Recommendations for Evolving Legacy Databases

Julien Delplanque

juliendelplanque@inria.fr

Université de Lille

Inria inventeurs du monde numérique

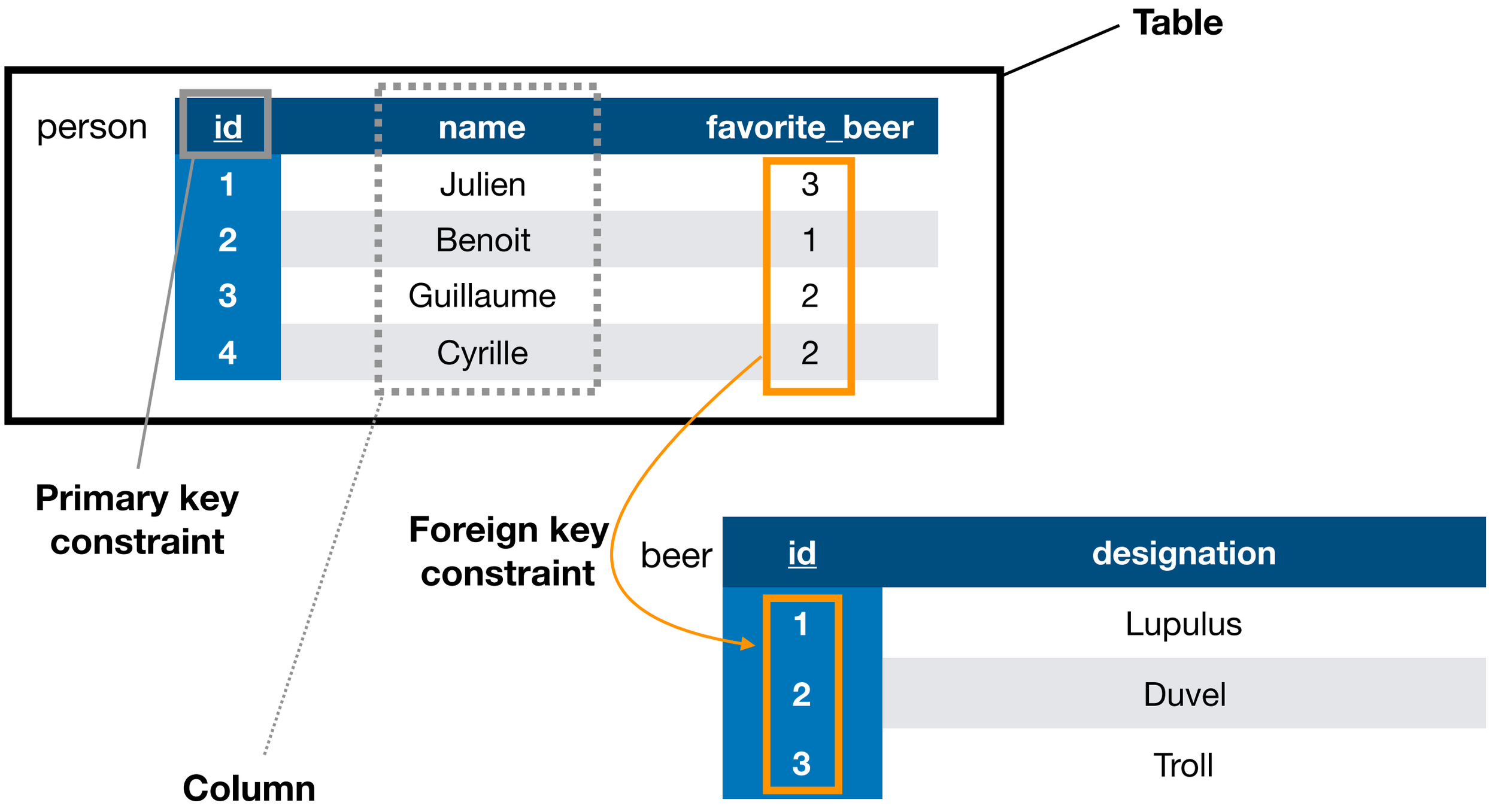
RMod

https://rmod.inria.fr/web/

Overview

1. Reminder about relational databases
2. Recommendations for Evolving Legacy Databases
3. Conclusions and future work

RDBMS - Basic Entities



RDBMS - Behavioural entities

- Views: Named SELECT queries stored in the database.
- Stored procedures: Functions, written in a programming language, implementing arbitrary computation.
- Triggers: Entity listening to events happening on a table and reacting to them.
- …

Problems during DB evolution

- No inconsistency is tolerated by the DBMS
- Stored procedures are black boxes

Examples

Example: remove beer.id

person

id	name	favorite_beer
1	Julien	3
2	Benoit	1
3	Guillaume	2
4	Cyrille	2

beer

id	designation
1	Lupulus
2	Duvel
3	Troll

Example: remove beer.id

person

id	name	favorite_beer
1	Julien	3
2	Benoit	1
3	Guillaume	2
4	Cyrille	2

Forbidden by the DBMS!

beer

id	designation
1	Lupulus
2	Duvel
3	Troll

Example: remove column beer.designation referenced in view

person

id	name	favorite_beer
1	Julien	3
2	Benoit	1
3	Guillaume	2
4	Cyrille	2

beer

id	designation
1	Lupulus
2	Duvel
3	Troll

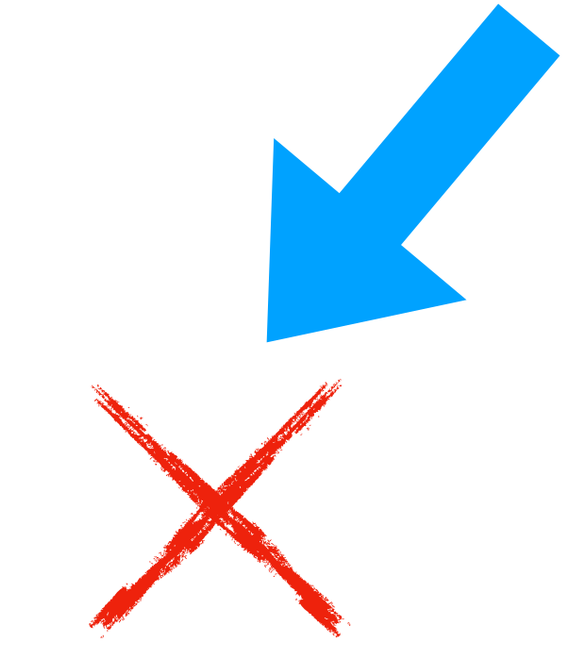
VIEW person_to_favorite_designation

```
SELECT id, name, designation
FROM person, beer
WHERE person.favorite_beer = beer.id;
```

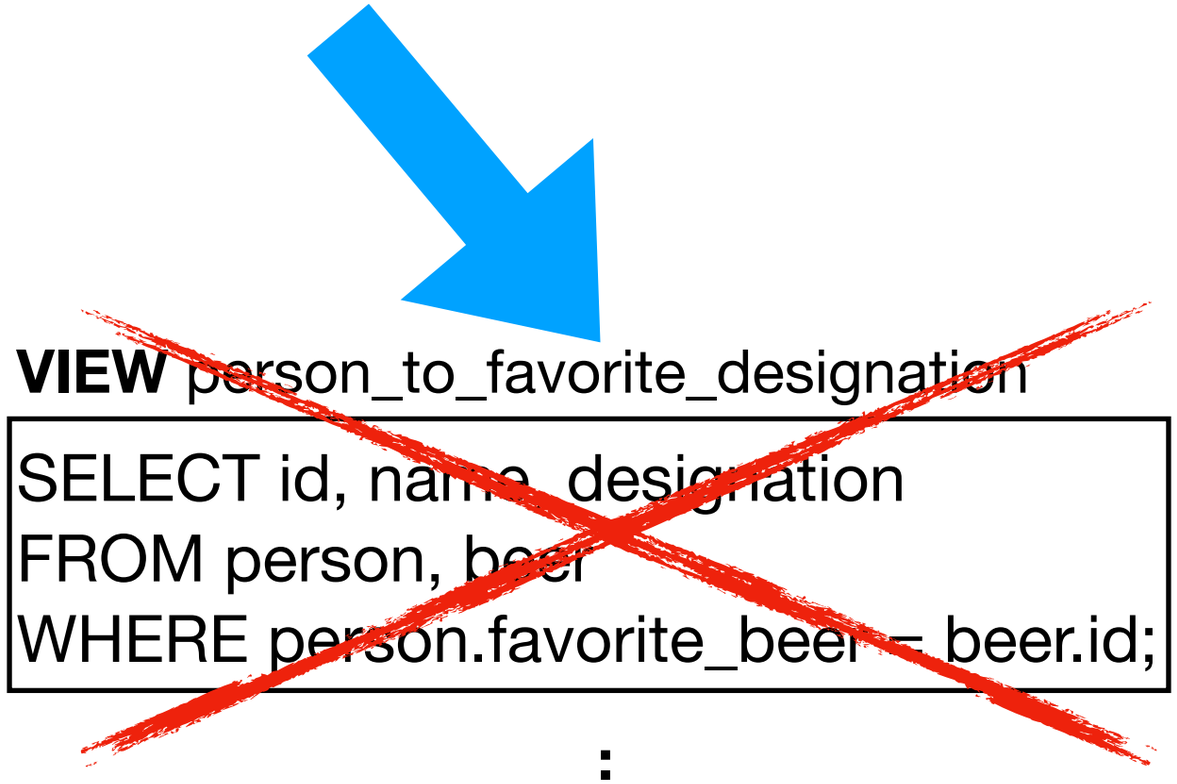
Example: remove column beer.designation referenced in view

VIEW person_to_favorite_designation

```
SELECT id, name, designation
FROM person, beer
WHERE person.favorite_beer = beer.id;
```



Refuse change



Continues in cascade

Example: remove column beer.designation referenced in view

VIEW person_to_favorite_designation

```
SELECT id, name, designation
FROM person, beer
WHERE person.favorite_beer = beer.id
```

Not convenient p + behaviour

Refuse change

VIEW person_to_favorite_designation

```
SELECT id, name, designation
FROM person, beer
WHERE person.favorite_beer = beer.id;
```

⋮

Continues in cascade

Example: remove person.name

person

id	name	favorite_beer
1	Julien	3
2	Benoit	1
3	Guillaume	2
4	Cyrille	2

beer

id	designation
1	Lupulus
2	Duvel
3	Troll

STOR. PROC. id_of_troll_lovers

```
RETURN SELECT person.id
FROM person, beer
WHERE person.favorite_beer = beer.id
    AND beer.designation = ‘Troll’;
```

Example: remove person.name

person

id	name	favorite_beer
1	Julien	3
2	Benoit	1
3	Guillaume	2
4	Cyrille	2

Unseen by the DBMS!

beer

id	designation
1	Lupulus
2	Duvel
3	Troll

STOR PROC. nb_of_troll_lovers

```
RETURNS SELECT person.id
FROM person, beer
WHERE person.favorite_beer = beer.id
   AND beer.designation = ‘Troll’;
```

Recommendations for Evolving Legacy Databases

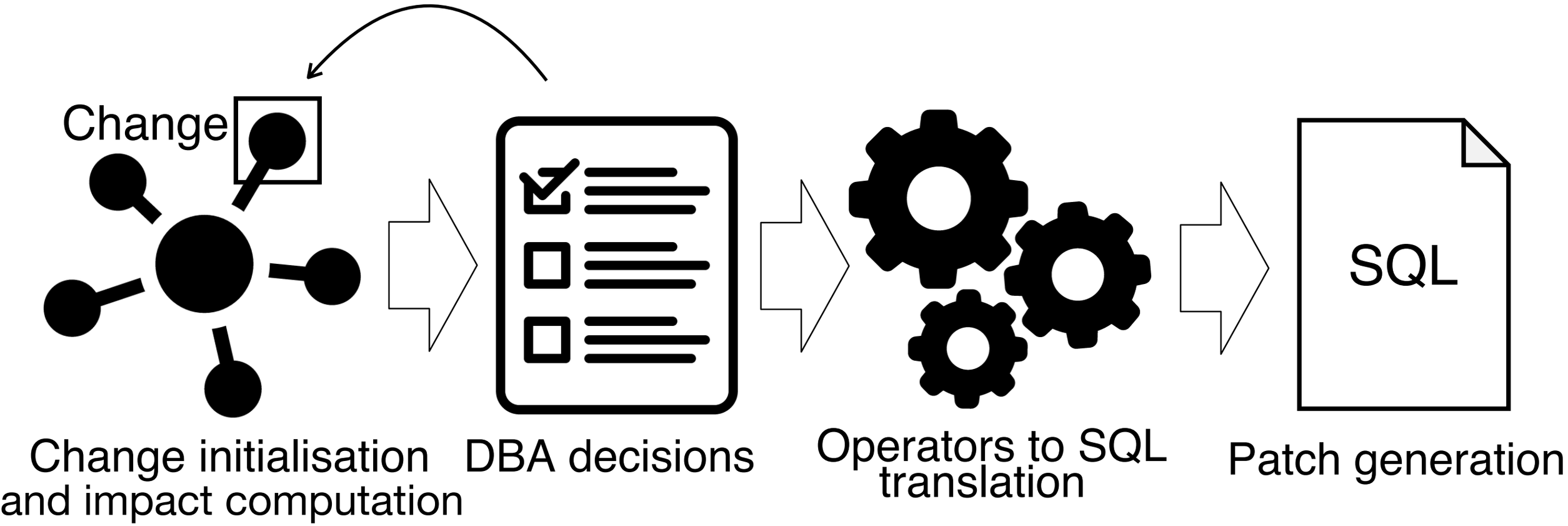
Refactoring v.s. Change

	Refactoring	Change
Code evolution	✅	✅
Behaviour preserving	✅	❌
Modify software features	❌	✅

Refactoring v.s. Change

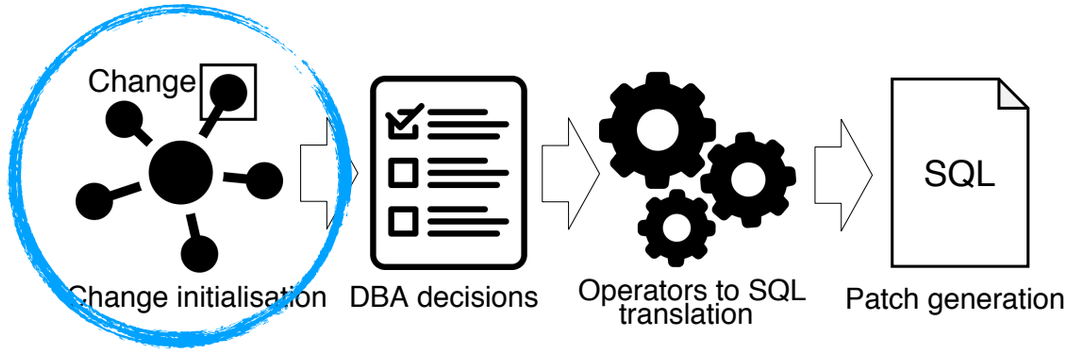
	Refactoring	Change
Code evolution	✅	✅
Behaviour preserving	✅	❌
Modify software features	❌	✅

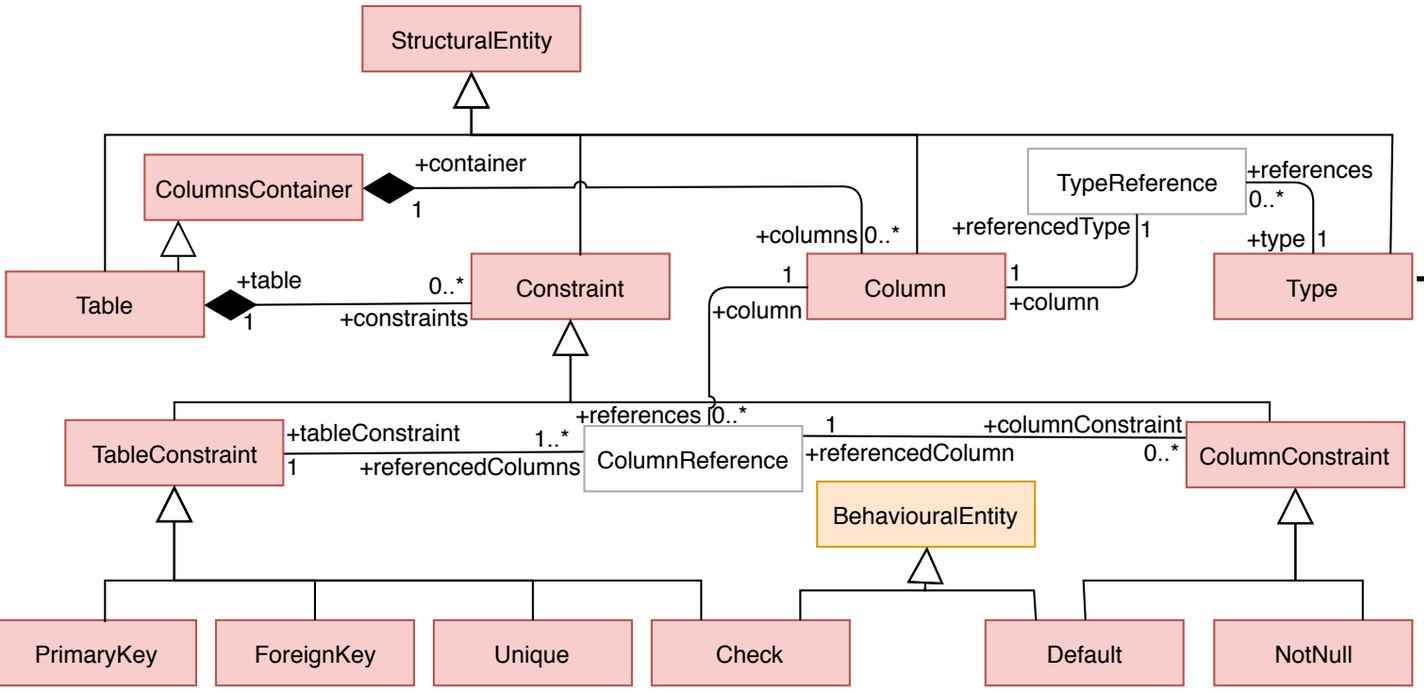
Approach overview



Meta-model

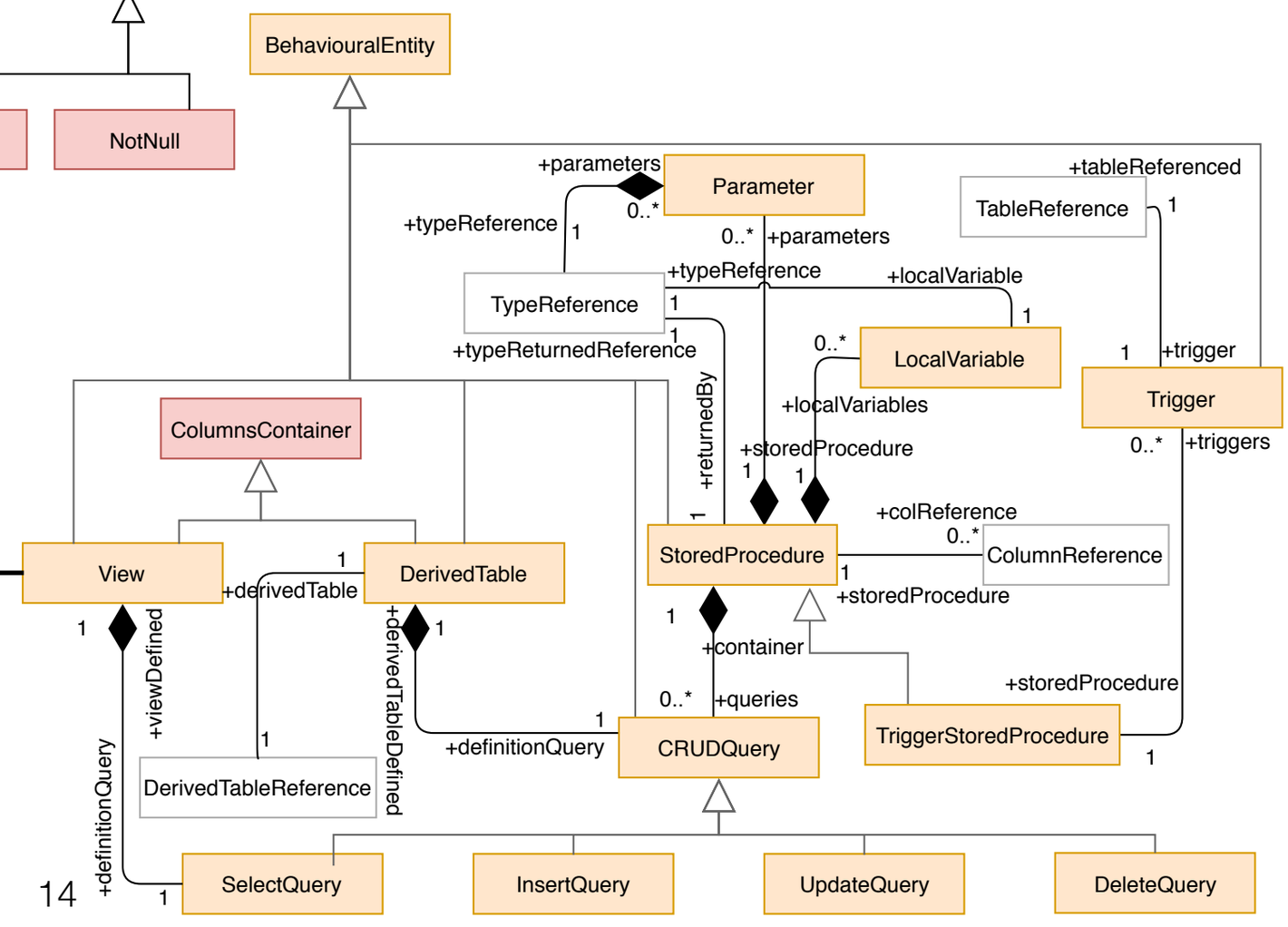
You are here



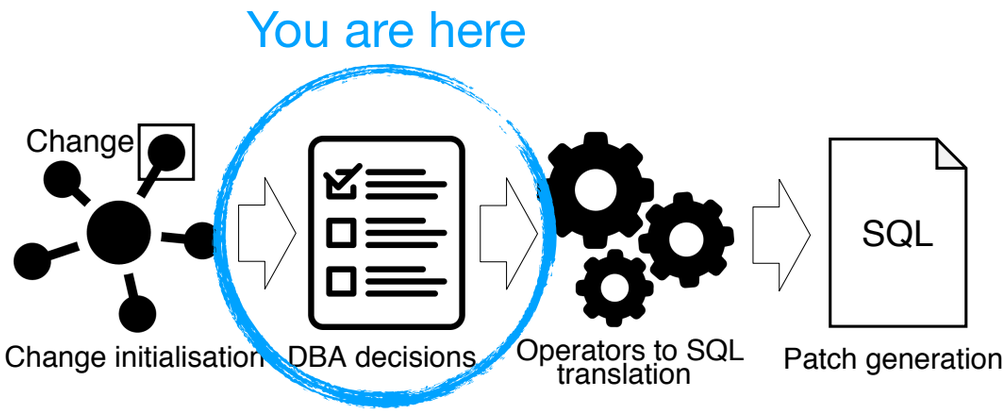


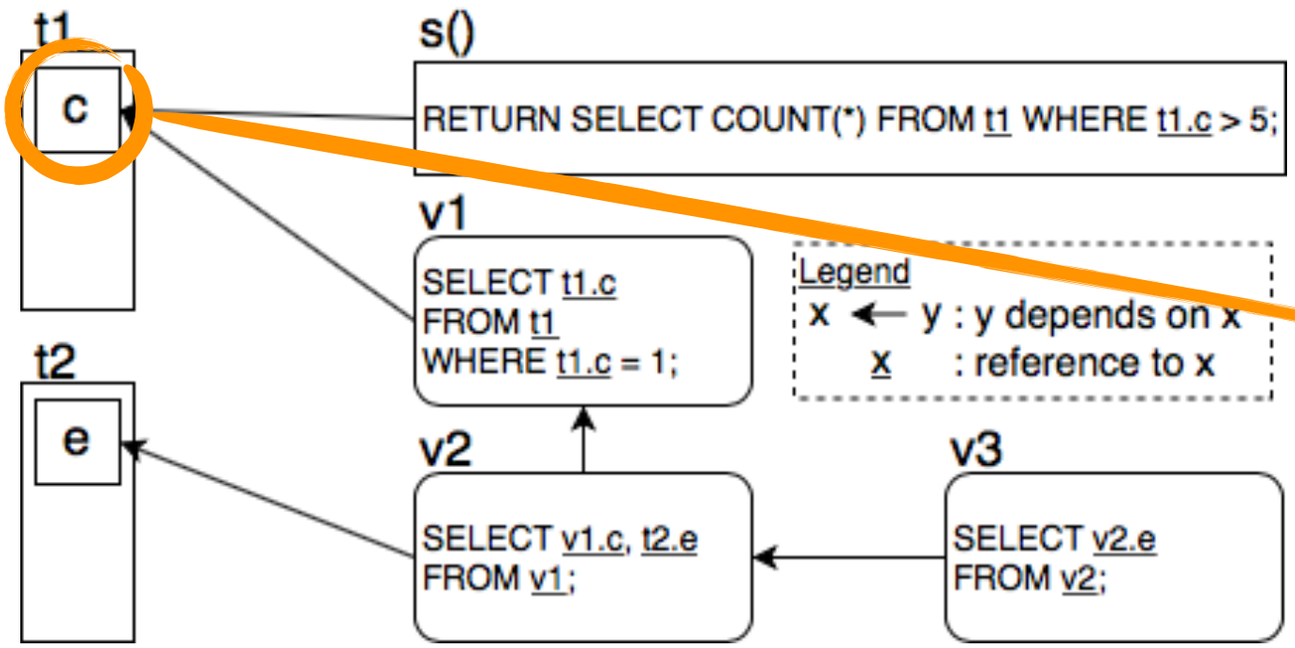
Structural entities

Behavioural entities



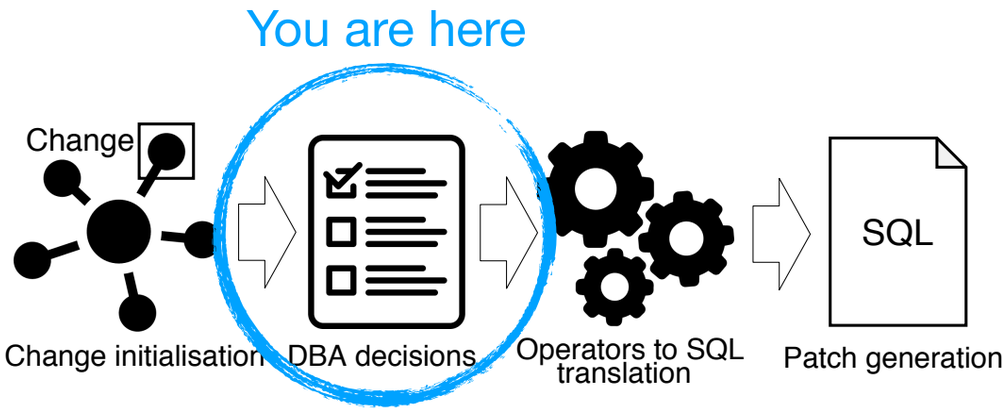
Impact computation & Recommendations selection

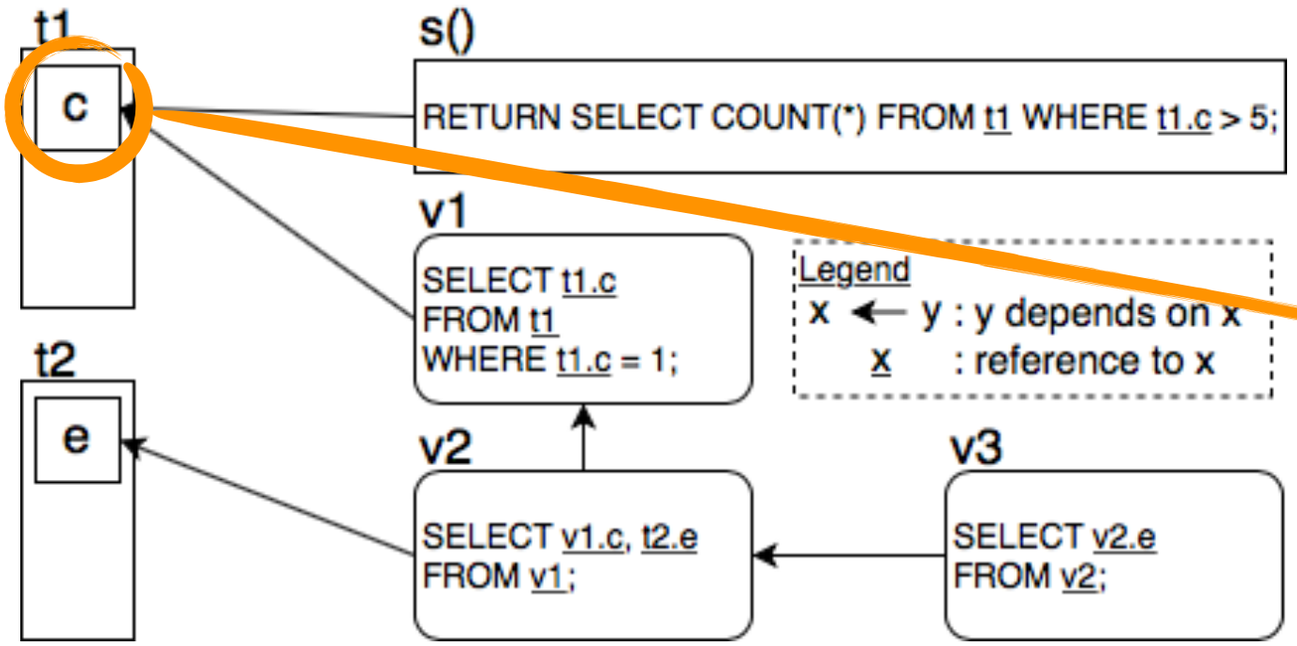




Rename t.c column as t.d

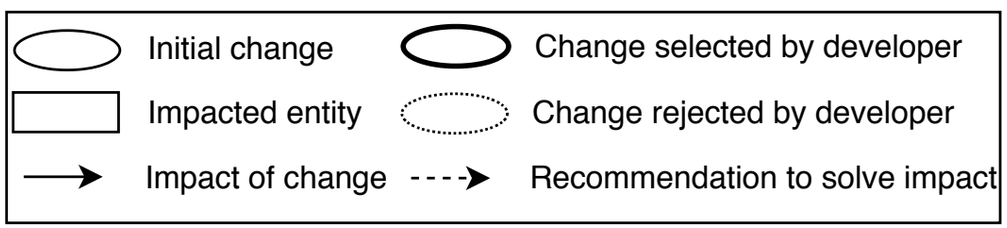
Impact computation & Recommendations selection



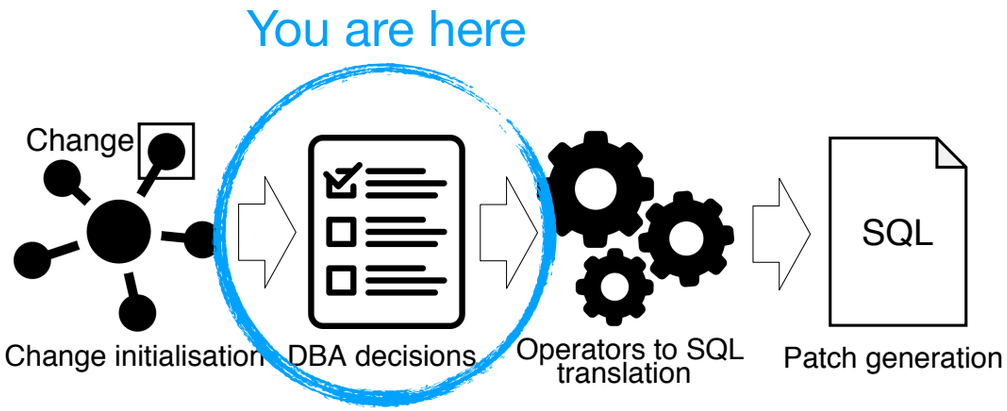


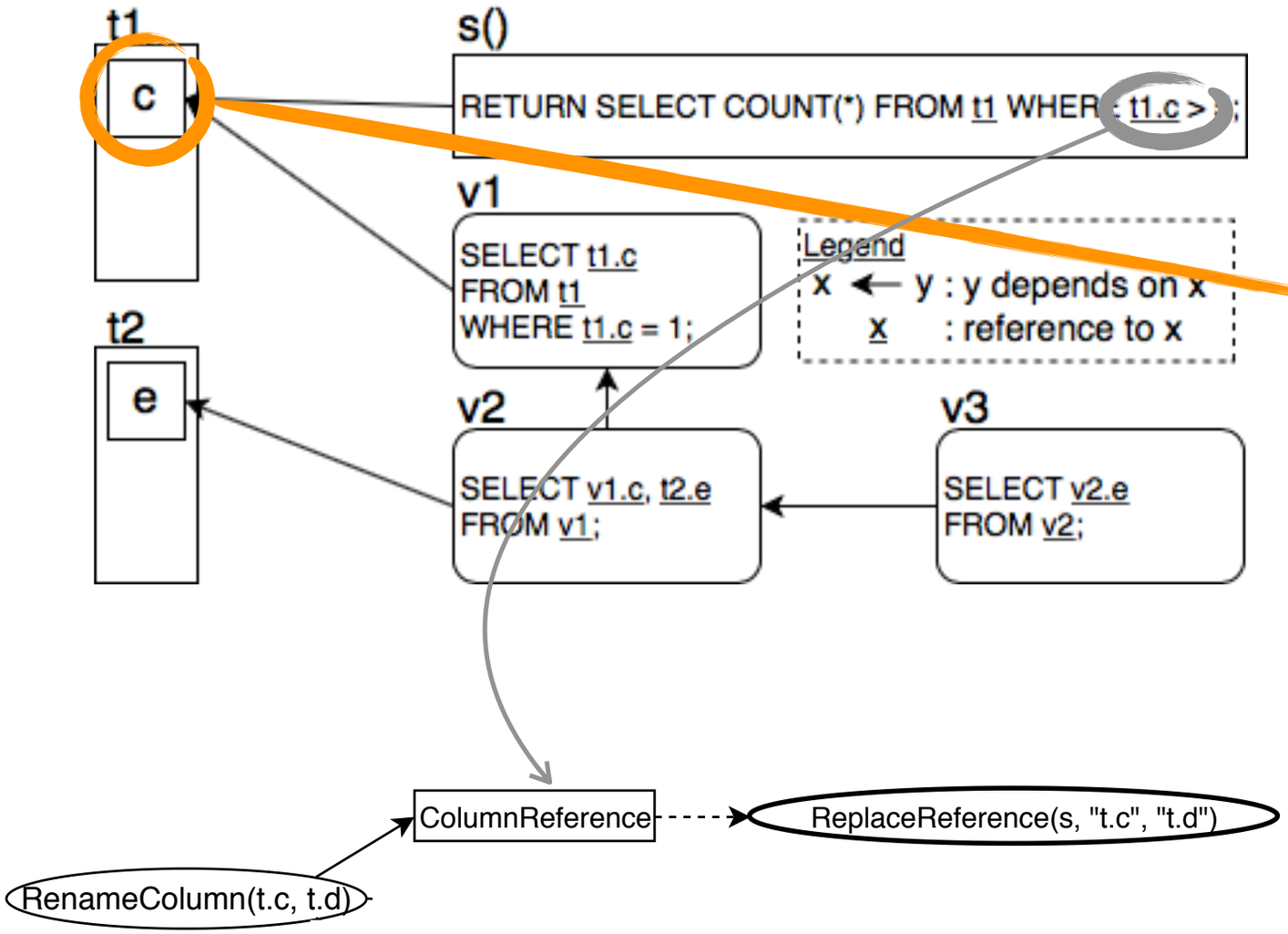
Rename t.c column as t.d

RenameColumn(t.c, t.d)

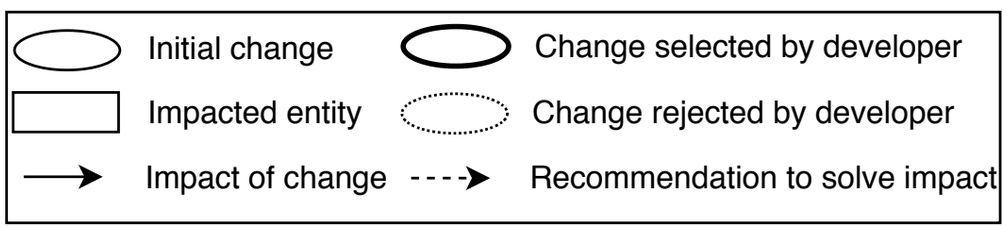


Impact computation & Recommendations selection

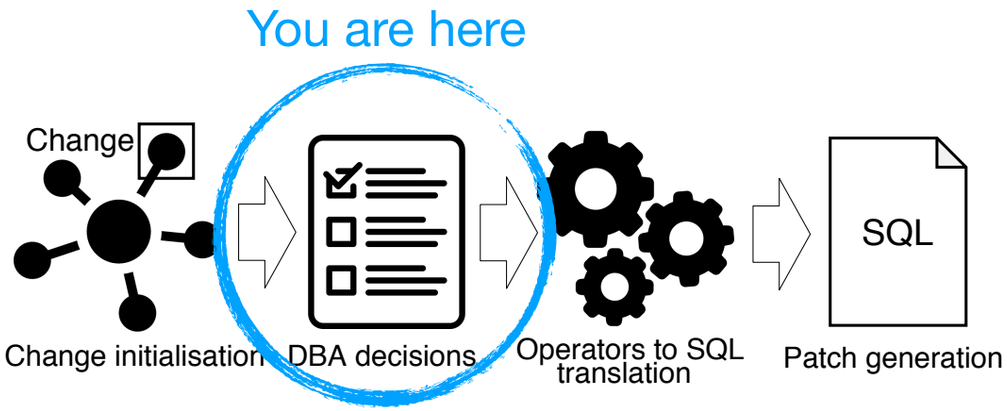


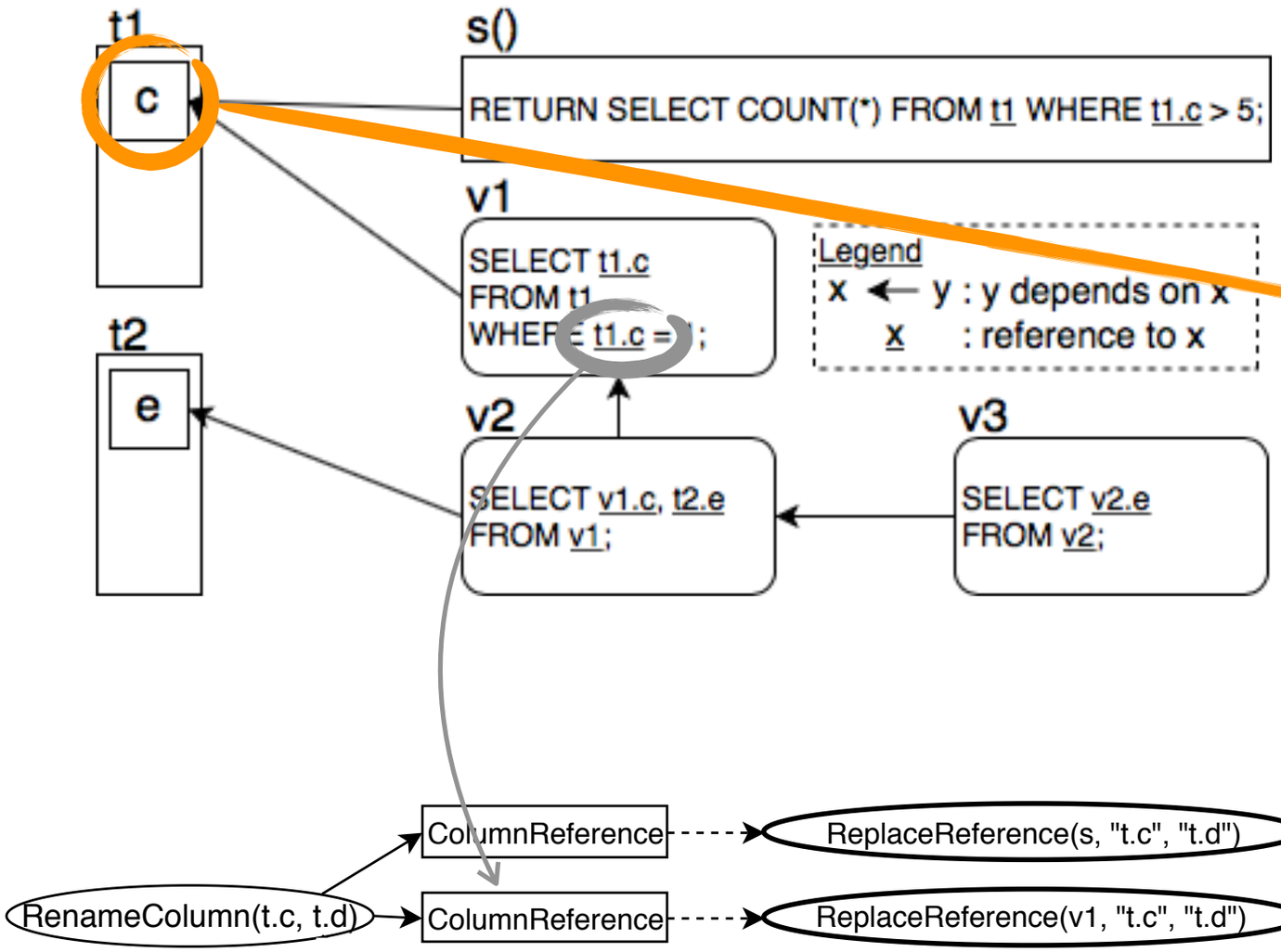


Rename t.c column as t.d

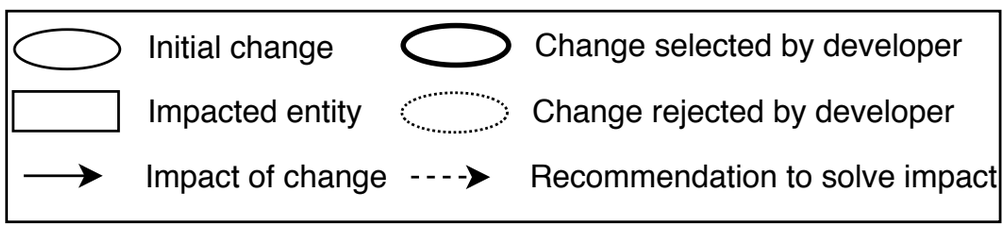


Impact computation & Recommendations selection

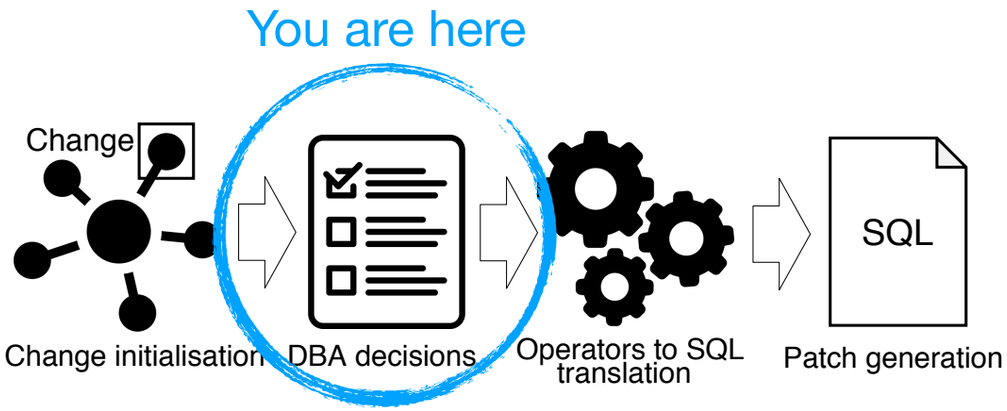


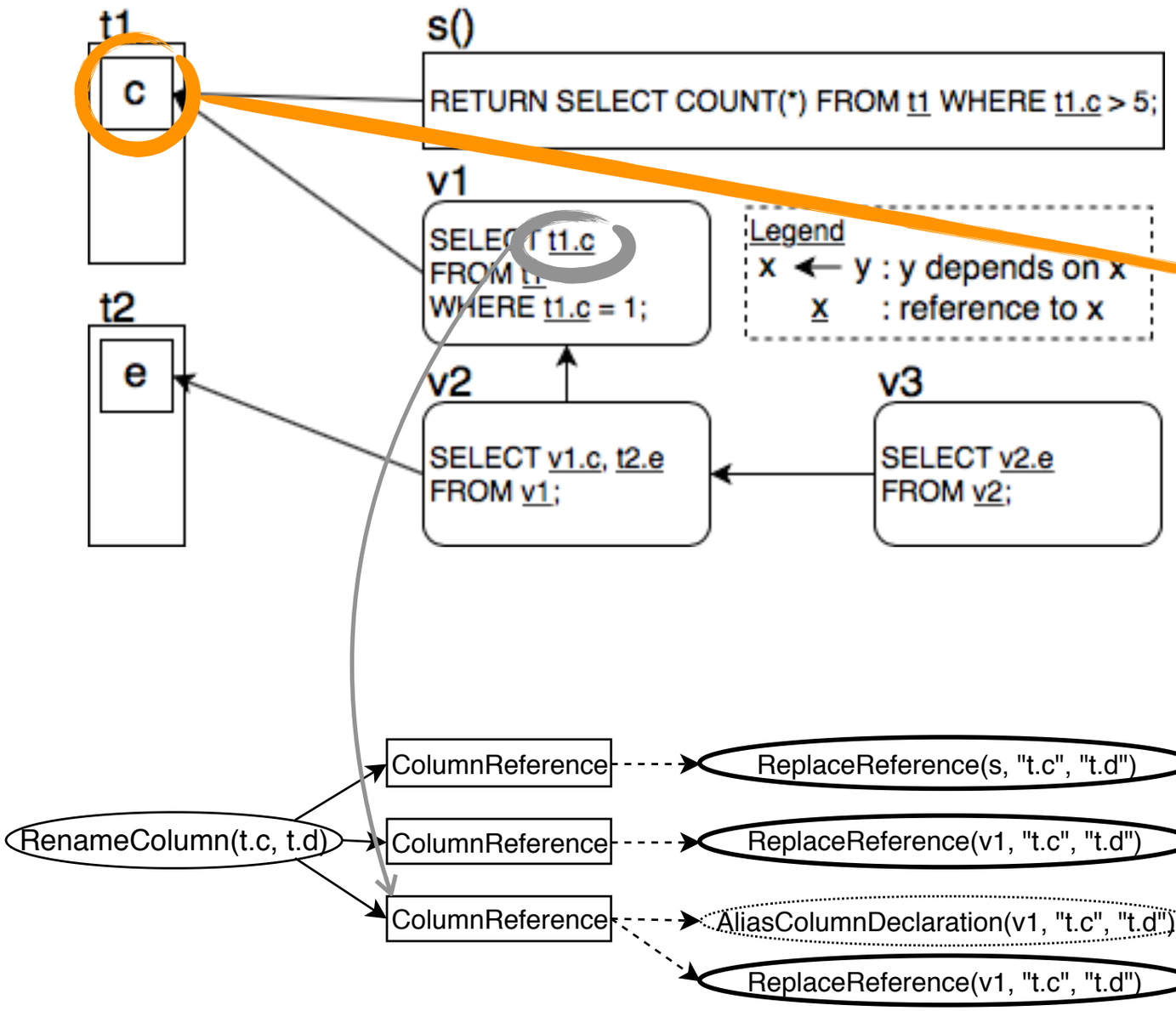


Rename t.c column as t.d

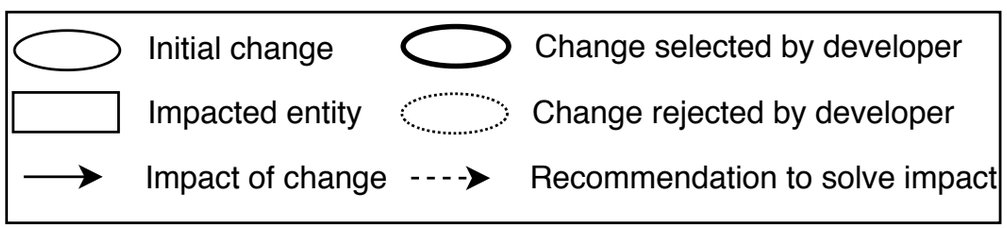


Impact computation & Recommendations selection

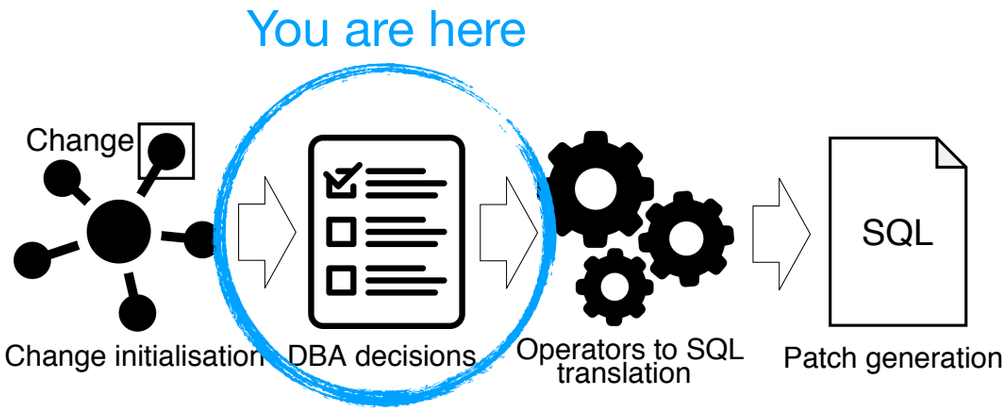


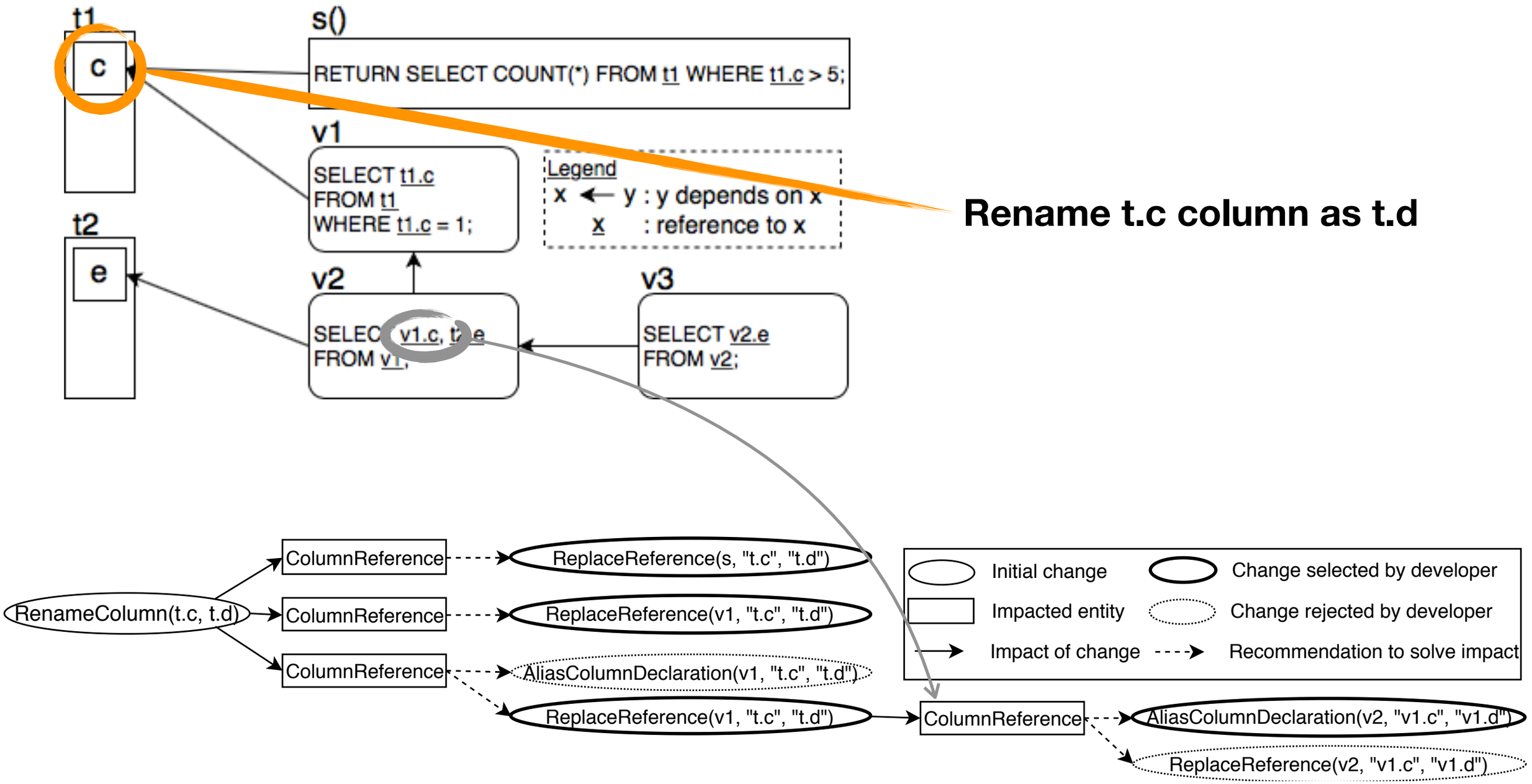


Rename t.c column as t.d

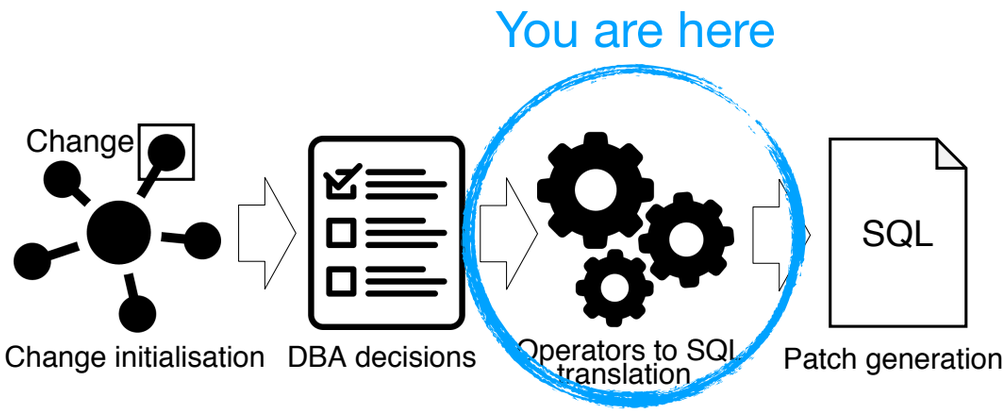


Impact computation & Recommendations selection

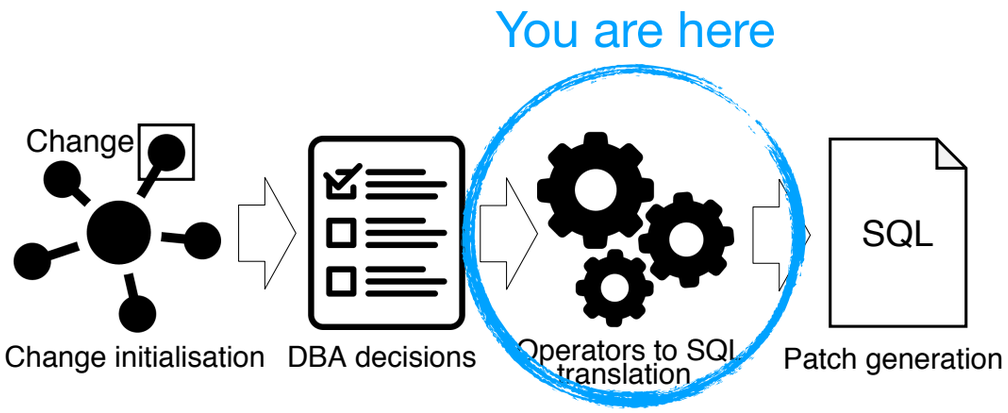


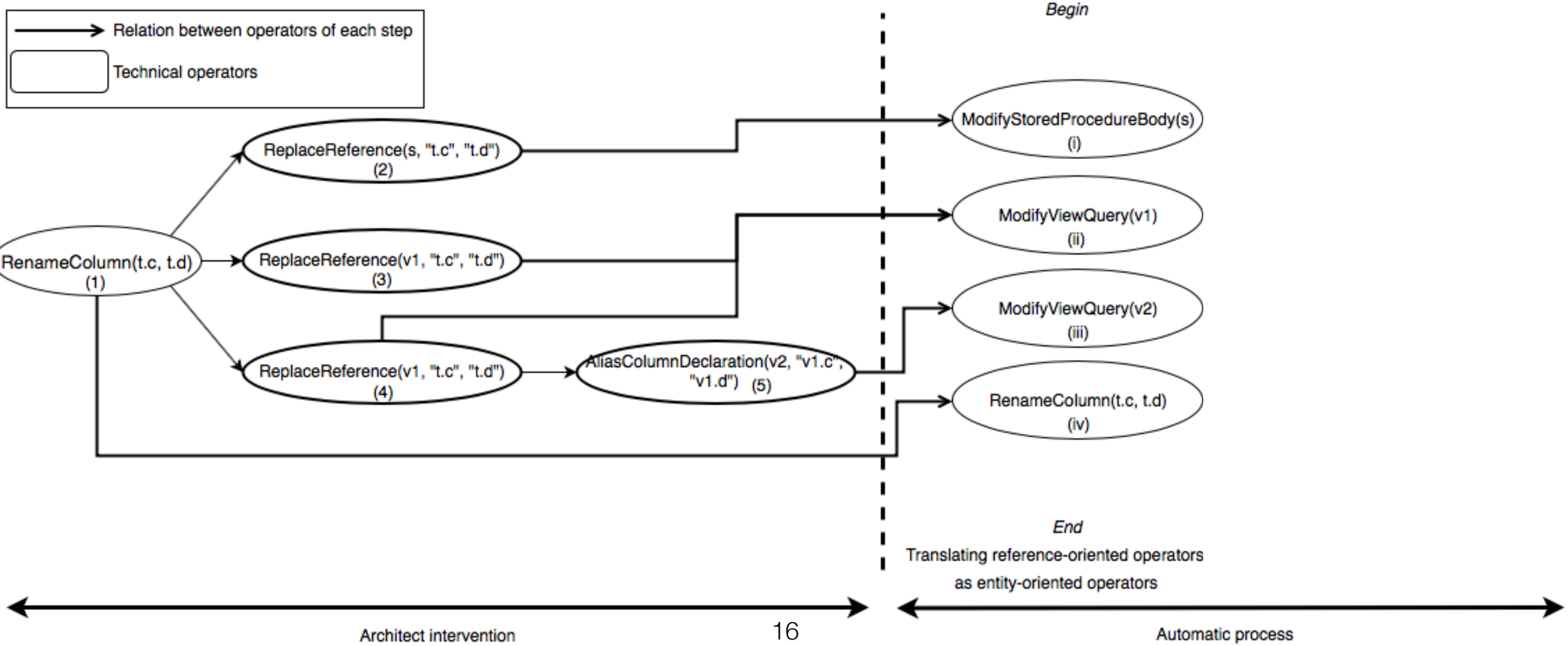


Architect choices compilation

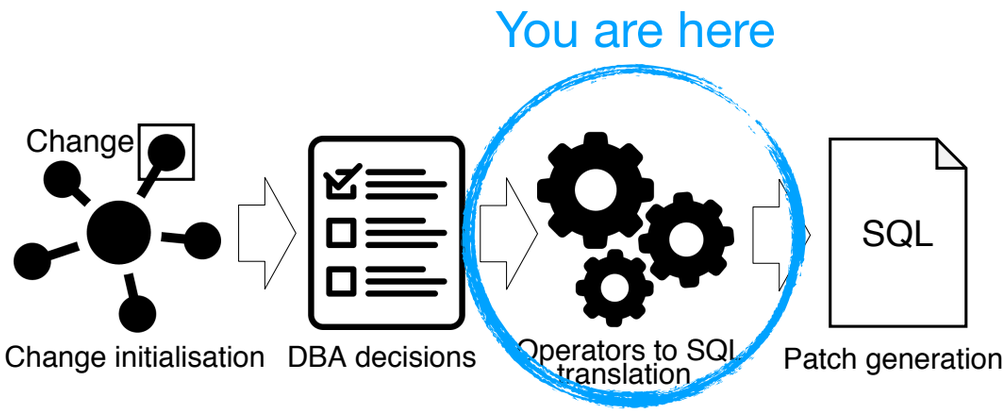


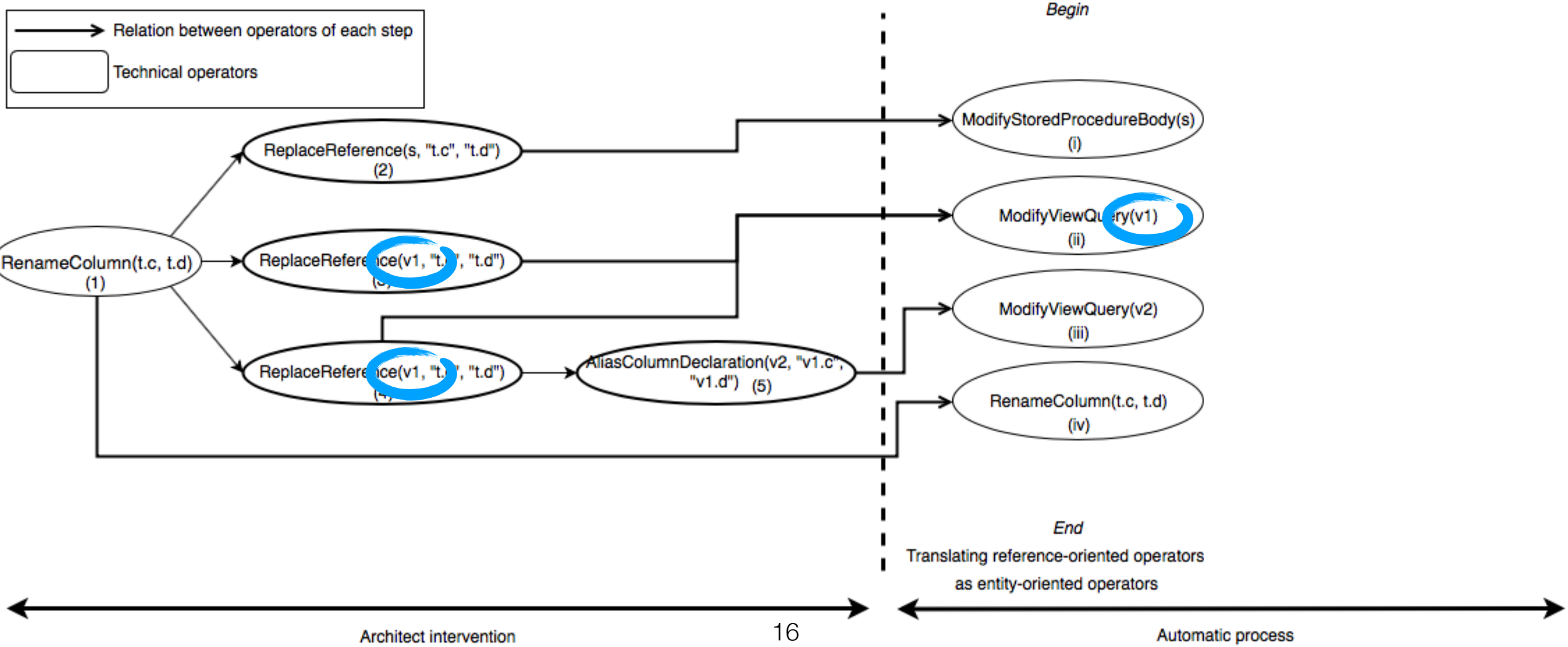
Architect choices compilation



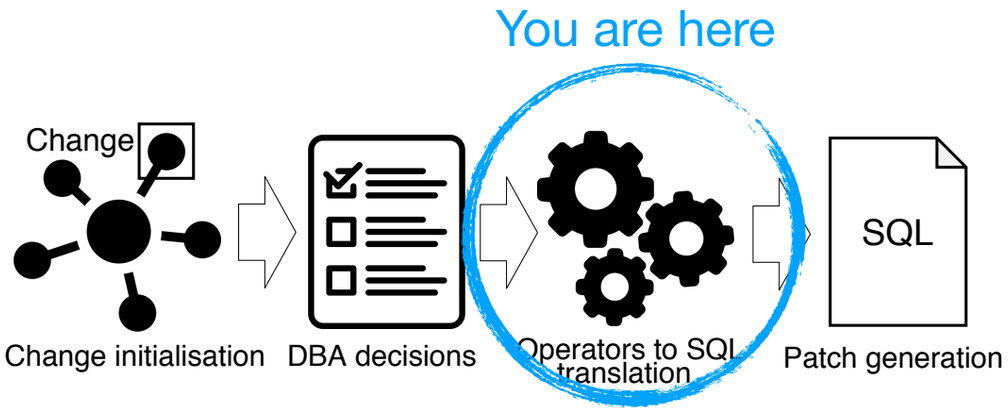


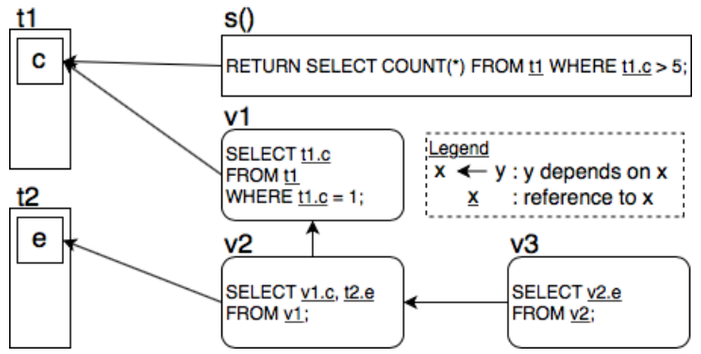
Architect choices compilation

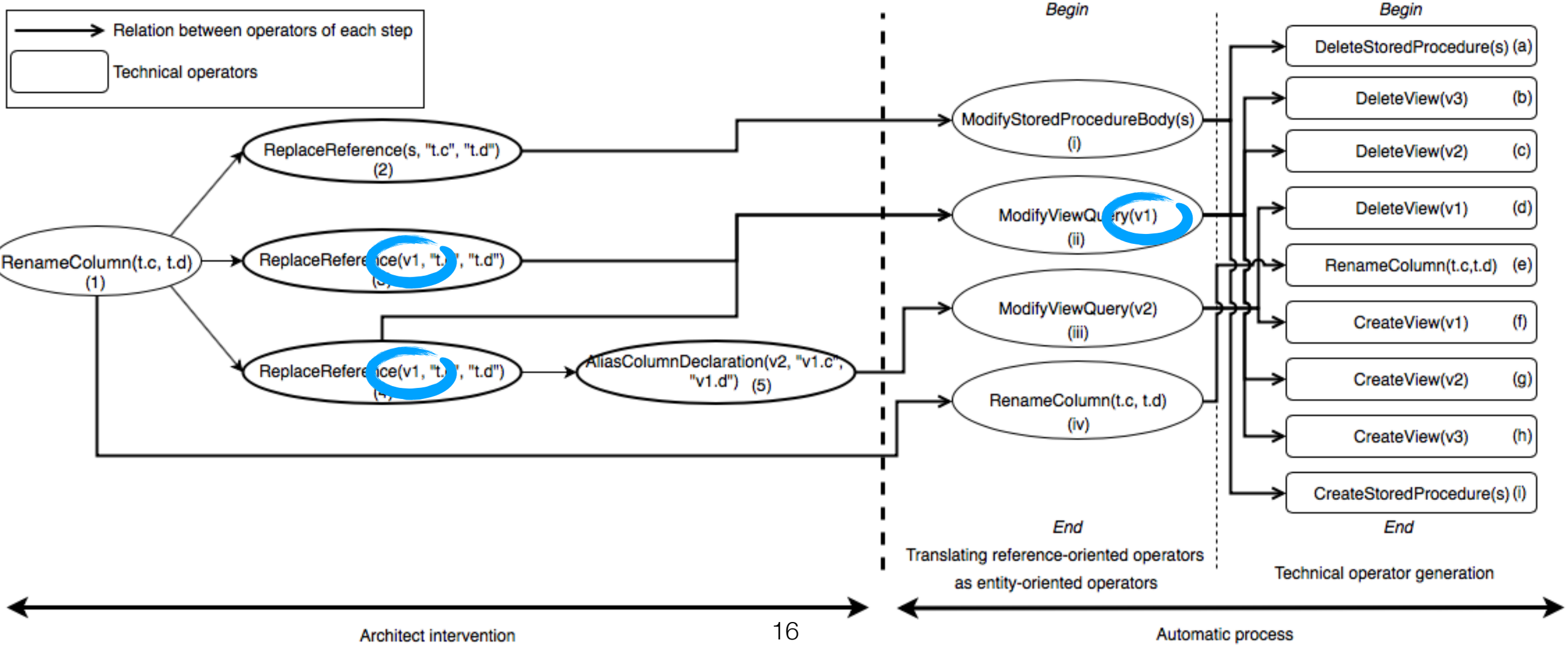




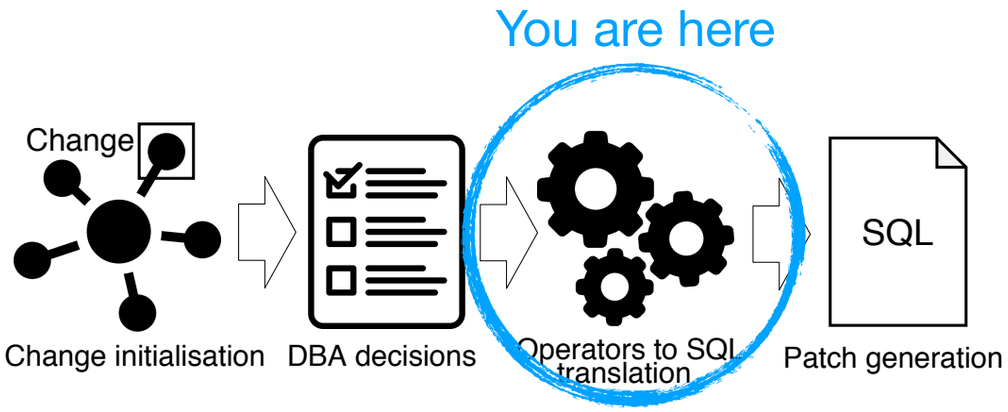
Architect choices compilation

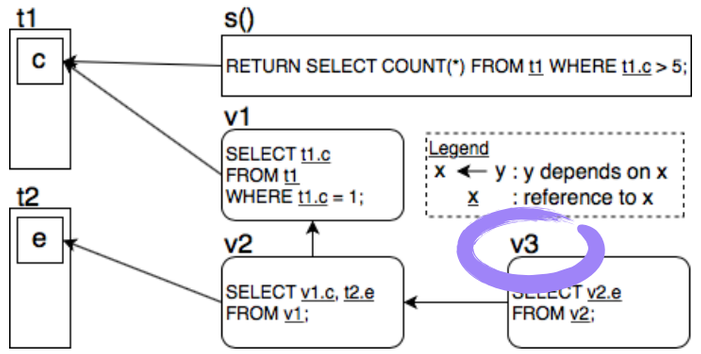


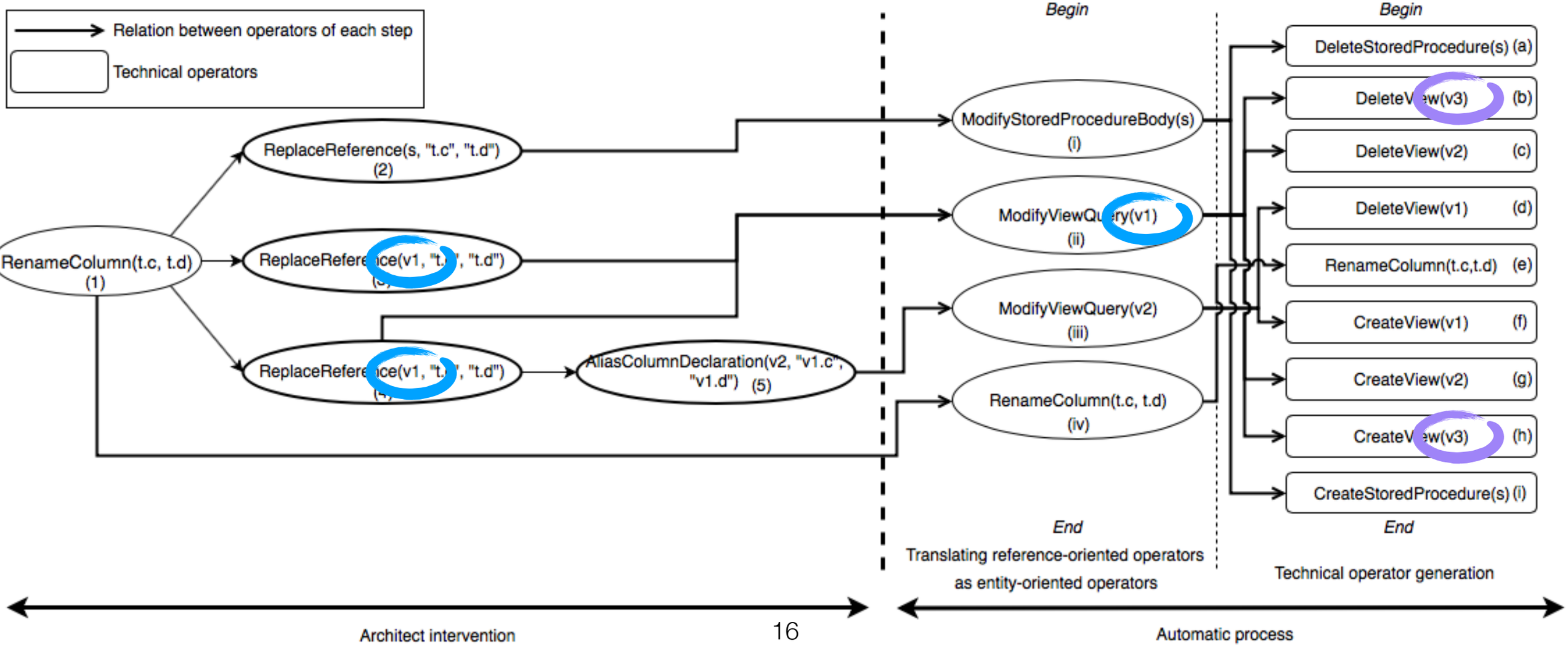




Architect choices compilation







Experiment

- Information system to manage members of our university department

- 95 tables, 63 views, 109 stored procedures, 20 triggers

- Post-mortem analysis of a SQL patch on this database

- We observed trial-and-error process from DBA to find dependencies between entities in a previous study*

- 1h to achieve a script of 200 LOC / 19 statements

* Julien Delplanque, Anne Etien, Nicolas Anquetil, and Olivier Auverlot. Relational Database Schema Evolution: An Industrial Case Study. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018.

Experimental protocol

1. **Extract initial** semantic **operators** from SQL script

2. Take decisions using **DBA's policy** when multiple possibilities

3. **Execute** generated **SQL** script on a database in the same state as information system's **database before the migration**

Experiment - Initial semantic operators extracted

- RenameColumn(person.uid, login)
- RemoveFunction(key_for_uid(varchar))
- RemoveFunction(is_responsible_of(int4))
- RemoveFunction(is_responsible_of(int4,int4))
- RenameFunction(uid(integer), login(integer))
- RenameLocalVariable(login.uidperson, login.loginperson)
- RemoveView(test_member_view)

Results

- **15 decisions** taken concerning the choice between renaming and aliasing.

- **270 LOC** script with **27 SQL statements**.

- Generated script **executed without error**.

- **Single difference** between the dump of the original database and our clone: **a comment**.

- **Time to implement** the evolution using our tool: **15 min** (v.s. 60 min without tool).

Conclusion

- Approach developed to address 2 main constraints set by DBMS:

    1. **No inconsistency is allowed**

    2. **Stored procedure bodies are black box**

- Main contributions:

    A. **Meta-model** for relational databases **easing impact computation**

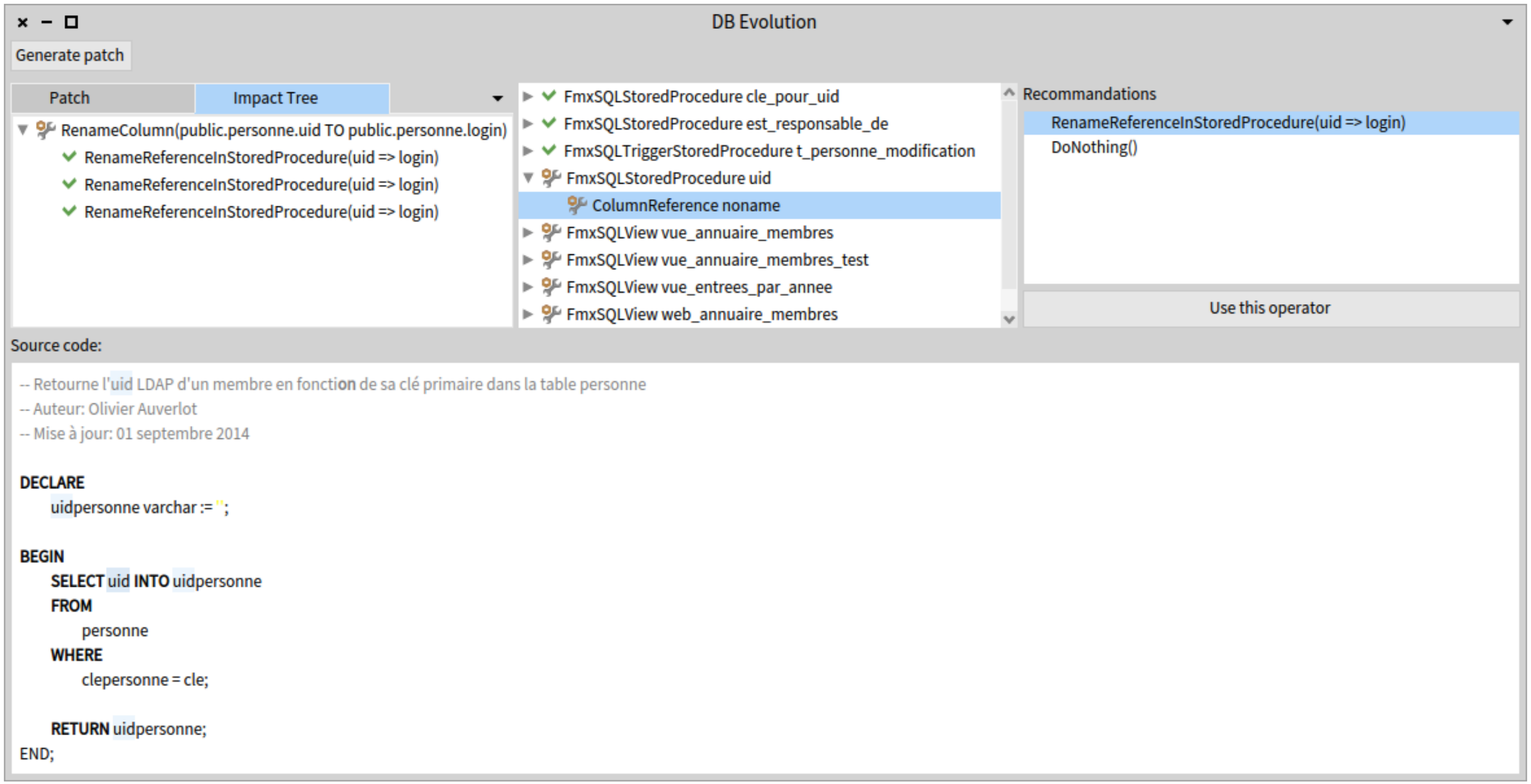
    B. **Semi-automatic approach** to help in database **evolution**

    C. **Experiment** to **assess** the **effectiveness** of our approach

Future work

- Extend supported semantic operators
- Compare the performance of architects using or not our tool to achieve the same evolution
- Empirical study on multiple open-source projects using relational a database
  - Assess if our tool can reproduce various changes
  - Compare changes proposed by our tool with changes historically applied on the database

Prototype


DB Evolution
Generate patch
Patch
Impact Tree
RenameColumn(public.personne.uid TO public.personne.login)
RenameReferenceInStoredProcedure(uid => login)
RenameReferenceInStoredProcedure(uid => login)
RenameReferenceInStoredProcedure(uid => login)
FmxSQLStoredProcedure cle_pour_uid
FmxSQLStoredProcedure est_responsable_de
FmxSQLTriggerStoredProcedure t_personne_modification
FmxSQLStoredProcedure uid
ColumnReference noname
FmxSQLView vue_annuaire_membres
FmxSQLView vue_annuaire_membres_test
FmxSQLView vue_entrees_par_annee
FmxSQLView web_annuaire_membres
Recommandations
RenameReferenceInStoredProcedure(uid => login)
DoNothing()
Use this operator
Source code:
-- Retourne l'uid LDAP d'un membre en fonction de sa clé primaire dans la table personne
-- Auteur: Olivier Auverlot
-- Mise à jour: 01 septembre 2014
DECLARE
uidpersonne varchar := '';
BEGIN
SELECT uid INTO uidpersonne
FROM
personne
WHERE
clepersonne = cle;
RETURN uidpersonne;
END;

Example: remove person.name

person

id	~~name~~	favorite_beer
1	~~Julien~~	3
2	~~Benoit~~	1
3	~~Guillaume~~	2
4	~~Cyrille~~	2

beer

id	designation
1	Lupulus
2	Duvel
3	Troll

Example: remove person.name

person

id	~~name~~	favorite_beer
1	Julien	3
2	Benoit	1
3	Guillaume	2
4	Cyrille	2

Change applied by DBMS

beer

id	designation
1	Lupulus
2	Duvel
3	Troll

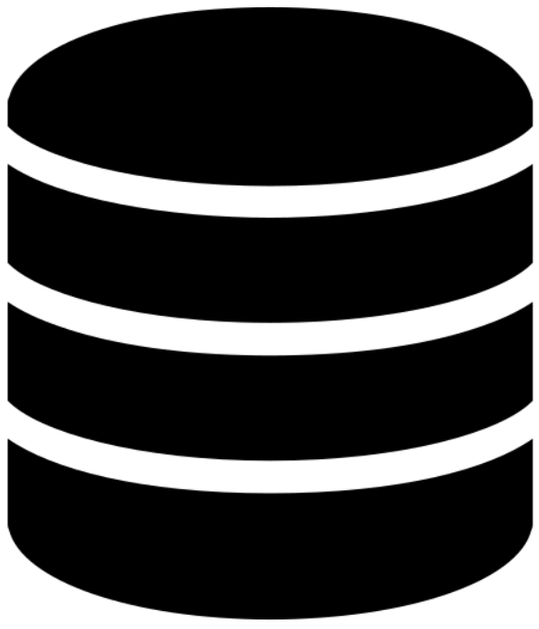
Approach Description

A. Impact computation

B. Recommendations selection

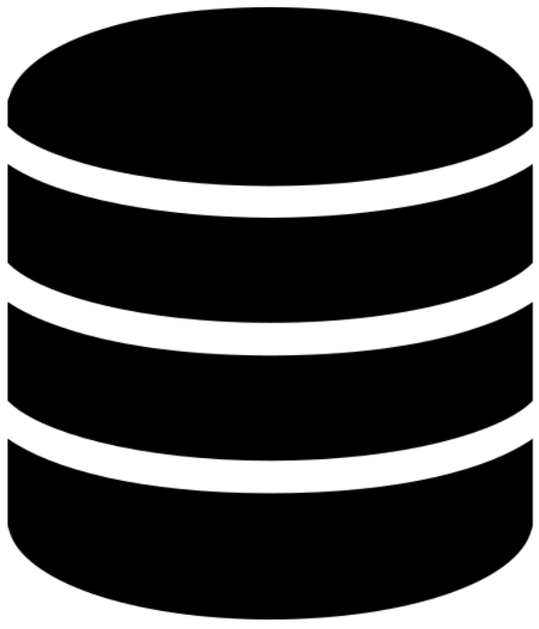
C. Compiling architect choices as a valid SQL script

About my PhD

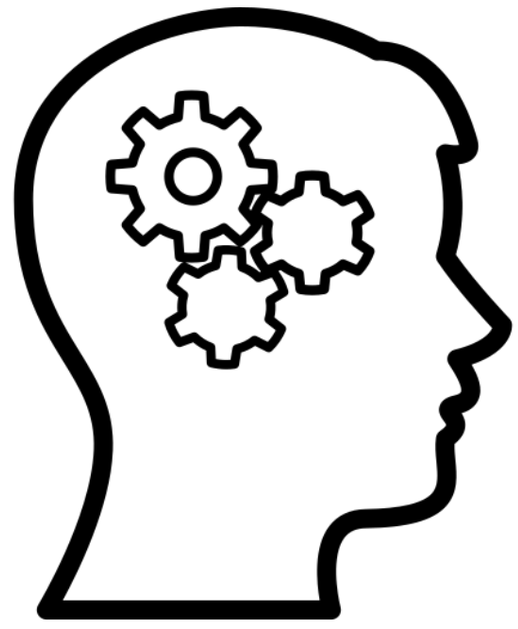


What quality problems databases have?

About my PhD

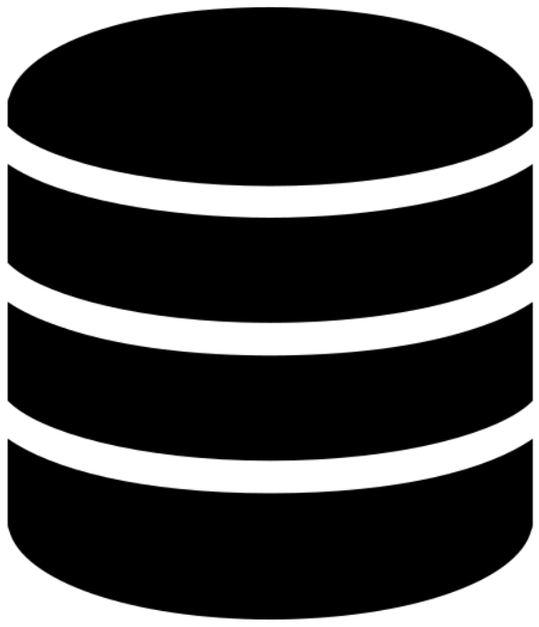


What quality problems databases have?

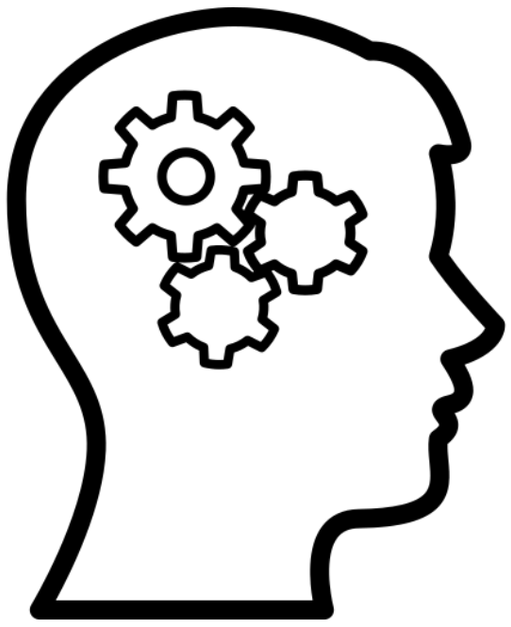


How DBAs make their DB evolve?

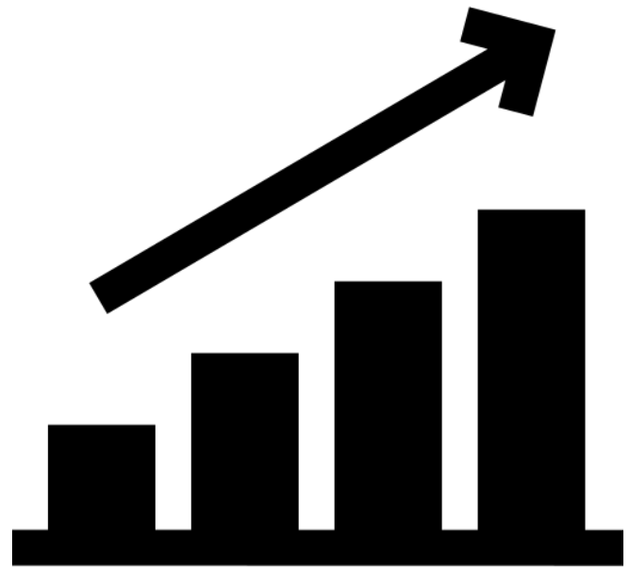
About my PhD



What quality problems databases have?

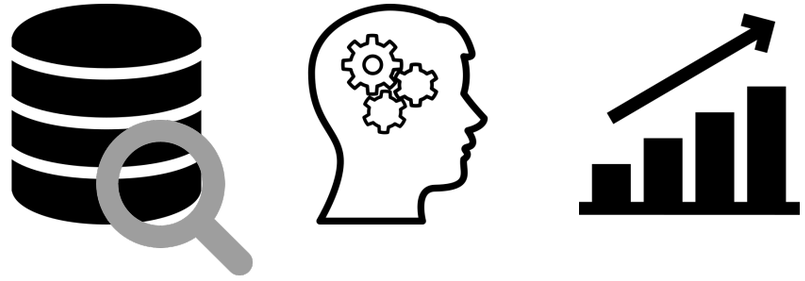


How DBAs make their DB evolve?



How to sustain DB evolution using software engineering techniques?

About my PhD



CodeCritics Applied to Database Schema: Challenges and First Results

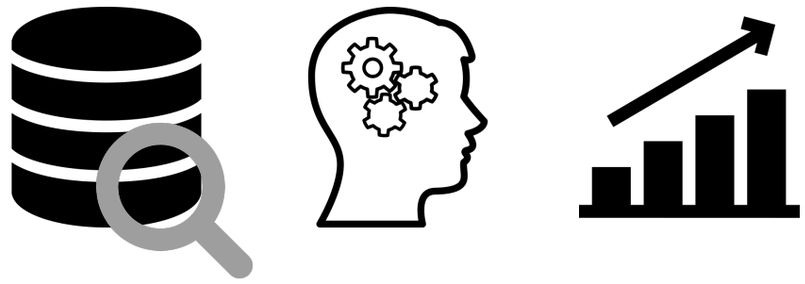
Julien Delplanque*†, Anne Etien*, Olivier Auverlot*, Tom Mens†, Nicolas Anquetil* and Stéphane Ducasse*
* Université de Lille, CRIStAL, CNRS, UMR 9189, Lille, France
RMoD Team, Inria Lille Nord Europe
{firstname.lastname}@inria.fr
† Software Engineering Lab, Université de Mons, Belgium
julien.delplanque@student.umons.ac.be, tom.mens@umons.ac.be

***Abstract*—Relational databases (DB) play a critical role in many information systems. For different reasons, their schemas gather not only tables and columns but also views, triggers or stored functions (*i.e.,* fragments of code describing treatments). As for any other code-related artefact, software quality in a DB schema helps avoiding future bugs. However, few tools exist to analyse DB quality and prevent the introduction of technical debt. Moreover, these tools suffer from limitations like the difficulty to deal with some entities (*e.g.,* functions) or dependencies between entities. This paper presents research issues related to assessing the software quality of a DB schema by adapting existing source code analysis research to database schemas. We present preliminary results that have been validated through the implementation of *DBCritics*, a prototype tool to perform static analysis on the SQL source code of a database schema. *DBCritics* addresses the**

The usual answer to DB evolution is to rely on metadata describing the DB schema. But these metadata do not consider, for example, the body of internal functions nor the requests building views. As a consequence the issues described above (similar to code smells), cannot be detected. Yet they may have important impact on the application using the DB. Quality analysis should therefore go beyond metadata only, and also consider internal treatments (*e.g.* stored procedure, triggers).

This paper adopts a software engineering approach by considering a DB similarly to a program and proposing to adapt existing software quality tools and techniques to it. Such an approach should be language dependent as is the case for most static analysis techniques in reverse engineering. It requires to reverse engineer the DB schema and to analyse its

About my PhD



CodeCritics Applied to Database Schema: Challenges and First Results

Julien Delplanque*†, Anne Etien*, Olivier Auverlot*, Tom Mens†, Nicolas Anquetil* and Stéphane Ducasse*
* Université de Lille, CRIStAL, CNRS, UMR 9189, Lille, France

Abstr
informa
not only
function
any oth
helps av
quality
these to
some er
This pa
software
analysis
results
DBCriti
SQL so

Database Critics Browser - /home/julien/Documents/Pharo/DCB/AppSI/schema20160125.

Rules

- Column not key (PK/FK) including "cle" in name. (11)
- Foreign key referencing a non primary key. (0)
- High number of columns in a table/view. (2)
- High number of columns is SELECT request. (1)
- SELECT request using *. (0)
- Stub entity. (11)
- Table alone. (13)
- Table without primary key. (9)
- View using another view. (4)
- View using only one table. (10)

Entities

- diplome.cle_coencadrant (Column)
- soutenance.cle_diplome (Column)
- personne.cle_nationalite (Column)
- quotite_support.cle_quotite (Column)
- quotite_support.cle_support (Column)
- web_equipes_cristal.cle_theme (Column)
- web_equipes_cristal.cle_type_equipe (Column)
- vue_liste_personnel_affectations.clepersonne (Column)
- vue_annuaire_membres_test.clepersonne (Column)
- vue_annuaire_membres.clepersonne (Column)
- vue_liste_personnel_affectations.cletypesupport (Column)

Mark as false positive | Inspect

Generally, when a column name uses "cle", it is either a PK or a FK. If not the case, it may be an error.

About my PhD



Relational Database Schema Evolution: An Industrial Case Study

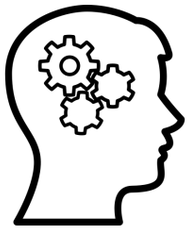
Julien Delplanque, Anne Etien, Nicolas Anquetil and Olivier Auverlot
Université de Lille, CRIStAL, CNRS, UMR 9189,
RMoD Team, Inria Lille Nord Europe
Lille, France
{firstname}.{lastname}@inria.fr, olivier.auverlot@univ-lille1.fr

***Abstract*—Modern relational database management systems provide advanced features allowing, for example, to include behaviour directly inside the database (stored procedures). These features raise new difficulties when a database needs to evolve (*e.g.* adding a new table). To get a better understanding of these difficulties, we recorded and studied the actions of a database architect during a complex evolution of the database at the core of a software system. From our analysis, problems faced by the database architect are extracted, generalized and explored through the prism of software engineering. Six problems are identified: (1) difficulty in analysing and visualising dependencies between database's entities, (2) difficulty in evaluating the impact of a modification on the database, (3) replicating the evolution of the database schema on other instances of the database, (4) difficulty in testing database's functionalities, (5) lack of synchronization between the IDE's internal model of the database and the database actual state and (6) absence of an integrated tool**

introduce new complexity in the management of the databases. We were asked by a database architect in our university to look at the matter and see if we could propose solutions to help him evolve a large PostgreSQL database (95 tables). This database has some characteristics that make it difficult to evolve:

- It has many views (62). In PostgreSQL, modifying a table used by a view might requires deleting the view first, then modifying the table and finally recreating the view. If this view itself is used by another view, the other one also has to be removed and recreated (in cascade).
- It has many stored functions (64). In PostgreSQL, stored functions are just text, so if a stored function accesses a table (or view) that has been modified, there is no check or warning from the RDBMS. The validity of the

The topic today


Recommendations for Evolving Legacy Databases

***Abstract*—Relational databases play a central role in many information systems. Their schemas usually contain structural and behavioral entity descriptions. However, as any piece of software, they must continuously evolve to adapt to new requirements of a world in constant change. From an evolution point of view, problems are twofold: (1) relational database management systems do not allow inconsistencies *i.e.,* no entity can reference a non existing entity; (2) stored procedure bodies are black boxes *i.e.,* DBMS as PostgreSQL consider stored procedure bodies as plain text and references to entities are unknown. As a consequence, evaluating the impact of an evolution of the database schema is a difficult task. In this article, we present a semi-automatic approach based on recommendations (sort of nested code transformations). Recommendations are proposed to architects who select the ones fitting their needs. Selected recommendations are then analysed and compiled to generate SQL script respecting the constraints imposed by the RDBMS. To support recommendations, we designed a meta-model for relational databases easing computation of change impact. We performed an experiment to validate the approach by reproducing a real evolution on a database. Thus, contributions are: 1) a meta-model for relational databases easing the computation of the impact of a change, 2) a semi-automatic approach for evolving a database while managing the impact of subsequent nested code**

For example, to remove a column of a table from a database, different cases occur:

(i) If the column is not referenced, the change can be performed.

(ii) If the column is a primary key and is referenced as foreign key in another table, the removal is not allowed.

(iii) If the column is referenced in views, the DBMS offers two possibilities to the architect. Either the change is refused or all the views referencing the column to remove must be dropped. In the later case, all the views referencing the views to drop must also be transitively dropped to keep the schema in a consistent state. The list is provided to the architect before the removal. However, the architect can only accept all the drops in cascade or the initial change (removing the column of a table) is refused.

(iv) If the column is referenced in a function, the DBMS accepts the removal of the column, but an error might occur at run time.

Cases ii and iii result from the first problem, *i.e.,* the way