27_1_, demographi1_.roster_termination_reason as roster31_27_1_, demographi1_.pcn_indicator as pcn32_27_1_,
demographi1_.family_doctor as family33_27_1_, demographi1_.alias as alias27_1_, demographi1_.previousAddress as previou35_27_1_,
demographi1_.children as children27_1_, demographi1_.sourceOfIncome as sourceO37_27_1_, demographi1_.citizenship as citizen38_27_1_,
demographi1_.sin as sin27_1_, demographi1_.anonymous as anonymous27_1_, demographi1_.spoken_lang as spoken41_27_1_,
demographi1_.official_lang as official42_27_1_, demographi1_.lastUpdateUser as lastUpd43_27_1_, demographi1_.lastUpdateDate as
lastUpd44_27_1_, demographi1_.newsletter as newsletter27_1_, demographi1_.country_of_origin as country46_27_1_, (select lst.description
from lst_gender lst where lst.code=demographi1_.sex) as formula21_1_, (select d.merged_to from demographic_merged d where d.deleted
= 0 and d.demographic_no = demographi1_.demographic_no) as formula22_1_, (select count(*) from admission a where
a.client_id=demographi1_.demographic_no and a.admission_status='current' and a.program_id in (select p.id from program p where
p.type='Bed' )) as formula23_1_, (select count(*) from health_safety h where h.demographic_no=demographi1_.demographic_no) as
formula24_1_ from appointment appointmen0_, demographic demographi1_ where
appointmen0_.demographic_no=demographi1_.demographic_no and demographi1_.hin<>' ' and
appointmen0_.appointment_date>='2014-10-23' and appointmen0_.appointment_date<='2014-10-23' and
(upper(demographi1_.province)='ONTARIO' or demographi1_.province='ON') group by demographi1_.demographic_no order by
demographi1_.last_name;
```

The problem of dynamically generated queries

The problem of dynamically generated queries

SQL queries are not always written in the programs

The problem of dynamically generated queries

SQL queries are not always written in the programs

SQL queries are most often generated by the programs

The problem of dynamically generated queries

SQL queries are not always written in the programs

SQL queries are most often generated by the programs

```
1  public class ProviderMgr {
2      private Statement st;
3      private ResultSet rs;
4      private boolean ordering;
5      ...
6
7      public void executeQuery(String x, String y){
8          String sql = getQueryStr(x);
9          if(ordering)
10             sql += " order by " + y;
11             rs = st.execute(sql);
12     }
13
14     public String getQueryStr(String str){
15         return "select * from " + str;
16     }
17
18     public Provider[] getAllProviders(){
19         String tableName = "Provider";
20         String columnName;
21         if (...)
22             columnName = "provider_id";
23         else
24             columnName = "provider_name";
25         executeQuery(tableName, columnName);
26         ...
27     }
28 }
```

JDBC

The problem of dynamically generated queries

SQL queries are not always written in the programs

SQL queries are most often generated by the programs

```
1  private static Session session;  
2  ...  
3  
4  public static void saveCustomer(Customer myCustomer){  
5      saveObject(myCustomer);  
6  }  
7  
8  public static void saveObject(Object o){  
9      session.save(o);  
10 }
```

Hibernate

The problem of dynamically generated queries

SQL queries are not always written in the programs

SQL queries are most often generated by the programs

```
1  EntityManager entityManager = entityManagerFactory.  
    createEntityManager();  
2  entityManager.getTransaction().begin();  
3  Order order= createNewOrder();  
4  entityManager.persist( order );  
5  entityManager.getTransaction().commit();  
6  entityManager.close();
```

JPA

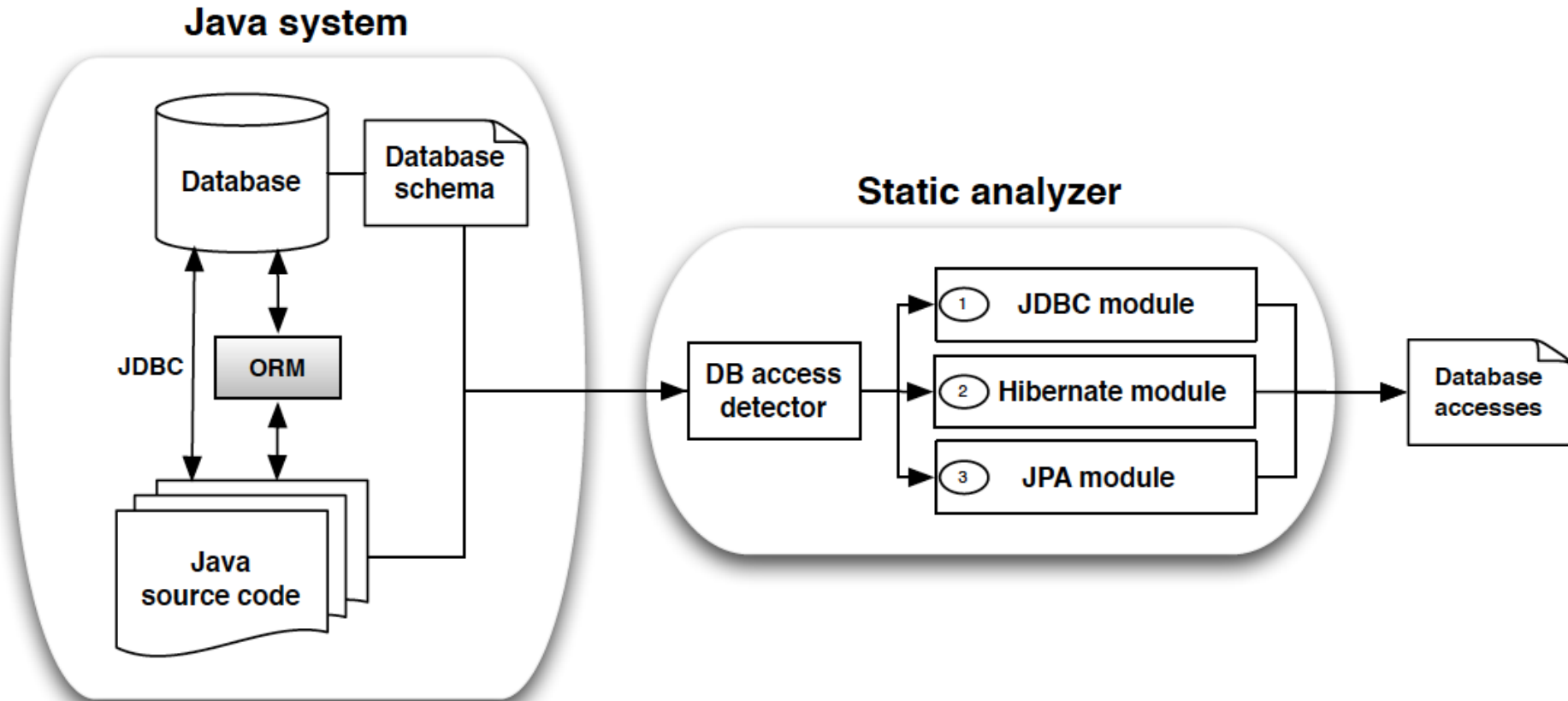
Research question

How can we extract and analyze the (generated) database queries from the source code of dynamic programs?

Research question

*How can we extract and analyze the (generated) **SQL** database queries from the source code of dynamic **Java** programs?*

SQL query extraction and analysis



SQL query extraction (JDBC)

```
1 public class ProviderMgr {
2     private Statement st;
3     private ResultSet rs;
4     private boolean ordering;
5     ...
6
7     public void executeQuery(String x, String y){
8         String sql = getQueryStr(x);
9         if(ordering)
10             sql += " order by " + y;
11         rs = st.execute(sql);
12     }
13
14     public String getQueryStr(String str){
15         return "select * from " + str;
16     }
17
18     public Provider[] getAllProviders(){
19         String tableName = "Provider";
20         String columnName;
21         if (...)
22             columnName = "provider_id";
23         else
24             columnName = "provider_name";
25         executeQuery(tableName, columnName);
26         ...
27     }
28 }
```



3 possible SQL queries at line 11:

select * from Provider

select * from Provider
order by provider_id

select * from Provider
order by provider_name

SQL query extraction (Hibernate)

```
1  private static Session session;  
2  ...  
3  
4  public static void saveCustomer(Customer myCustomer){  
5      saveObject(myCustomer);  
6  }  
7  
8  public static void saveObject(Object o){  
9      session.save(o);  
10 }
```

+ class Customer is mapped with table CLIENT



SQL query at line 9 (among others):

```
insert into CLIENT values (...)
```

SQL query extraction (JPA)

```
1  EntityManager entityManager = entityManagerFactory.  
    createEntityManager();  
2  entityManager.getTransaction().begin();  
3  Order order= createNewOrder();  
4  entityManager.persist( order );  
5  entityManager.getTransaction().commit();  
6  entityManager.close();
```

+ class Order is mapped with table ORDERS

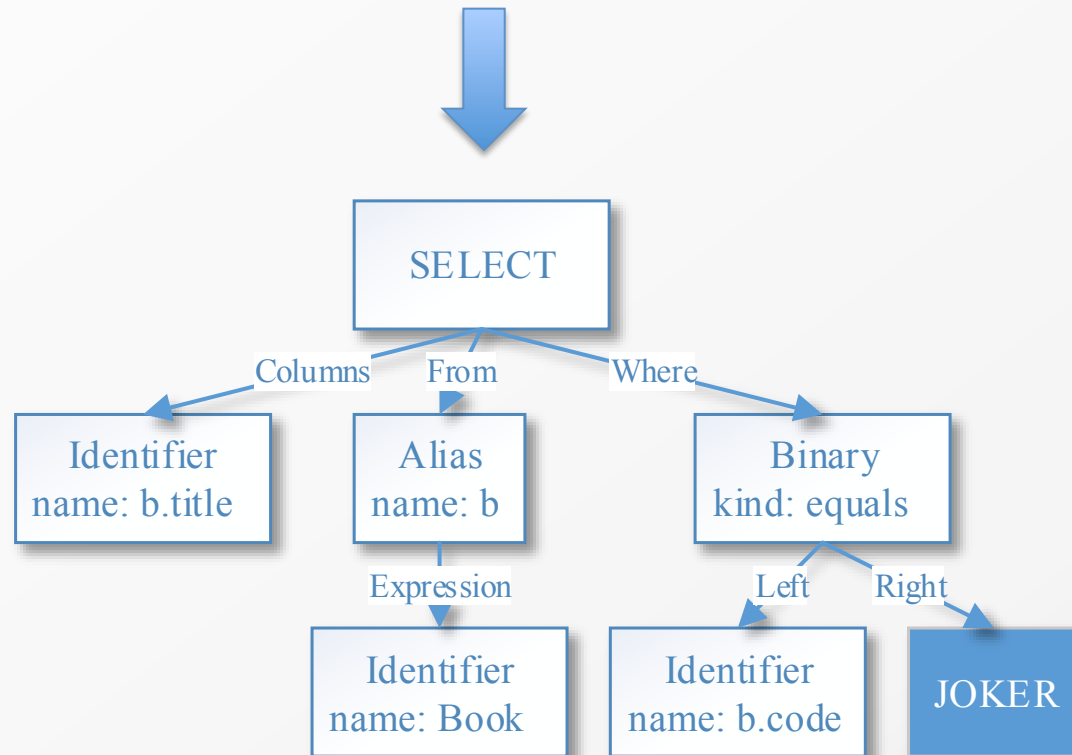


SQL query at line 4 :

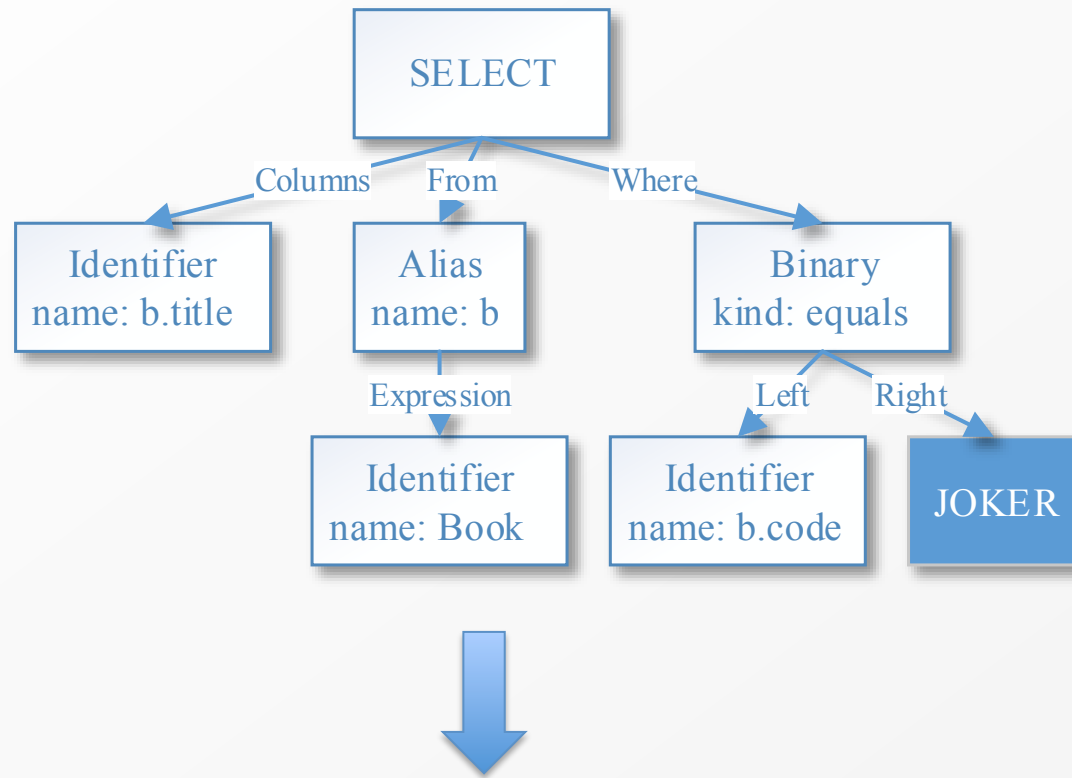
insert into ORDERS values (...)

SQL parsing

```
SELECT b.title FROM Book b
WHERE b.code=:code
```



SQL analysis



Accessed table: Book

Accessed columns: Book.title, Book.code

Evaluation

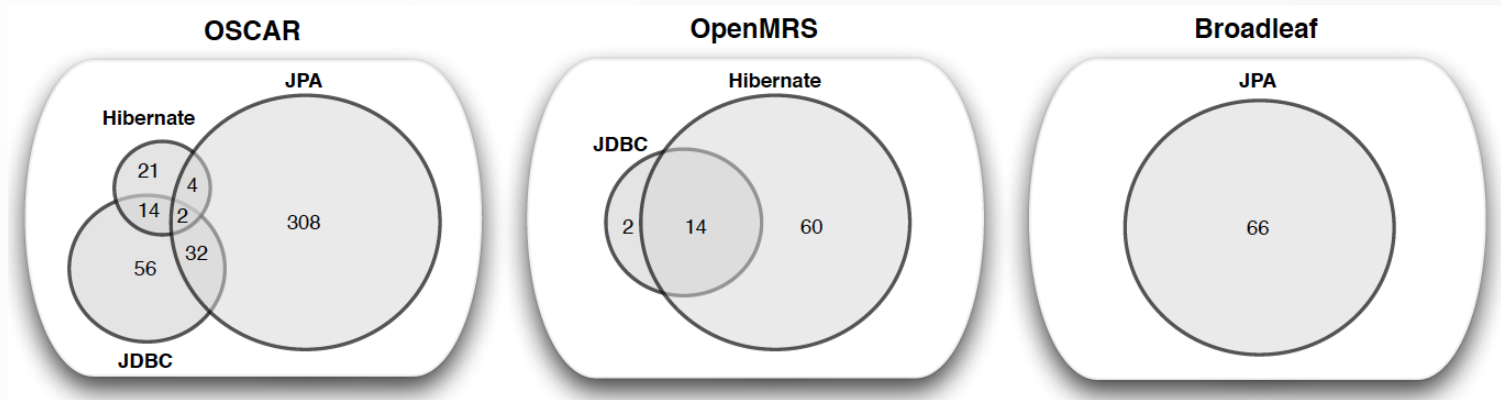
Metrics about the case studies

System	Description	LOC	Tables	Columns
Oscar	Medical record system	2 054 940	480	13 822
OpenMRS	Medical record system	301 232	88	951
Broadleaf	E-commerce framework	254 027	179	965

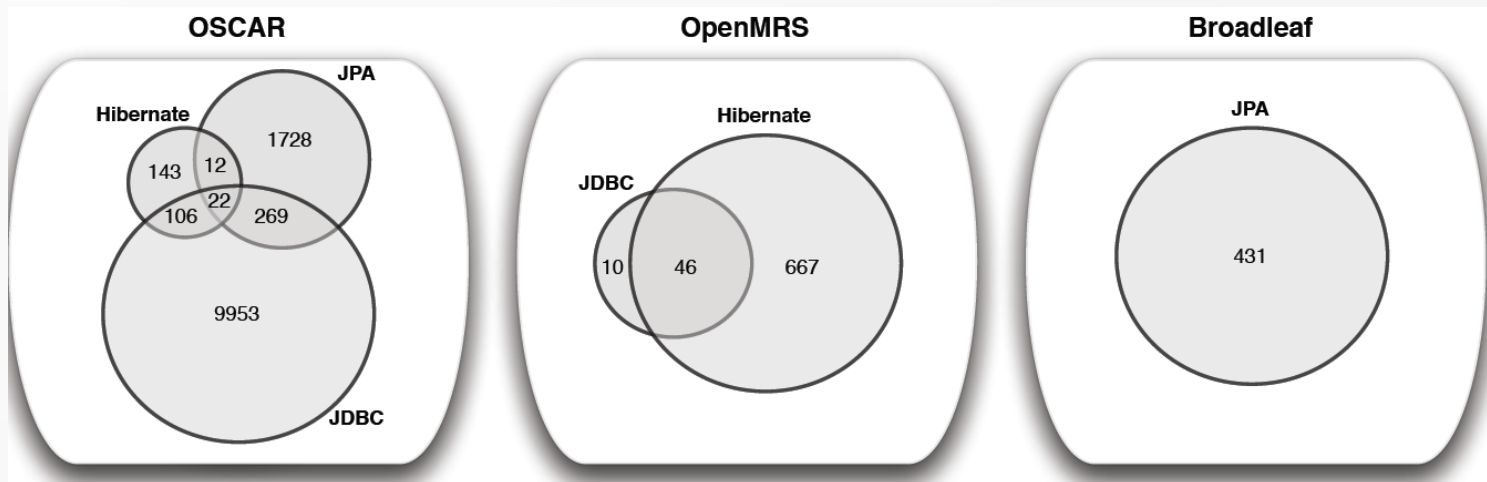
System	Database Accesses		
	JDBC	Hibernate	JPA
Oscar	123 661	727	31 729
OpenMRS	77	687	0
Broadleaf	0	0	930

Evaluation

Distribution of tables accessed per technology



Distribution of columns accessed per technology



Evaluation

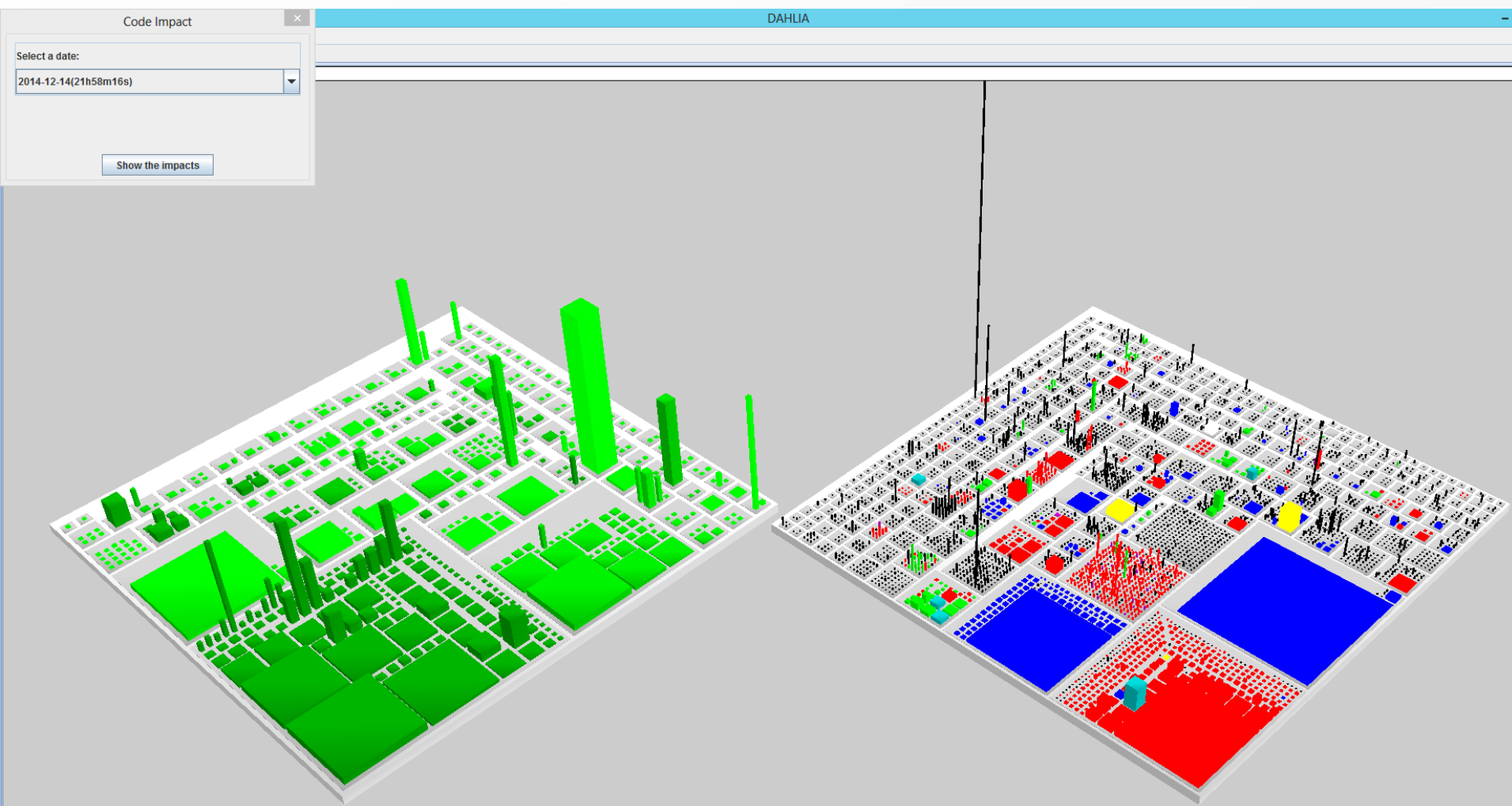
Precision of the query extraction process (computed based on the test cases)

System	Technologies		Total
	JDBC	Hibernate/JPA	
Oscar	14/17	656/689	95.2%
OpenMRS	8/8	86/99	86.8%
Broadleaf	-	29/29	100%

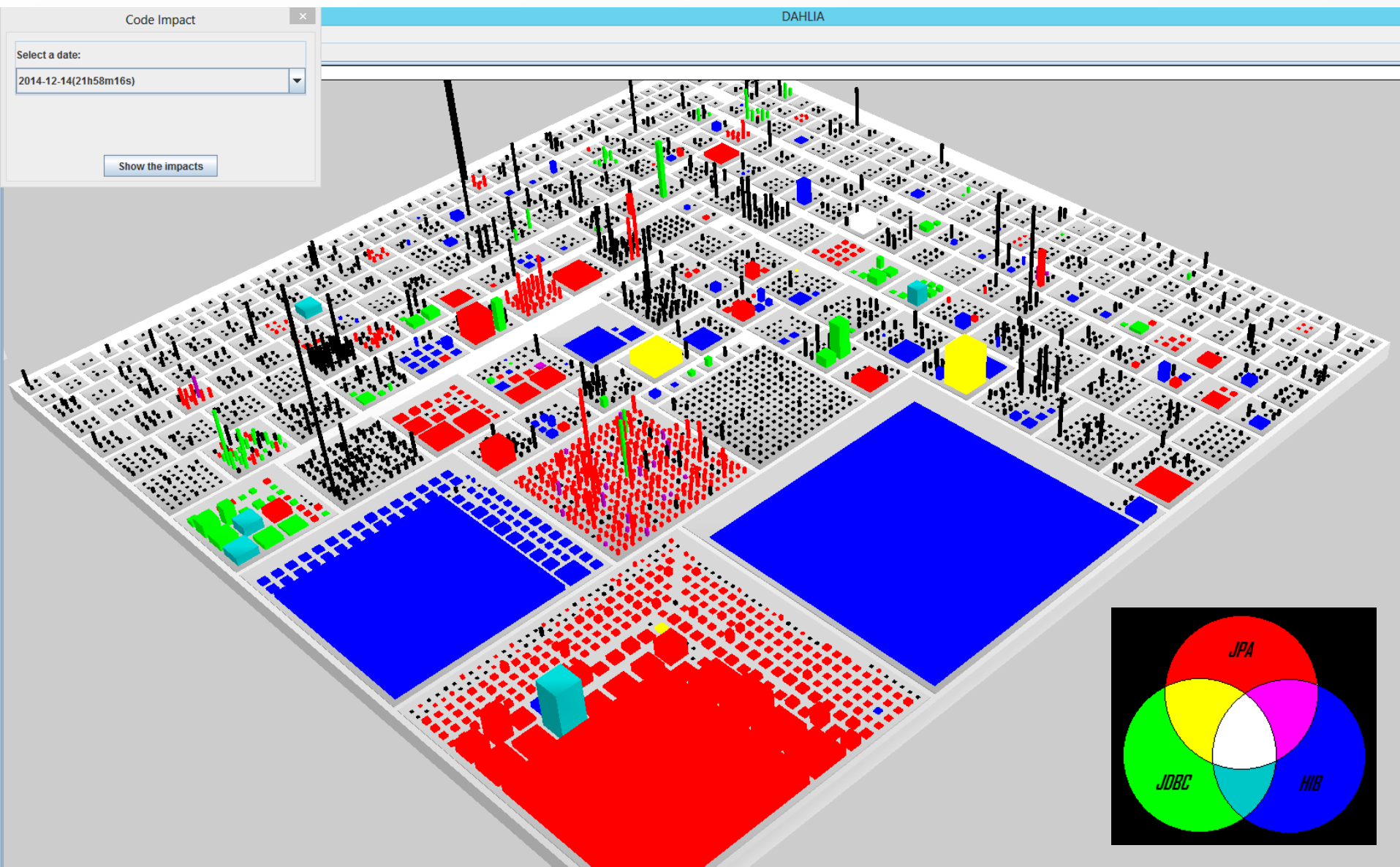
Recall of the query extraction process (computed based on the test cases)

System	Technologies		Total
	JDBC	Hibernate/JPA	
Oscar	1681/2038	892/1558	71,5%
OpenMRS	31/41	268/322	82,4%
Broadleaf	-	94/95	99%

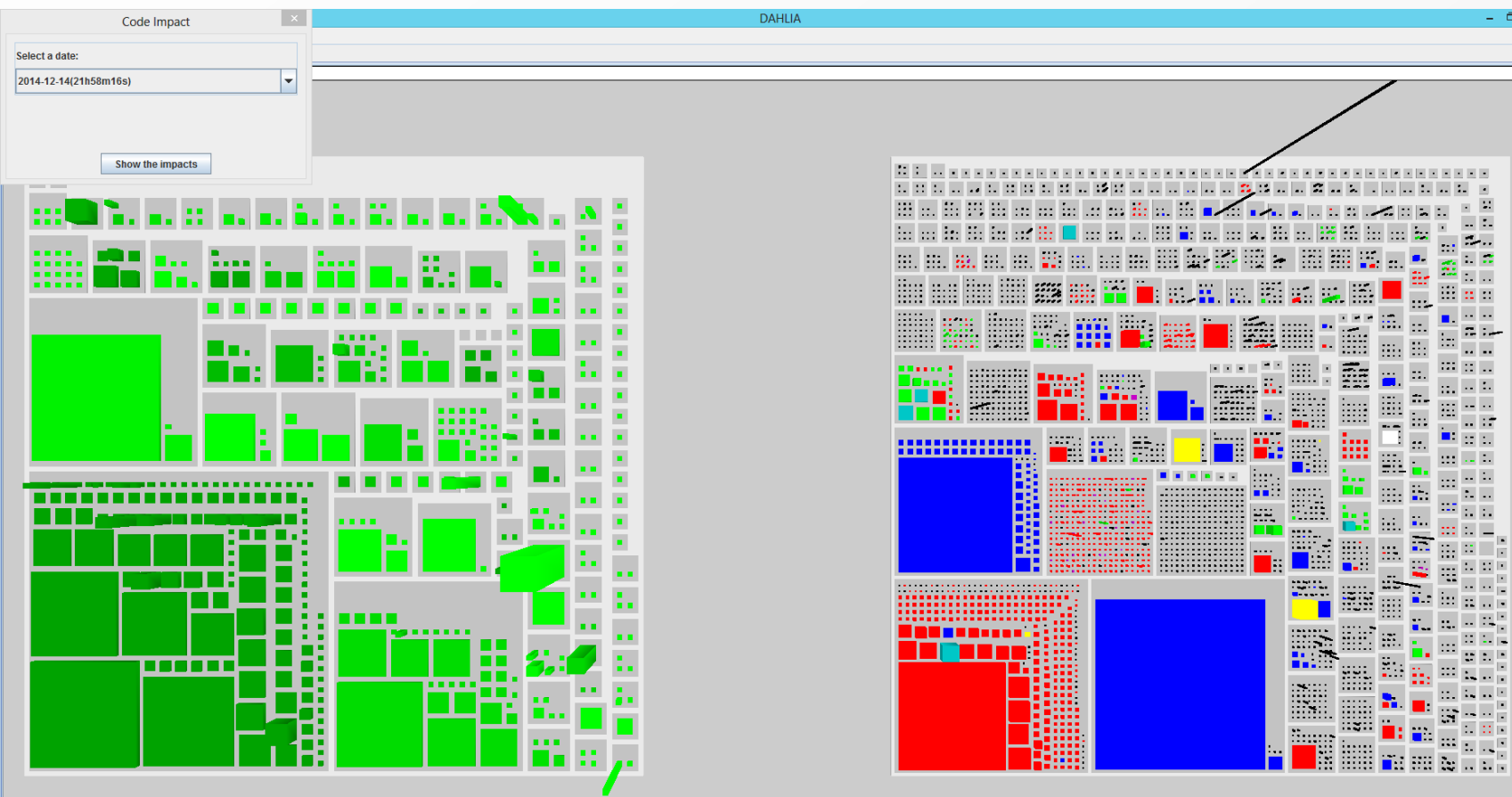
Visualization within DAHLIA (OSCAR)



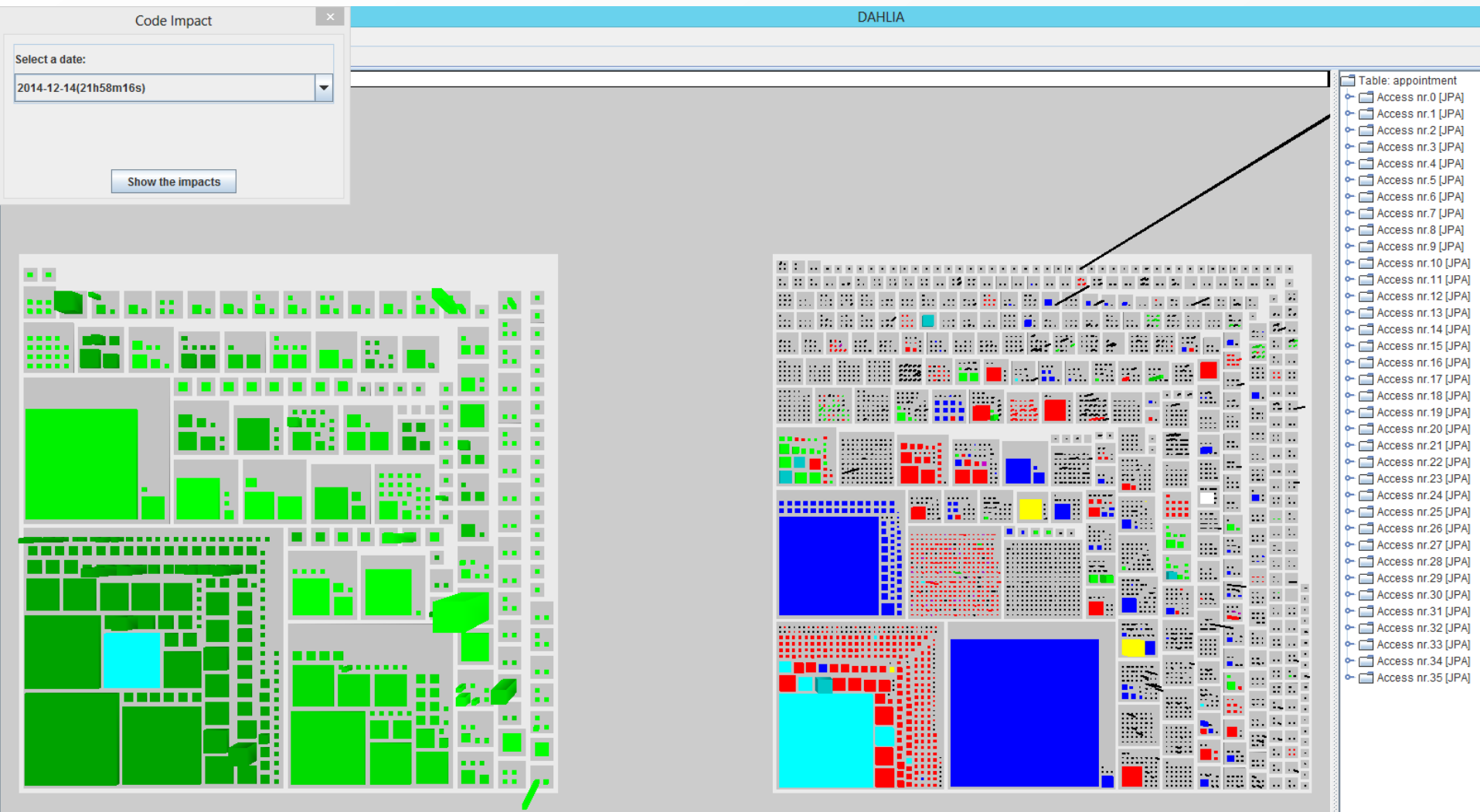
Visualization within DAHLIA (OSCAR)



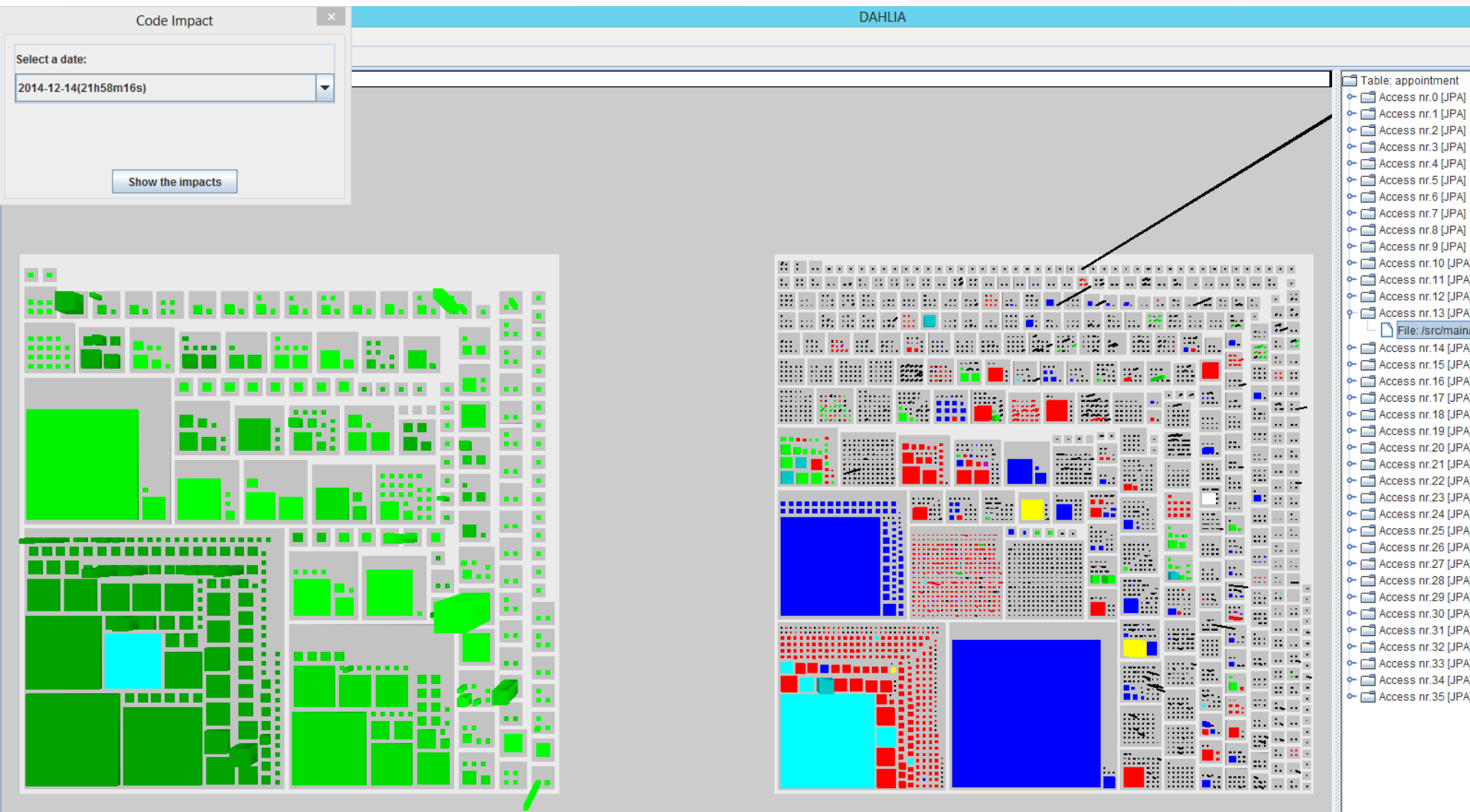
Visualization within DAHLIA (OSCAR)



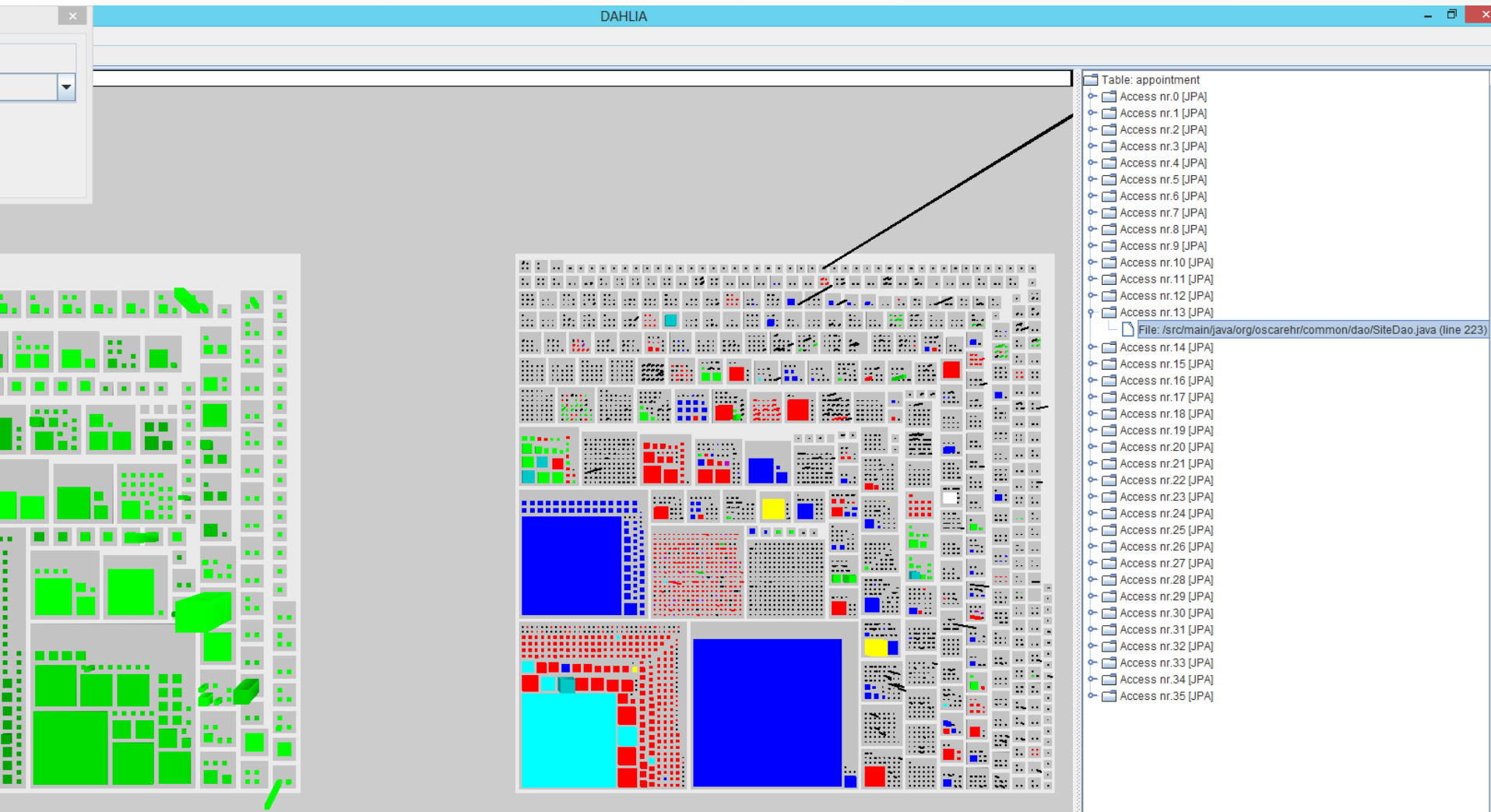
Visualization within DAHLIA (OSCAR)



Visualization within DAHLIA (OSCAR)



Visualization within DAHLIA (OSCAR)



Visualization within DAHLIA (OSCAR)

The screenshot displays the DAHLIA (OSCAR) interface. On the left, a project tree shows a table named 'appointment' with 35 access points, each labeled 'Access nr. [X] [JPA]'. The tree is expanded to show the file structure, including 'File: /src/main/java/org/oscarehr/common/dao/SiteDao.java (line 223)'. On the right, the Java code for 'SiteDao.java' is shown, with line 223 highlighted. The code includes methods for retrieving site information and appointment details.

```
189         return pList;
190     }
191
192     public List<String> getGroupBySiteManagerProviderNo(String providerNo) {
193
194         Query query = entityManager.createNativeQuery(
195             "select distinct g.mylgroup_no from mygroup g " +
196             " inner join provider p on p.provider_no = g.provider_no and p.status = 1 " +
197             " inner join providersite ps on ps.provider_no = g.provider_no " +
198             " where ps.site_id in (select site_id from providersite where provider_no = :providerNo)");
199         query.setParameter("providerNo", providerNo);
200
201         @SuppressWarnings("unchecked")
202         List<String> groupList = query.getResultList();
203
204
205         return groupList;
206     }
207
208     public Long site_searchmygroupcount(String myGroupNo, String siteName) {
209         Query query = entityManager.createNativeQuery("select count(provider_no) from mygroup where mygroup_no=:groupno and provider_no=:siteName");
210         query.setParameter("groupno", myGroupNo);
211         query.setParameter("siteName", siteName);
212
213         Long result = ((BigInteger)query.getSingleResult()).longValue();
214         return result;
215     }
216
217     public String getSiteNameByAppointmentNo(String appointmentNo) {
218
219         Query query = entityManager.createNativeQuery("select location from appointment where appointment_no = :appointmentno");
220         query.setParameter("appointmentno", appointmentNo);
221
222         @SuppressWarnings("unchecked")
223         List<String> list = query.getResultList();
224         if(list.size()>0) {
225             return list.get(0);
226         }
227
228         return "";
229     }
230 }
```

Online demo

Episod IV

DAHLIA++

Episod IV – DAHLIA++

Analyzing and Supporting Database/Program Co-Evolution

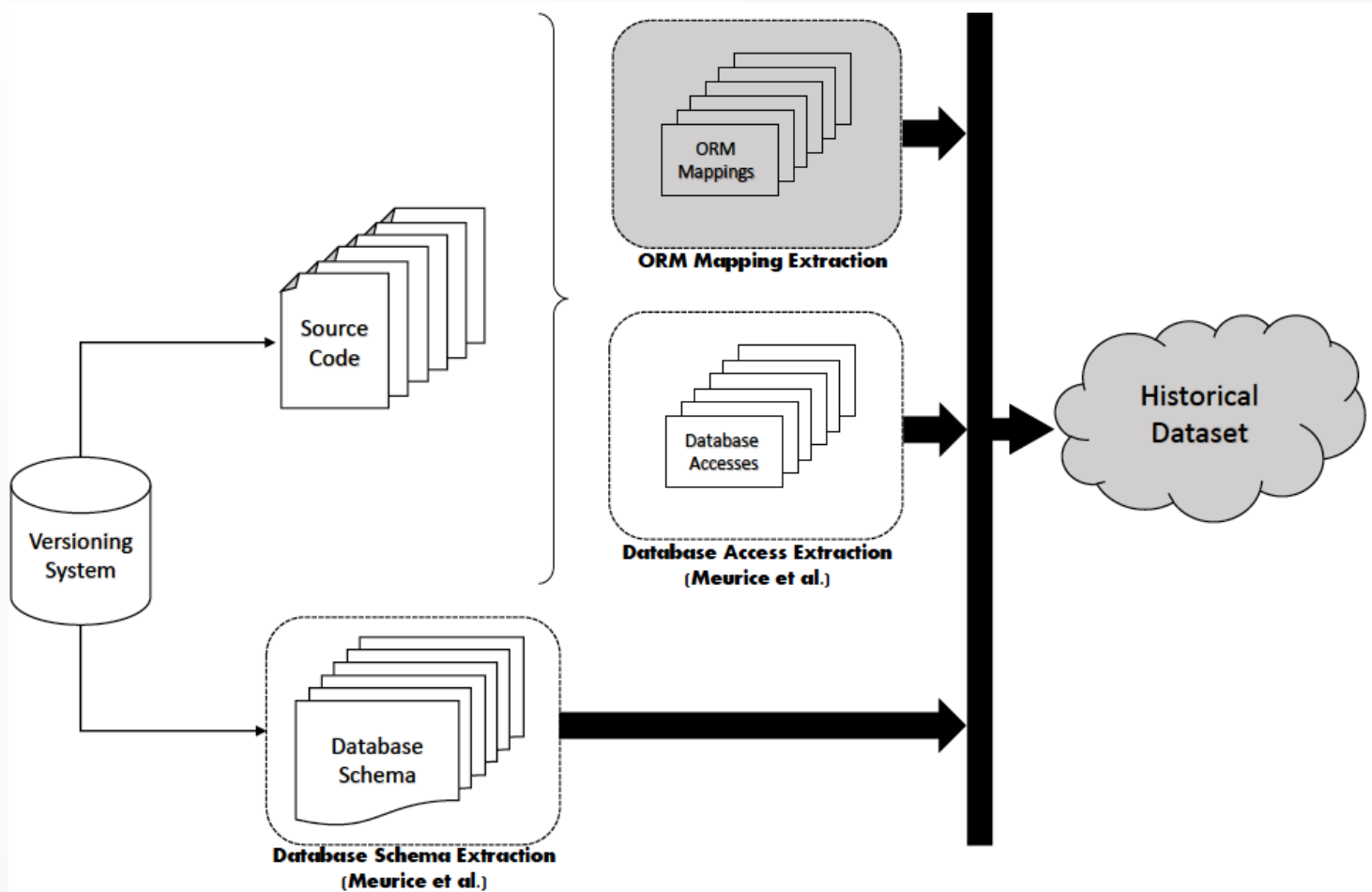
Goals:

Identify program inconsistencies due to past database schema changes

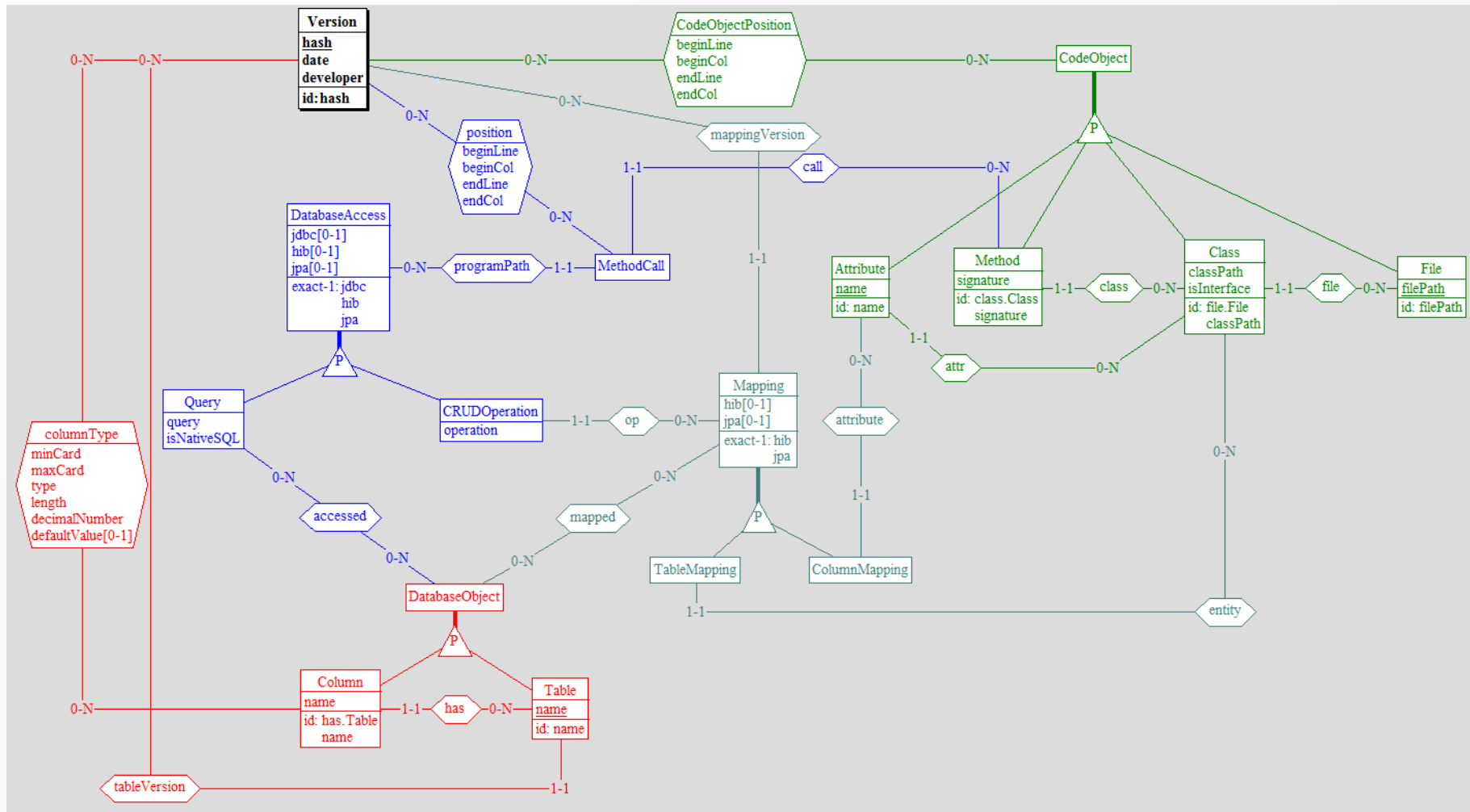
Prevent such program inconsistencies in the future, by helping developers propagating schema changes to programs

Episod IV

Analyzing & supporting database/program co-evolution

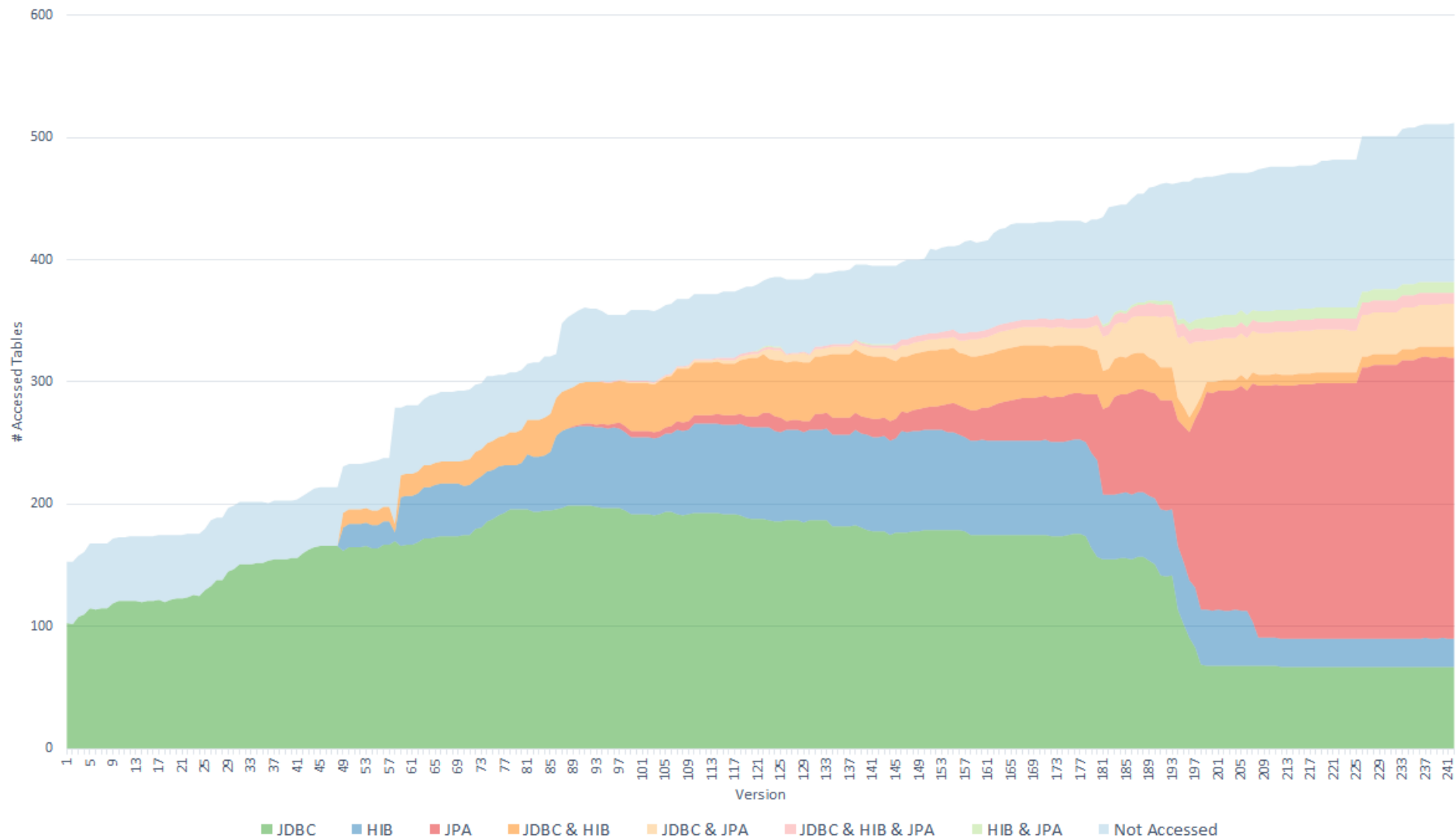


Data model of the historical dataset



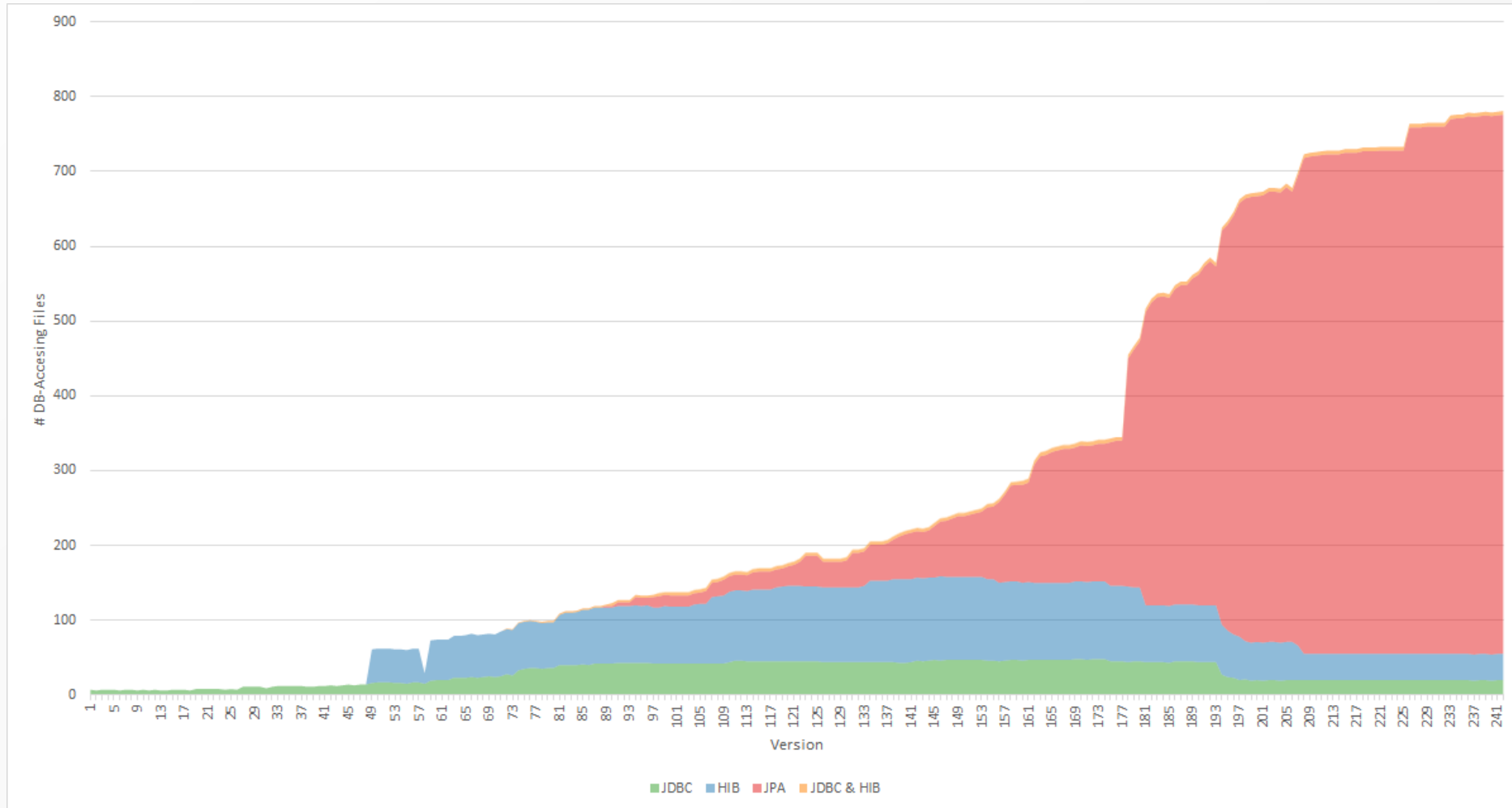
Episod IV

Analyzing the evolution of database access technologies



Episod IV

Analyzing the evolution of database access technologies



Episod IV

Identifying co-evolution inconsistencies

System	#Table Deletions	#Unsolved	Propagation Time avg~max	#Accesses avg~max
OpenMRS	11	1	1.6 ~ 134	7.5 ~ 9
Broadleaf	86	0	1.1 ~ 6	2.8 ~ 14
OSCAR	33	5	1.4 ~ 90	2.9 ~ 9

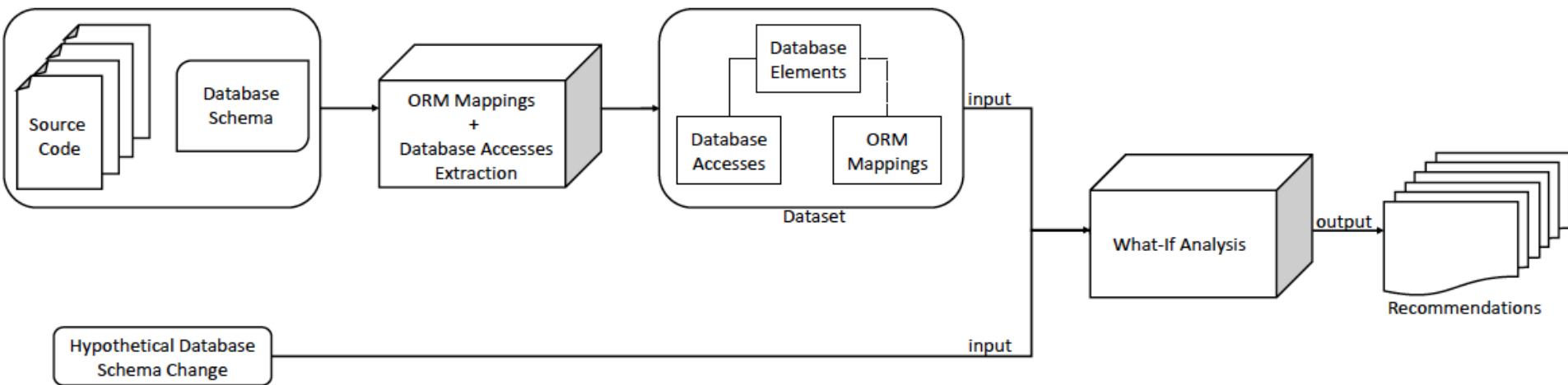
System	Renaming	Solution Time avg~max	#Accesses avg~max
OpenMRS	1	1 ~ 1	0 ~ 0
Broadleaf	14	2.6 ~ 6	3.3 ~ 8
OSCAR	7	1.9 ~ 89	1.6 ~ 132

System	#Column Deletions	#Unsolved	Propagation Time avg~max	#Accesses avg~max
OpenMRS	32	4	1.6 ~ 134	2.2 ~ 4
Broadleaf	154	0	1.1 ~ 2	4 ~ 15
OSCAR	170	0	1.1 ~ 24	1.6 ~ 132

System	Renaming	Solution Time avg~max	#Accesses avg~max
OpenMRS	10	1 ~ 1	0 ~ 0
Broadleaf	16	1.1 ~ 2	1.7 ~ 3
OSCAR	321	1.2 ~ 38	1.7 ~ 389

Episod IV

Preventing co-evolution inconsistencies



Evaluation

130 selected schema changes

	TR	TD	CR	CD
Broadleaf	12	17	12	52
OpenMRS	0	2	0	5
OSCAR	5	9	7	9
Total	17	28	19	66

Correctness of recommendations

	TR	TD	CR	CD	Total	Perc.
Correct recommendations	17	90	24	71	202	99%
Wrong recommendations	0	0	0	2	2	1%
Missing recommendations	2	0	1	3	6	5%

Online demo

Epilogue

conclusions and todo list

Conclusions

Observations

Data-intensive systems are indeed large and complex

Continuously increasing size and complexity over time

Several database access technologies may co-exist

Database access can be highly dynamic

Co-evolving database and programs is non-trivial → inconsistencies

Automated support for developers is more than welcome !

Conclusions

Achievements

- Analyzing the evolution history of database schemas
- Analyzing database usage in dynamic Java programs
- Analyzing co-evolution between databases and programs
- Supporting co-evolution between databases and programs
- Current implementation for Java systems using MySQL
- Promising case studies and evaluations for large-scale systems

Conclusions

Future work

Support other programming languages and database platforms

Consider other information sources (e.g., data, developers, user interface)

Support other database evolution scenarios (e.g., migration)

Partly automate program adaptation under database schema change

References

Episod I

Maxime Gobert, Jerome Maes, Anthony Cleve, and Jens Weber. Understanding Schema Evolution as a Basis for Database Reengineering. In *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM 2013)*. IEEE Computer Society, 2013.

Episod II

Loup Meurice and Anthony Cleve. DAHLIA – A Visual Analyzer of Database Schema Evolution. In *Proceedings of the IEEE CSMR/WCRE 2014 Software Evolution Week*, pages 464–468. IEEE Computer Society, 2014.

Anthony Cleve, Maxime Gobert, Loup Meurice, Jerome Maes, and Jens Weber. Understanding Database Schema Evolution: A Case Study. *Science of Computer Programming*, 97:113–121, 2015.

Episod III

Csaba Nagy, Loup Meurice, and Anthony Cleve. Where Was this SQL Query Executed? A Static Concept Location Approach. In *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2015)*, pages 580–584. IEEE, 2015.

Loup Meurice, Csaba Nagy, and Anthony Cleve. Static Analysis of Dynamic Database Usage in Java Systems. In *Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016)*, Lectures Notes on Computer Science. Springer, 2016. *(to appear)*

Episod IV

Loup Meurice, Csaba Nagy, and Anthony Cleve. Detecting and Preventing Program Inconsistencies Under Database Schema Evolution. *(submitted for publication)*

merci



beaucoup

Email: anthony.cleve@unamur.be

Twitter: @anthonymcleve

Skype: anthonymcleve

Analyzing the Evolution of Data-Intensive Software Systems *in support to software maintenance*

Anthony Cleve
PReCISE Research Center
University of Namur, Belgium