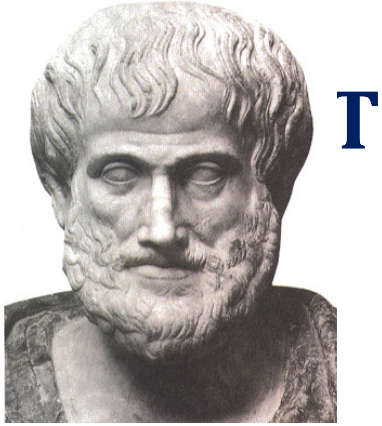
Schema evolution for traditional databases and data warehouses

Panos Vassiliadis

also: Apostolos Zarras, Petros Manousis, Ioannis Skoulis, George Papastefanatos*

Department of Computer Science and Engineering
University of Ioannina, Hellas

* Research Center “Athena”, IMIS, Athens



The nature that needs change is vicious;
for it is not simple nor good…

Nicomachean Ethics, Book VII, Aristotle

SWEBOK Maintenance

	Correction	**Enhancement**
Proactive	Preventive	**Perfective**
Reactive	**Corrective**	Adaptive

- **Corrective** maintenance: reactive modification (or repairs) of a software product performed after delivery to **correct discovered problems**.
- **Adaptive** maintenance: modification of a software product performed after delivery **to keep a software product usable in a changed or changing environment.**
- **Perfective** maintenance: modification of a software product after delivery to provide enhancements for users, improvement of program documentation, and recoding **to improve software performance, maintainability, or other software attributes.**
- Preventive maintenance: modification of a software product after delivery to detect and correct latent faults in the software product before they become operational faults.

Database Evolution: why and what

- Software systems and, thus, databases are dynamic environments and can evolve due
  - Changes of requirements
  - Internal restructuring due to performance reasons
  - migration / integration of data from another system
  - …
- Database evolution concerns
  - changes in the content (**data**) of the databases as time passes by
  - changes in the internal structure, or **schema**, of the database
  - changes in the operational environment of the database

What evolves in DBMS…

- Data

EMP_ID	SALARY
100	1500

→

EMP_ID	SALARY
100	1650

```
UPDATE EMP
SET SALARY = SALARY *1.10
WHERE...
```

- Metadata – Schemata – Models

```
ALTER TABLE EMP
ADD COLUMN PHONE VARCHAR ...
```

EMP_ID	SALARY
100	1500

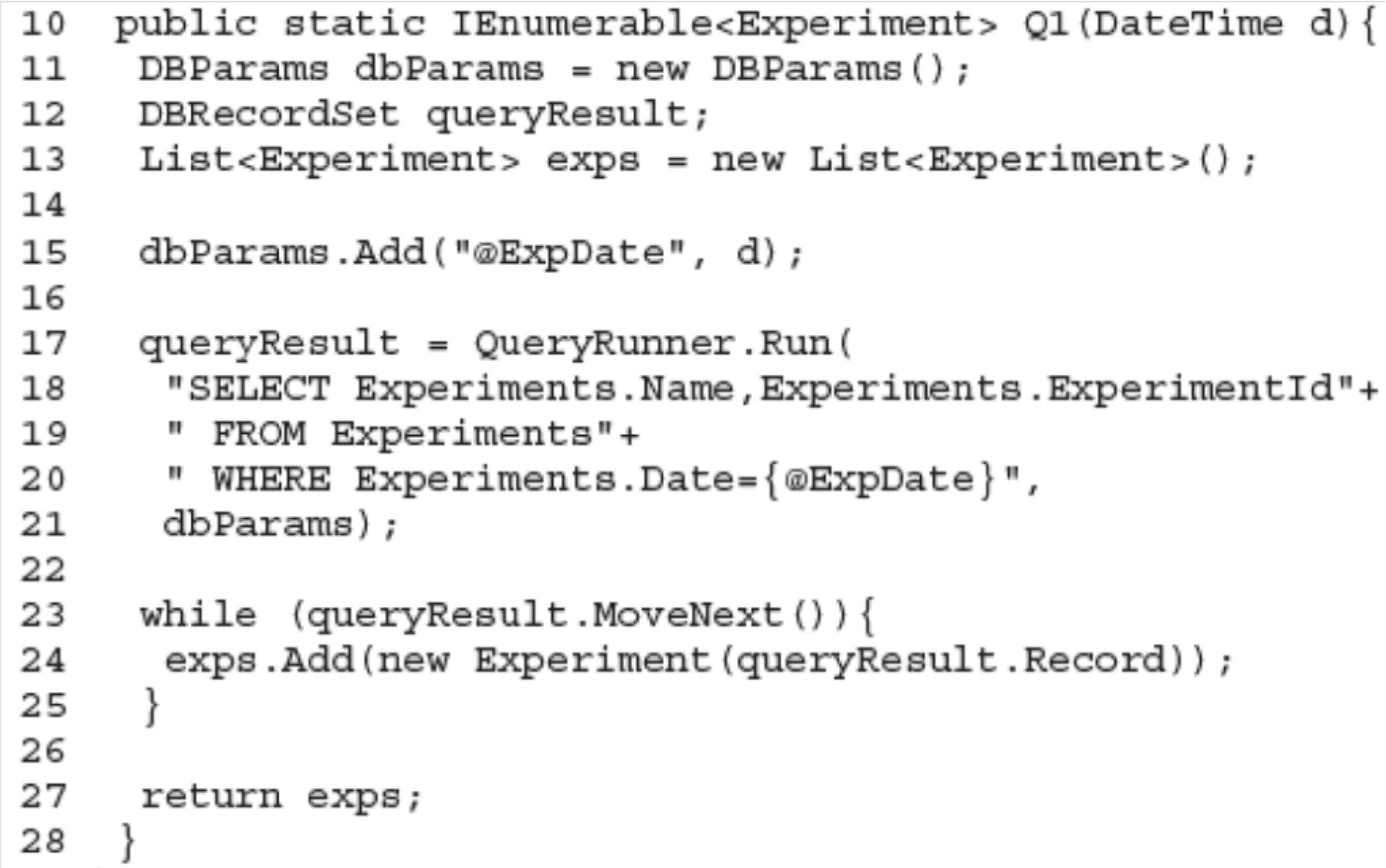
→

EMP_ID	SALARY	PHONE
100	1650	210777777

Why is (schema) evolution so important?

- Software and DB **maintenance** makes up for **at least 50% of all resources spent in a project**.
- Changes are more frequent than you think
- **Databases are rarely stand-alone: typically, an entire ecosystem of applications is structured around them =>**
- **Changes in the schema can impact a large (typically, not traced) number of surrounding app's, without explicit identification of the impact**

Some exemplary code [Maule+08]



```
public static IEnumerable<Experiment> Q1(DateTime d){
 DBParams dbParams = new DBParams();
 DBRecordSet queryResult;
 List<Experiment> exps = new List<Experiment>();

 dbParams.Add("@ExpDate", d);

 queryResult = QueryRunner.Run(
  "SELECT Experiments.Name,Experiments.ExperimentId"+
  " FROM Experiments"+
  " WHERE Experiments.Date={@ExpDate}",
  dbParams);

 while (queryResult.MoveNext()){
  exps.Add(new Experiment(queryResult.Record));
 }

 return exps;
}
```

Evolution taxonomy

- **Schema evolution**, itself, can be addressed at
  - the conceptual level (req's, goals, conc. model, …. Evolve)
  - the logical level, where the main constructs of the database structure evolve
    - E.g.,: relations and views in the relational area, classes in the object-oriented database area, or (XML) elements in the XML/semi-structured area),
  - the physical level, involving data placement and partitioning, indexing, compression, archiving etc.

Evolution taxonomy: areas

- Relational databases
- Object Oriented db’s
- Conceptual models
- XML
- Ontologies
- …

- Special case of relational: data warehouses

… To probe further …

- Michael Hartung, James F. Terwilliger, Erhard Rahm: Recent Advances in Schema and Ontology Evolution. In Schema Matching and Mapping (Zohra Bellahsene, Angela Bonifati, Erhard Rahm), 149-190, Springer 2011, ISBN 978-3-642-16517-7

- Matteo Golfarelli, Stefano Rizzi: A Survey on Temporal Data Warehousing. IJDWM 5(1): 1-17 (2009)
- Robert Wrembel: A Survey of Managing the Evolution of Data Warehouses. IJDWM 5(2): 24-56 (2009)

Roadmap

- Evolution of views
- Data warehouse Evolution
  - A case study if time
- Impact assessment in ecosystems
- Empirical studies for database evolution
- Open Issues and discussions

Roadmap

- **Evolution of views**
- Data warehouse Evolution
- A case study (if time)
- Impact assessment in ecosystems
- Empirical studies concerning database evolution
- Open Issues and discussions

- What views and mat. views are
- Traditional research problems related to views
- View adaptation
- Significant works

VIEW ADAPTATION

Views

- **Virtual** views: macros that allow the developers to construct queries easier by using them as tables in subsequent queries

```
CREATE VIEW sales_vv AS
SELECT t.calendar_year, p.prod_id, SUM(s.amount_sold) AS
  sum_sales
FROM times t, products p, sales s
WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
GROUP BY t.calendar_year, p.prod_id;
```

Query:

```
SELECT * FROM sales_vv WHERE calendar_year > 2012;
```

http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_6002.htm#SQLRF01302

Views

- **Materialized** views are not macros, however, as they actually store (precompute) the result in persistent storage

```
CREATE MATERIALIZED VIEW sales_mv
BUILD IMMEDIATE
REFRESH FAST ON COMMIT
AS
SELECT t.calendar_year, p.prod_id,
  SUM(s.amount_sold) AS sum_sales
FROM times t, products p, sales s
WHERE t.time_id = s.time_id AND p.prod_id =
  s.prod_id
GROUP BY t.calendar_year, p.prod_id;
```

Traditional research problems with views

- **Query answering**: how to integrate views (of all kinds) in the optimizer's plan?
- **View selection**: which views to materialized given query and update workloads?
- **View maintenance**: how to update the stored extent of the mat. view when changes occur at the sources?
  - For which views can I do it? (query class)
  - How: Full or Incremental?
  - When: On update? On demand? Periodically?
  - Available info: deltas only? Int. constraints?

Oracle 11g and Materialized Views

```
CREATE MATERIALIZED VIEW view-name
```

```
BUILD [IMMEDIATE | DEFERRED]
```

- Compute extent at view definition or at query time

```
REFRESH [FAST | COMPLETE | FORCE ]
```

- FAST: incremental (needs log def. on source tables); COMPLETE: full; FORCE: if FAST fails, then COMPLETE

```
ON [COMMIT | DEMAND ]
```

- Trigger refresh when sources are updated, or on-demand

```
[[ENABLE | DISABLE] QUERY REWRITE]
```

- Used by the optimizer during Query Optimization

```
AS SELECT ... query definition …;
```

```
CREATE MATERIALIZED VIEW LOG ON times
WITH ROWID, SEQUENCE (time_id,
calendar_year)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON products
WITH ROWID, SEQUENCE (prod_id)
INCLUDING NEW VALUES;
```

View adaptation

- What if there is a change in
  - the view definition?
  - the schema of the sources?

- Can we maintain the view's
  - definition
  - extent
- correctly and efficiently?

Gupta et al @ Inf. Systems, 26(5), 2001

- **Assume the view definition changes**
- Given
  - the old and the new view definition
  - the existing data that are stored in the view
  - the source tables
  - (when needed: auxiliary information, like indexes on PK’s, aux. relations, …)
- Produce the extent corresponding to the new view definition
- Such that
  - It is done incrementally rather than via a complete recomputation

A “taxonomy” of atomic changes to SPJ and SPJG+ views

Method: The authors assume a comprehensive set of potential **atomic changes.**

- Addition or deletion of an attribute in the SELECT clause.
- Addition, deletion, or modification of a predicate in the WHERE clause (with and without aggregation).
- Addition or deletion of a join operand (in the FROM clause), with associated equijoin predicates and attributes in the SELECT clause.
- Addition or deletion of an attribute from the GROUPBY list.
- Addition or deletion of an aggregation function to a GROUPBY view.
- Addition, deletion, or modification of a predicate in the HAVING clause. Addition of the first predicate or deletion of the last predicate corresponds to addition and deletion of the HAVING clause itself.
- Addition or deletion of an operand to the UNION and EXCEPT operators.
- Addition or deletion of the DISTINCT operator.

For each type of change the authors propose a set of steps required to maintain the view’s extent

Example: Adding an atomic selection to the WHERE clause

Assume we add a filter **Q** to a view V which becomes V’

```
CREATE VIEW V’ AS
SELECT A1, …, An
FROM R1 & … & Rm
WHERE Q AND C1 AND … AND Ck
```

We want to maintain V0 given its old extent and the source relations.

Algebraically: $\mathbf{V' = V - V^{-} \cup V^{+}}$

where V+ are the tuples that should be inserted in the view and V^{-} are the tuples to be removed

```
DELETE FROM V WHERE NOT Q                //delete V-
INSERT INTO V ( SELECT A1, …, An          //add V+ (here: empty)
FROM R1 & … & Rm
WHERE Q1 AND NOT C1 AND … AND NOT Ck )
```

Important notes

- **Maintenance is incremental**: you try to recompute V by checking out only the existing data
- “Taxonomy” of atomic changes with locality principle: if you are given a complex redefinition, you can process it one change at a time (atomic changes are composable)

Nica et al., EDBT 1998

- What if the schema in one of the relations participating to the view definition changes?
- The method by Nica et al., proposes an algorithm (heavily oriented towards handling **deletions**) for rewritting the view to address the change
- Two pillars:
  - A Meta Knowledge Base keeping semantic properties of the database
  - The annotation of views with directives on how to respond to changes

Meta Knowledge Base

- Information on
  - Available relations and views
  - Implicit join conditions
  - Semantic equivalences: which attribute/relation can be regarded as a potential replacement for another
- For example:
  - Join conditions:
    - `product.prod_id = sales.prod_id`
  - Equivalence assertions:
    - `sales.prod_id = product.prod_id`
    - `times.calendar_year = year(sales.time_id)`

View annotation

- E-SQL: language to annotate parts of a view (exported attributes, underlying relations and filters) wrt:
  - Dispensability: if the part can be removed from the view definition completely
  - Replaceability with an another equivalent part.

```
CREATE VIEW empProj_VV AS
SELECT e.EName, e.Ephone (AD true, AR true) p.PName,
  w.PDuration
FROM EMP e (RR true), PROJECT p,  WORKS w
WHERE (e.Eid = w.Eid) AND (p.Pid = w.Pid) AND
  (p.Plocation=Barcelona) (CD true)

//assuming a relation EMP_ContactInfo duplicating id, name,
  phone of EMP's, possibly with other contact info means
```

Complex View Synchronization algorithm

- Input : (0) an SPJ view V, (1) a change in a relation, (2) old MKB entities, and, (3) new MKB entities.
- Output: view rewritings to adapt to new MKB providing the same result
- Means: model that represents attributes as hyper-nodes and (i) relations, (ii) join cond., and (iii) equivalence assertions as hyper-edges
- Steps:
  - find all entities affected for Old MKB to become New MKB,
  - for each one of these entities find a replacement from Old MKB,
  - rewrite the view over these replacements.

Roadmap

- Evolution of views
- **Data warehouse Evolution**
- A case study (if time)
- Impact assessment in ecosystems
- Empirical studies concerning database evolution
- Open Issues and discussions

- DWs as Views

- Evolving dimensions & SCD

- Multiversion DWs & cross-version queries

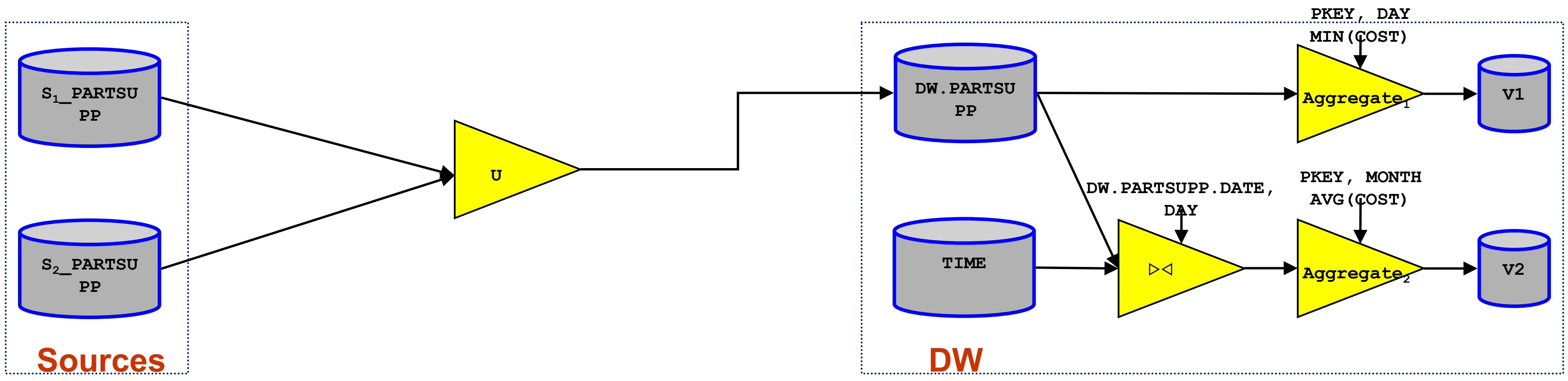
- Bonus: a case study, if time permits

DATA WAREHOUSE EVOLUTION

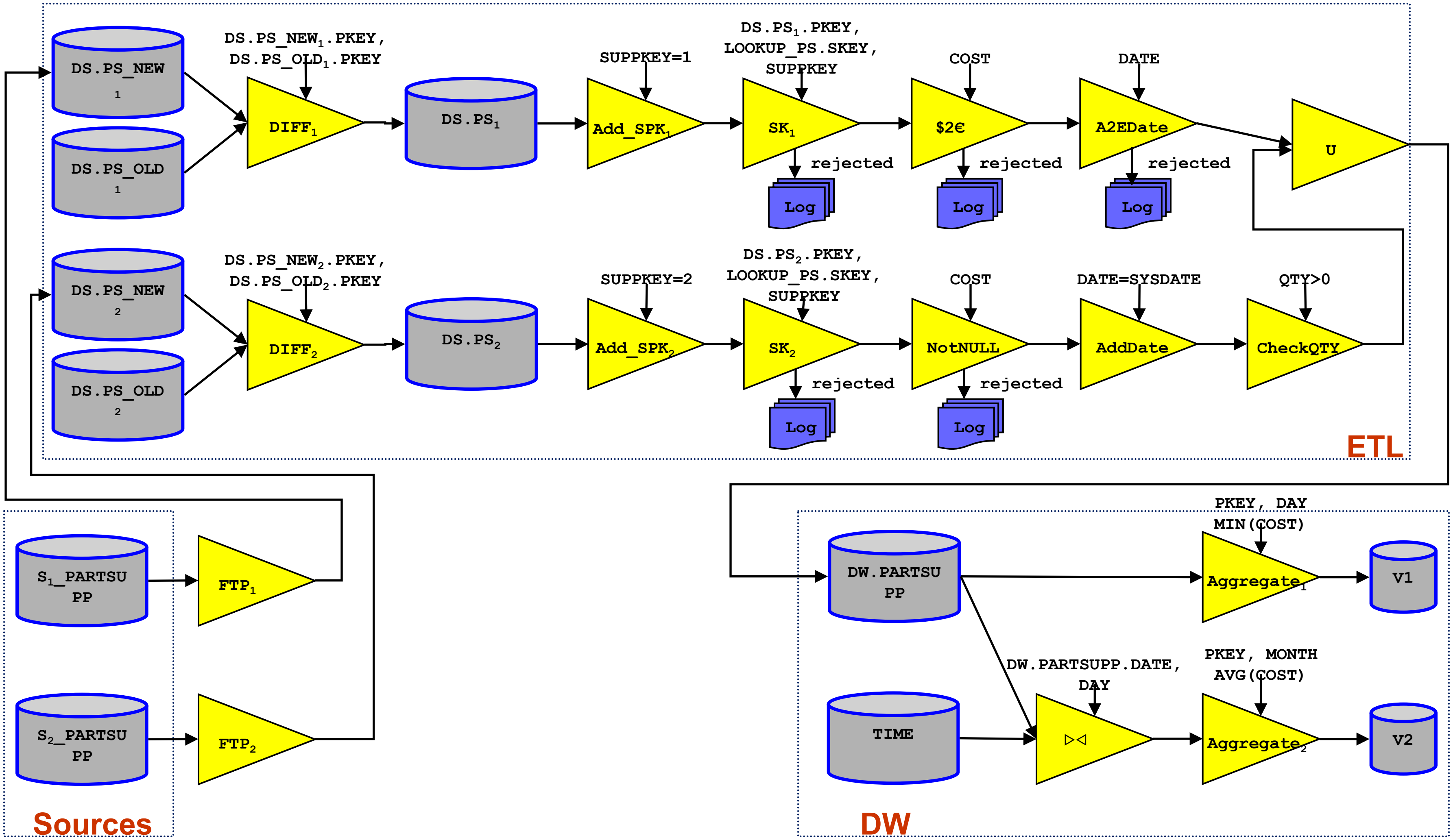
Early days (late ‘90s)

- Back then, people continued to think that DWs were **collections of materialized views**, defined over sources.
- In this case, evolution is mostly an issue of adapting the views’ definitions whenever sources changes

DW = set of mat. views?



NO!! DW ≠ Mat. views!



Bellahsene (DEXA’98, KAIS02)

- Annotate views with a HIDE clause that works oppositely to SELECT (i.e., you project all attributes except for the hidden ones)
- Also: ADD ATTRIBUTE to equip views with attributes not present in the sources (e.g., timestamps, calculations)
- What if **sources** change? The author considers attribute/relation addition & deletion and the impact it has to view rematerialization (how to recompute the materialized extent via SQL commands)
- Cost model to estimate the cost of different options

Bellahsene (DEXA'98, KAIS02)

```
[Create] view <view name> As <query expression>;
[Add Attribute <attribute name>:<query expression>=<type>]
[Hide Attribute <attribute name>];
```

Used for schema evolution too. See how to change the type of an attribute

```
Create View V' As Select From V;
Hide attribute A;
Add attribute A : T /* new type for A in V' */
Drop view V;
Rename V' as V;
```

Example stolen from paper

Quix @ DMDW '99

- Context: DW schemata annotated with quality factors
- Metadata that track the history of changes and provide a set of rules to enforce when a quality factor (completeness, consistency, correctness, …) has to be reevaluated.
- Basic **taxonomy of changes**

	Relation	View	Attribute	Constraint
Add	✓	✓	✓	✓
Delete	✓	✓	✓	✓
Rename	✓	✓	✓	
Redefine semantics		✓		

… and then came dimension buses and multidimensional models …

- … which treat the DW as a collection of
  - **cubes**, representing clean, undisputed facts that are to be loaded from the sources, cleaned and transformed, and eventually queried by the client applications
  - defined over **unambiguous, consolidated dimensions** that uniquely and commonly define the context of the facts

- **… The idea of a central DW schema acting as reference for the back-stage loading and front-end querying completely changed the perspective of DW research …**

Slowly Changing Dimensions

What you ‘ve probably heard for dimension updates is SCD’s

- Type 0: no change allowed
- Type 1: new value overwrites old
- Type 2: new record; valid time timestamps + status columns indicate which row is current and what happened
  - New Surrogate Key (so joins with facts work as if these are different dimension records)
  - Same natural / detailed key (to be used in group by’s)
  - Status attribute: Current vs Old (aka Type 6)
- Type 3: add new column “PreviousValueForAttributeXXX” and update cells with new and old values respectively

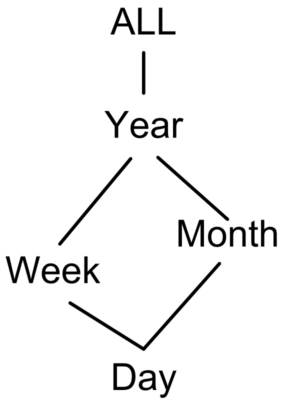
Slowly Changing Dimensions

- Type 4: definitions vary
  - Split type 2 table in two tables subsets of the data set: the historical one and the current one (single row)
  - Kimball’s: if some attributes of the dimension change frequently, export a new table (called “profile”) just for them; facts have two FK’s for the dimension, one for the dim table and another for the profile table

Quick guide to dimensional modeling

For each dimension:

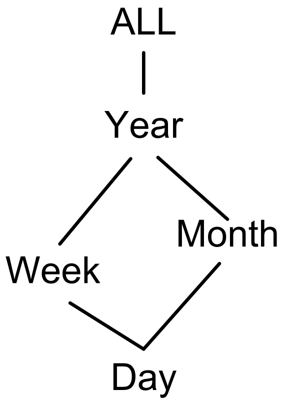
- Levels for “granularity degrees” of information
- Each level L with a domain dom(L) (typically isom. to integers)
- Can have attributes too
- Typically form a **lattice** with
  - a detailed level at bottom and
  - A single-valued ‘ALL’ level at the top
- **Rollup functions** between subsequent levels
- Have to be fully defined at the domain level and consistent under composition
- Drill-down relations (not functions): their inverse



Hurtado, Mendelzon and Vaisman @ DOLAP99, ICDE'99

Research issue:

- What is the **algebra** of operators to change the dimensions of a DW?
- What are the **operators**?
- How do they **affect the schema and data** of dimensions and cubes?



- [HuMN99a,b] Set of **operators** for evolving dimensions prescribing what should be done to have both a consistent schema and a consistent set of instances

Hurtado, Mendelzon and Vaisman @ DOLAP99, ICDE’99

Generalize	Adds a new level above a preexisting one, + a rollup function
Specialize	Adds a new level below the current bottom level + a rollup function
Relate	Adds a new edge, between two parallel levels. The associated rollup function, if it exists, is determined automatically. If not possible to do so uniquely, the operator is not applicable.
Unrelate	Deletes an edge between two levels.
Delete Level	Deletes a level with the precondition that the new hierarchy must have a unique bottom level (ALL cannot be deleted).
Add Instance	Adds a value, say x, + a pair of the form (x,y) for each rollup function
Delete Instance	Deletes a value x from a level L + rollup functions
Reclassify	Update rollup-memberships (e.g., a brand moves to a new company)
Split & Merge	Czechoslovakia <-> Czechia & Slovakia + rollup functions
Update	Rename value without structural changes

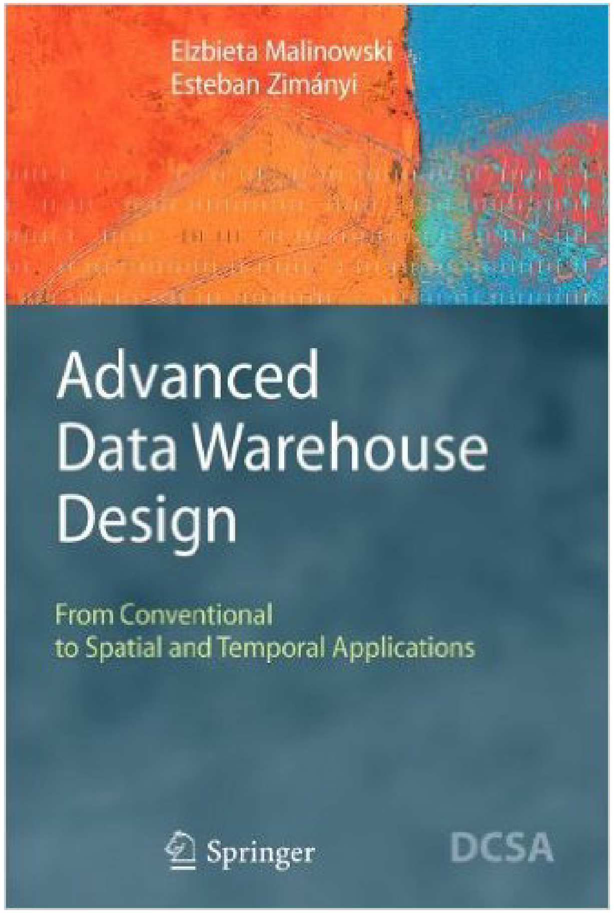
Blaschka, Sapia and Höfling @DaWaK'99

- Data model + an **evolution algebra** :
  - **Evolution operators** for multi-dimensional schemata and
  - Spec. of their effects to both schema and instances.
- Operators: **atomic evolution operations**,
- that can be used for complex operations.

Algebraic Operator
Insert level
Delete level
Insert Attribute
Delete Attribute
Connect attribute to dimension level
Disconnect attribute from dimension level
Connect attribute to fact
Disconnect attribute to fact
Insert classification relationship
Delete classification relationship
Insert fact
Delete fact
Insert dimension into fact
Delete Dimension

… and then came versioning…

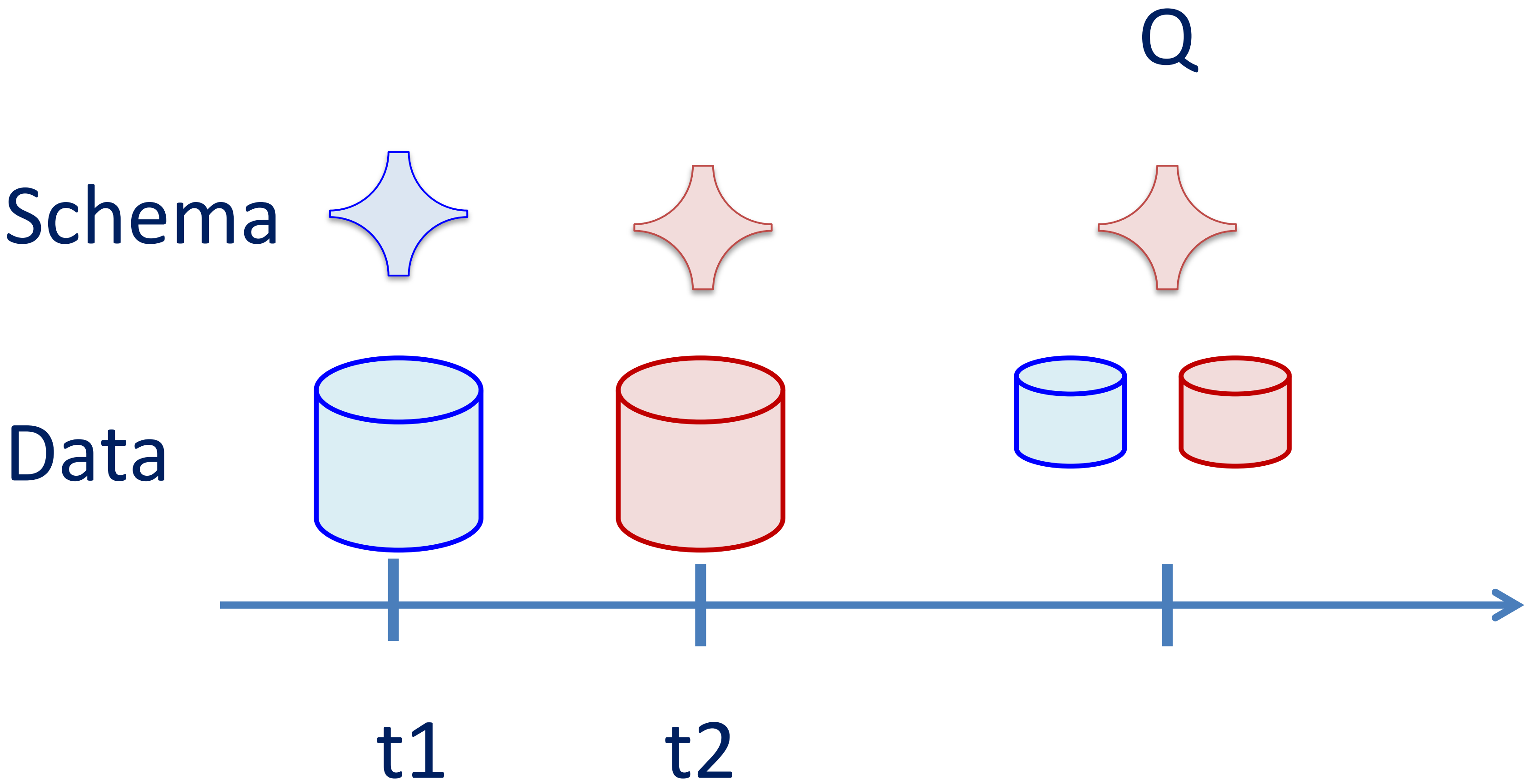
- After we had obtained a basic understanding of how multidimensional schemata are restructured, people thought:
- “what if we keep track of the history of all the versions of a DW schema as it evolves?”
- Then, **we can ask a query that spans versions**, and transform its results into a convenient schema to show to the users


Elzbieta Malinowski
Esteban Zimányi
Advanced Data Warehouse Design
From Conventional to Spatial and Temporal Applications
Springer
DCSA

Closely related to **temporal management in DW’s**

See later today presentation by Waqas Ahmed

Cross-Version Querying



Q

Schema

Data

t1

t2

Eder and Koncilia @ DaWaK 2001

- Multidimensional data model that allows the registration of temporal versions of dimension data in data warehouses.
- To navigate between temporal versions: mappings as **transformation matrices**. Each matrix is a mapping of data from structure Vi to V i+1 for a dimension D. For example, table T describes a split of value a into values a1 and a2 respectively. There is an mapping function that describes that the 30% of the fact –values for A should be placed to a1 and the remaining should be placed in a2 .

T	A1	A2
A	30%	70%

- This mapping function is described in a transformation matrix T that says exactly that in order to go from A to A1 we need to take 30% of the tuples of table A and what remains goes to table A2.
- Queries are posed over snapshots of the database. For each query the appropriate snapshots are computed.

Eder and Koncilia @ DaWaK 2001

- We can transform each cuboid C (with facts) over a set of dimensions from version Vi to version Vi+1, by sequentially transforming each of its dimensions one at a time.
- **Original cube** (dimension values on the side)

$$C = \begin{array}{c|ccc} & a_1 & a_2 & a_3 \\ \hline b_1 & 3 & 7 & 5 \\ b_2 & 10 & 8 & 6 \\ b_3 & 20 & 13 & 5 \end{array}$$

- **Transformation matrices for dimensions**

$$T_A = \begin{array}{c|cccc} & a_{11} & a_{12} & a_2 & a_3 \\ \hline a_1 & 0.3 & 0.7 & 0 & 0 \\ a_2 & 0 & 0 & 1 & 0 \\ a_3 & 0 & 0 & 0 & 1 \end{array}$$

$$T_B = \begin{array}{c|ccc} & b_1 & b_2 & b_3 \\ \hline b_{12} & 1 & 1 & 0 \\ b_3 & 0 & 0 & 1 \end{array}$$

- **Final cube** with the **values** of the **original version** over the "structure" of the new version

$$\begin{array}{c|cccc} & a_{11} & a_{12} & a_2 & a_3 \\ \hline b_{12} & 3.9 & 9.1 & 15 & 11 \\ b_3 & 6 & 14 & 13 & 5 \end{array}$$

Photos stolen from the paper

Eder, Koncilia and Mitsche @ DaWaK’03, CAiSE’04

- Making use of three basic operations (INSERT, UPDATE and DELETE), the authors are able to represent more complex operations on dimension values such as: SPLIT, MERGE, CHANGE, MOVE, NEW-MEMBER, and DELETE-MEMBER.
- Also: data mining techniques for the detection of structural changes in data warehouses.

Golfarelli, Lechtenbörger, Rizzi and Vossen @ DKE 2006

- **How to facilitate cross-version queries?**
- A graph model for DW multidimensional schemata
- Nodes : (i) fact tables and (ii) their attributes of fact tables (including properties and measures),
- Edges: functional dependencies (aka dimension hierarchies) defined over the nodes

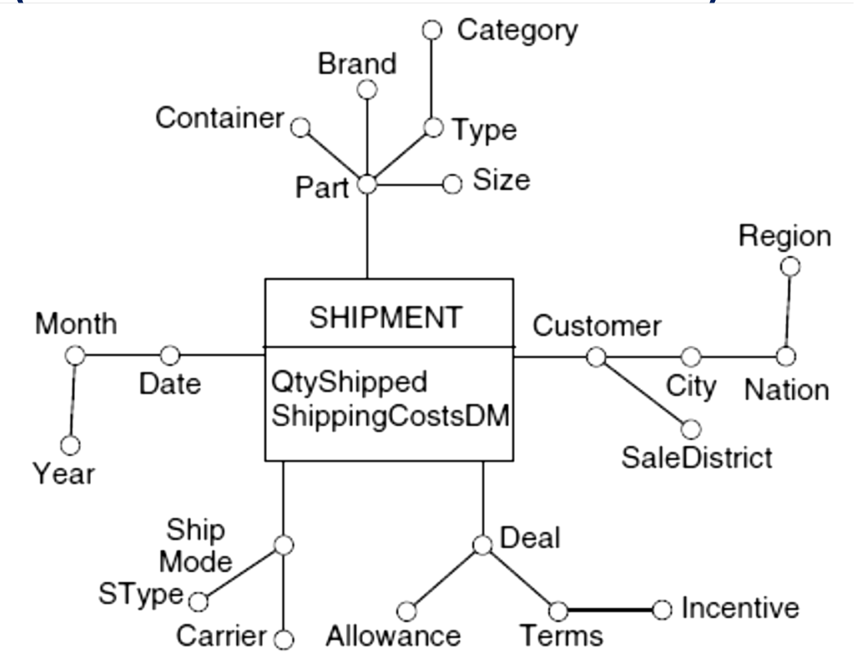


Photo stolen from the paper

Golfarelli, Lechtenbörger, Rizzi and Vossen @ DKE 2006

- Taxonomy of changes:
  - Add / delete node (i.e., tables and attributes)
  - Add / delete edge (i.e., restructure dimensions)
- Transactions = sequences of atomic changes

Golfarelli, Lechtenbörger, Rizzi and Vossen @ DKE 2006

Augmented schema of a previous version

- Assume a version S_k
- Assume a set of changes M_1,…, M_n
- Then you get to a version S_{k+n}
- The **augmented version** of S_k wrt S_{k+n} is the schema and data of S_k, along with all the extra attributes and FD's added at S_{k+n}
- So basically, we are **adapting the previous schema+data to the structure of the new version**
- This might require aggregations or disaggregations (and estimations of the necessary values), addition of default values, …

Golfarelli, Lechtenbörger, Rizzi and Vossen @ DKE 2006

- Assume a fact
  - **`SALES(ProdID, DayId, CustId, Price, Qty)`**
- With a set of dimensions
  - **`Product (Product, Type, Family)`**
  - **`Customer(Customer, CustGroup)`**
  - **`Time(Day, Month, Year)`**
- and a set of changes
  - Add attribute Salesman and a hierarchy Salesman -> Store
  - Remove day from the time hierarchy and replace it with Month
  - SumSales = Qty*Price
- Then, the new fact is

`SALES’(ProdID,MonthId,CustId,SalesmanId,Price,Qty,SumSales)`

Golfarelli, Lechtenbörger, Rizzi and Vossen @ DKE 2006

SALES(ProdID, DayId, CustId, Price, Qty)

SALES’(ProdID,MonthId,CustId,SalesmanId,Price,Qty,SumSales)

- We can compute the augmented version of the OLD schema

SALES^{Aug}(ProdID,DayId,MonthId,CustId,SalesmanId,Price,Qty,SumSales)

- …that includes @ schema level
  - The old attributes & FD’s
  - The new attributes & FD’s added during evolution
  - … hoping that all FD’s hold (otherwise there is no augmentation)
- … and at data level: values of SALES (the old v.) with interpolation for the measures due to dimension addition

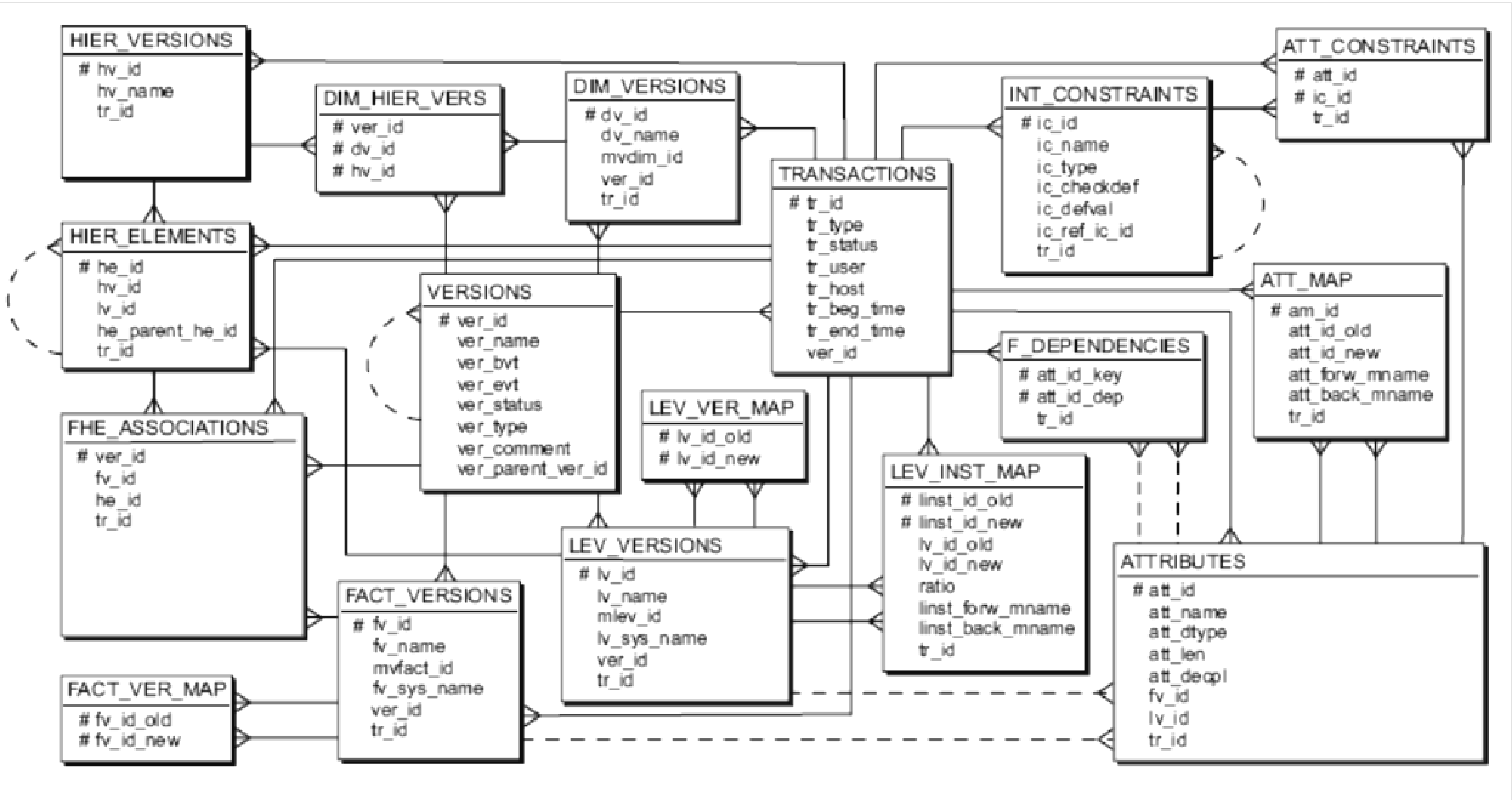
Golfarelli, Lechtenbörger, Rizzi and Vossen @ DKE 2006

- **History** is a sequence of versions H = (v_1, …, v_n) . Each version has
  - Its own schema
  - The augmented schema wrt v_n //needs modification if v_{n+1} comes
  - The timestamp of change
- Why bother?
- Because at query time, we can transform the old schema and data to the last one.
- Then we can pose queries to the old data based on the structure of the new one and a get a uniform result under the last known schema.
- If differences (e.g., because of attribute deletions), we retain the common set of attributes

Wrembel and Bebel @ JoDS’07

- **How to handle changes that come up on the external data sources (EDS) of a data warehouse?**
- Deal with it via a **multiversion** technique!
- Everything has a version (each with a valid time):
  - Dimensions, levels and hierarchies
  - Facts
  - Attributes
  - Integrity constraints
- Mappings are between versioned objects. E.g.,
  - level versions are mapped to dimension versions
  - Fact versions to level versions
  - …
- Both real and alternative (for simulation) versions are supported

Wrembel and Bebel @ JoDS'07



HIER_VERSIONS
# hv_id
hv_name
tr_id

DIM_HIER_VERS
# ver_id
# dv_id
# hv_id

DIM_VERSIONS
# dv_id
dv_name
mvdim_id
ver_id
tr_id

TRANSACTIONS
# tr_id
tr_type
tr_status
tr_user
tr_host
tr_beg_time
tr_end_time
ver_id

INT_CONSTRAINTS
# ic_id
ic_name
ic_type
ic_checkdef
ic_defval
ic_ref_ic_id
tr_id

ATT_CONSTRAINTS
# att_id
# ic_id
tr_id

HIER_ELEMENTS
# he_id
hv_id
lv_id
he_parent_he_id
tr_id

VERSIONS
# ver_id
ver_name
ver_bvt
ver_evt
ver_status
ver_type
ver_comment
ver_parent_ver_id

ATT_MAP
# am_id
att_id_old
att_id_new
att_forw_mname
att_back_mname
tr_id

F_DEPENDENCIES
# att_id_key
# att_id_dep
tr_id

LEV_VER_MAP
# lv_id_old
# lv_id_new

FHE_ASSOCIATIONS
# ver_id
fv_id
he_id
tr_id

LEV_INST_MAP
# linst_id_old
# linst_id_new
lv_id_old
lv_id_new
ratio
linst_forw_mname
linst_back_mname
tr_id

LEV_VERSIONS
# lv_id
lv_name
mlev_id
lv_sys_name
ver_id
tr_id

ATTRIBUTES
# att_id
att_name
att_dtype
att_len
att_decpl
fv_id
lv_id
tr_id

FACT_VERSIONS
# fv_id
fv_name
mvfact_id
fv_sys_name
ver_id
tr_id

FACT_VER_MAP
# fv_id_old
# fv_id_new

Photo stolen from the paper

Wrembel and Bebel @ JoDS'07

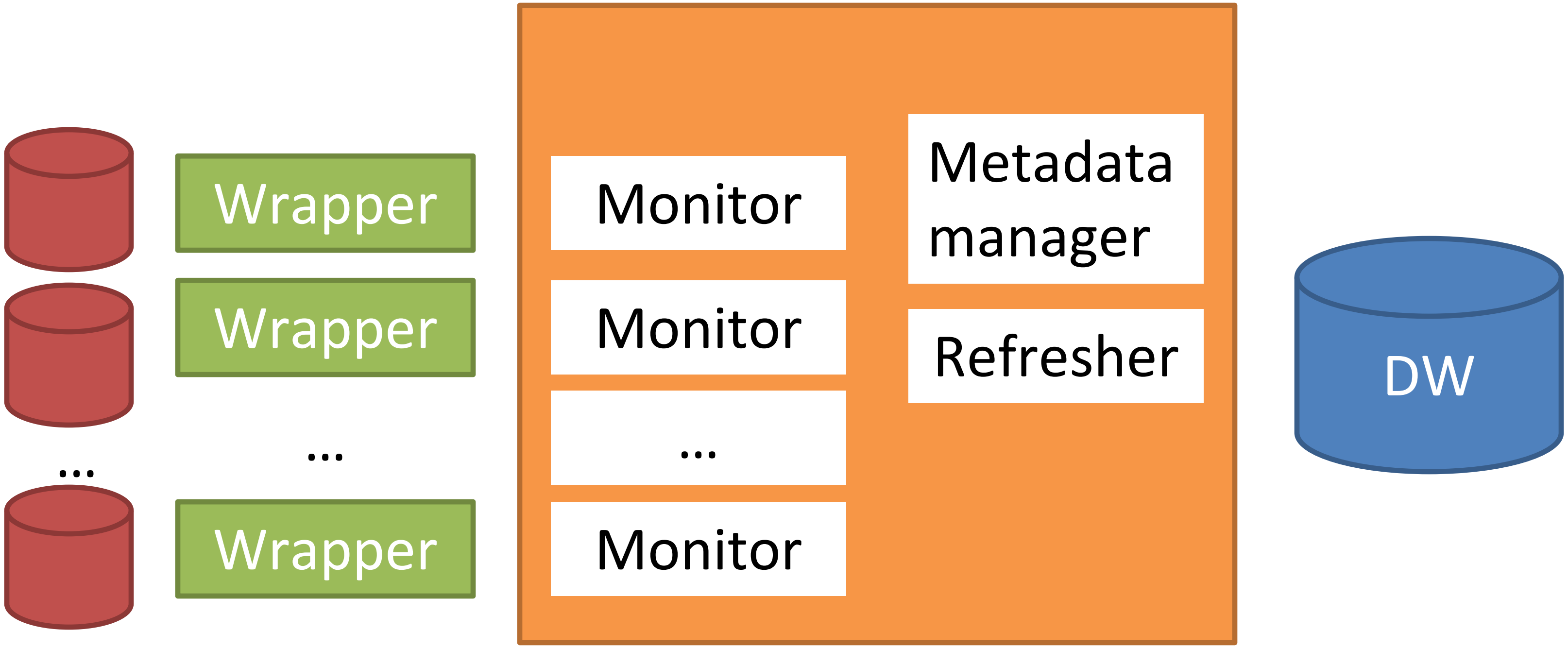
- Schema Change Operations
  - the addition / deletion of attribute @ dimension level table
  - the creation of a new fact table + the association of a fact with a dimension
  - the renaming of a table
  - snowflake changes:
    - the creation of a new dimension level table with a given structure
    - the inclusion of a parent dimension level table into its child dimension level table,
    - the creation of a parent dimension level table based on its child level table.

- Instance change operations
  - Add/del level instance
  - Change parent of a level
  - Merge many instances of a level into a single one / split(inverse)

Wrembel and Bebel @ JoDS’07

- Querying multiple versions
- Split original query to a set of single version queries
- For each single version query, do a best-effort approach:
  - if attributes are missing, omit them;
  - use metadata for renames
  - ignore v. if a grouping is impossible
  - …
- If possible, the collected results are integrated under the intersection of attributes common to all versions (if this is the case of the query);
- Else they are presented as a set of results, each with its own metadata

Wrembel and Bebel @ JoDS'07



monitored
External
Data
Sources

Roadmap

- Evolution of views
- Data warehouse Evolution
- **A case study** (if time)
- Impact assessment in ecosystems
- Empirical studies concerning database evolution
- Open Issues and discussions

George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, Yannis Vassiliou. Metrics for the Prediction of Evolution Impact in ETL Ecosystems: A Case Study. Journal on Data Semantics, August 2012, Volume 1, Issue 2, pp 75-97

A CASE STUDY OF DW EVOLUTION

Context of the Study

- We have studied a data warehouse scenario from a Greek public sector’s data warehouse maintaining information for farming and agricultural statistics.
- The warehouse maintains statistical information collected from surveys, held once per year via questionnaires.
- Our study is based on the evolution of the source tables and their accompanying ETL flows, which has happened in the *context of maintenance due to the change of requirements at the real world*.
- Practically this is due to the update of the questionnaires from year to year

Internals of the monitored scenario

- The environment involves a set of 7 ETL workflows:
  - – 7 source tables, (S1 to S7)
  - – 3 lookup tables(L1 to L3),
  - – 7 target tables, (T1 to T7), stored in the data warehouse.
  - – 7 temporary tables (each target table has a temporary replica) for keeping data in the data staging area,
  - – 58 ETL activities in total for all the 7 workflows.

PL/SQL to graph transformation

- All ETL scenarios were source coded as PL\SQL stored procedures in the data warehouse.
  - We extracted embedded SQL code (e.g., cursor definitions, DML statements, SQL queries) from activity stored procedures
  - Each activity was represented in our graph model as a view defined over the previous activities
  - Table definitions were represented as relation graphs.

Method of assessment

- We have represented the ETL workflows in our graph model
- We have recorded evolution events on the nodes of the source, lookup and temporary tables.
- We have applied each event sequentially on the graph and monitored the impact of the change towards the rest of the graph by recording the times that a node has been affected by each change

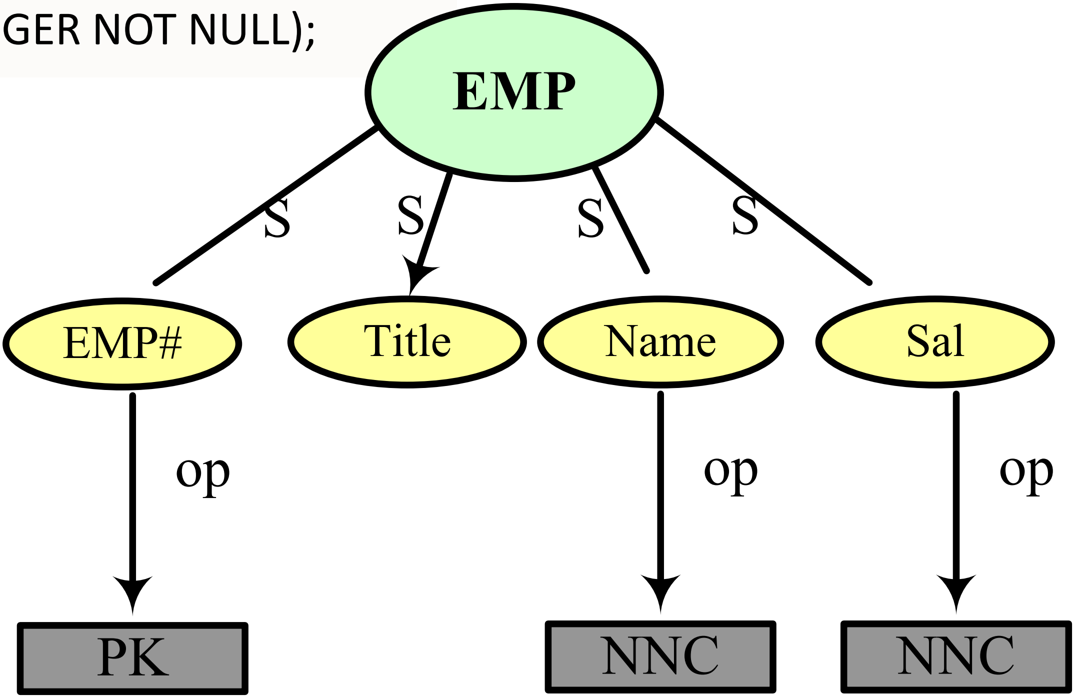
Graph modeling of a data-intensive ecosystem

- The **entire data-intensive ecosystem**, comprising **databases** and their internals, as well as **applications** and their data-intensive parts, is modeled via a graph that we call **Architecture Graph**

- Why Graph modeling?
  - Completeness: graphs can model everything
  - Uniformity: we would like to module everything **uniform** manner
  - Detail and Grand-View: we would like to capture **parts** and **dependencies** at the very **finest level**; at same time, we would like to have the ability to **zoom-out** at higher levels of abstraction
  - Exploit graph management techniques and toolkits

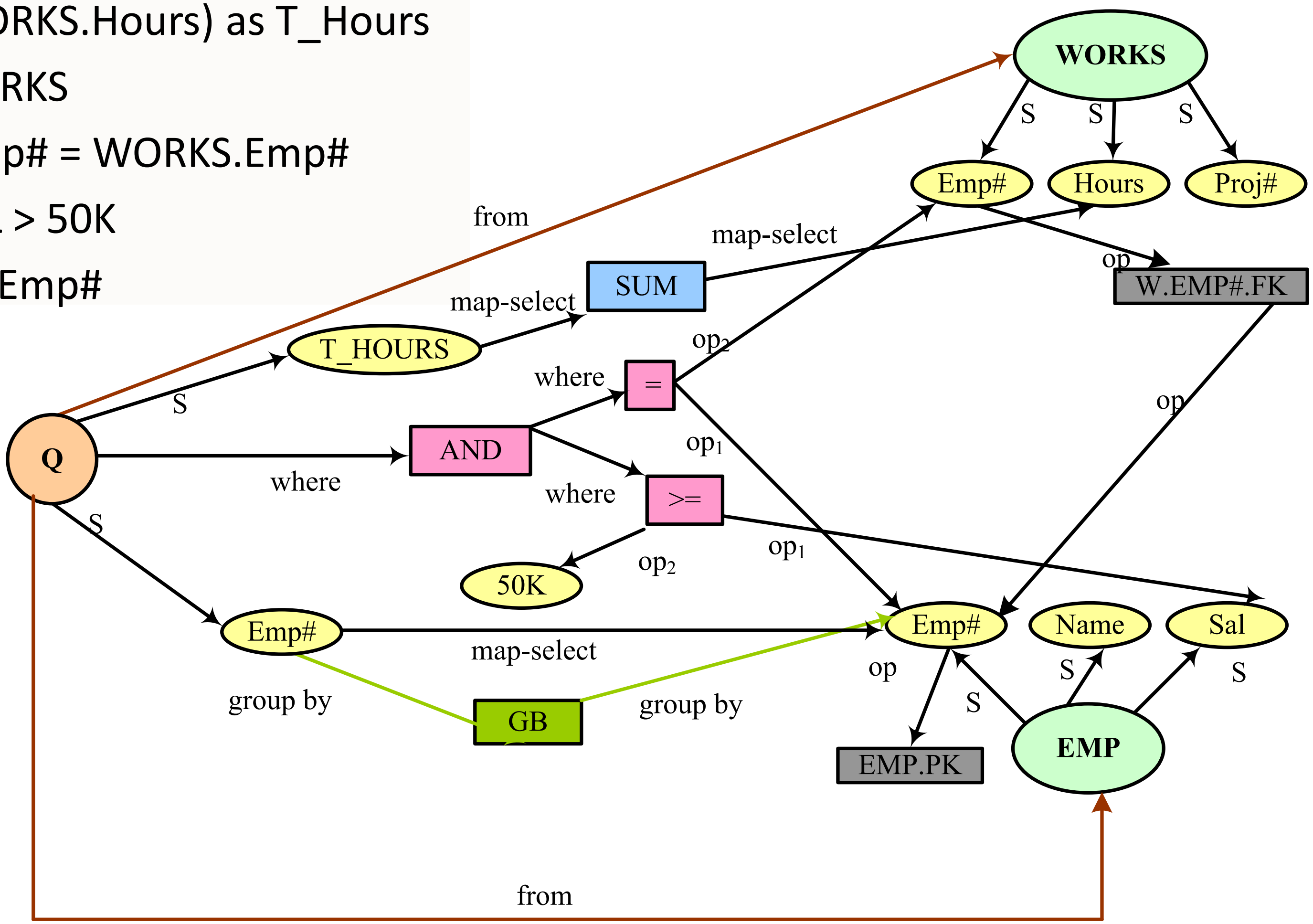
Relations – Attributes - Constraints

```
CREATE TABLE EMP (EMP#     INTEGER  PRIMARY KEY,
                  NAME     VARCHAR(25) NOT NULL,
                  TITLE    VARCHAR(10),
                  SAL      INTEGER NOT NULL);
```

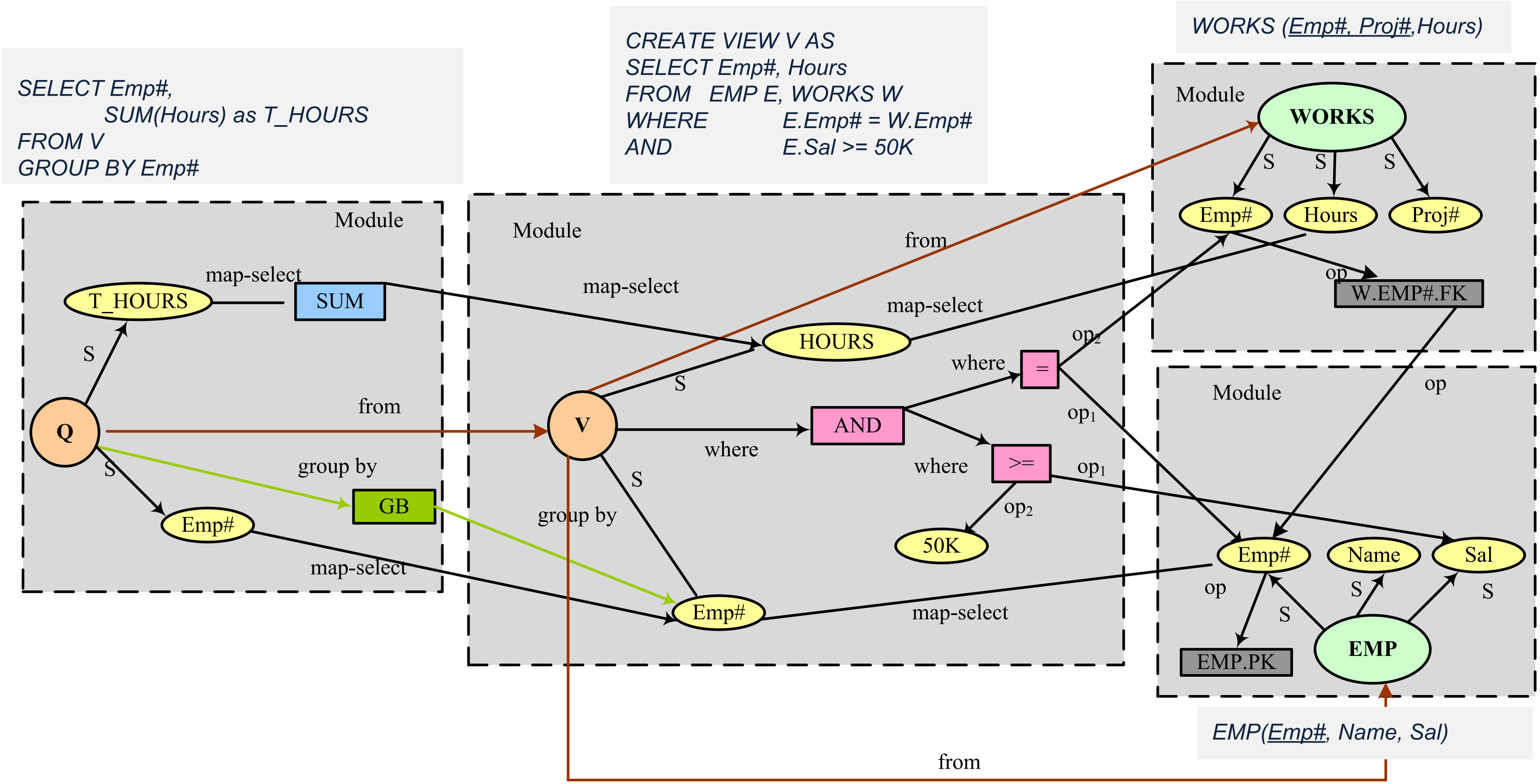


Queries & Views

```
Q: SELECT  EMP.Emp# as Emp#,
           Sum(WORKS.Hours) as T_Hours
   FROM EMP, WORKS
   WHERE EMP.Emp# = WORKS.Emp#
     AND  EMP.SAL > 50K
   GROUP BY EMP.Emp#
```



Modules: relations, queries, views



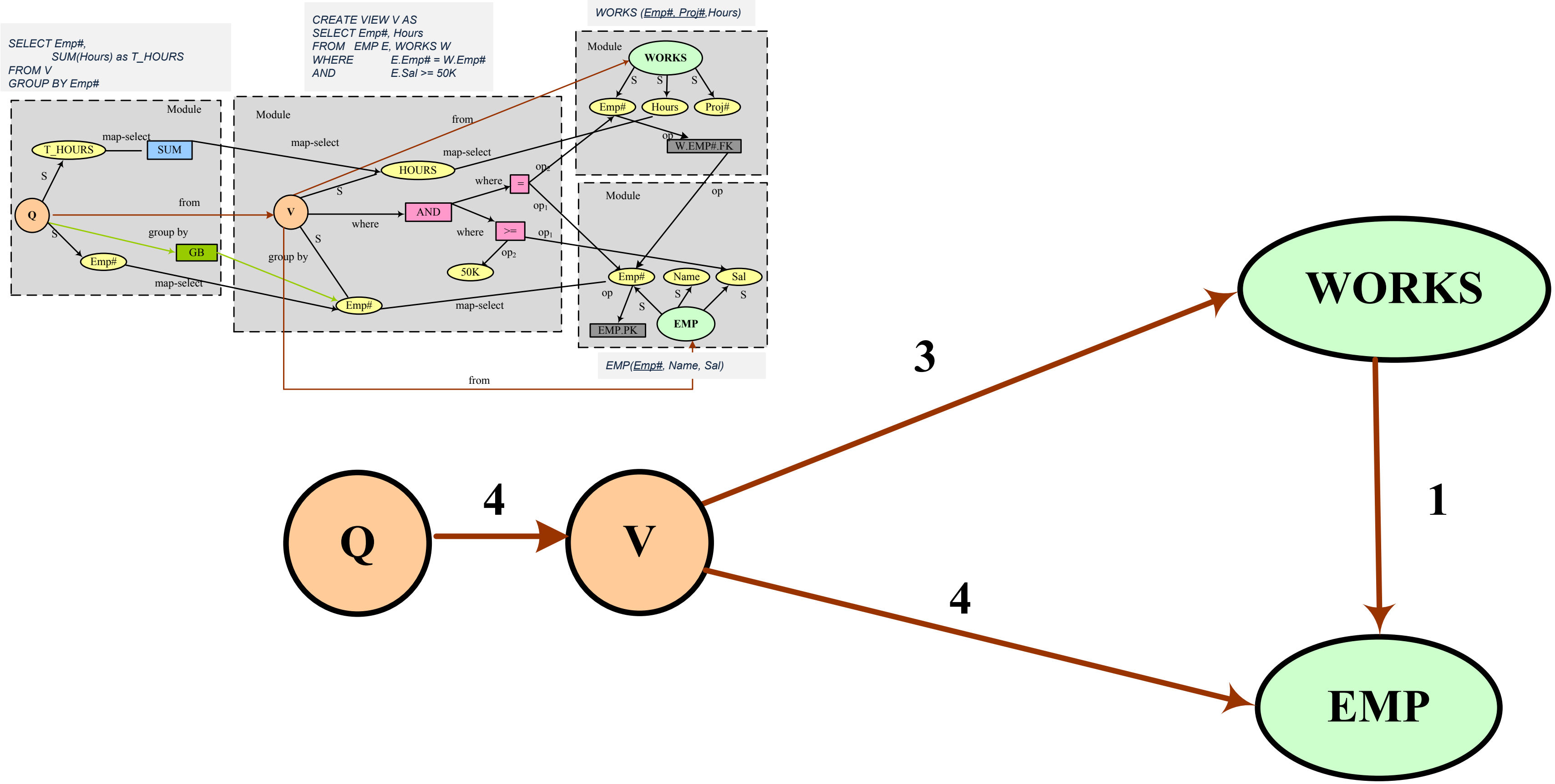
```
SELECT Emp#,
        SUM(Hours) as T_HOURS
FROM V
GROUP BY Emp#
```

```
CREATE VIEW V AS
SELECT Emp#, Hours
FROM   EMP E, WORKS W
WHERE         E.Emp# = W.Emp#
AND           E.Sal >= 50K
```

WORKS (Emp#, Proj#,Hours)

EMP(Emp#, Name, Sal)

Zooming out to top-level nodes (modules)



SELECT Emp#,
SUM(Hours) as T_HOURS
FROM V
GROUP BY Emp#

CREATE VIEW V AS
SELECT Emp#, Hours
FROM EMP E, WORKS W
WHERE E.Emp# = W.Emp#
AND E.Sal >= 50K

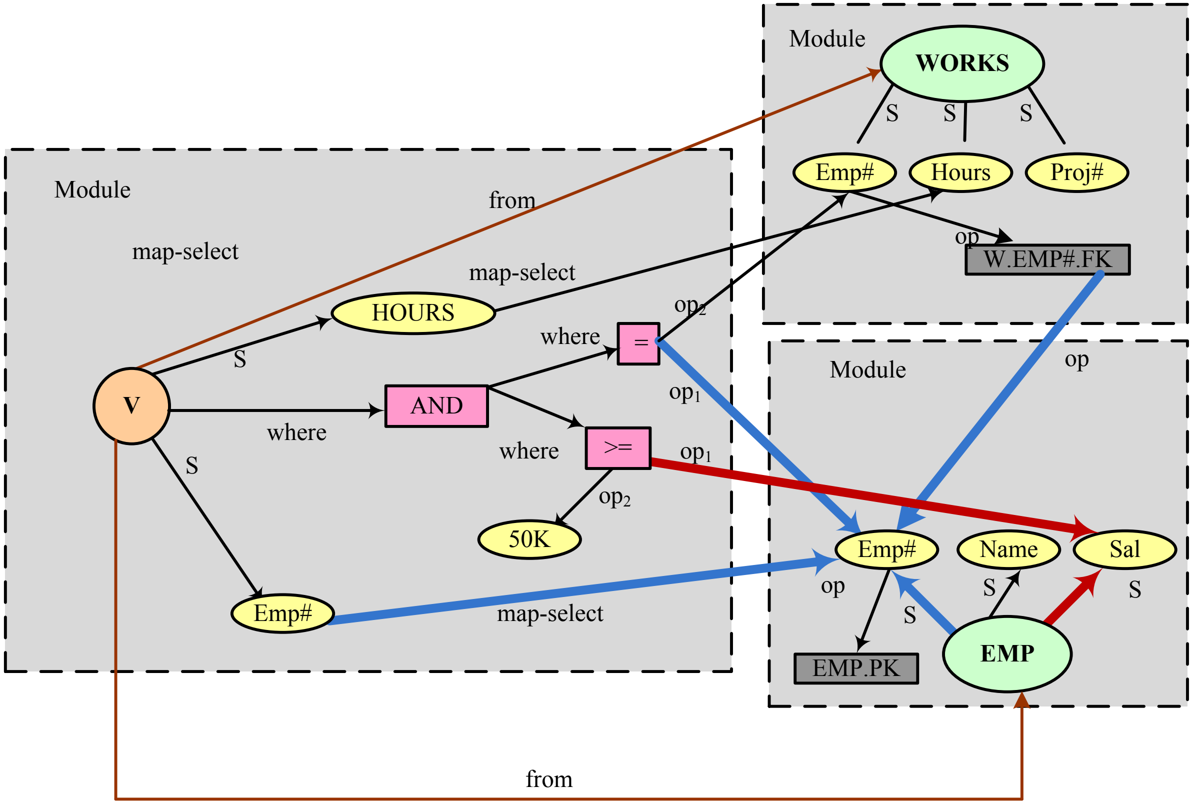
WORKS (Emp#, Proj#,Hours)

EMP(Emp#, Name, Sal)

Metrics: Node Degree

EMP.Emp# is the most important attribute of EMP.SAL, if one considers how many nodes depend on it.

Simple metrics:
in-degree, out-degree, degree

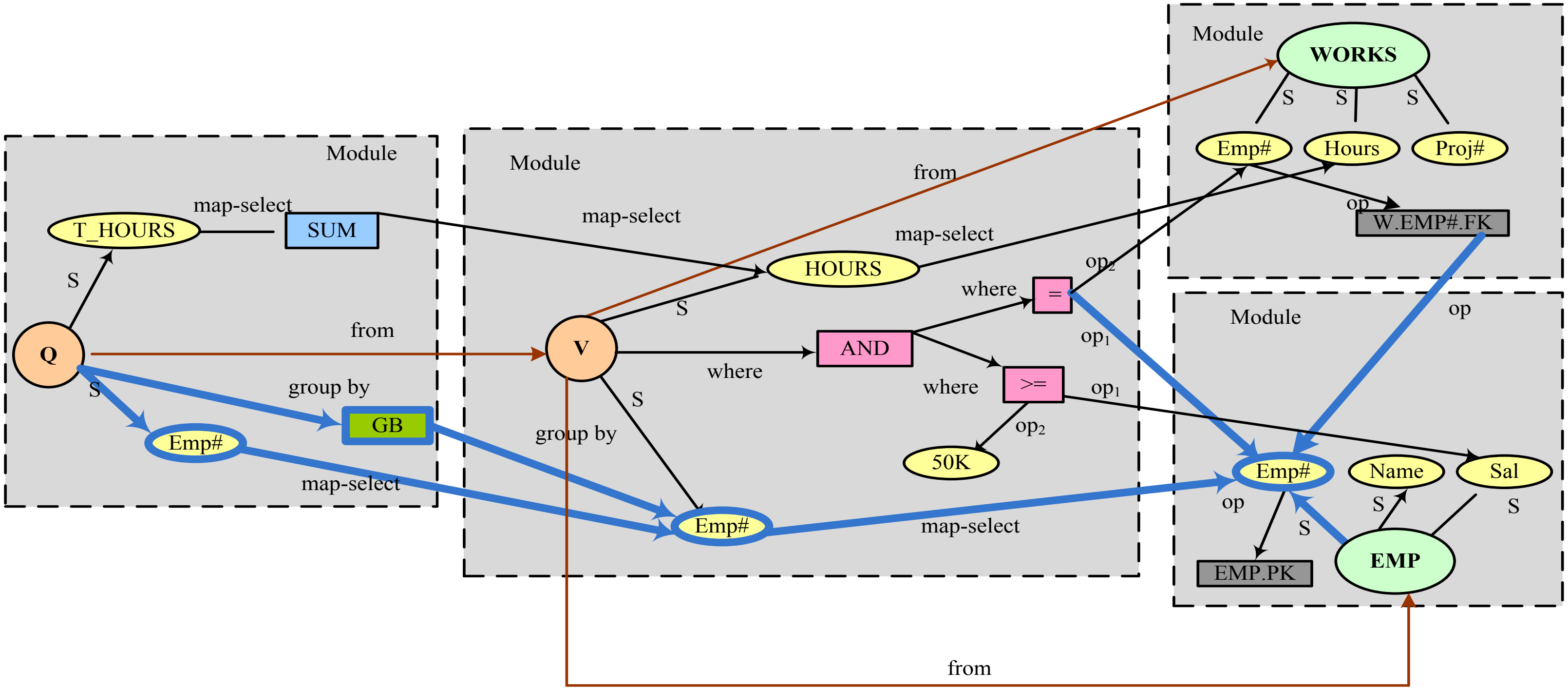


Edge direction:
from dependant
to depended upon

Metrics: Transitive Node Degree

Observe that there is both a view and a query with nodes dependent upon attribute *EMP.Emp#.*

Transitive Metrics:
in-degree, out-degree, degree

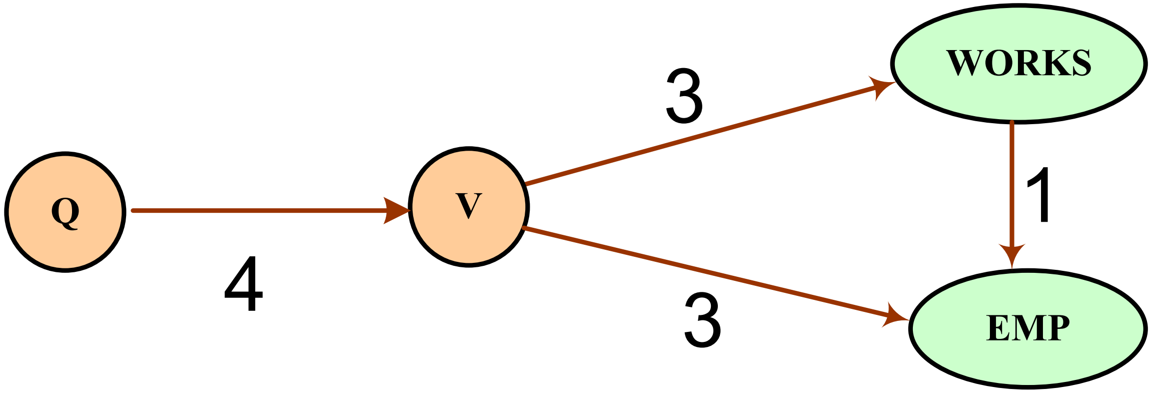


Strength: Zooming out to modules

A zoomed out graph highlights the dependence between modules (relations, queries, views), incorporating the detailed dependencies as the weight of the edges

Again, for modules, we can have both:

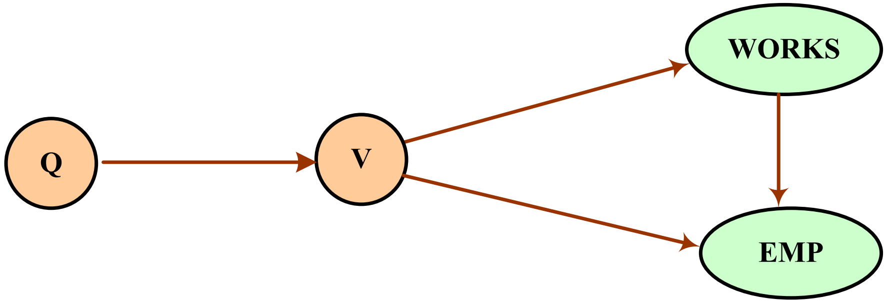
- Simple strength
- Transitive strength



Metrics: Node Entropy

The probability a node *v* being affected by an evolution event on node y_i :

$$P(v|y_k) = \frac{paths(v, y_k)}{\sum_{y_i \in V} paths(v, y_i)}, \text{ for all nodes } y_i \in V.$$



Examples
P(Q|*V*) = 1/4,
P(Q|*EMP*) = 2/4,
P(*V*|*WORKS*) = 1/3

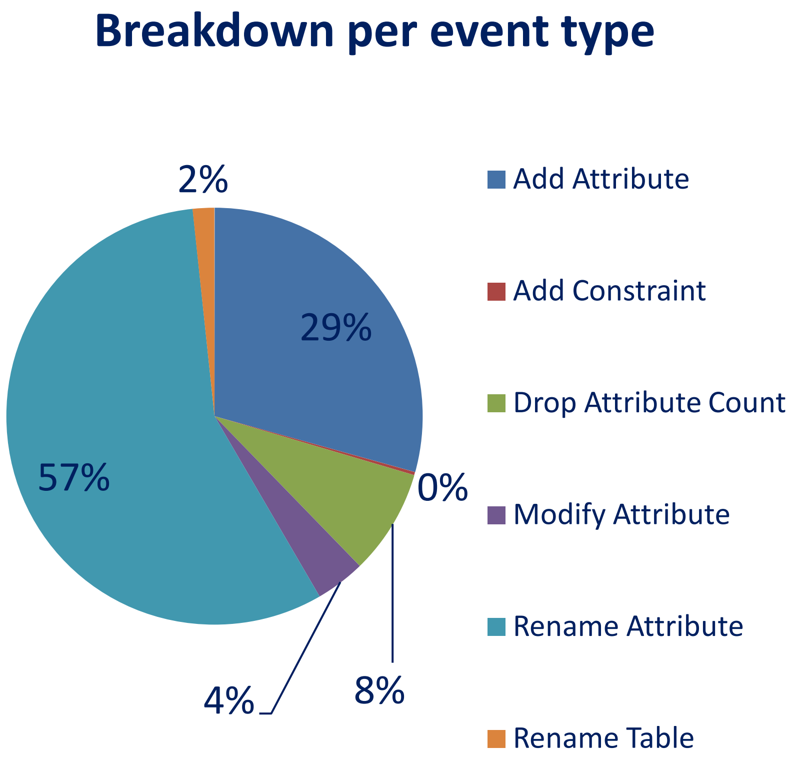
Entropy of a node *v* : How sensitive the node *v* is by an arbitrary event on the graph.

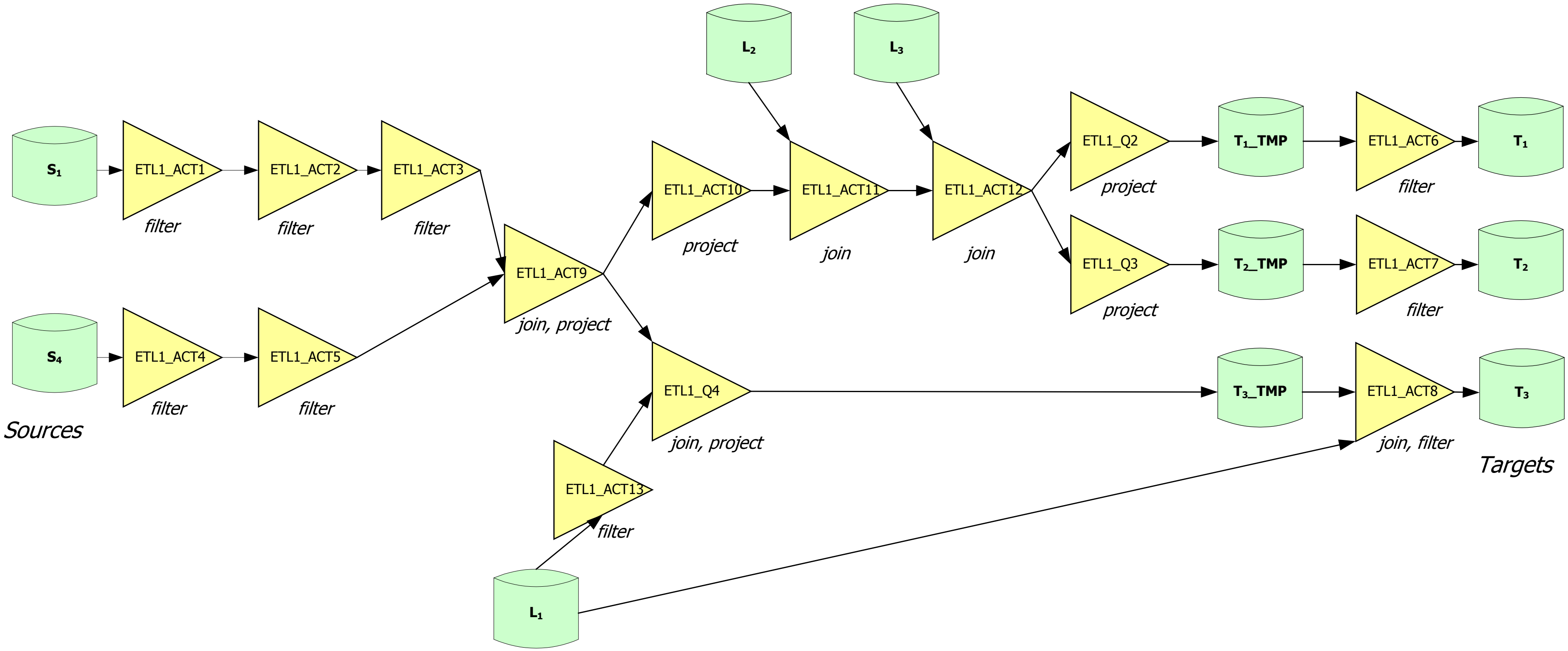
$$H(v) = -\sum_{y_i \in V} P(v \mid y_i) \log_2 P(v \mid y_i), \text{ for all nodes } y_i \in V.$$

Macroscopic view

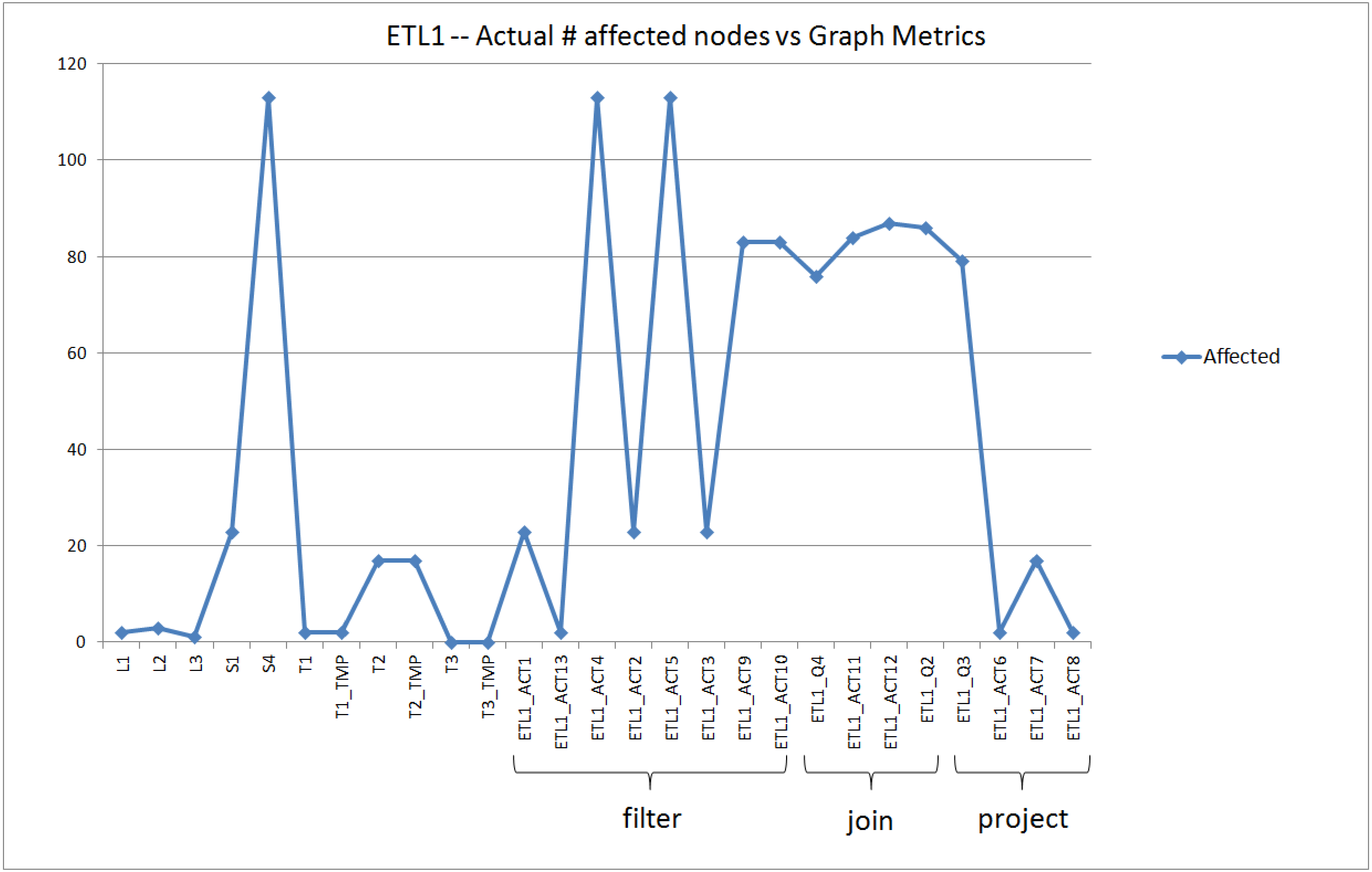
ATTN: change of requirements at the real world determines pct breakdown!!

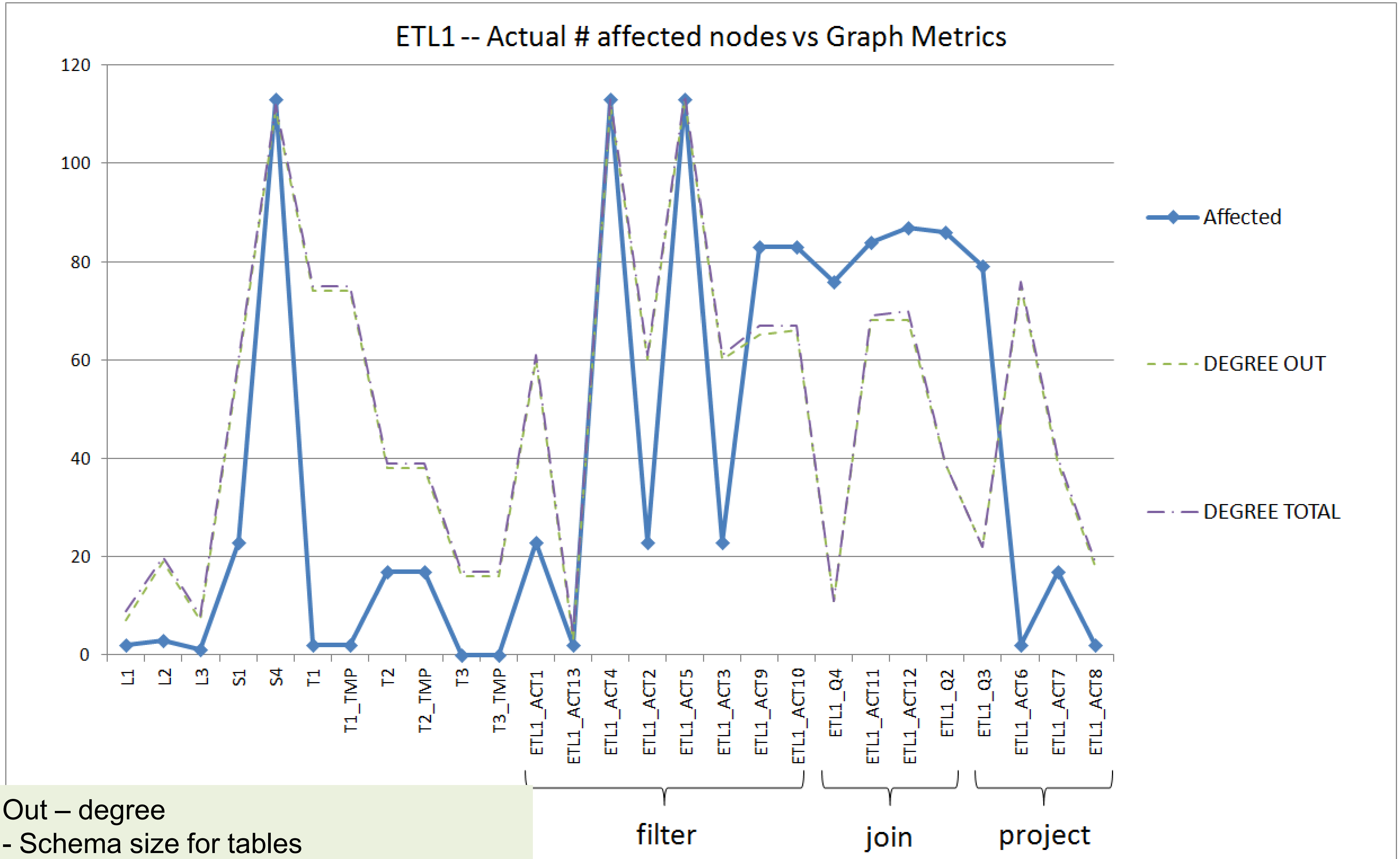
	# tables affected	Occurrences	pct
Add Attribute	8	122	**29**%
Add Constraint	1	1	0%
Drop Attribute Count	5	34	8%
Modify Attribute	9	16	4%
Rename Attribute	5	236	**57%**
Rename Table	7	7	2%
		416	


Breakdown per event type

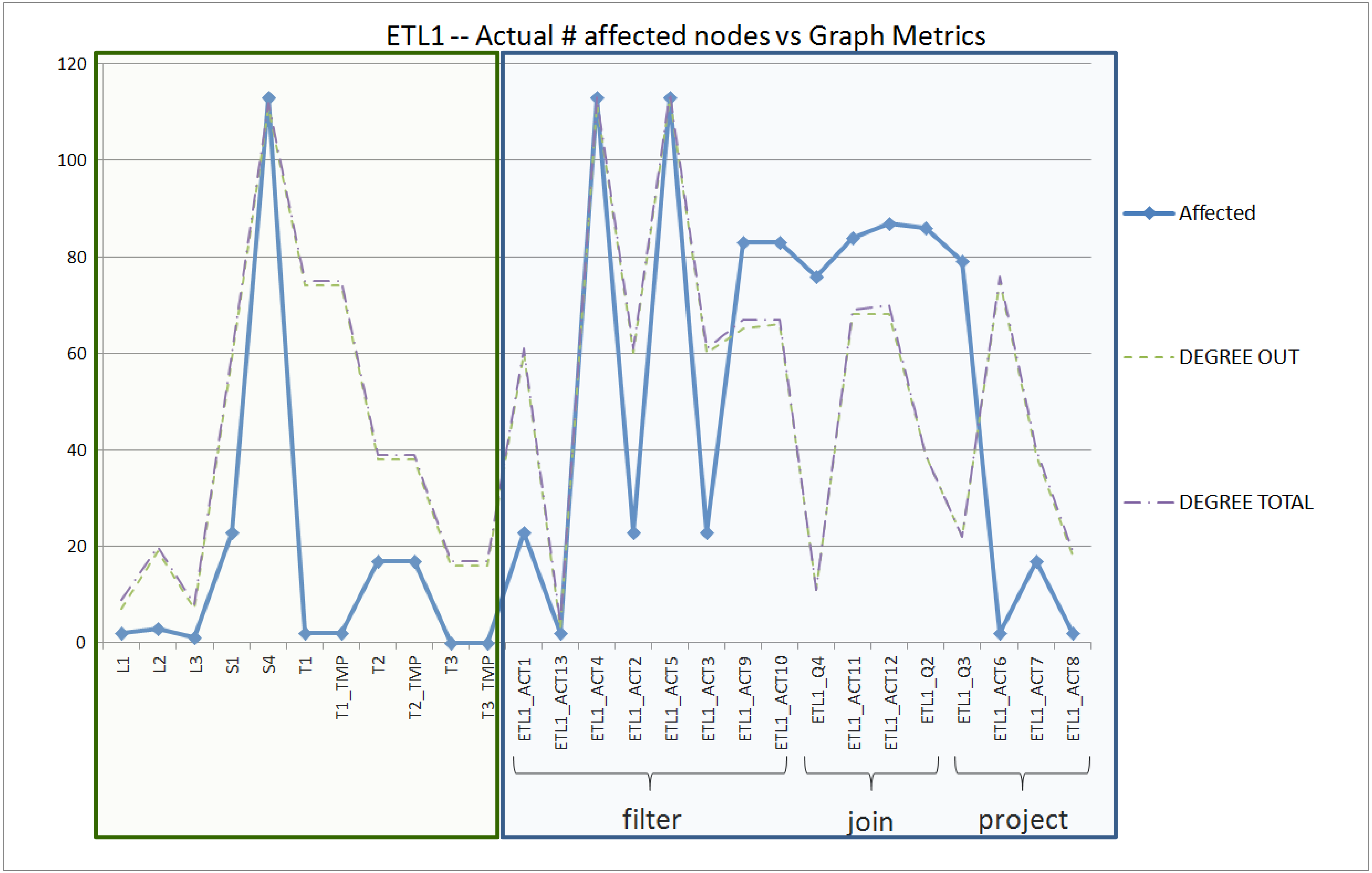


Workflow of the first ETL scenario, ETL1


ETL1 -- Actual # affected nodes vs Graph Metrics
120
100
80
60
40
20
0
L1
L2
L3
S1
S4
T1
T1_TMP
T2
T2_TMP
T3
T3_TMP
ETL1_ACT1
ETL1_ACT13
ETL1_ACT4
ETL1_ACT2
ETL1_ACT5
ETL1_ACT3
ETL1_ACT9
ETL1_ACT10
ETL1_Q4
ETL1_ACT11
ETL1_ACT12
ETL1_Q2
ETL1_Q3
ETL1_ACT6
ETL1_ACT7
ETL1_ACT8
filter
join
project
Affected



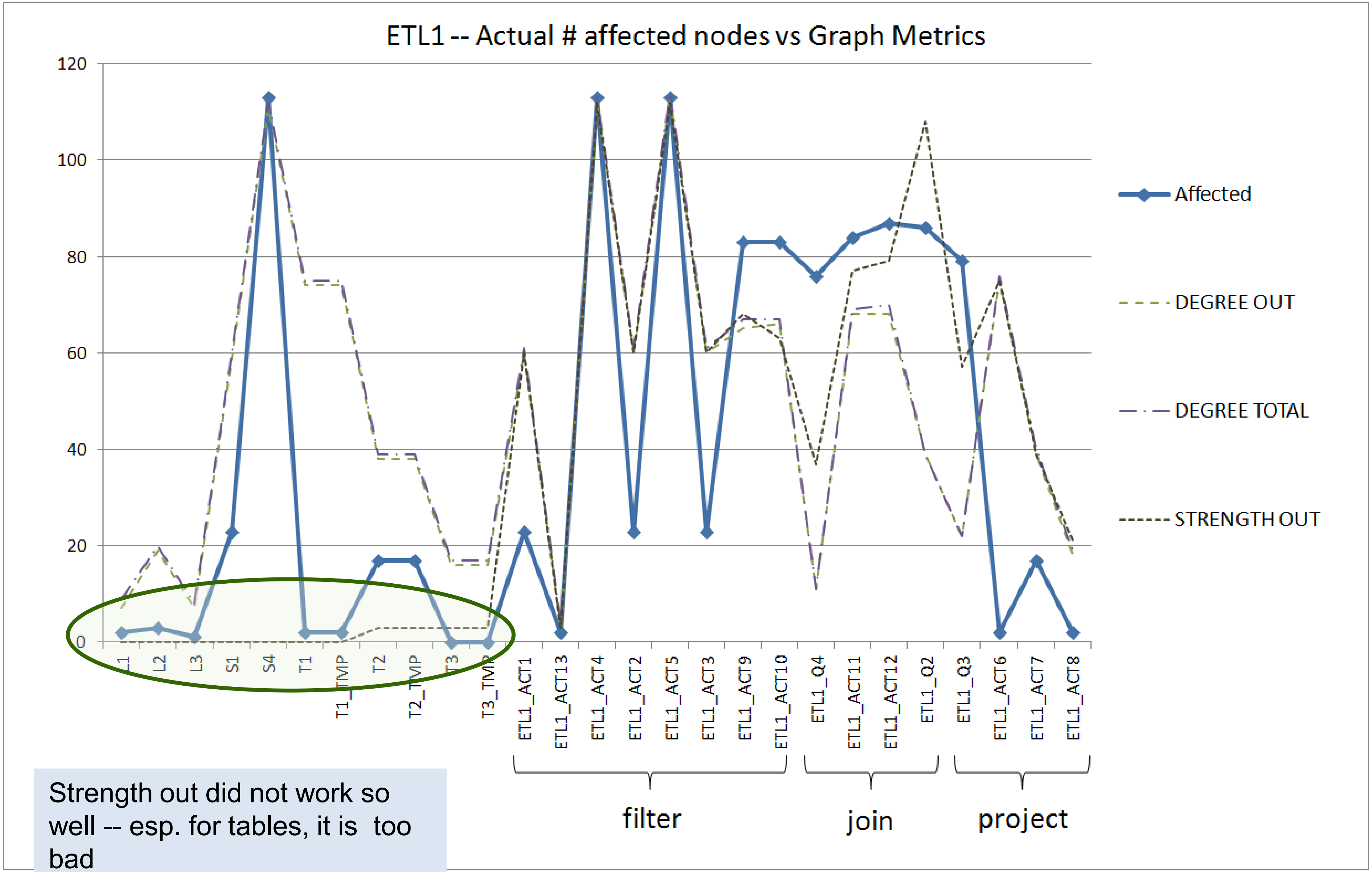
Out – degree
- Schema size for tables
- Output schema size for activities



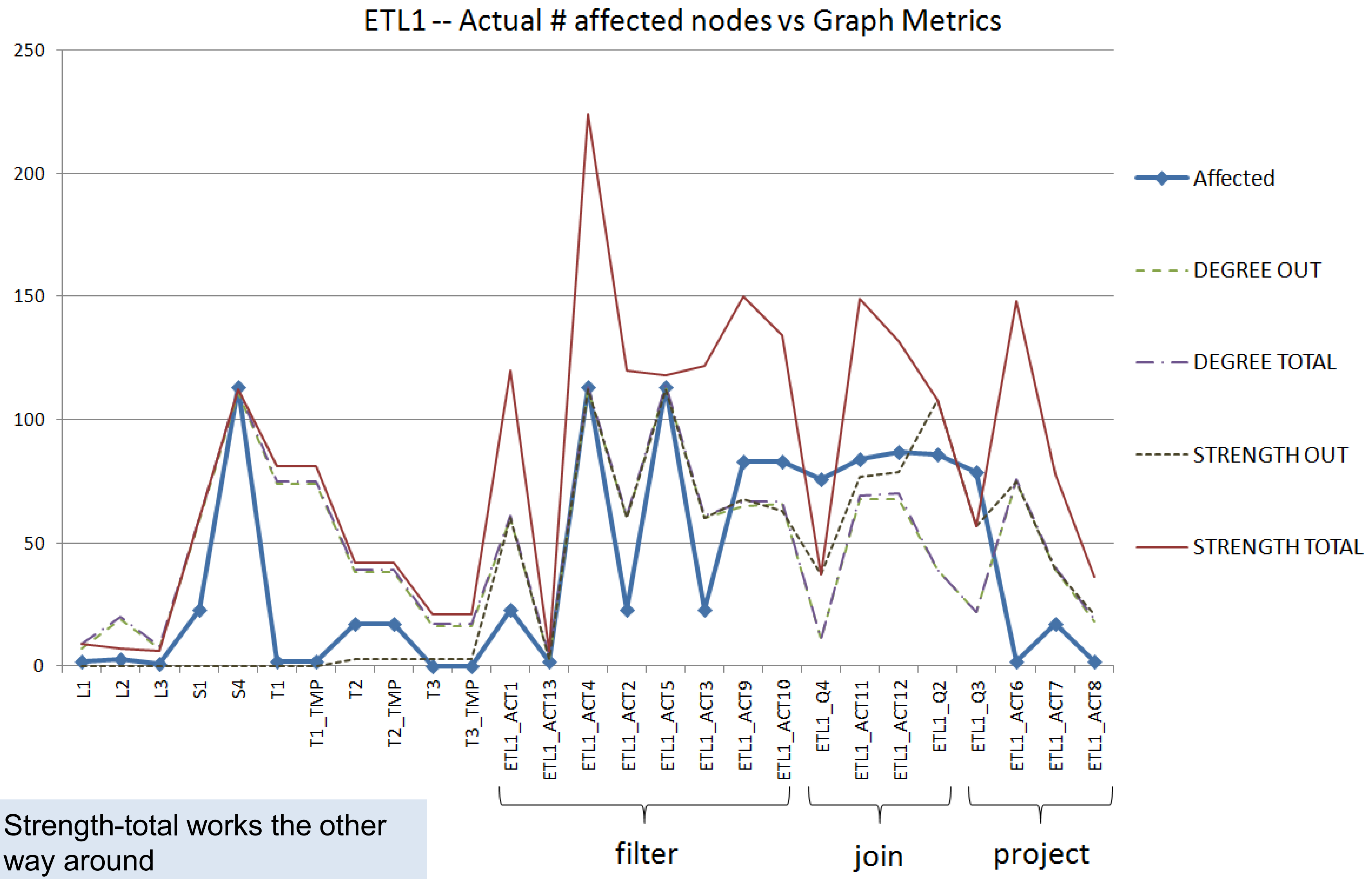
Pretty good job
for tables

Decent job for filters
and joins

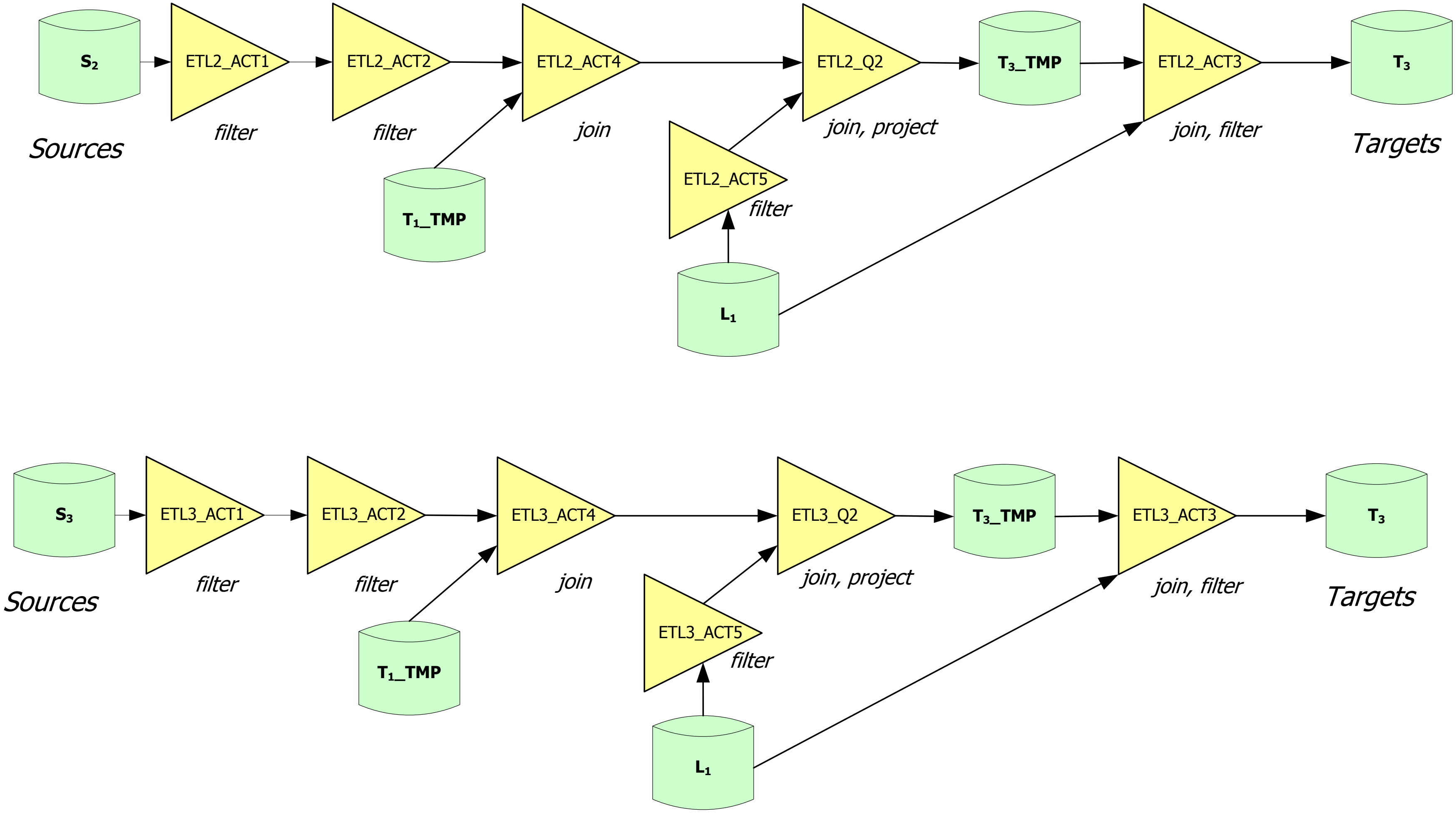
Not so good for
projection activities



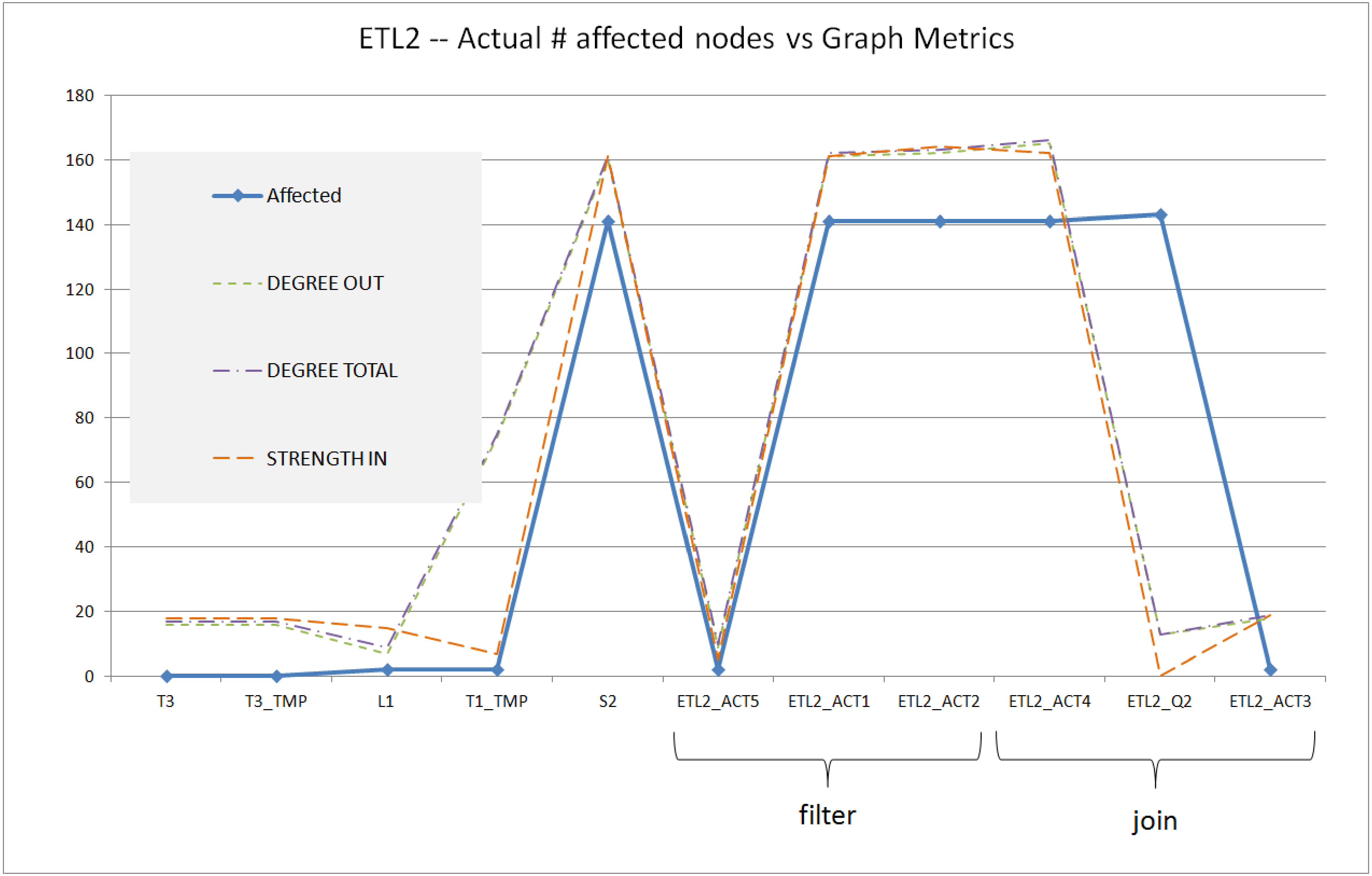
Strength out did not work so well -- esp. for tables, it is too bad

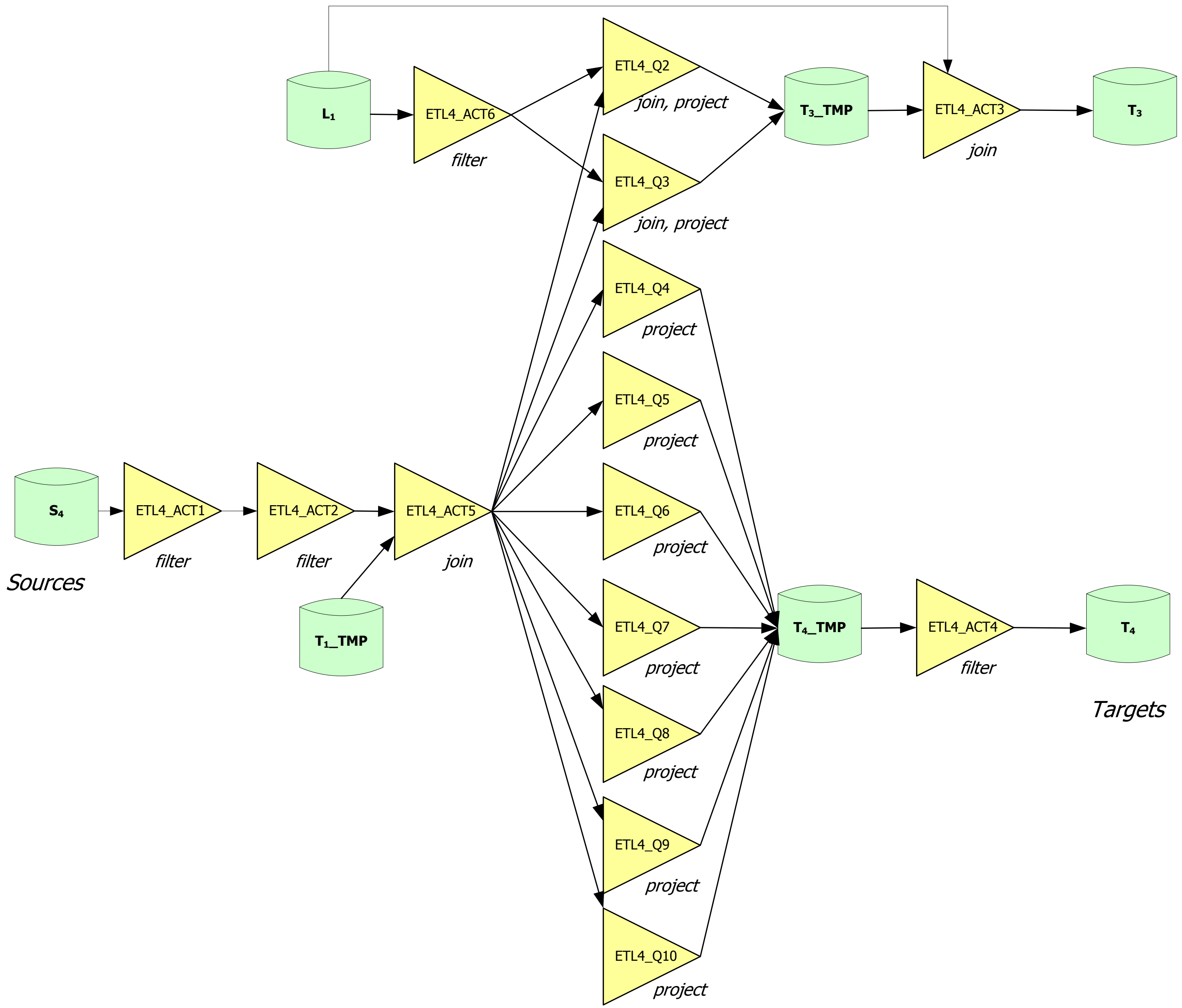


Strength-total works the other way around

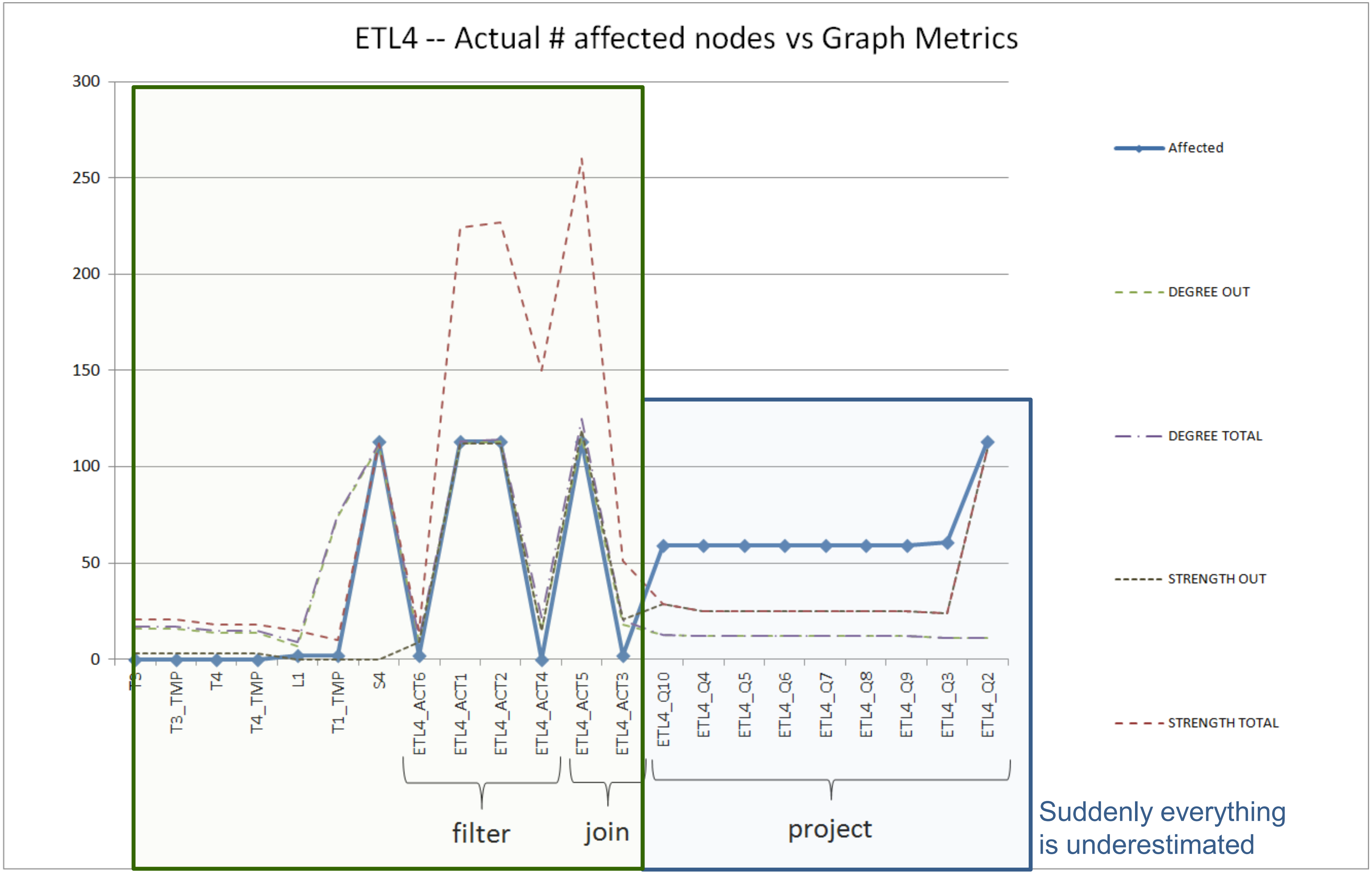


Workflows of the second & third ETL scenarios, ETL2 – ETL3


ETL2 -- Actual # affected nodes vs Graph Metrics
Affected
DEGREE OUT
DEGREE TOTAL
STRENGTH IN
180
160
140
120
100
80
60
40
20
0
T3
T3_TMP
L1
T1_TMP
S2
ETL2_ACT5
ETL2_ACT1
ETL2_ACT2
ETL2_ACT4
ETL2_Q2
ETL2_ACT3
filter
join

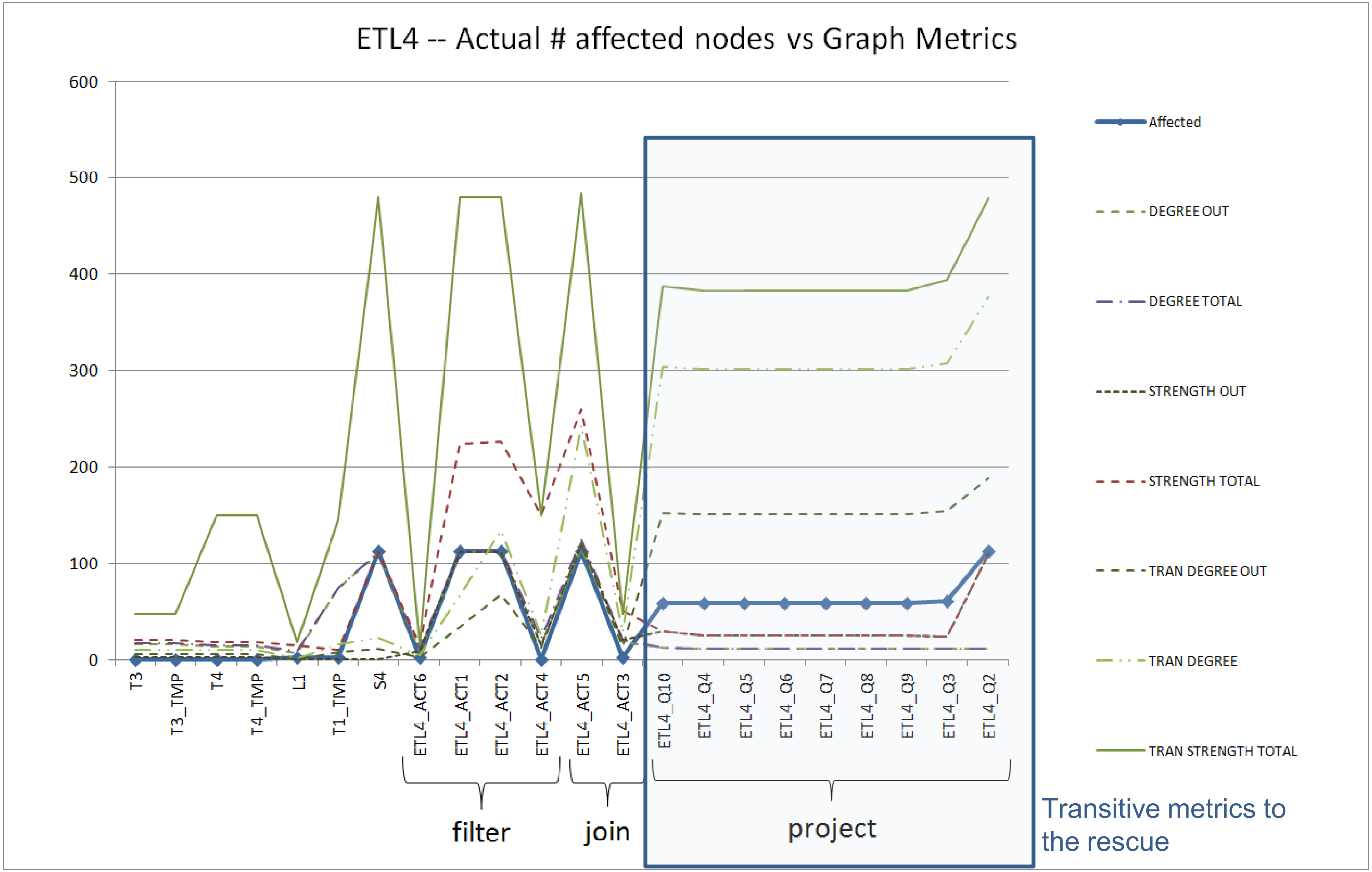

L$_1$
ETL4_ACT6
filter
ETL4_Q2
join, project
ETL4_Q3
join, project
T$_3$_TMP
ETL4_ACT3
join
T$_3$
ETL4_Q4
project
ETL4_Q5
project
S$_4$
ETL4_ACT1
filter
ETL4_ACT2
filter
ETL4_ACT5
join
Sources
T$_1$_TMP
ETL4_Q6
project
ETL4_Q7
project
T$_4$_TMP
ETL4_ACT4
filter
T$_4$
Targets
ETL4_Q8
project
ETL4_Q9
project
ETL4_Q10
project

ETL 4

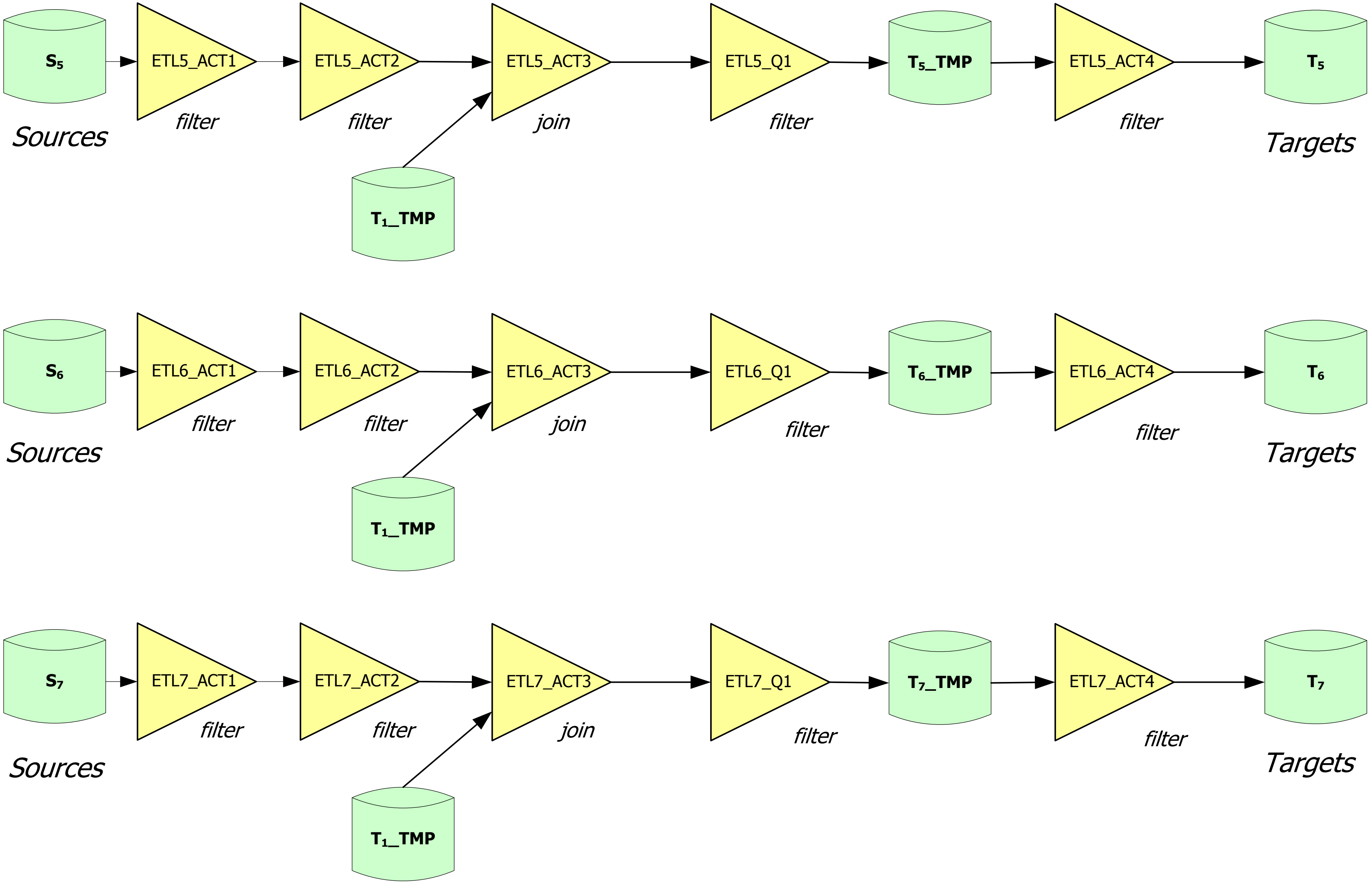


Suddenly everything is underestimated

Pretty good job in the left part

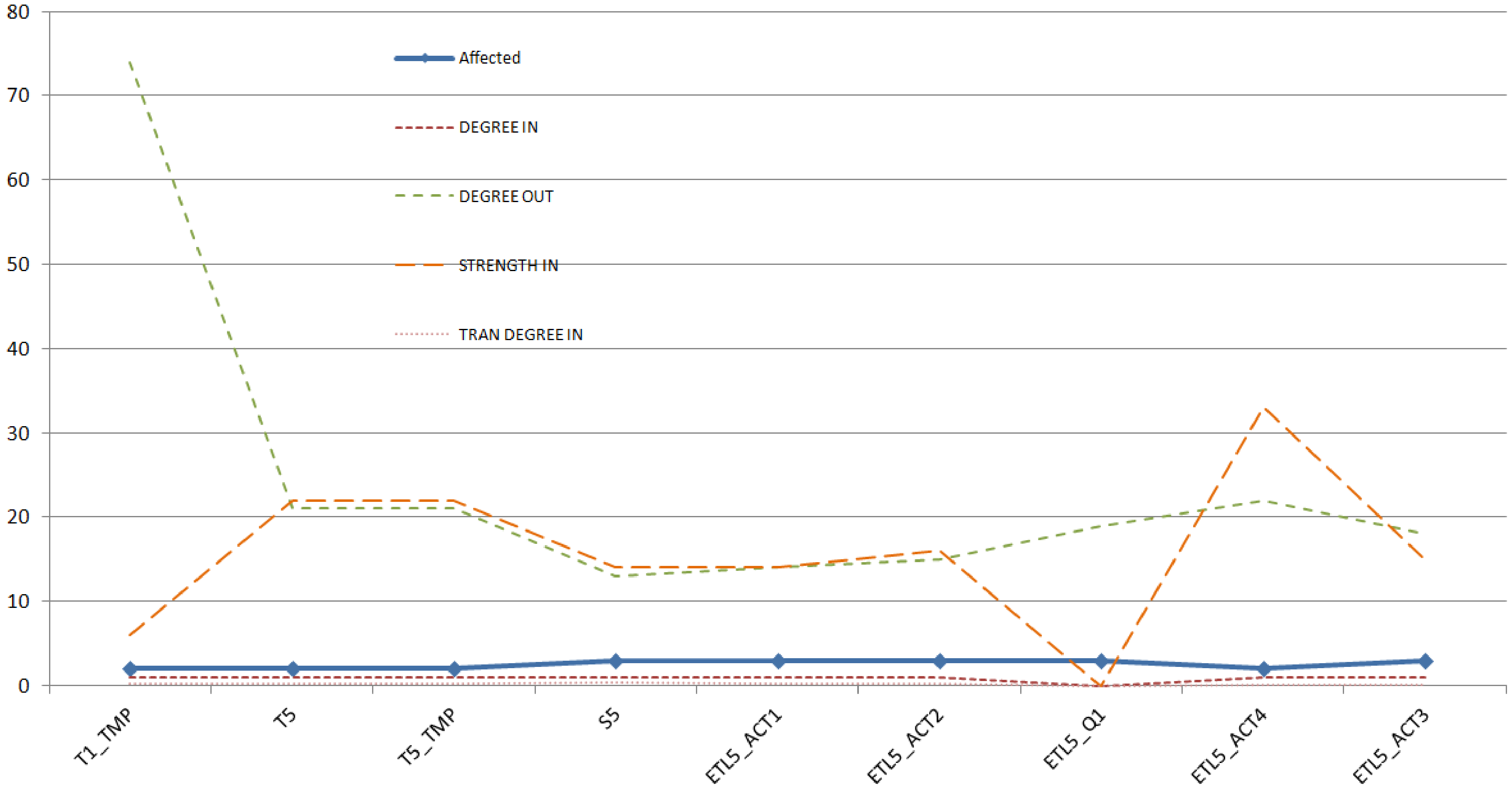


Transitive metrics to the rescue


S5
ETL5_ACT1
filter
ETL5_ACT2
filter
ETL5_ACT3
join
ETL5_Q1
filter
T5_TMP
ETL5_ACT4
filter
T5
Sources
Targets
T1_TMP
S6
ETL6_ACT1
filter
ETL6_ACT2
filter
ETL6_ACT3
join
ETL6_Q1
filter
T6_TMP
ETL6_ACT4
filter
T6
Sources
Targets
T1_TMP
S7
ETL7_ACT1
filter
ETL7_ACT2
filter
ETL7_ACT3
join
ETL7_Q1
filter
T7_TMP
ETL7_ACT4
filter
T7
Sources
Targets
T1_TMP

ETL 5,6,7

ETL5 -- Actual # affected nodes vs Graph Metrics



Schema size and module complexity as predictors for the vulnerability of a system

- The **size of the schemas** involved in an ETL design significantly affects the design vulnerability to evolution events.
  - For example, source or intermediate tables with many attributes are more vulnerable to changes at the attribute level.
  - The **out-degree** captures the projected attributes by an activity, whereas the **out-strength** captures the total number of dependencies between an activity and its sources.
- The **internal structure of an ETL activity** plays a significant role for the impact of evolution events on it.
  - Activities with high out-degree and out-strengths tend to be more vulnerable to evolution
  - Activities performing **attribute reduction** (e.g., through either a group-by or a projection operation) are in general, less vulnerable to evolution events.
  - Transitive degree and entropy metrics capture the dependencies of a module with its various non-adjacent sources. Useful for activities which act as "hubs" of various different paths from sources in complex workflows.
- The **module-level design** of an ETL flow also affects the overall evolution impact on the flow.
  - For example, it might be worthy to **place schema reduction activities early in an ETL flow** to restrain the flooding of evolution events.

Summary & Guidelines

ETL Construct	Most suitable Metric	Heuristic
Source Tables	*out-degree*	*Retain small schema size*
Intermediate & Target Tables	*out-degree*	*Retain small schema size in intermediate tables*
Filtering activities	*out-degree*, *out-strength*	*Retain small number of conditions*
Join Activities	*out-degree*, *out-strength*, *trans. out-degree, trans. out-strength, entropy*	*Move to early stages of the workflow*
Project Activities	*out-degree*, *out-strength*, *trans. out-degree, trans. out-strength, entropy*	*Move attribute reduction activities to early stages of the workflow and attribute increase activities to later stages*

Roadmap

- Evolution of views
- Data warehouse Evolution
- A case study (if time)
- **Impact assessment in ecosystems**
- Empirical studies concerning database evolution
- Open Issues and discussions

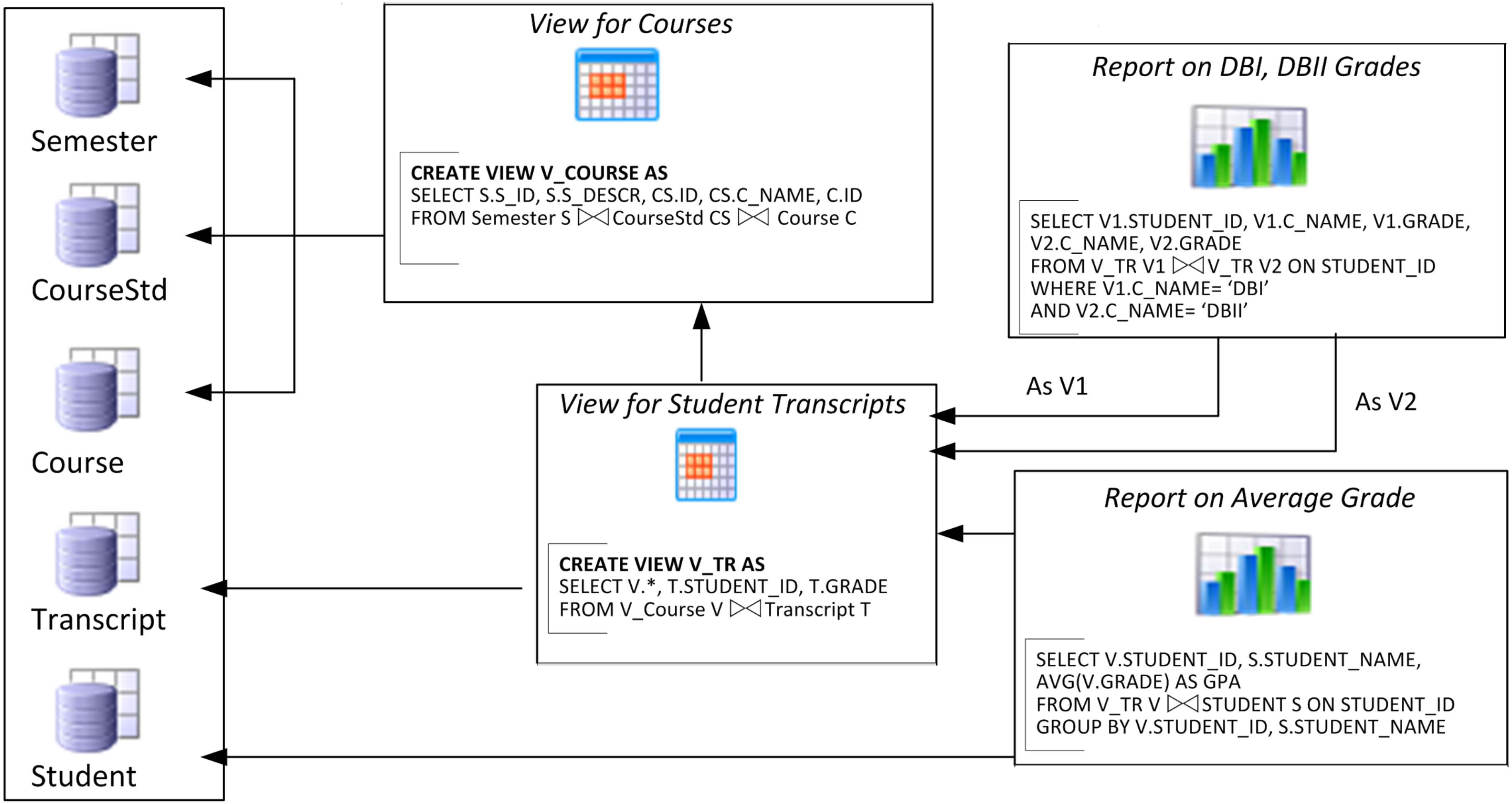
… and data intensive ecosystems…

IMPACT ASSESSMENT

Data intensive ecosystems

- Ecosystems of **applications**, built on top of one or more **databases** and strongly dependent upon them
- Like all software systems, they too change…

Evolving data-intensive ecosystem



Semester

CourseStd

Course

Transcript

Student

View for Courses

```
CREATE VIEW V_COURSE AS
SELECT S.S_ID, S.S_DESCR, CS.ID, CS.C_NAME, C.ID
FROM Semester S ⋈CourseStd CS ⋈ Course C
```

View for Student Transcripts

```
CREATE VIEW V_TR AS
SELECT V.*, T.STUDENT_ID, T.GRADE
FROM V_Course V ⋈Transcript T
```

Report on DBI, DBII Grades

```
SELECT V1.STUDENT_ID, V1.C_NAME, V1.GRADE,
V2.C_NAME, V2.GRADE
FROM V_TR V1 ⋈V_TR V2 ON STUDENT_ID
WHERE V1.C_NAME= 'DBI'
AND V2.C_NAME= 'DBII'
```

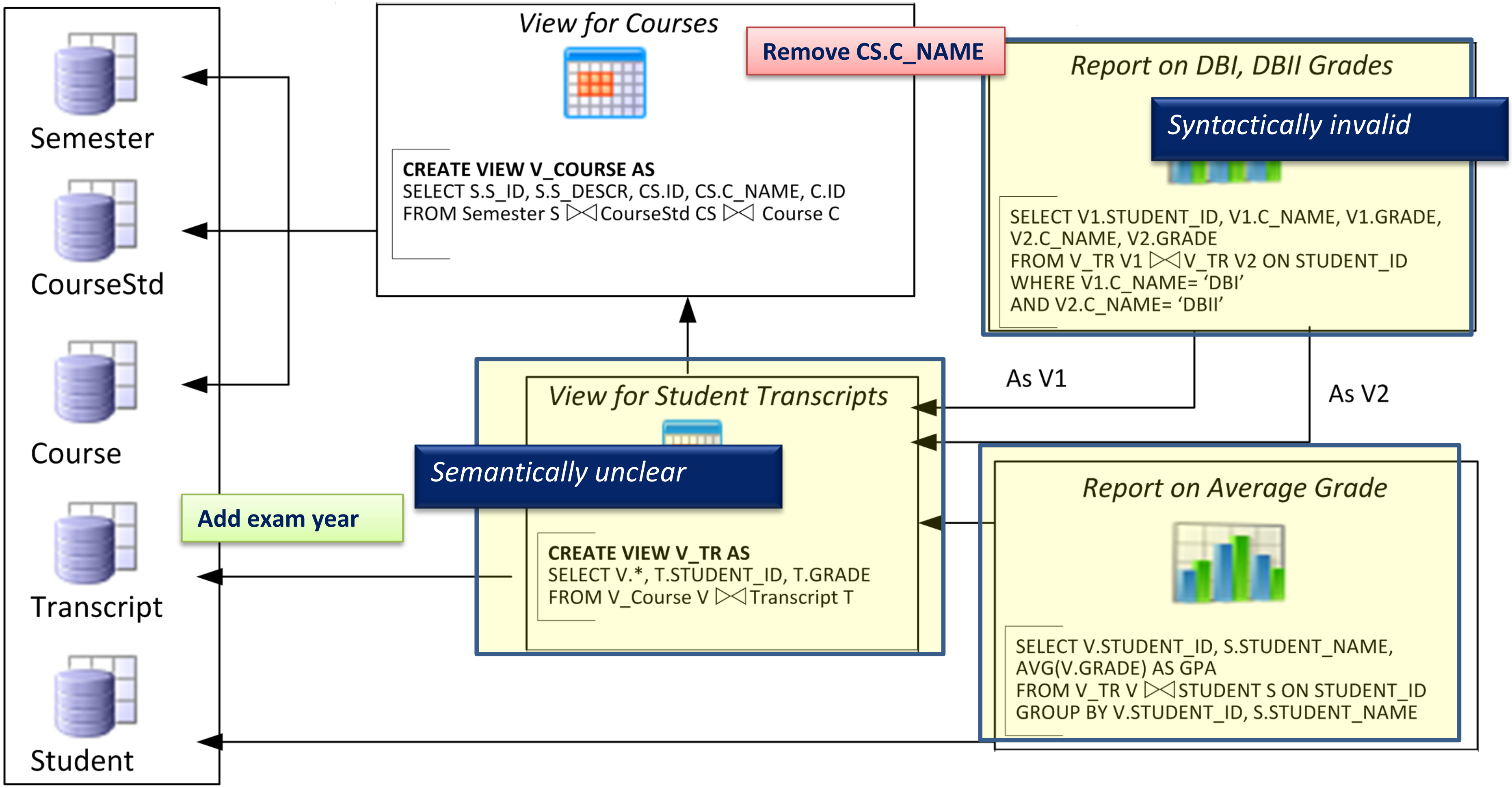
As V1

As V2

Report on Average Grade

```
SELECT V.STUDENT_ID, S.STUDENT_NAME,
AVG(V.GRADE) AS GPA
FROM V_TR V ⋈STUDENT S ON STUDENT_ID
GROUP BY V.STUDENT_ID, S.STUDENT_NAME
```

Evolving data-intensive ecosystem



```
CREATE VIEW V_COURSE AS
SELECT S.S_ID, S.S_DESCR, CS.ID, CS.C_NAME, C.ID
FROM Semester S ⋈CourseStd CS ⋈ Course C
```

```
SELECT V1.STUDENT_ID, V1.C_NAME, V1.GRADE,
V2.C_NAME, V2.GRADE
FROM V_TR V1 ⋈V_TR V2 ON STUDENT_ID
WHERE V1.C_NAME= 'DBI'
AND V2.C_NAME= 'DBII'
```

```
CREATE VIEW V_TR AS
SELECT V.*, T.STUDENT_ID, T.GRADE
FROM V_Course V ⋈Transcript T
```

```
SELECT V.STUDENT_ID, S.STUDENT_NAME,
AVG(V.GRADE) AS GPA
FROM V_TR V ⋈STUDENT S ON STUDENT_ID
GROUP BY V.STUDENT_ID, S.STUDENT_NAME
```

The impact can be syntactical (causing crashes), semantic (causing info loss or inconsistencies) and related to the performance

The impact of changes & a wish-list

- Syntactic: scripts & reports simply crash
- Semantic: views and applications can become inconsistent or information losing
- Performance: can vary a lot

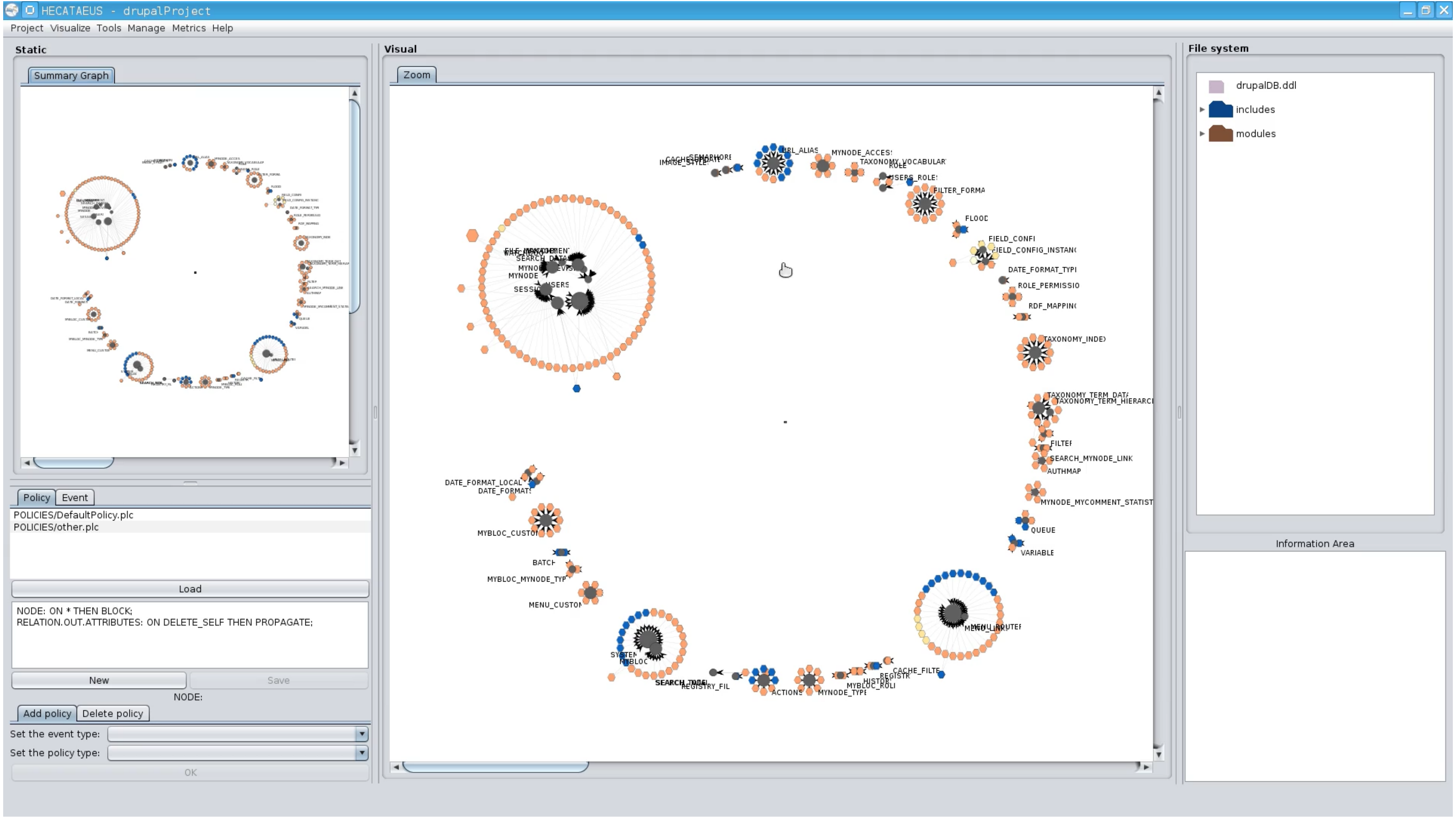
We would like: evolution predictability

i.e., control of what will be affected

before changes happen

- Learn what changes & how
- Find ways to quarantine effects

The **Hecataeus** tool & method.
Here: a first map of Drupal

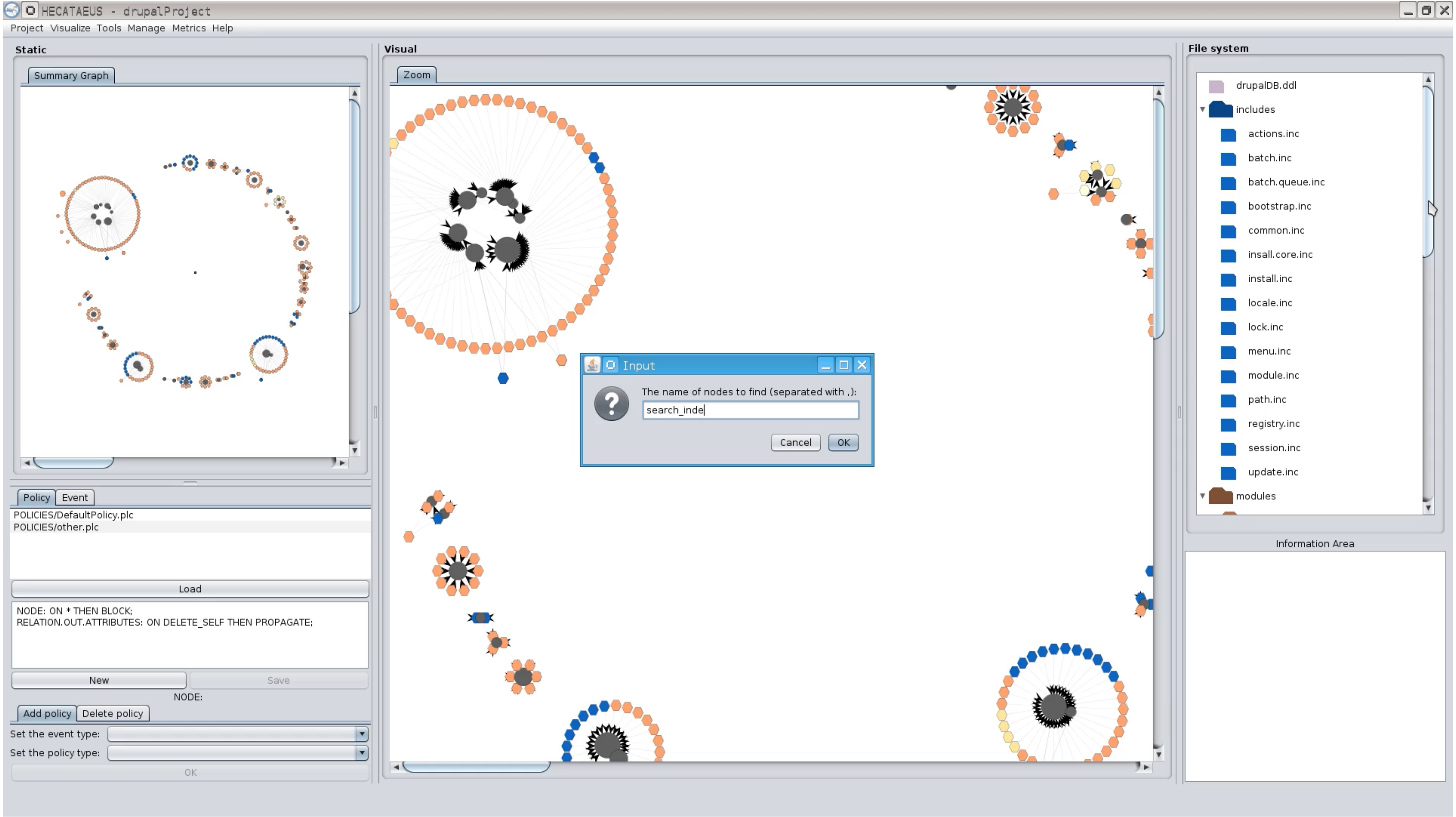


POLICIES/DefaultPolicy.plc
POLICIES/other.plc

NODE: ON * THEN BLOCK;
RELATION.OUT.ATTRIBUTES: ON DELETE_SELF THEN PROPAGATE;

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

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



POLICIES/DefaultPolicy.plc
POLICIES/other.plc

NODE: ON * THEN BLOCK;
RELATION.OUT.ATTRIBUTES: ON DELETE_SELF THEN PROPAGATE;

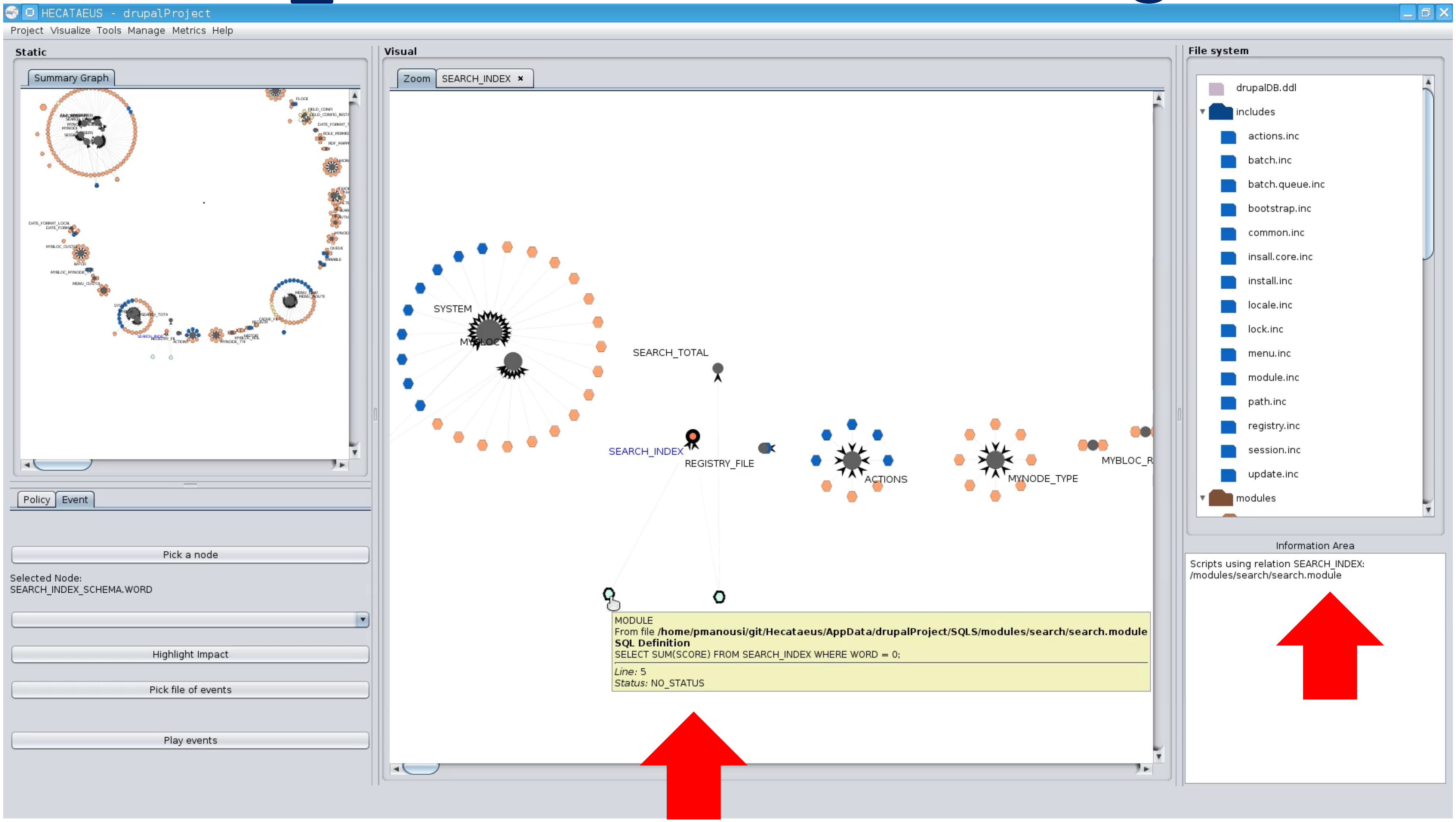
NODE:

Set the event type:

Set the policy type:

The name of nodes to find (separated with ,): search_inde

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



Selected Node:
SEARCH_INDEX_SCHEMA.WORD

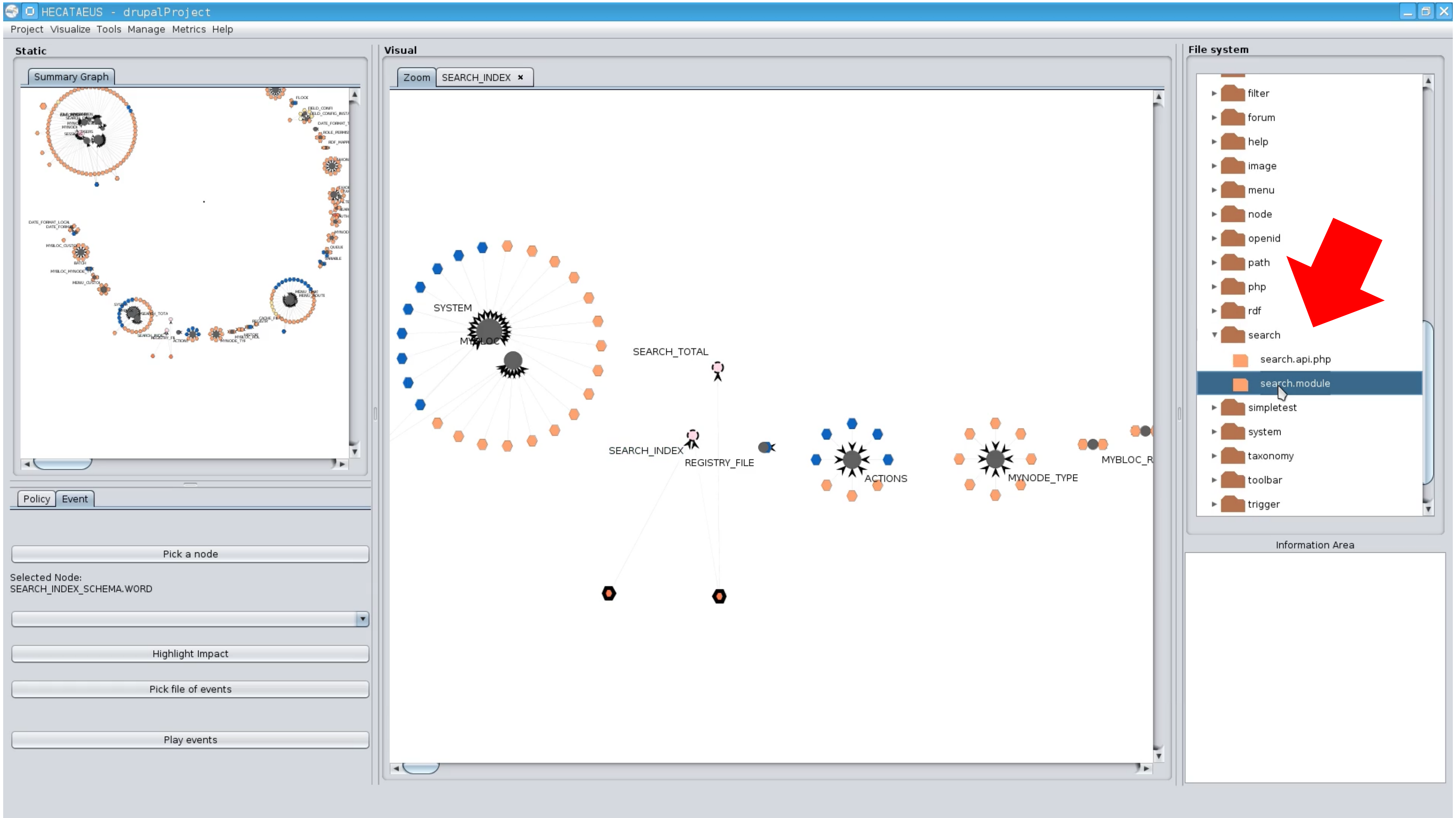
MODULE
From file **/home/pmanousi/git/Hecataeus/AppData/drupalProject/SQLS/modules/search/search.module**
SQL Definition
SELECT SUM(SCORE) FROM SEARCH_INDEX WHERE WORD = 0;

Line: 5
Status: NO_STATUS

Scripts using relation SEARCH_INDEX:
/modules/search/search.module

Tooltips with info on the script & query

In the file structure too…



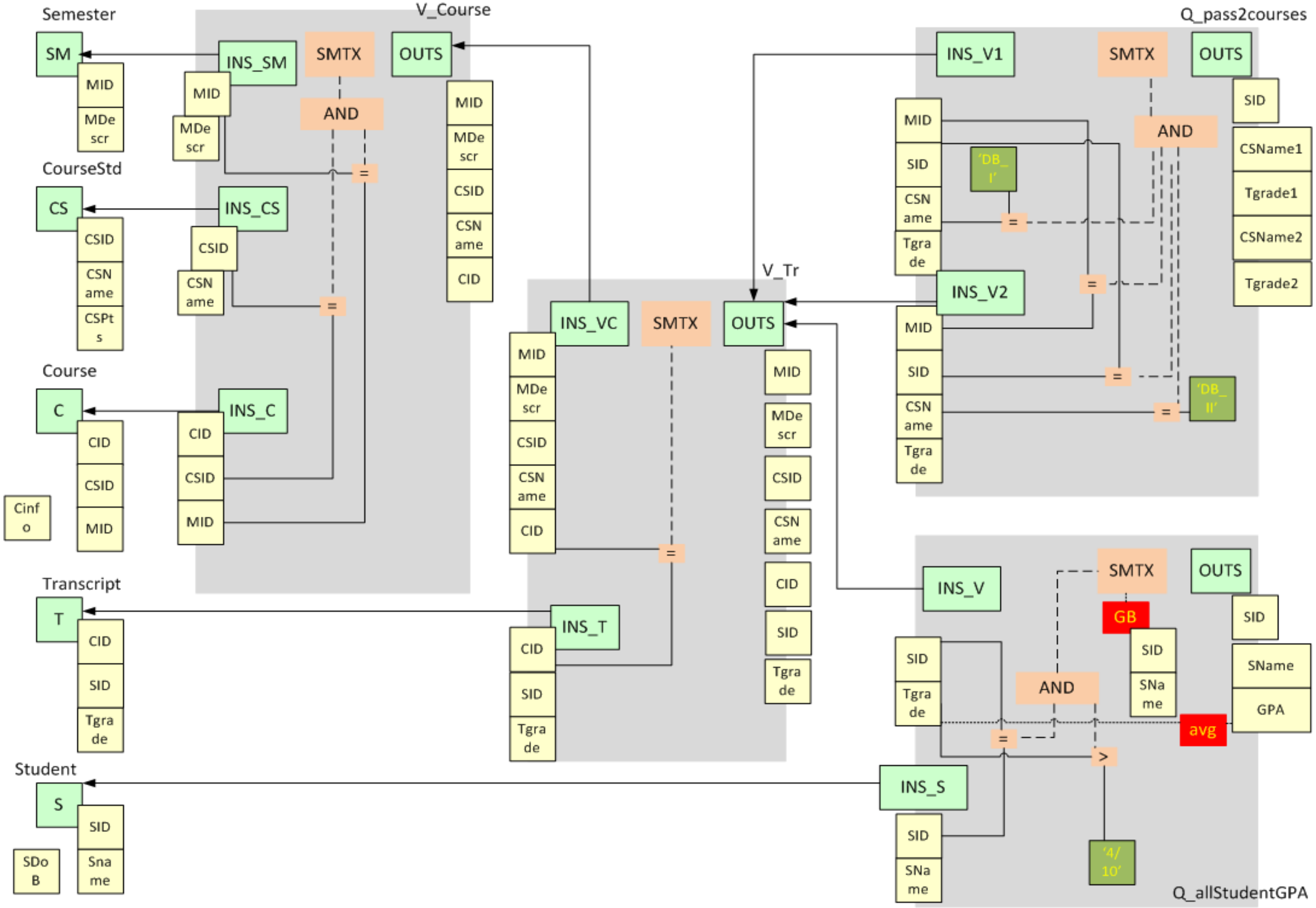
Selected Node:
SEARCH_INDEX_SCHEMA.WORD

How to handle evolution?

- Architecture Graphs: graph with the data flow between modules (i.e., relations, views or queries) at the detailed (attribute) level; module internals are also modeled as subgraphs of the Architecture Graph
- Policies, that annotate a module with a reaction for each possible event that it can withstand, in one of two possible modes:
  - (a) block, to veto the event and demand that the module retains its previous structure and semantics, or,
  - (b) propagate, to allow the event and adapt the module to a new internal structure.
- Given a potential change in the ecosystem
  - we identify which parts of the ecosystem are affected via a “change propagation” algorithm
  - we rewrite the ecosystem to reflect the new version in the parts that are affected and do not veto the change via a rewriting algorithm
    - Within this task, we resolve conflicts (different modules dictate conflicting reactions) via a conflict resolution algorithm

Manousis+ @ ER 2013 for the details of impact analysis (summary coming)
ER 2014 for the visualization (not here)

University E/S Architecture Graph

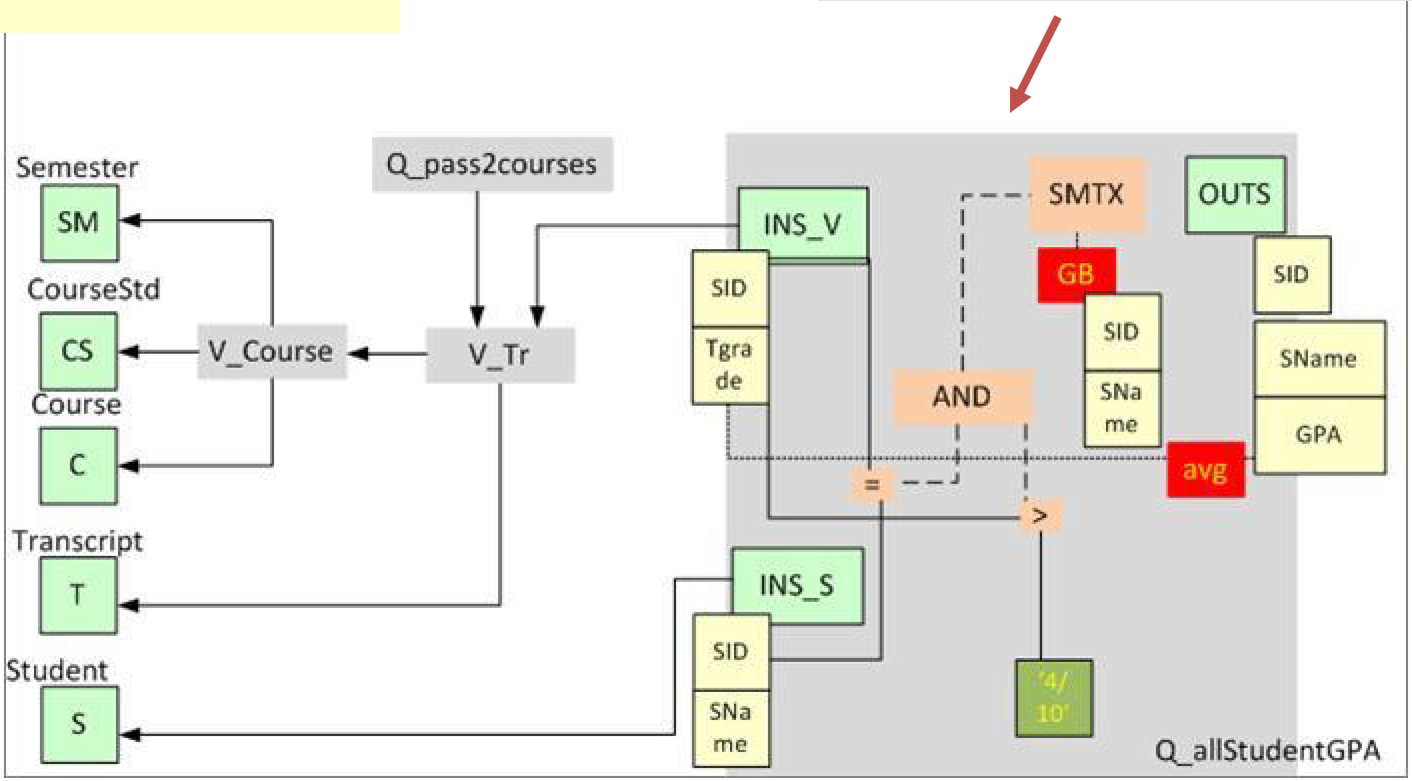


Architecture Graph

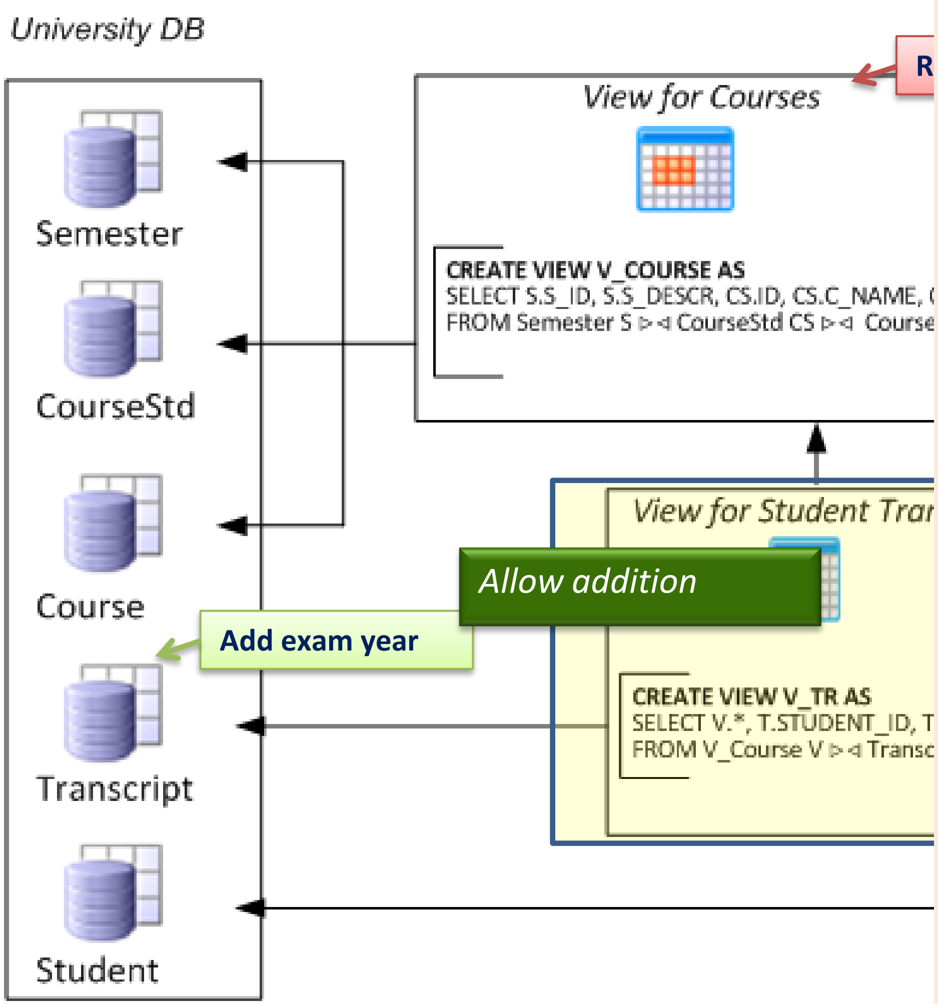
Modules and Module Encapsulation

Observe the input and output schemata!!

```
SELECT  V.STUDENT_ID, S.STUDENT_NAME,
                AVG(V.TGRADE) AS GPA
FROM V_TR V |><| STUDENT S ON STUDENT_ID
WHERE V.TGRADE > 4 / 10
GROUP BY V.STUDENT_ID, S.STUDENT_NAME
```



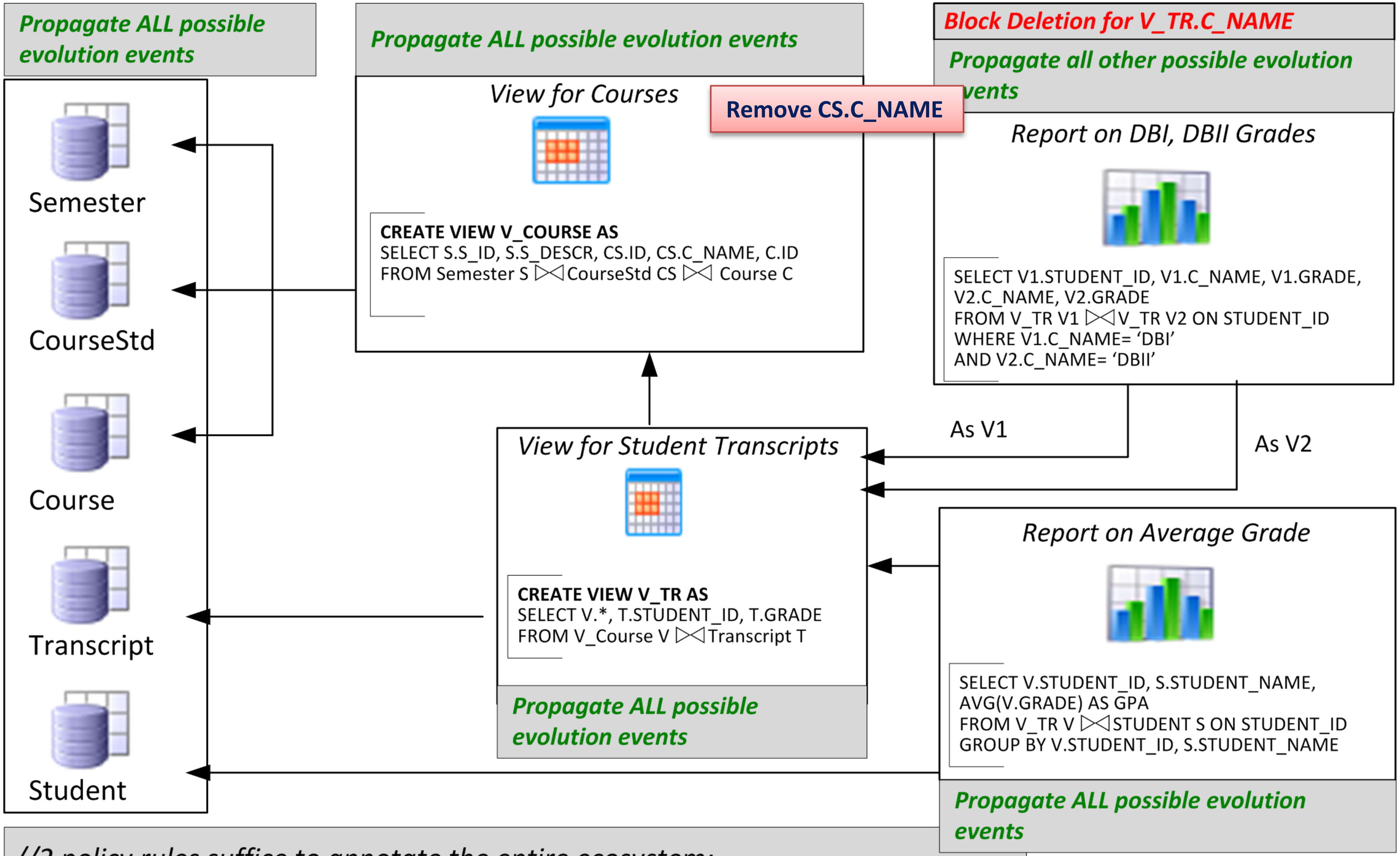
Policies to predetermine reactions



```
RELATION.OUT.SELF: on ADD_ATTRIBUTE then PROPAGATE;
RELATION.OUT.SELF: on DELETE_SELF then PROPAGATE;
RELATION.OUT.SELF: on RENAME_SELF then PROPAGATE;
RELATION.OUT.ATTRIBUTES: on DELETE_SELF then PROPAGATE;
RELATION.OUT.ATTRIBUTES: on RENAME_SELF then PROPAGATE;

VIEW.OUT.SELF: on ADD_ATTRIBUTE then PROPAGATE;
VIEW.OUT.SELF: on ADD_ATTRIBUTE_PROVIDER then PROPAGATE;
VIEW.OUT.SELF: on DELETE_SELF then PROPAGATE;
VIEW.OUT.SELF: on RENAME_SELF then PROPAGATE;
VIEW.OUT.ATTRIBUTES: on DELETE_SELF then PROPAGATE;
VIEW.OUT.ATTRIBUTES: on RENAME_SELF then PROPAGATE;
VIEW.OUT.ATTRIBUTES: on DELETE_PROVIDER then PROPAGATE;
VIEW.OUT.ATTRIBUTES: on RENAME_PROVIDER then PROPAGATE;
VIEW.IN.SELF: on DELETE_PROVIDER then PROPAGATE;
VIEW.IN.SELF: on RENAME_PROVIDER then PROPAGATE;
VIEW.IN.SELF: on ADD_ATTRIBUTE_PROVIDER then PROPAGATE;
VIEW.IN.ATTRIBUTES: on DELETE_PROVIDER then PROPAGATE;
VIEW.IN.ATTRIBUTES: on RENAME_PROVIDER then PROPAGATE;
VIEW.SMTX.SELF: on ALTER_SEMANTICS then PROPAGATE;

QUERY.OUT.SELF: on ADD_ATTRIBUTE then PROPAGATE;
QUERY.OUT.SELF: on ADD_ATTRIBUTE_PROVIDER then PROPAGATE;
QUERY.OUT.SELF: on DELETE_SELF then PROPAGATE;
QUERY.OUT.SELF: on RENAME_SELF then PROPAGATE;
QUERY.OUT.ATTRIBUTES: on DELETE_SELF then PROPAGATE;
QUERY.OUT.ATTRIBUTES: on RENAME_SELF then PROPAGATE;
QUERY.OUT.ATTRIBUTES: on DELETE_PROVIDER then PROPAGATE;
QUERY.OUT.ATTRIBUTES: on RENAME_PROVIDER then PROPAGATE;
QUERY.IN.SELF: on DELETE_PROVIDER then PROPAGATE;
QUERY.IN.SELF: on RENAME_PROVIDER then PROPAGATE;
QUERY.IN.SELF: on ADD_ATTRIBUTE_PROVIDER then PROPAGATE;
QUERY.IN.ATTRIBUTES: on DELETE_PROVIDER then PROPAGATE;
QUERY.IN.ATTRIBUTES: on RENAME_PROVIDER then PROPAGATE;
QUERY.SMTX.SELF: on ALTER_SEMANTICS then PROPAGATE;
```

Policies to predetermine the modules' reaction to a hypothetical event?

How to handle evolution?


Propagate ALL possible evolution events
Semester
CourseStd
Course
Transcript
Student
Propagate ALL possible evolution events
View for Courses
Remove CS.C_NAME
Block Deletion for V_TR.C_NAME
Propagate all other possible evolution events
Report on DBI, DBII Grades
As V1
As V2
View for Student Transcripts
Propagate ALL possible evolution events
Report on Average Grade
Propagate ALL possible evolution events

```
CREATE VIEW V_COURSE AS
SELECT S.S_ID, S.S_DESCR, CS.ID, CS.C_NAME, C.ID
FROM Semester S ⋈CourseStd CS ⋈ Course C
```

```
SELECT V1.STUDENT_ID, V1.C_NAME, V1.GRADE,
V2.C_NAME, V2.GRADE
FROM V_TR V1 ⋈V_TR V2 ON STUDENT_ID
WHERE V1.C_NAME= 'DBI'
AND V2.C_NAME= 'DBII'
```

```
CREATE VIEW V_TR AS
SELECT V.*, T.STUDENT_ID, T.GRADE
FROM V_Course V ⋈Transcript T
```

```
SELECT V.STUDENT_ID, S.STUDENT_NAME,
AVG(V.GRADE) AS GPA
FROM V_TR V ⋈STUDENT S ON STUDENT_ID
GROUP BY V.STUDENT_ID, S.STUDENT_NAME
```

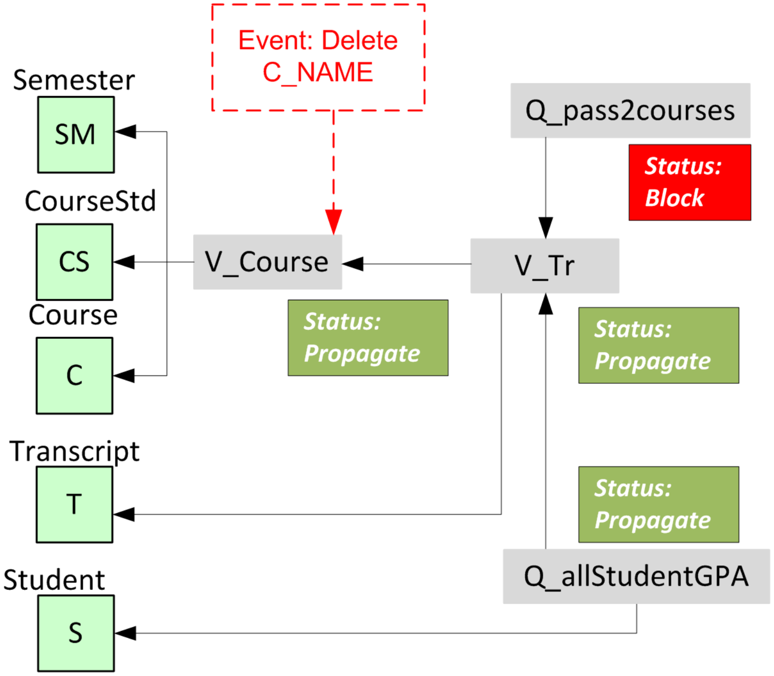
//2 policy rules suffice to annotate the entire ecosystem:

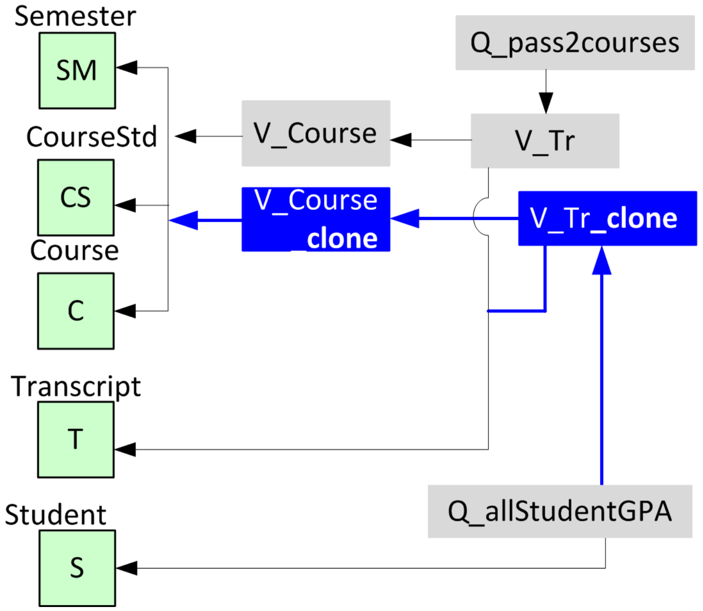
```
NODE: ON * THEN PROPAGATE;
Q_pass2courses_IN_V1.C_SNAME ON DELETE_SELF THEN BLOCK;
```

Internals of impact assess. & rewriting

1. **Impact assessment**. Given a potential event, a status determination algorithm makes sure that the nodes of the ecosystem are assigned a status concerning (a) whether they are affected by the event or not and (b) what their reaction to the event is (block or propagate).
2. **Conflict resolution and calculation of variants**. Algorithm that checks the affected parts of the graph in order to highlight affected nodes with whether they will adapt to a new version or retain both their old and new variants.
3. **Module Rewriting**. Our algorithm visits affected modules sequentially and performs the appropriate restructuring of nodes and edges.

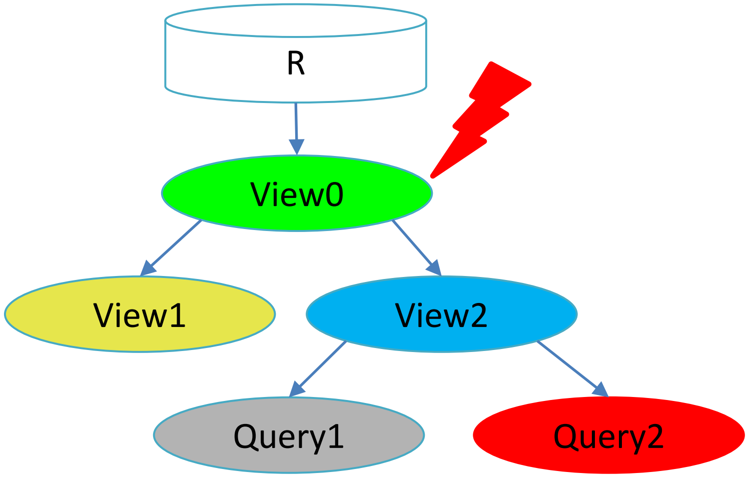
Impact assessment & rewriting



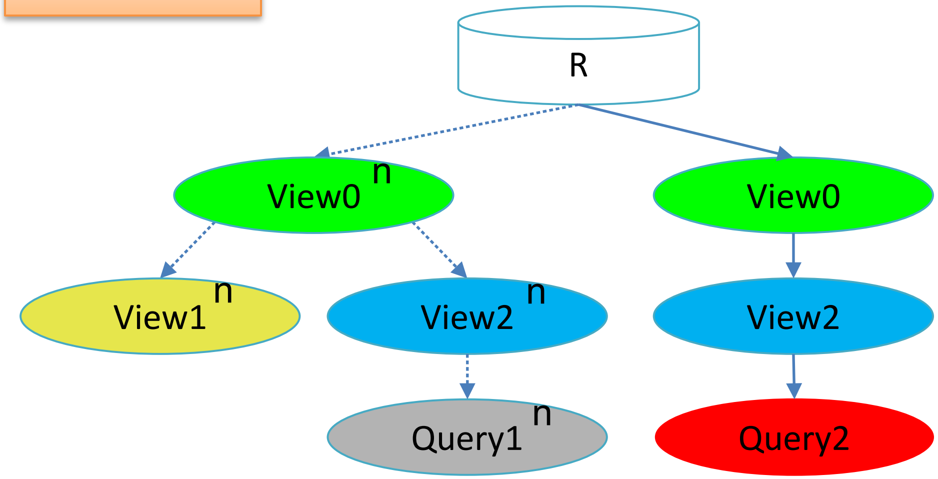


Conflicts: what they are and how to handle them (more than flooding)

BEFORE



AFTER

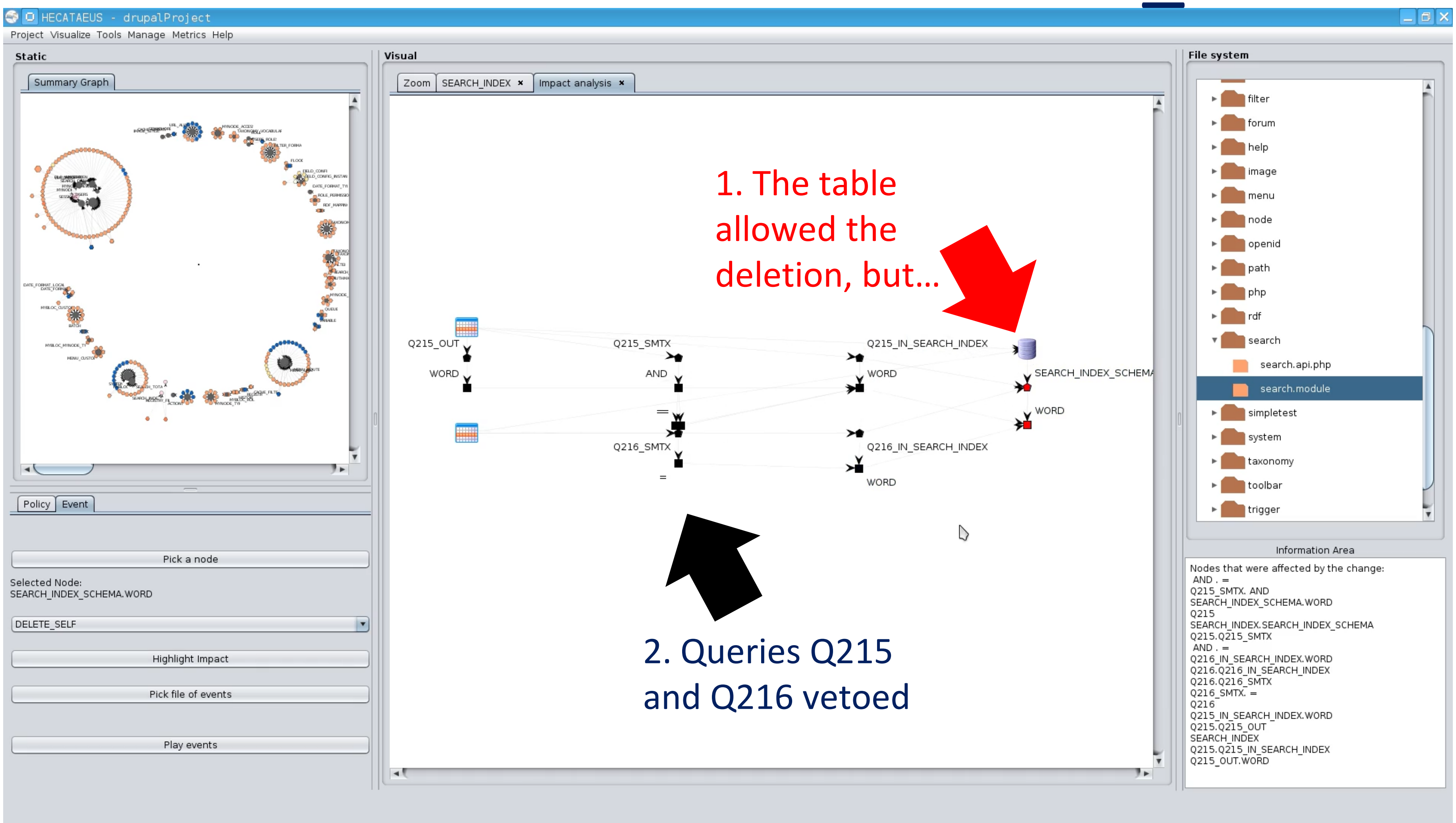


- View0 initiates a change
- View1 and View 2 accept the change

- Query2 rejects the change
- Query1 accepts the change

- The path to Query2 is left intact, so that it retains it semantics
- View1 and Query1 are adapted
- View0 and View2 are adapted too, however, we need two version for each: one to serve Query2 and another to serve View1 and Query1

Played an impact analysis scenario: delete attr. ‘word’ from search_index



Selected Node:
SEARCH_INDEX_SCHEMA.WORD

DELETE_SELF

1. The table allowed the deletion, but…

2. Queries Q215 and Q216 vetoed

```
Nodes that were affected by the change:
 AND . =
Q215_SMTX. AND
SEARCH_INDEX_SCHEMA.WORD
Q215
SEARCH_INDEX.SEARCH_INDEX_SCHEMA
Q215.Q215_SMTX
 AND . =
Q216_IN_SEARCH_INDEX.WORD
Q216.Q216_IN_SEARCH_INDEX
Q216.Q216_SMTX
Q216_SMTX. =
Q216
Q215_IN_SEARCH_INDEX.WORD
Q215.Q215_OUT
SEARCH_INDEX
Q215.Q215_IN_SEARCH_INDEX
Q215_OUT.WORD
```

Other efforts

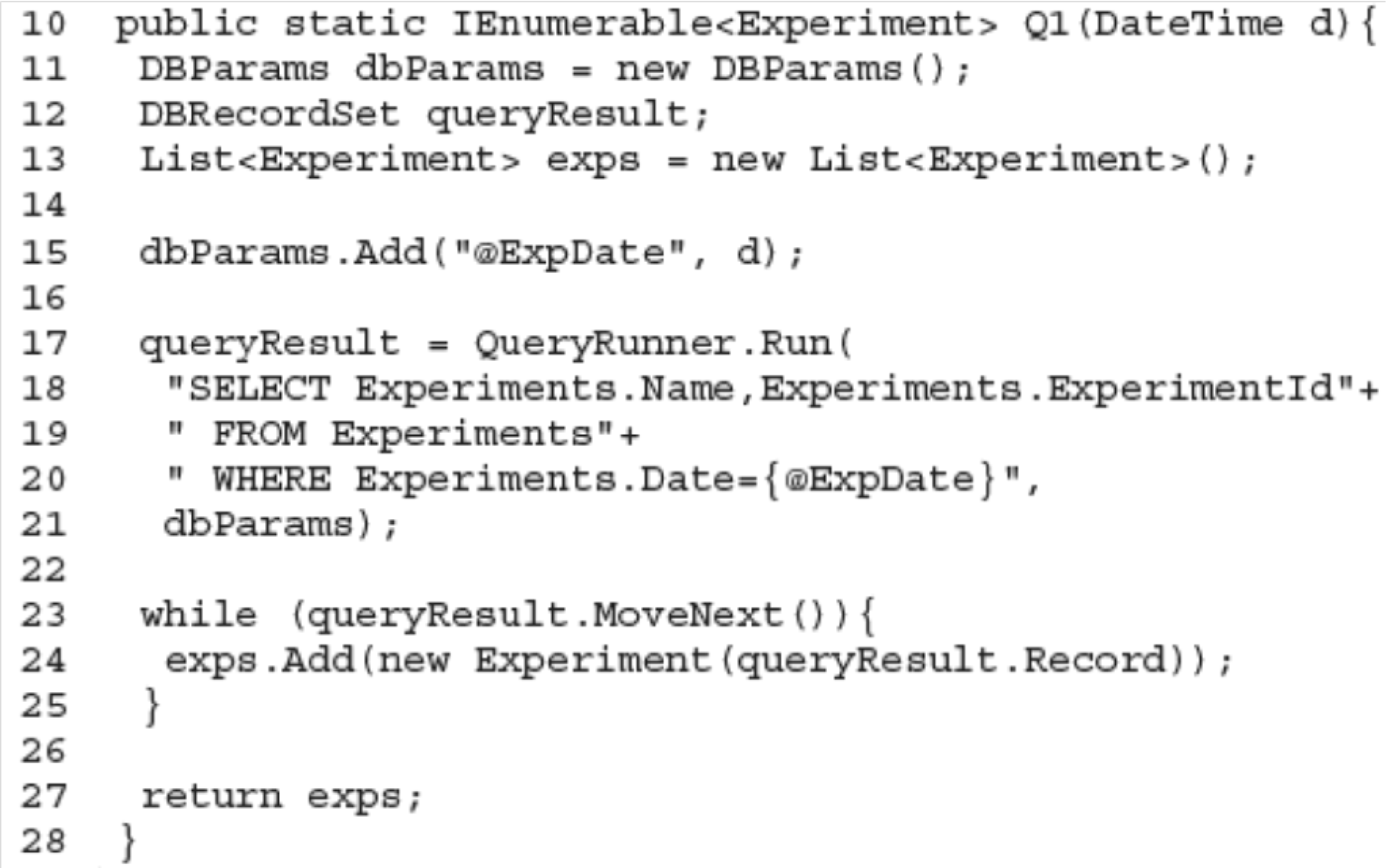
- Maule et al @ ICSE 2008
- The Prism/Prism++ line of research

Maule et al. @ ICSE’08

- Given an OO app. built on top of a relational db schema and a change type
- Produce the locations of the code that are affected
- Method:

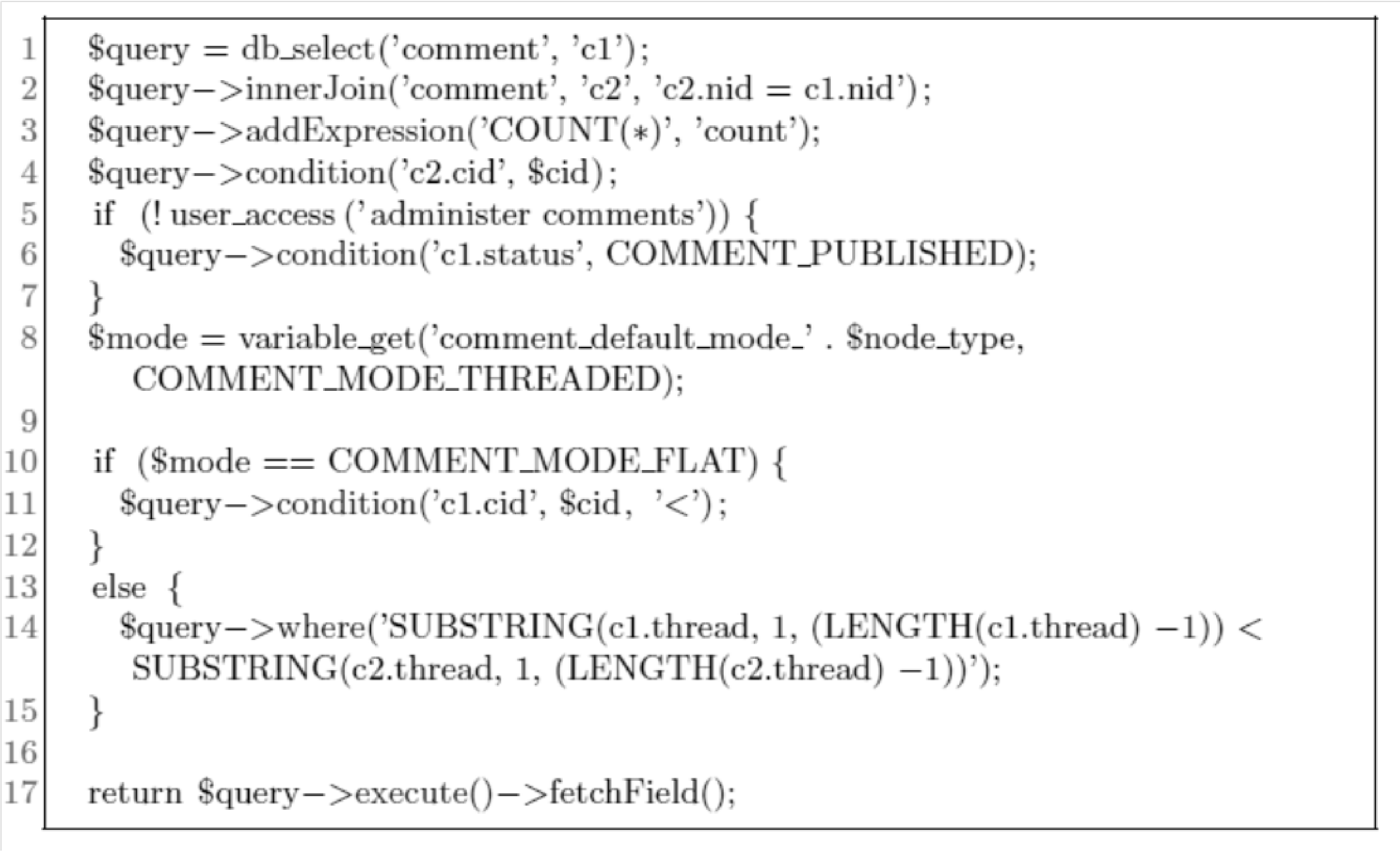
1. Slicing. A prototype slicing implementation to identify the database queries of the program.
2. A data-flow analysis algorithm to estimate all the possible runtime values for the parameters of the query.
3. Use an impact assessment tool, Crocopat, with a reasoning language (RML). Depending on the type of change, a different RML program that assesses impact over the stored data of the previous step is run: this isolates the lines of code affected by the change.

[On the side] code assumed to be of the form:



```
public static IEnumerable<Experiment> Q1(DateTime d){
 DBParams dbParams = new DBParams();
 DBRecordSet queryResult;
 List<Experiment> exps = new List<Experiment>();

 dbParams.Add("@ExpDate", d);

 queryResult = QueryRunner.Run(
  "SELECT Experiments.Name,Experiments.ExperimentId"+
  " FROM Experiments"+
  " WHERE Experiments.Date={@ExpDate}",
  dbParams);

 while (queryResult.MoveNext()){
  exps.Add(new Experiment(queryResult.Record));
 }

 return exps;
}
```

… whereas what you can get is …



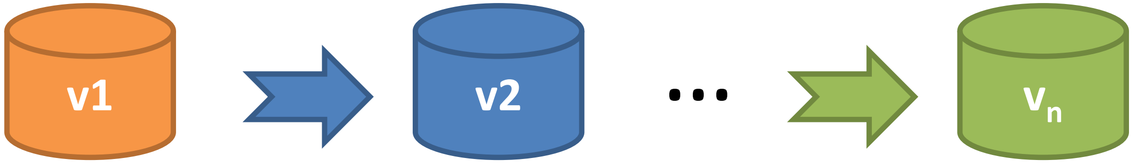
```
$query = db_select(’comment’, ’c1’);
$query->innerJoin(’comment’, ’c2’, ’c2.nid = c1.nid’);
$query->addExpression(’COUNT(*)’, ’count’);
$query->condition(’c2.cid’, $cid);
if (! user_access (’administer comments’)) {
  $query->condition(’c1.status’, COMMENT_PUBLISHED);
}
$mode = variable_get(’comment_default_mode_’ . $node_type,
   COMMENT_MODE_THREADED);

if ($mode == COMMENT_MODE_FLAT) {
  $query->condition(’c1.cid’, $cid, ’<’);
}
else {
  $query->where(’SUBSTRING(c1.thread, 1, (LENGTH(c1.thread) -1)) <
   SUBSTRING(c2.thread, 1, (LENGTH(c2.thread) -1))’);
}

return $query->execute()->fetchField();
```

Prism/Prism++

- Series of works from the same authors
- Carlo Curino, Hyun Jin Moon, Carlo Zaniolo. Graceful database schema evolution: the PRISM workbench. PVLDB 1(1): 761-772 (2008)
- Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++. *PVLDB, 4(2):117–128,* 2010.
- Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. Automating the database schema evolution process. *VLDB J.,* 22(1):73–98, 2013.

Prism/Prism++ motivation

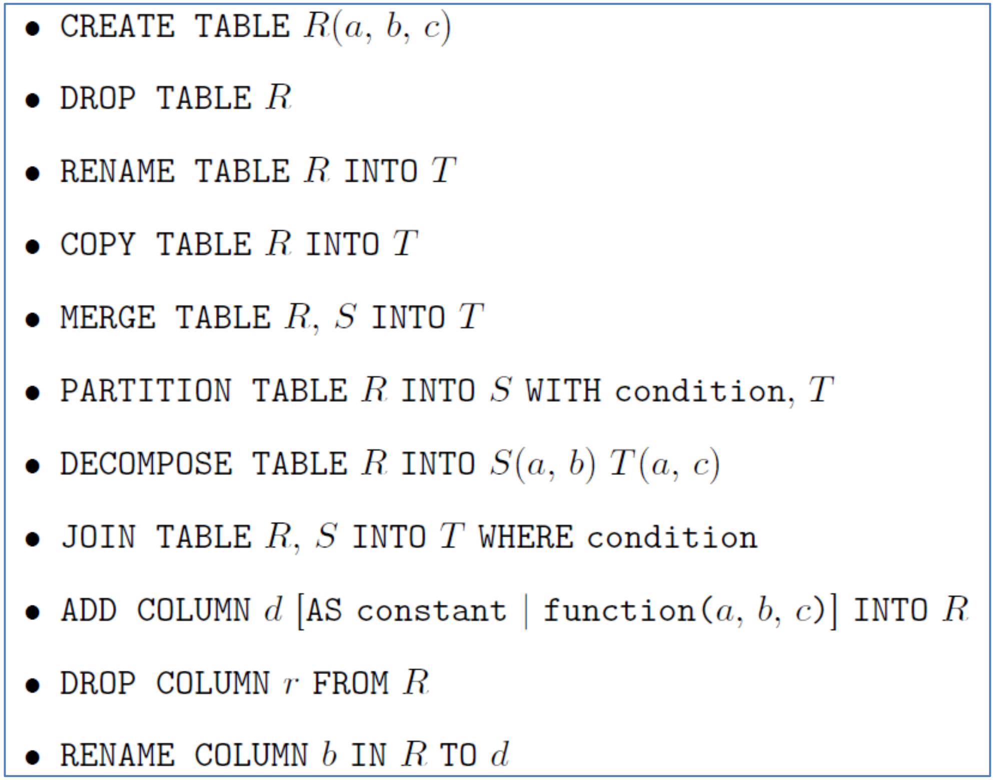
- Evolution happens all the time => can be viewed as a sequence of changes



Automatically migrate schema + data+ surrounding queries

- Schema Modification Operators (SMO's) are a principled set of operators to describe evolution steps, s.t.:
  - The evolution DDL is implied by the SMO's
  - The DML for data migration can be automatically produced from the SMO's
  - The surrounding queries can be rewritten to the new schema

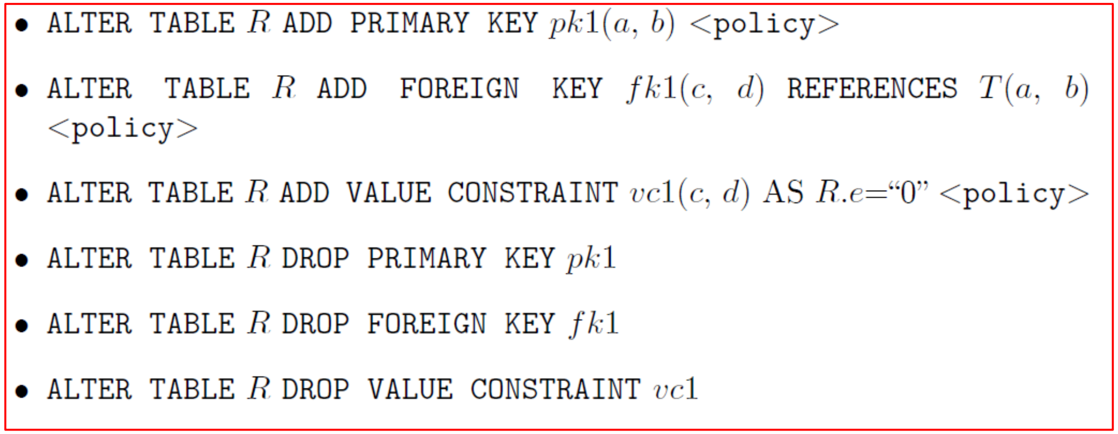
SMO's and ICSMO's



- CREATE TABLE $R(a, b, c)$
- DROP TABLE R
- RENAME TABLE R INTO T
- COPY TABLE R INTO T
- MERGE TABLE R, S INTO T
- PARTITION TABLE R INTO S WITH condition, T
- DECOMPOSE TABLE R INTO $S(a, b)$ $T(a, c)$
- JOIN TABLE R, S INTO T WHERE condition
- ADD COLUMN d [AS constant | function(a, b, c)] INTO R
- DROP COLUMN r FROM R
- RENAME COLUMN b IN R TO d

Automatic creation of
- DDL (schema evo)
- DML (data migration)
is feasible

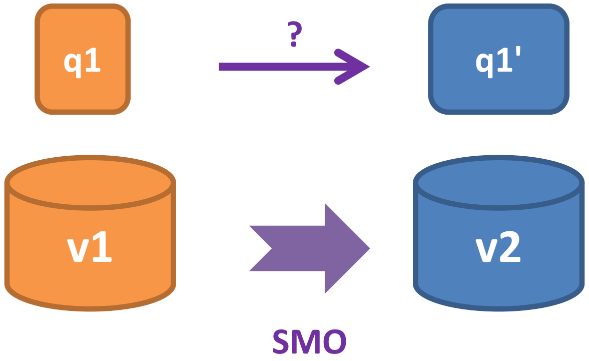
ICSMO's: the technique is extended to cover Integrity Constraints too.



- ALTER TABLE R ADD PRIMARY KEY $pk1(a, b)$ <policy>
- ALTER TABLE R ADD FOREIGN KEY $fk1(c, d)$ REFERENCES $T(a, b)$ <policy>
- ALTER TABLE R ADD VALUE CONSTRAINT $vc1(c, d)$ AS $R.e$="0" <policy>
- ALTER TABLE R DROP PRIMARY KEY $pk1$
- ALTER TABLE R DROP FOREIGN KEY $fk1$
- ALTER TABLE R DROP VALUE CONSTRAINT $vc1$

<policy>
(i) CHECK if current
db satisfies the constraint, else ICMO is rolled back,
(ii) ENFORCE the removal of all data violating the constraint,
(iii) IGNORE violating tuples + informs the user about this.

Answering old queries to new schemata without user noticing it



- Assume we migrate the schema + data from v1 to v2
- Can we rewrite the query q1 to q1' s.t. we get the same result, as if we were still in v1?

- SMO invertibility:
- q1/v1 = q1 /SMO^{-1}(v2) = q1' / v2

```
V1: R(…)
Q1: SELECT * FROM R

SMO: PARTITION R in S(…), T(…)

Q1': SELECT * FROM S,T WHERE S.ID =T.ID
```

Roadmap

- Evolution of views
- Data warehouse Evolution
- A case study (if time)
- Impact assessment in ecosystems
- **Empirical studies concerning database evolution**
- Open Issues and discussions

A timeline of efforts concerning empirical studies

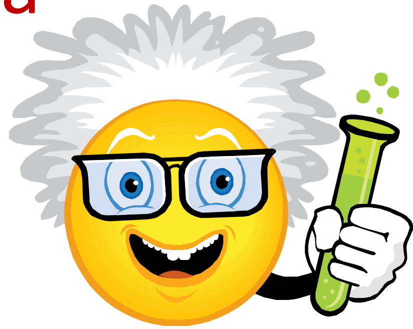
Results by Univ. Ioannina

EMPIRICAL STUDIES

WHAT ARE THE “LAWS” OF DATABASE SCHEMA EVOLUTION?

What are the “laws” of database (schema) evolution?

- How do databases change?
- In particular, how does the schema of a database evolve over time?



- Long term research goals:
  - Are there any “invariant properties” (e.g., patterns of repeating behavior) on the way database (schemata) change?
  - Is there a theory / model to explain them?

Why care for the “laws”/patterns of schema evolution?

- Scientific curiosity!
- Practical Impact: DB’s are dependency magnets. Applications have to conform to the structure of the db…
  - typically, development waits till the “db backbone” is stable and applications are build on top of it
  - slight changes to the structure of a db can cause several (parts of) different applications to crash, causing the need for emergency repairing

Imagine if we could predict how a schema will evolve over time…

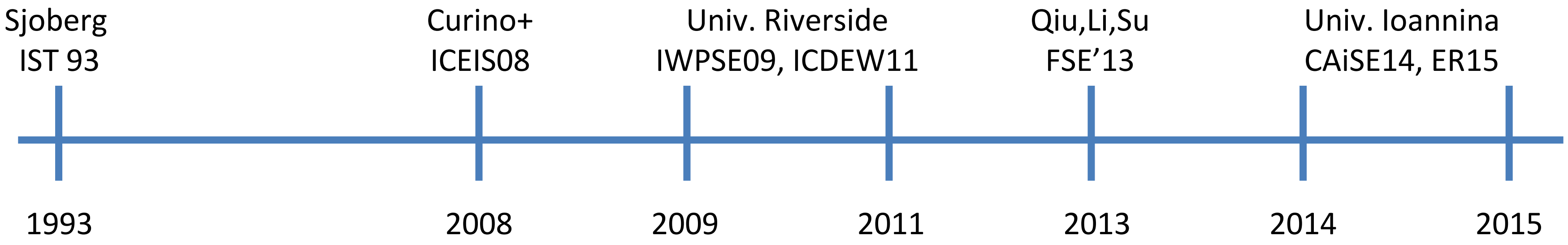
- … we would be able to **“design for evolution”** and **minimize the impact of evolution** to the surrounding applications
  - by applying design patterns
  - by avoiding anti-patterns & complexity increase

  … **in both the db and the code**
- … we would be able to **plan** administration and perfective maintenance tasks and resources, instead of responding to emergencies

Why aren’t we there yet?

- Historically, nobody from the research community had access + the right to publish to version histories of database schemata
- Open source tools internally hosting databases have changed this landscape:
  - not only is the code available, but also,
  - public repositories (git, svn, …) keep the entire history of revisions
- We are now presented with the opportunity to study the version histories of such “open source databases”

Timeline of empirical studies



Sjoberg
IST 93

Curino+
ICEIS08

Univ. Riverside
IWPSE09, ICDEW11

Qiu,Li,Su
FSE'13

Univ. Ioannina
CAiSE14, ER15

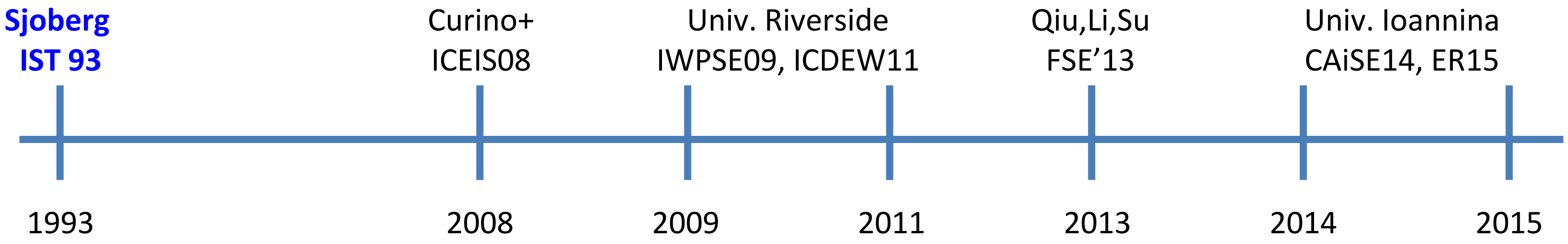
Timeline of empirical studies

Sjoberg @ IST 93: 18 months study of a health system.
139% increase of #tables ; 274% increase of the #attributes

Changes in the code (on avg):

- relation addition: 19 changes ; attribute additions: 2 changes
- relation deletion : 59.5 changes; attribute deletions: 3.25 changes

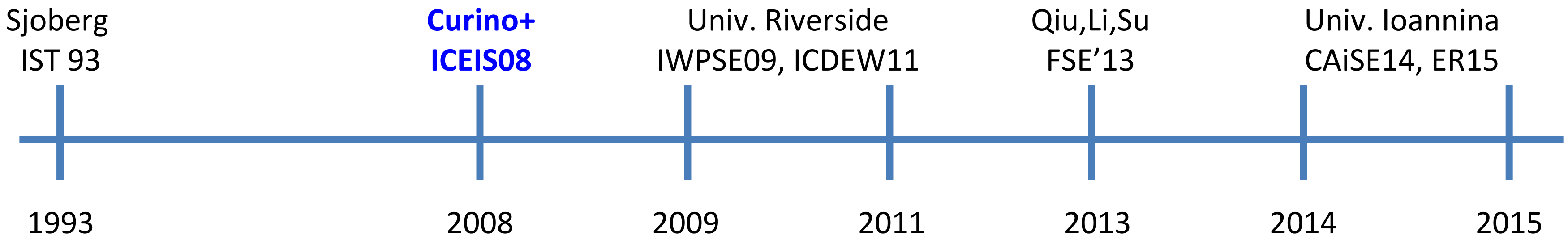
An **inflating period** during construction where almost all changes were additions, and a **subsequent period** where additions and deletions where balanced.



Timeline of empirical studies

Curino+ @ ICEIS08: Mediawiki for 4.5 years
100% increase in the number of tables
142% in the number of attributes.

45% of changes do not affect the information capacity of the schema
(but are rather index adjustments, documentation, etc)



Sjoberg
IST 93

**Curino+
ICEIS08**

Univ. Riverside
IWPSE09, ICDEW11

Qiu,Li,Su
FSE'13

Univ. Ioannina
CAiSE14, ER15

1993

2008

2009

2011

2013

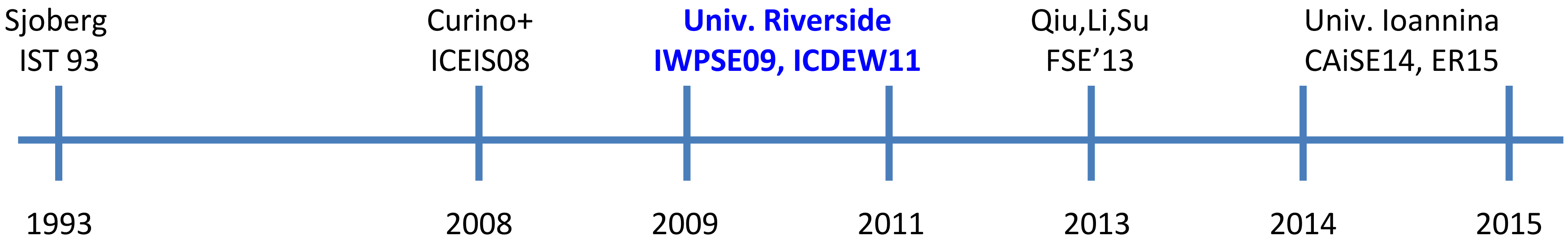
2014

2015

Timeline of empirical studies

IWPSE09: Mozilla and Monotone (a version control system)
Many ways to be out of synch between code and evolving db schema

ICDEW11: Firefox, Monotone , Biblioteq (catalogue man.) , Vienna (RSS)
Similar pct of changes with previous work
Frequency and timing analysis: **db schemata tend to stabilize over time**, as there is more change at the beginning of their history, but seem to converge to a relatively fixed structure later



Sjoberg
IST 93

Curino+
ICEIS08

Univ. Riverside
IWPSE09, ICDEW11

Qiu,Li,Su
FSE'13

Univ. Ioannina
CAiSE14, ER15

1993

2008

2009

2011

2013

2014

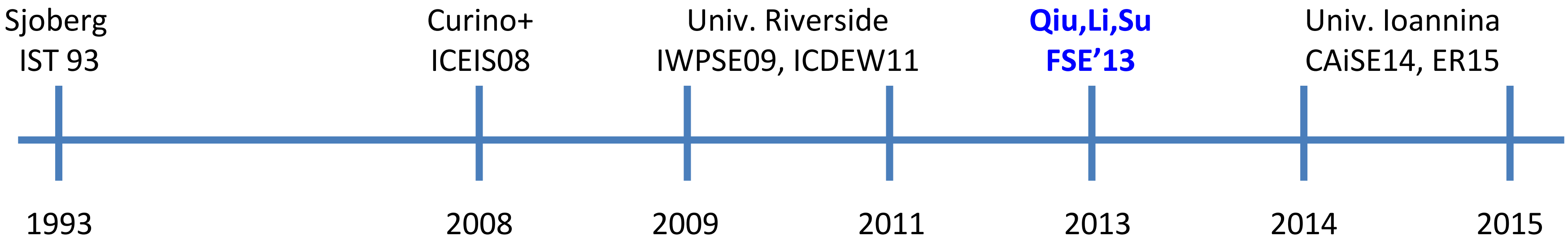
2015

Timeline of empirical studies

Qiu,Li,Su@ FSE 2013: 10 (!) database schemata studied.
Change is focused both (a) with respect to time and (b) with respect to the tables who change.

Timing: 7 out of 10 databases reached 60% of their schema size within 20% of their early lifetime.
Change is frequent in the early stages of the databases, with inflationary characteristics; then, the schema evolution process calms down.

Tables that change: 40% of tables do not undergo any change at all, and 60%-90% of changes pertain to 20% of the tables (in other words, 80% of the tables live quiet lives). The most frequently modified tables attract 80% of the changes.



Sjoberg IST 93 — 1993

Curino+ ICEIS08 — 2008

Univ. Riverside IWPSE09, ICDEW11 — 2009, 2011

Qiu,Li,Su FSE'13 — 2013

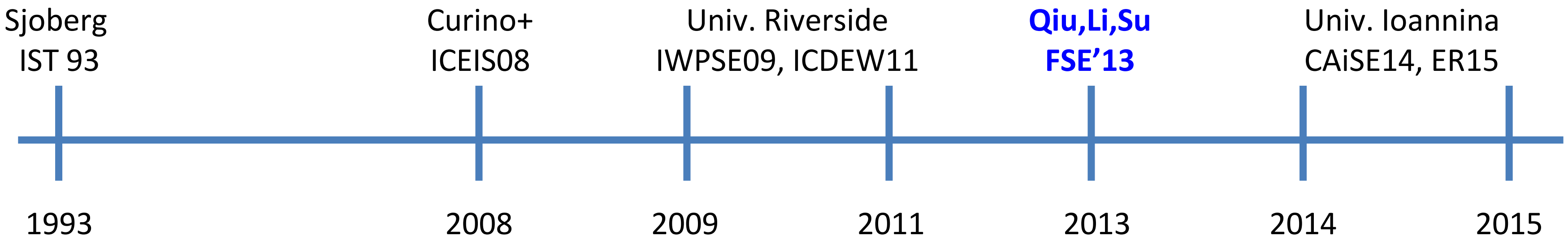
Univ. Ioannina CAiSE14, ER15 — 2014, 2015

Timeline of empirical studies

Qiu,Li,Su@ FSE 2013: **Code and db co-evolution, not always in synch**.

- Code and db changed in the same revision: 50.67% occasions
- Code change was in a previous/subsequent version than the one where the database schema change: 16.22% of occasions
- database changes not followed by code adaptation: 21.62% of occasions
- 11.49% of code changes were unrelated to the database evolution.

Each atomic change at the schema level is estimated to result in 10 -- 100 lines of application code been updated;
A valid db revision results in 100 -- 1000 lines of application code being updated



Sjoberg
IST 93

Curino+
ICEIS08

Univ. Riverside
IWPSE09, ICDEW11

Qiu,Li,Su
FSE'13

Univ. Ioannina
CAiSE14, ER15

1993

2008

2009

2011

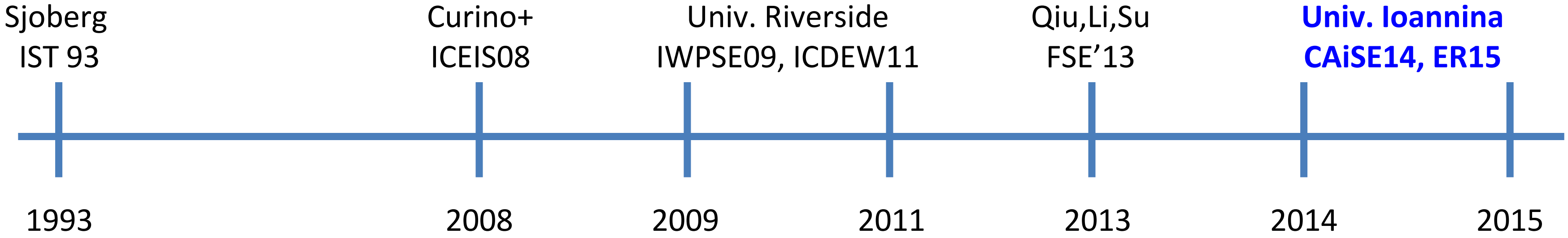
2013

2014

2015

Timeline of empirical studies

To be detailed next.
CAiSE14: DB level
ER’15: Table level



Sjoberg
IST 93

Curino+
ICEIS08

Univ. Riverside
IWPSE09, ICDEW11

Qiu,Li,Su
FSE’13

Univ. Ioannina
CAiSE14, ER15

1993

2008

2009

2011

2013

2014

2015

Our take on the problem

- Collected version histories for the schemata of 8 open-sour projects
  - CMS's: MediaWiki, TYPO3, Coppermine, phpBB, OpenCart
  - Physics: ATLAS Trigger --- Bio: Ensemble, BioSQL

- Preprocessed them to be parsable by our **HECATE schema comparison tool** and exported the transitions between each two subsequent versions and measures for them (size, growth, changes)

- Visualized the transitions in graphical representations and statistically studied the measures, both at the **macro (database)** and at the **micro (table)** level

.. What do we see if we observe the evolution of the entire schema?

http://www.cs.uoi.gr/~pvassil/publications/2014_CAiSE/

Skoulis, Vassiliadis, Zarras. Open-Source Databases: Within, Outside, or Beyond Lehman's Laws of Software Evolution? **CAiSE 2014**

Also: Growing up with stability: How open-source relational databases evolve. **Information Systems, Volume 53**, October–November 2015

SCHEMA EVOLUTION FOR O/S DB’S AT THE “MACRO” LEVEL

Datasets

https://github.com/DAINTINESS-Group/EvolutionDatasets

- Content management Systems
  - MediaWiki, TYPO3, Coppermine, phpBB, OpenCart
- Medical Databases
  - Ensemble, BioSQL
- Scientific
  - ATLAS Trigger

Data sets

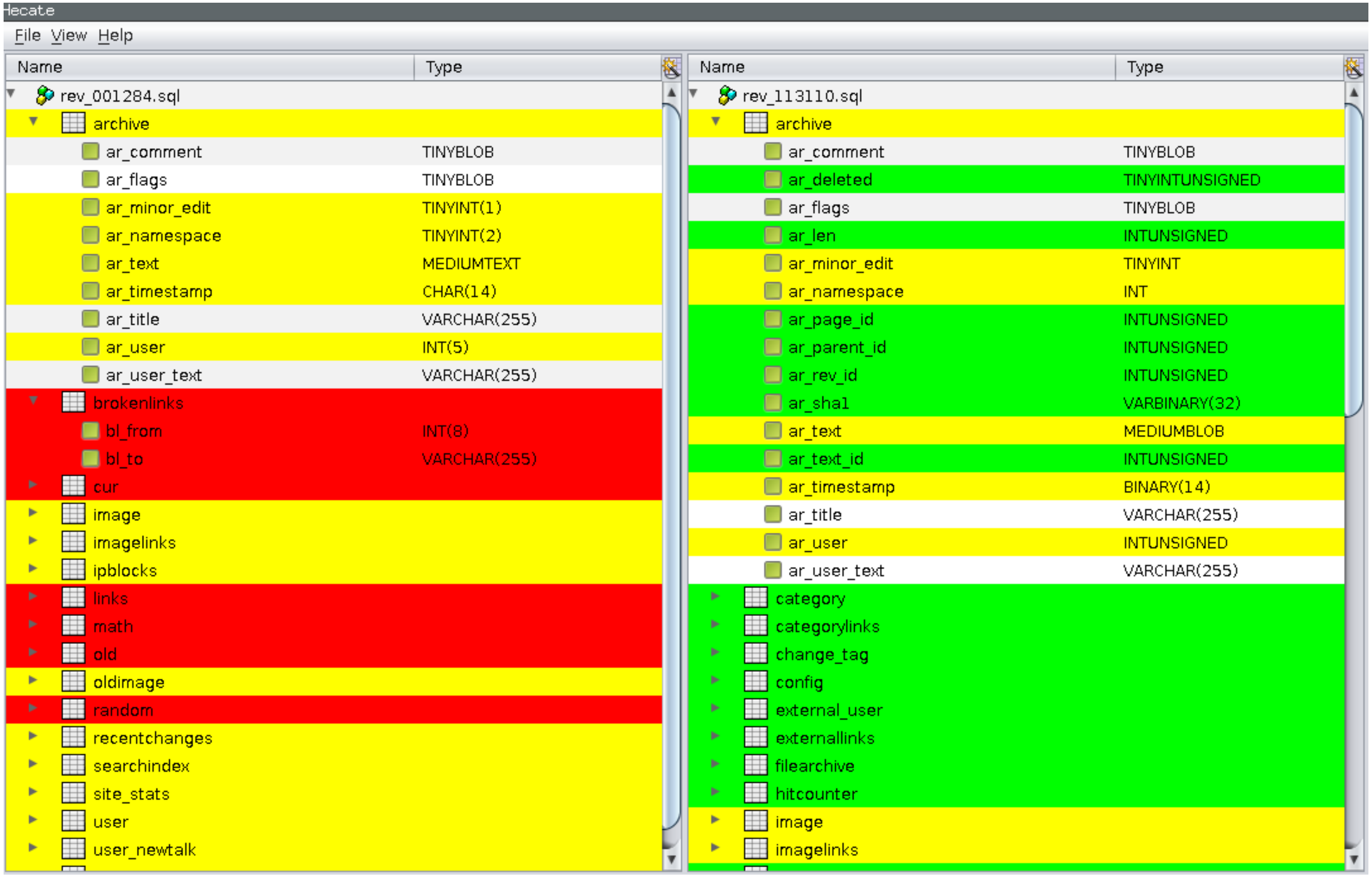
Dataset	Versions	Lifetime	Tables Start	Tables End	Attributes Start	Attributes End	Commits per Day	% commits with change	Repository URL
ATLAS Trigger	84	2 Y, 7 M, 2 D	56	73	709	858	0,089	82%	http://atdaq-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/Trigger/TrigConfiguration/TrigDb/share/sql/combined_schema.sql
BioSQL	46	10 Y, 6 M, 19 D	21	28	74	129	0,012	63%	https://github.com/biosql/biosql/blob/master/sql/biosqldb-mysql.sql
Coppermine	117	8 Y, 6 M, 2 D	8	22	87	169	0,038	50%	http://sourceforge.net/p/coppermine/code/8581/tree/trunk/cpg1.5.x/sql/schema.sql
Ensembl	528	13 Y, 3 M, 15 D	17	75	75	486	0,109	60%	http://cvs.sanger.ac.uk/cgi-bin/viewvc.cgi/ensembl/sql/table.sql?root=ensembl&view=log
MediaWiki	322	8 Y, 10 M, 6 D	17	50	100	318	0,100	59%	https://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/maintenance/tables.sql?view=log
OpenCart	164	4 Y, 4 M, 3 D	46	114	292	731	0,104	47%	https://github.com/opencart/opencart/blob/master/upload/install/opencart.sql
phpBB	133	6 Y, 7 M, 10 D	61	65	611	565	0,055	82%	https://github.com/phpbb/phpbb3/blob/develop/phpBB/install/schemas/mysql_41_schema.sql
TYPO3	97	8 Y, 11 M, 0 D	10	23	122	414	0,030	76%	https://git.typo3.org/Packages/TYPO3.CMS.git/history/TYPO3_6-0:/t3lib/stddb/tables.sql

Hecate: SQL schema diff viewer

- Parses DDL files
- Creates a model for the parsed SQL elements
- Compares two versions of the same schema
- Reports on the diff performed with a variety of metrics
- Exports the transitions that occurred in XML format

https://github.com/DAINTINESS-Group/Hecate

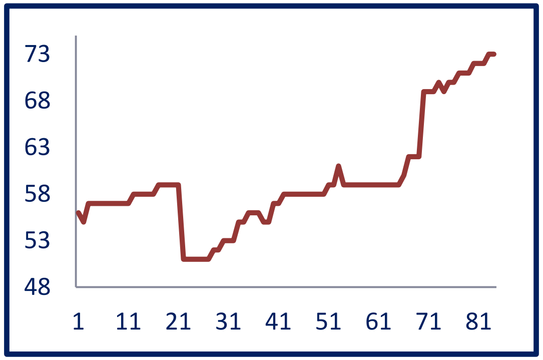
Hecate: SQL schema diff viewer

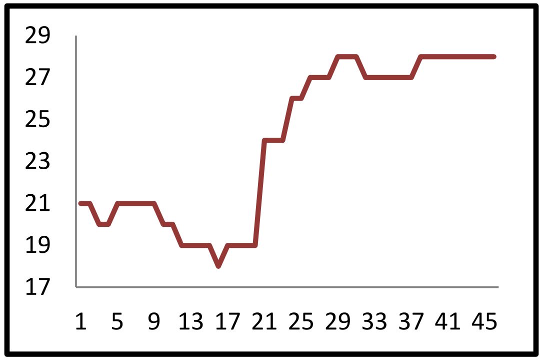


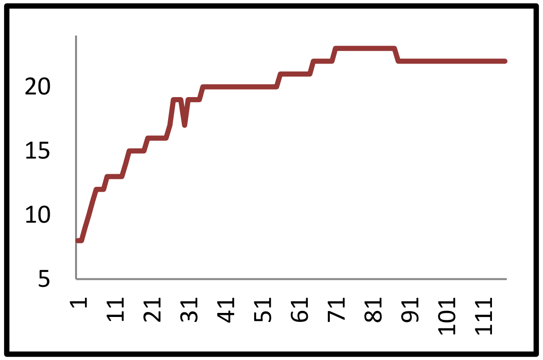
Name	Type
rev_001284.sql	
archive	
ar_comment	TINYBLOB
ar_flags	TINYBLOB
ar_minor_edit	TINYINT(1)
ar_namespace	TINYINT(2)
ar_text	MEDIUMTEXT
ar_timestamp	CHAR(14)
ar_title	VARCHAR(255)
ar_user	INT(5)
ar_user_text	VARCHAR(255)
brokenlinks	
bl_from	INT(8)
bl_to	VARCHAR(255)
cur	
image	
imagelinks	
ipblocks	
links	
math	
old	
oldimage	
random	
recentchanges	
searchindex	
site_stats	
user	
user_newtalk	

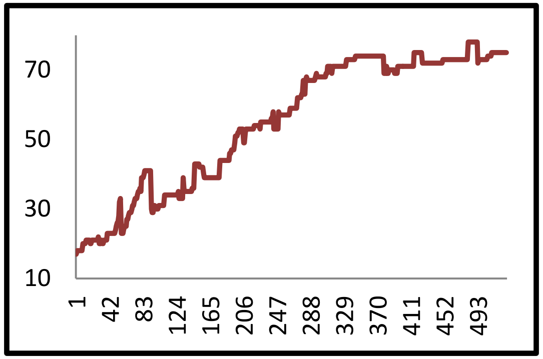
Name	Type
rev_113110.sql	
archive	
ar_comment	TINYBLOB
ar_deleted	TINYINTUNSIGNED
ar_flags	TINYBLOB
ar_len	INTUNSIGNED
ar_minor_edit	TINYINT
ar_namespace	INT
ar_page_id	INTUNSIGNED
ar_parent_id	INTUNSIGNED
ar_rev_id	INTUNSIGNED
ar_sha1	VARBINARY(32)
ar_text	MEDIUMBLOB
ar_text_id	INTUNSIGNED
ar_timestamp	BINARY(14)
ar_title	VARCHAR(255)
ar_user	INTUNSIGNED
ar_user_text	VARCHAR(255)
category	
categorylinks	
change_tag	
config	
external_user	
externallinks	
filearchive	
hitcounter	
image	
imagelinks	

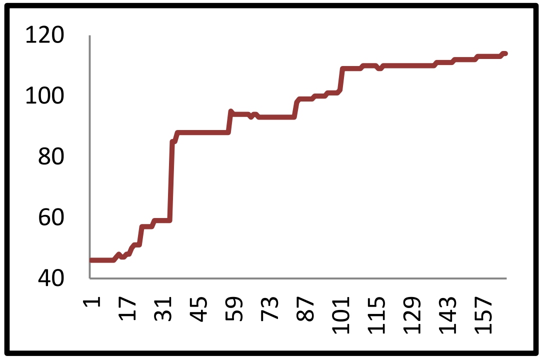
Schema Size (relations)

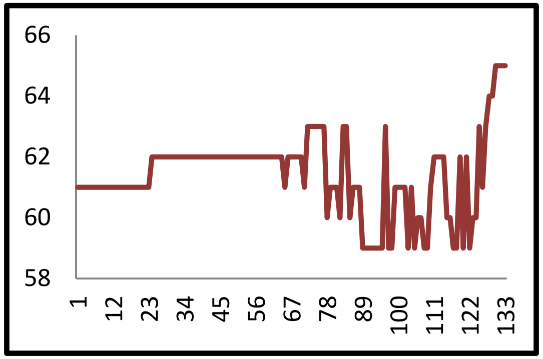


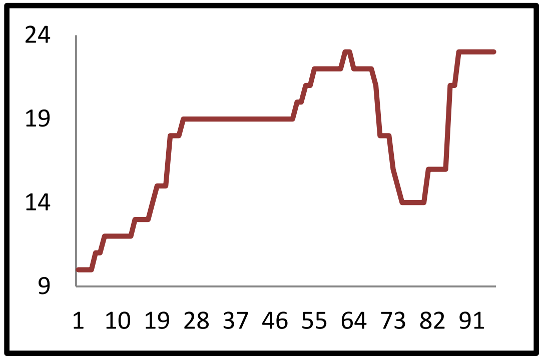


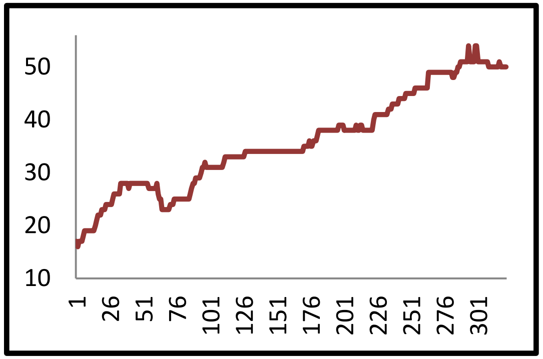




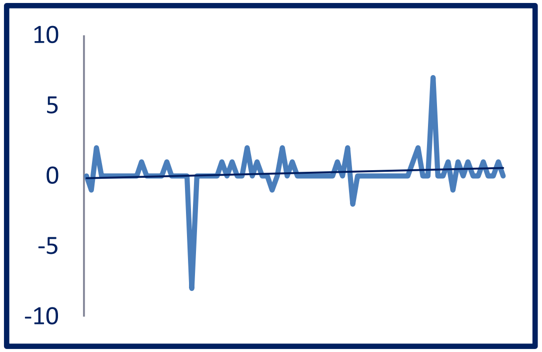


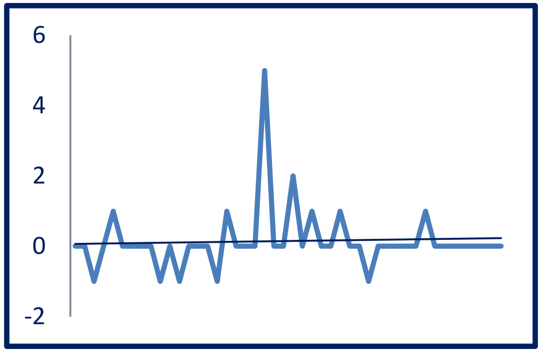


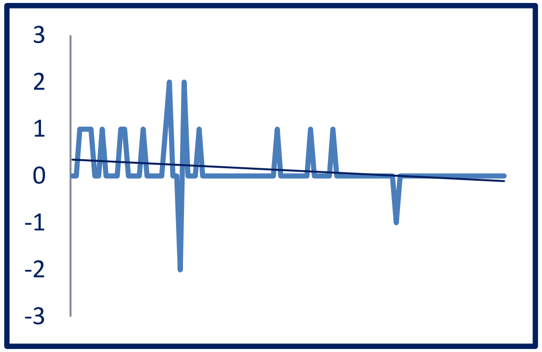


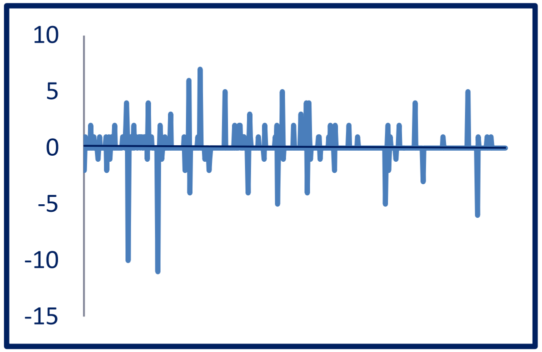


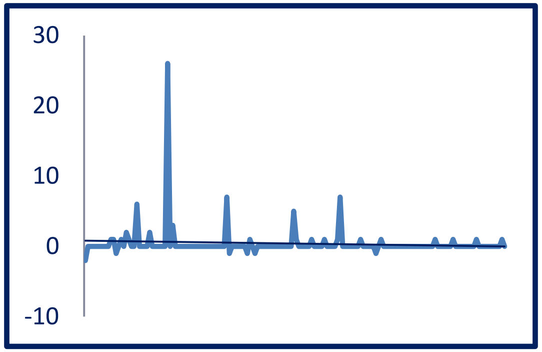
Schema Growth (diff in #tables)

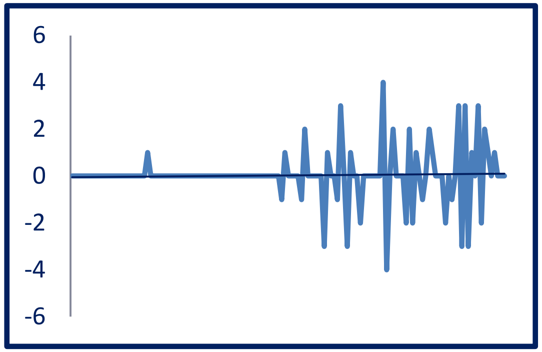


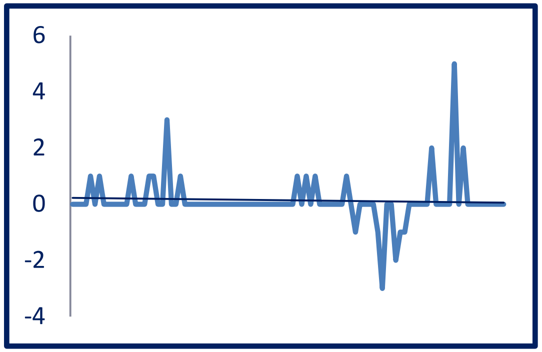


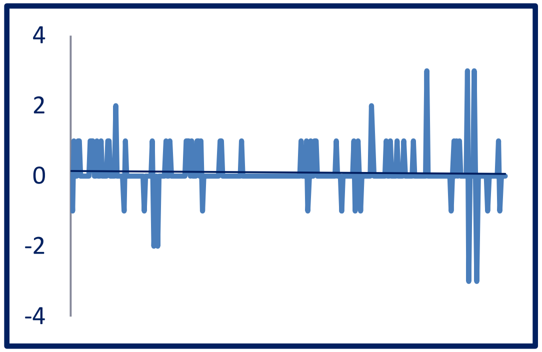






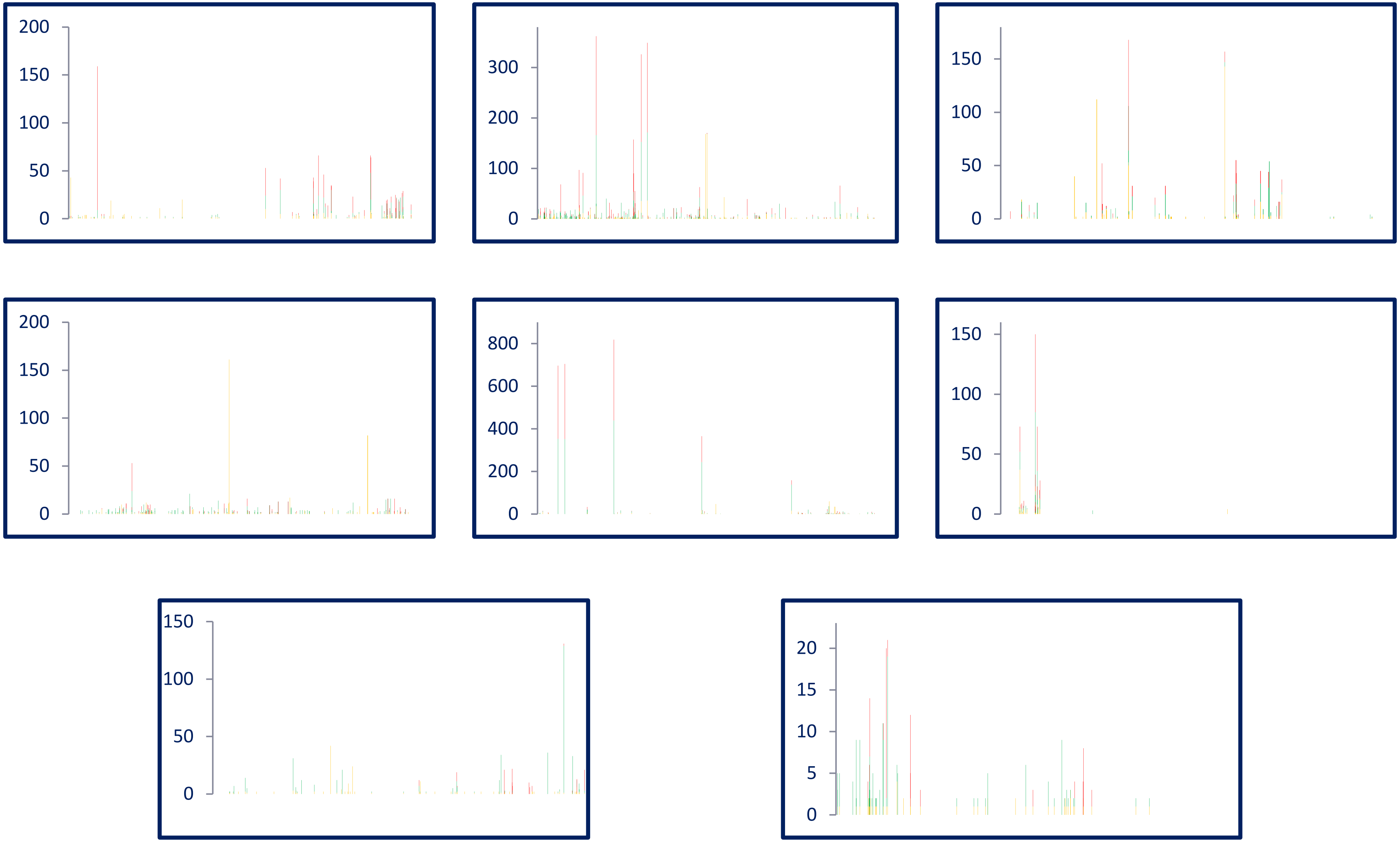






http://www.cs.uoi.gr/~pvassil/publications/2014_CAiSE/

Change over time



Main results

Schema size (#tables, #attributes) supports the assumption of a feedback mechanism

- Schema size grows over time; not continuously, but with bursts of concentrated effort
- Drops in schema size signifies the existence of perfective maintenance
- Regressive formula for size estimation holds, with a quite short memory

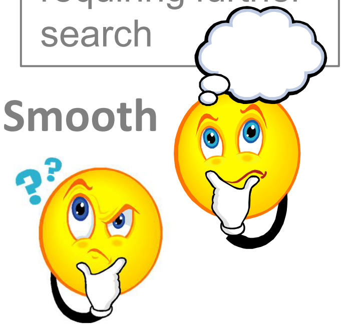
Schema Growth (diff in size between subsequent versions) is small!!

- Growth is small, smaller than in typical software
- The number of changes for each evolution step follows Zipf's law around zero
- Average growth is close (slightly higher) to zero

Patterns of change: no consistently constant behavior

- Changes reduce in density as databases age
- Change follows three patterns: **Stillness**, **Abrupt change** (up or down), **Smooth growth upwards**
- Change frequently follows **spike** patterns
- **Complexity** does **not** increase with age

Grey for results requiring further search



.. What do we see if we observe the evolution of individual tables?

http://www.cs.uoi.gr/~pvassil/publications/2015_ER

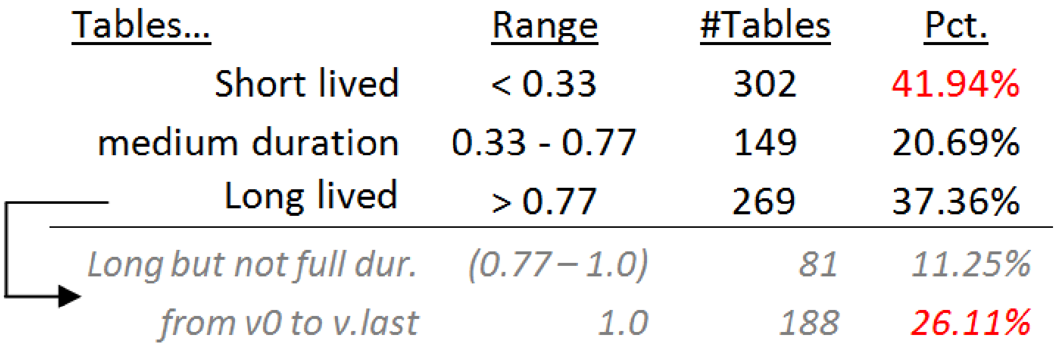
Vassiliadis, Zarras, Skoulis. **How is Life for a Table** in an Evolving Relational Schema? **Birth, Death & Everything in Between**.

To appear in **ER 2015**

OBSERVING THE EVOLUTION OF O/S DB SCHEMATA AT THE MICRO LEVEL

Statistical study of durations

- Short and long lived tables are practically equally proportioned
- Medium size durations are few!
- Long lived tables are mostly survivors (see on the right)



Tables…	Range	#Tables	Pct.
Short lived	< 0.33	302	41.94%
medium duration	0.33 - 0.77	149	20.69%
Long lived	> 0.77	269	37.36%
Long but not full dur.	*(0.77 – 1.0)*	*81*	*11.25%*
from v0 to v.last	*1.0*	*188*	*26.11%*

One of the fascinating revelations of this measurement was that there is a 26.11% fraction of tables that appeared in the beginning of the database and survived until the end.
In fact, if a table is long-lived there is a 70% chance (188 over 269 occasions) that it has appeared in the beginning of the database.

Tables are mostly thin

- On average, half of the tables (approx. 47%) are thin tables with less than 5 attributes.
- The tables with 5 to 10 attributes are approximately one third of the tables' population
- The large tables with more than 10 attributes are approximately 17% of the tables.

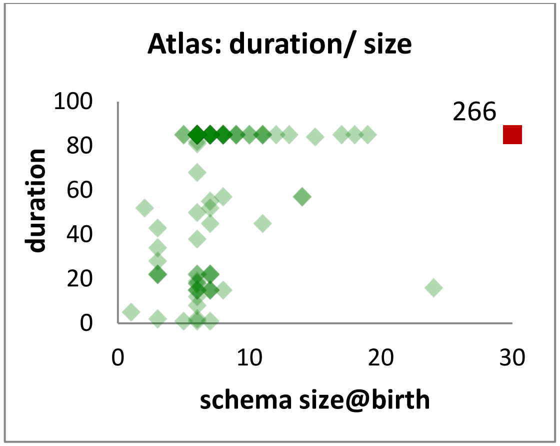
Pct of tables with num. of attributes …

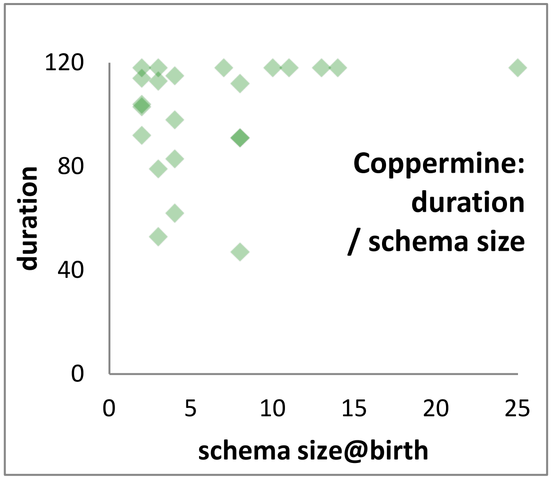
	<5	5-10	>10
atlas	10,23%	68,18%	21,59%
biosql	75,56%	24,44%	0,00%
coppermine	52,17%	30,43%	17,39%
ensembl	54,84%	38,06%	7,10%
mediawiki	61,97%	19,72%	18,31%
phpbb	40,00%	44,29%	15,71%
typo3	21,88%	31,25%	46,88%
opencart	57,20%	33,05%	9,75%
Average	46,73%	36,18%	17,09%

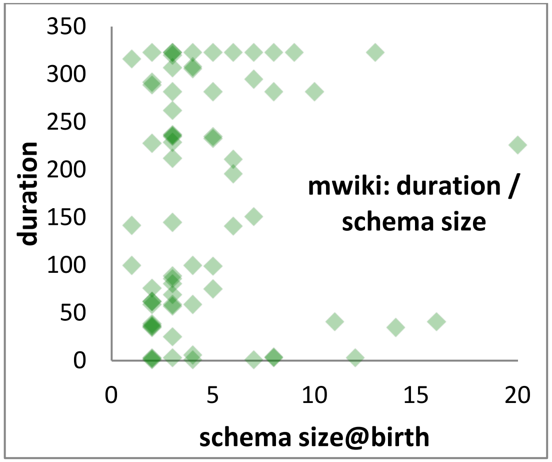
The $Gamma$ Γ Pattern: "if you 're wide, you survive"

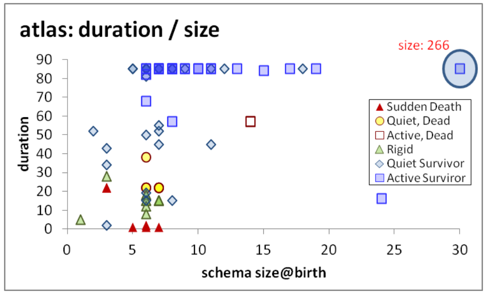
- The $Gamma$ phenomenon:
  - tables with small schema sizes can have arbitrary durations, //small size does not determine duration
  - larger size tables last long

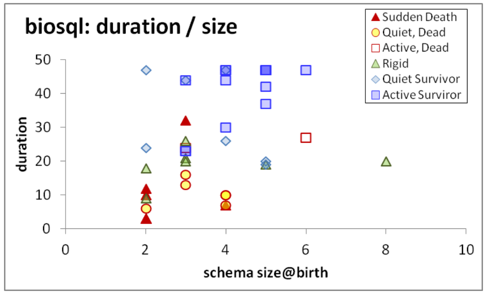
- Observations:
  - whenever a table exceeds the critical value of 10 attributes in its schema, its chances of surviving are high.
  - in most cases, the large tables are created early on and are not deleted afterwards.

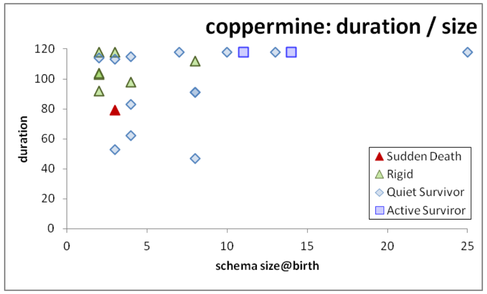


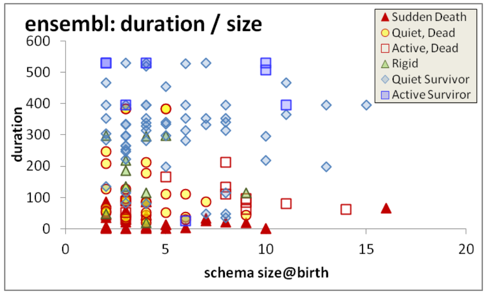


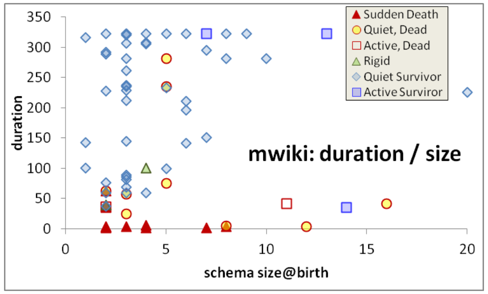


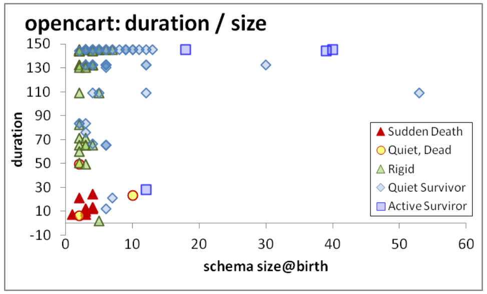


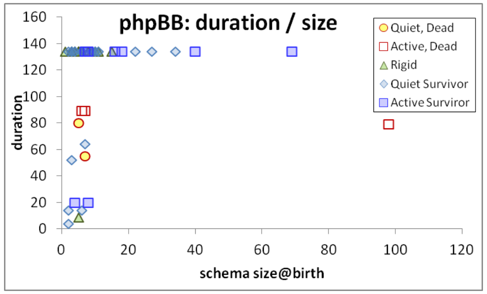


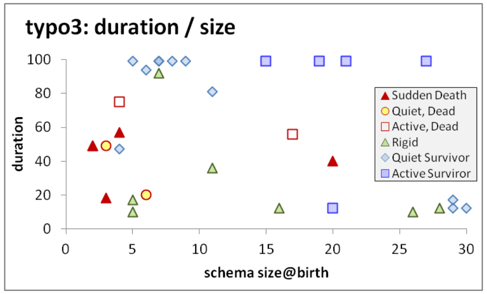










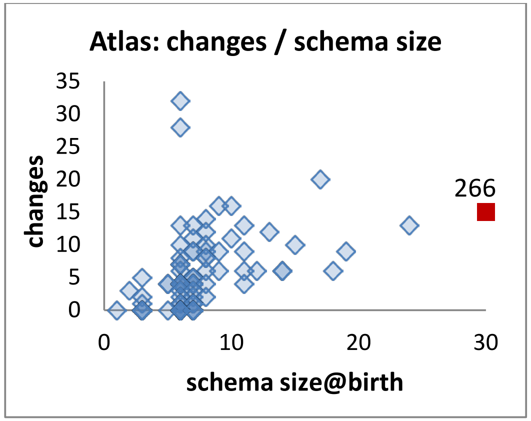


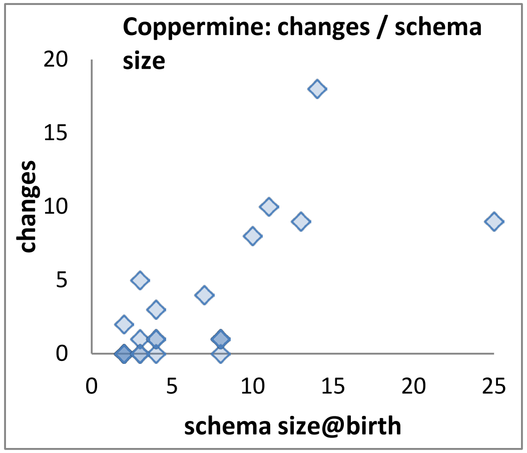
These exceptions are due to the fact that they do not exceed 10 attributes

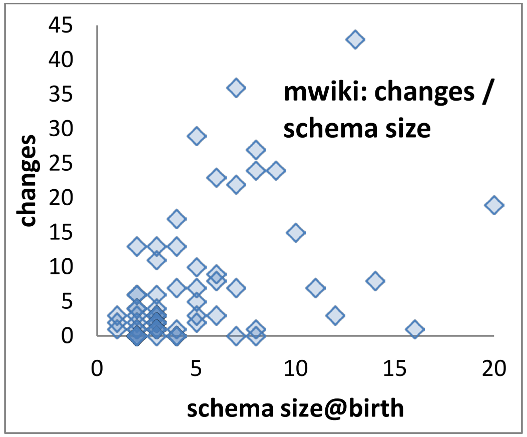
The Comet Pattern

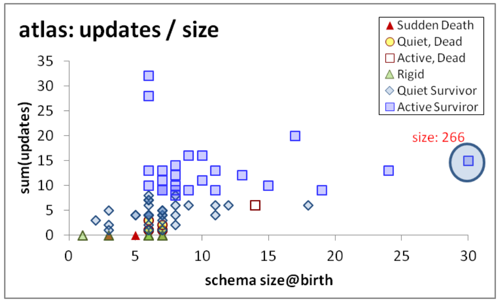
“Comet “ for change over schema size with:

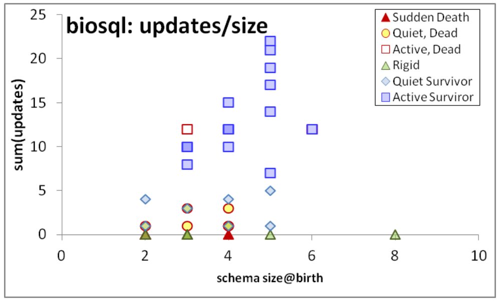
- a large, dense, nucleus cluster close to the beginning of the axes, denoting small size and small amount of change,
- medium schema size tables typically demonstrating medium to large change
- The tables with the largest amount of change are typically tables whose schema is on average one standard deviation above the mean
- wide tables with large schema sizes demonstrating small to medium (typically around the middle of the y-axis) amount of change.

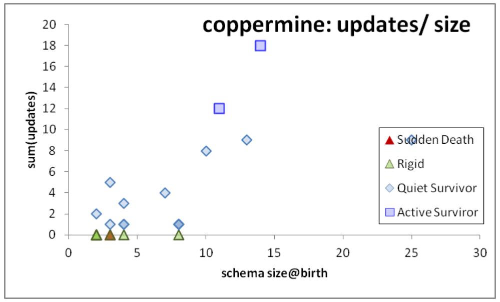


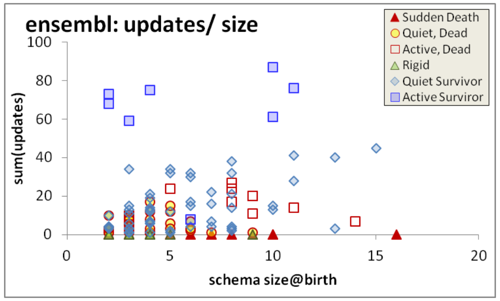


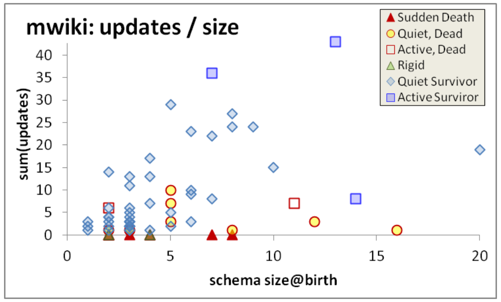


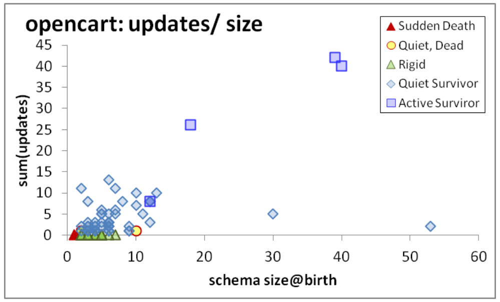


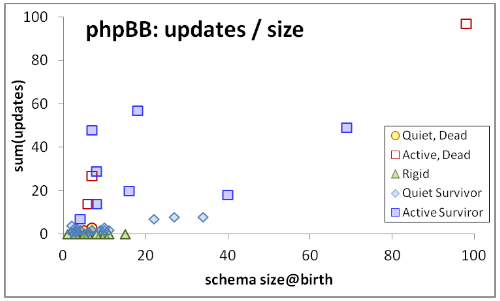


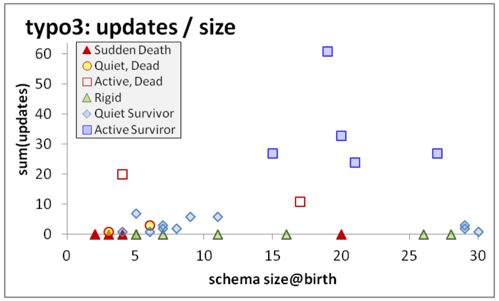




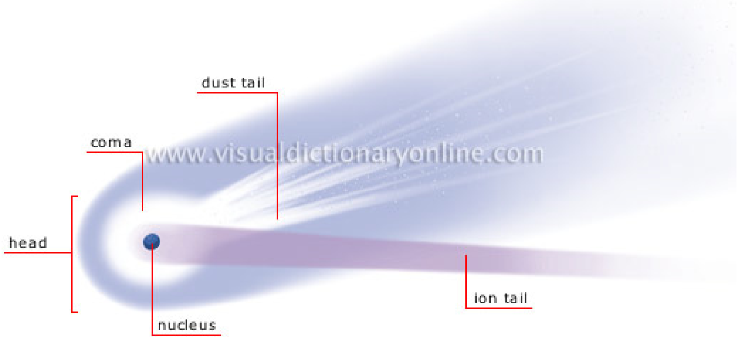








http://visual.merriam-webster.com/astronomy/celestial-bodies/comet.php



Comets have two tails: White one is made of comet dust particles. Blue one is made of electrically charged gas.

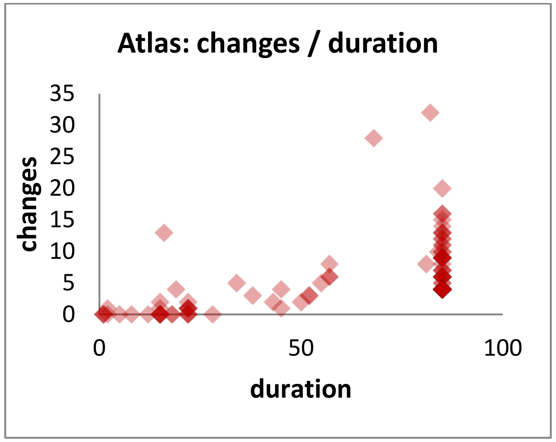
The coma is the cloud of comet dust particles surrounding the nucleus.

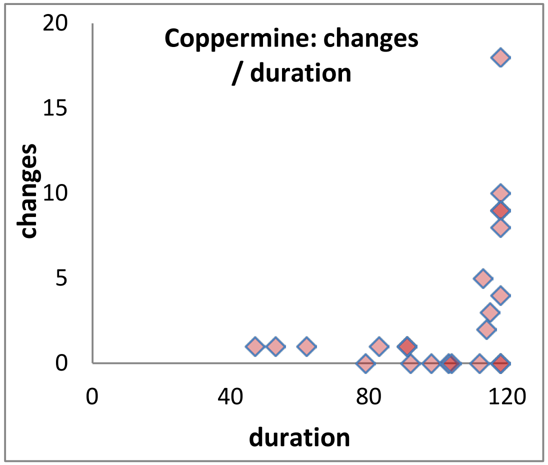
Nucleus is solid, icy heart of comet, inside the cloud of the coma

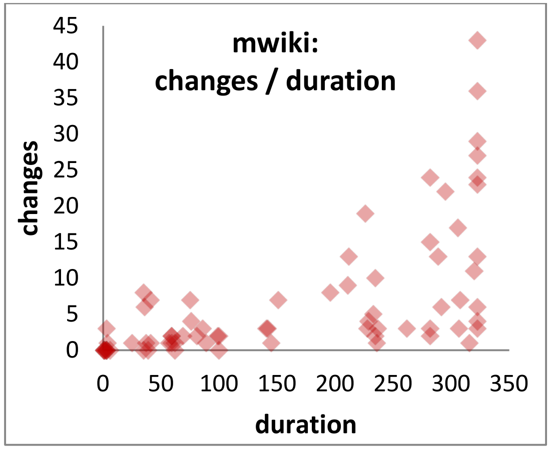
http://spaceplace.nasa.gov/comet-nucleus/en/

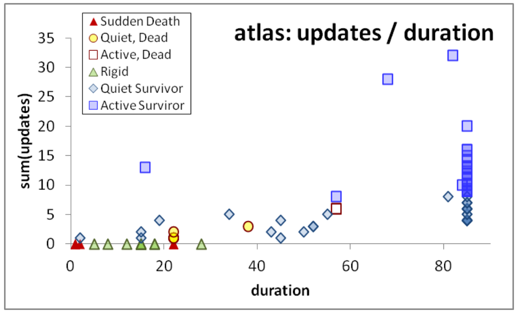
The inverse $Gamma$ ⅃ pattern

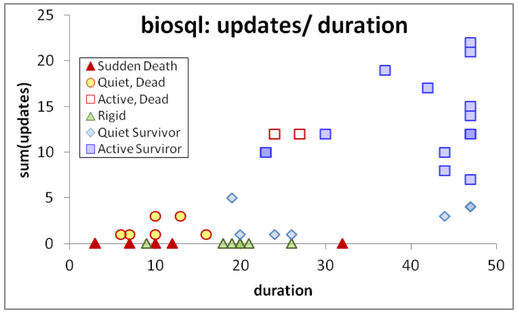
- The correlation of change and duration is as follows:
  - small durations come necessarily with small change,
  - large durations come with all kinds of change activity and
  - medium sized durations come mostly with small change activity (inverse $Gamma$).

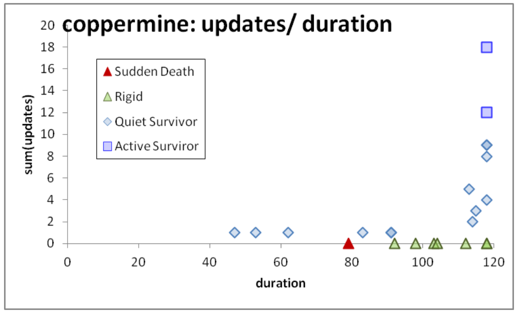


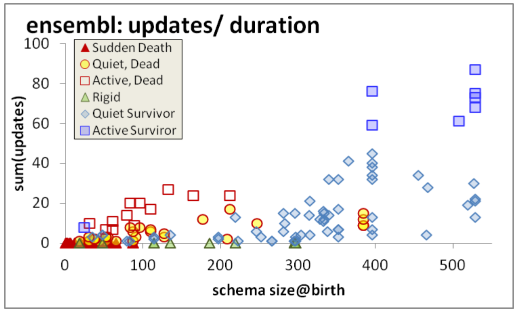


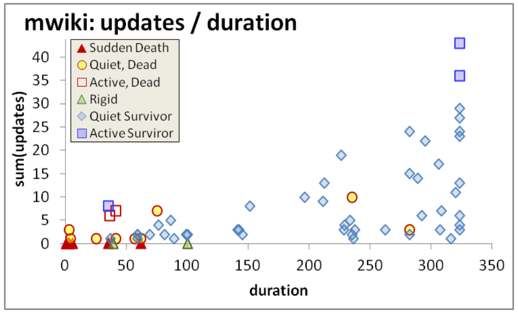


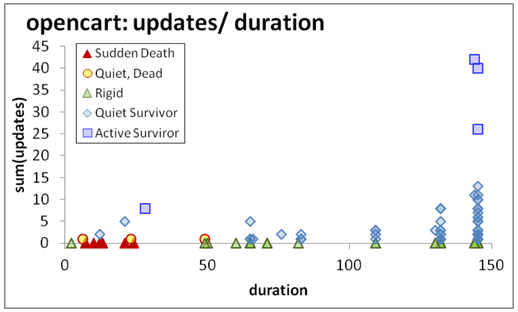

atlas: updates / duration

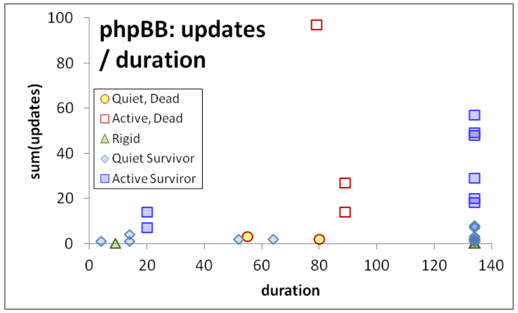

biosql: updates/ duration

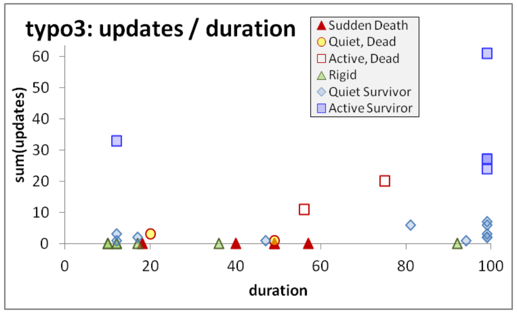

coppermine: updates/ duration


ensembl: updates/ duration


mwiki: updates / duration


opencart: updates/ duration


phpBB: updates / duration


typo3: updates / duration

Quiet tables rule, esp. for mature db's

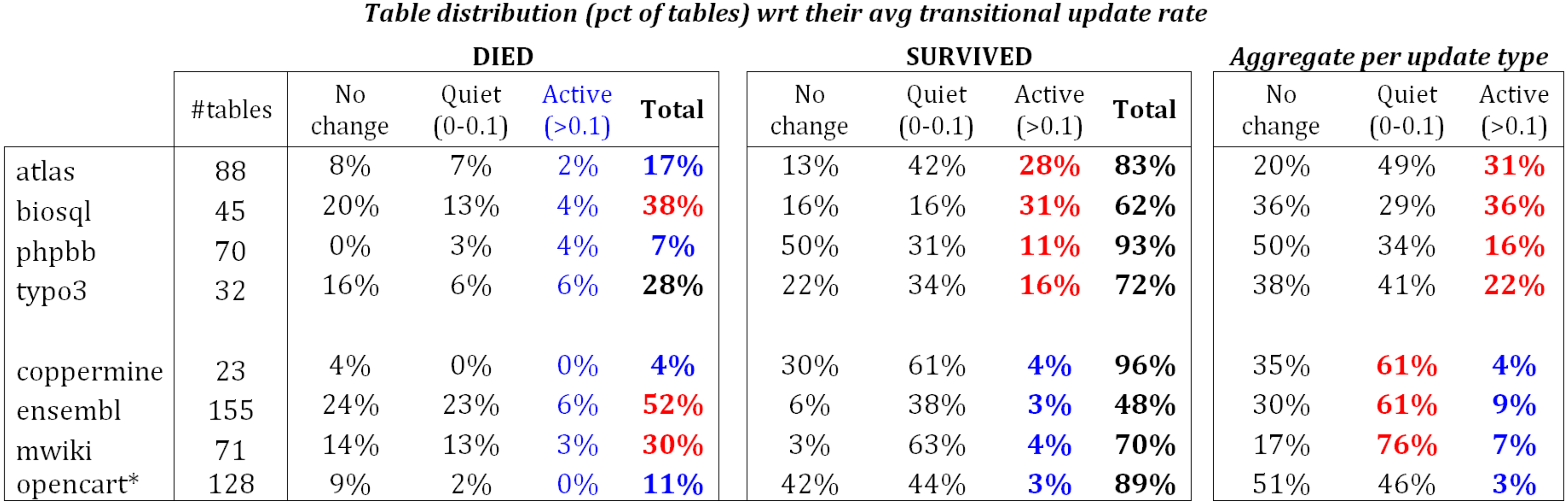


Table distribution (pct of tables) wrt their avg transitional update rate

		DIED				**SURVIVED**				***Aggregate per update type***		
	#tables	No change	Quiet (0-0.1)	Active (>0.1)	**Total**	No change	Quiet (0-0.1)	Active (>0.1)	**Total**	No change	Quiet (0-0.1)	Active (>0.1)
atlas	88	8%	7%	2%	**17%**	13%	42%	**28%**	**83%**	20%	49%	**31%**
biosql	45	20%	13%	4%	**38%**	16%	16%	**31%**	**62%**	36%	29%	**36%**
phpbb	70	0%	3%	4%	**7%**	50%	31%	**11%**	**93%**	50%	34%	**16%**
typo3	32	16%	6%	6%	**28%**	22%	34%	**16%**	**72%**	38%	41%	**22%**
coppermine	23	4%	0%	0%	**4%**	30%	61%	**4%**	**96%**	35%	**61%**	**4%**
ensembl	155	24%	23%	6%	**52%**	6%	38%	**3%**	**48%**	30%	**61%**	**9%**
mwiki	71	14%	13%	3%	**30%**	3%	63%	**4%**	**70%**	17%	**76%**	**7%**
opencart*	128	9%	2%	0%	**11%**	42%	44%	**3%**	**89%**	51%	46%	**3%**

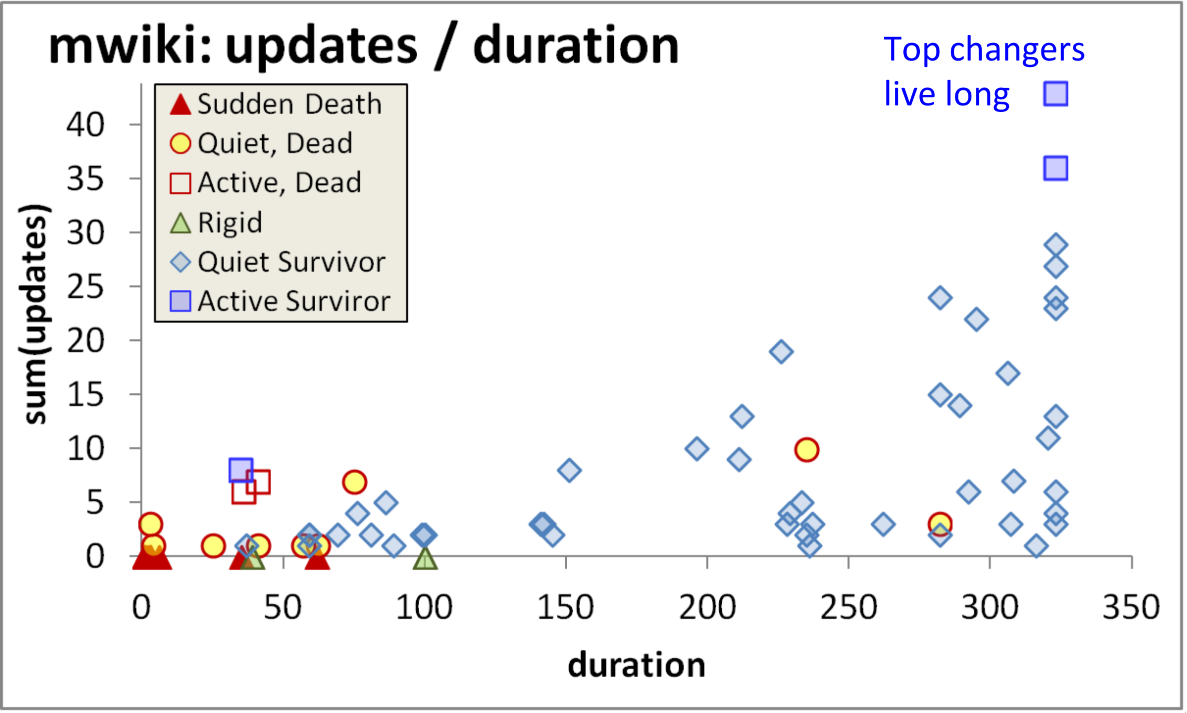
Non-survivors

- Sudden deaths mostly
- Quiet come ~ close
- Too few active

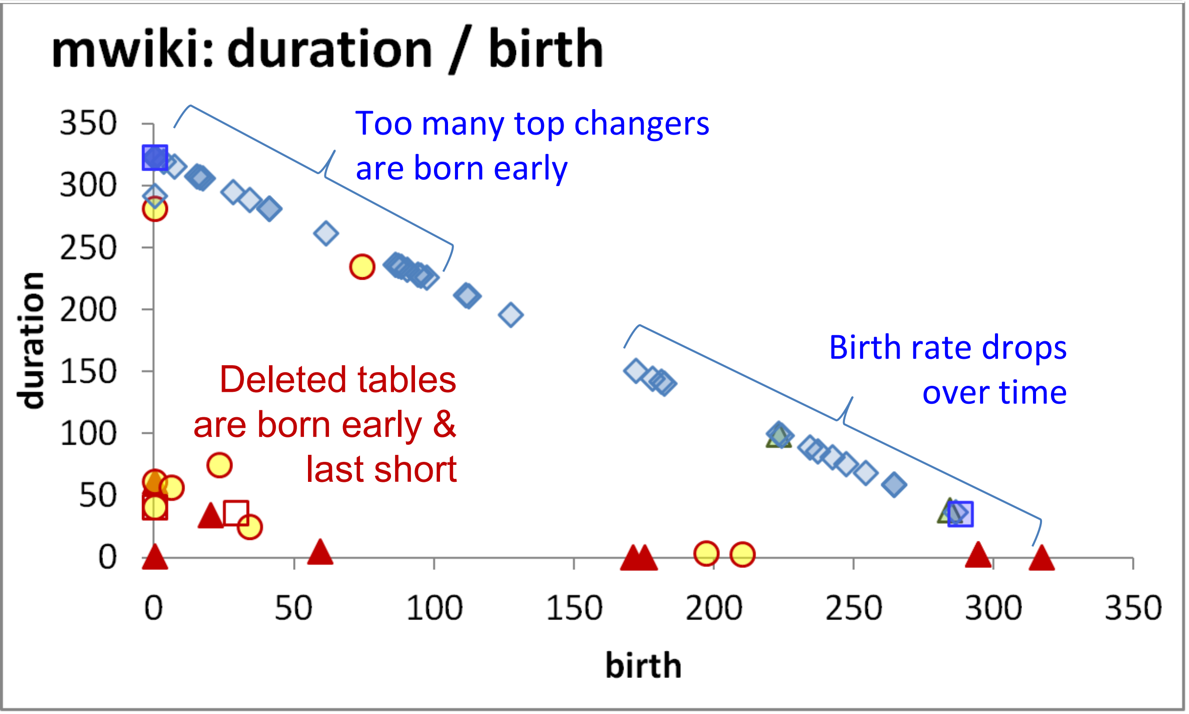
Survivors

- Quiet tables rule
- Rigid and active then
- Active mostly in "new" db's

Mature DB's: the pct of active tables drops significantly



Top changers live long



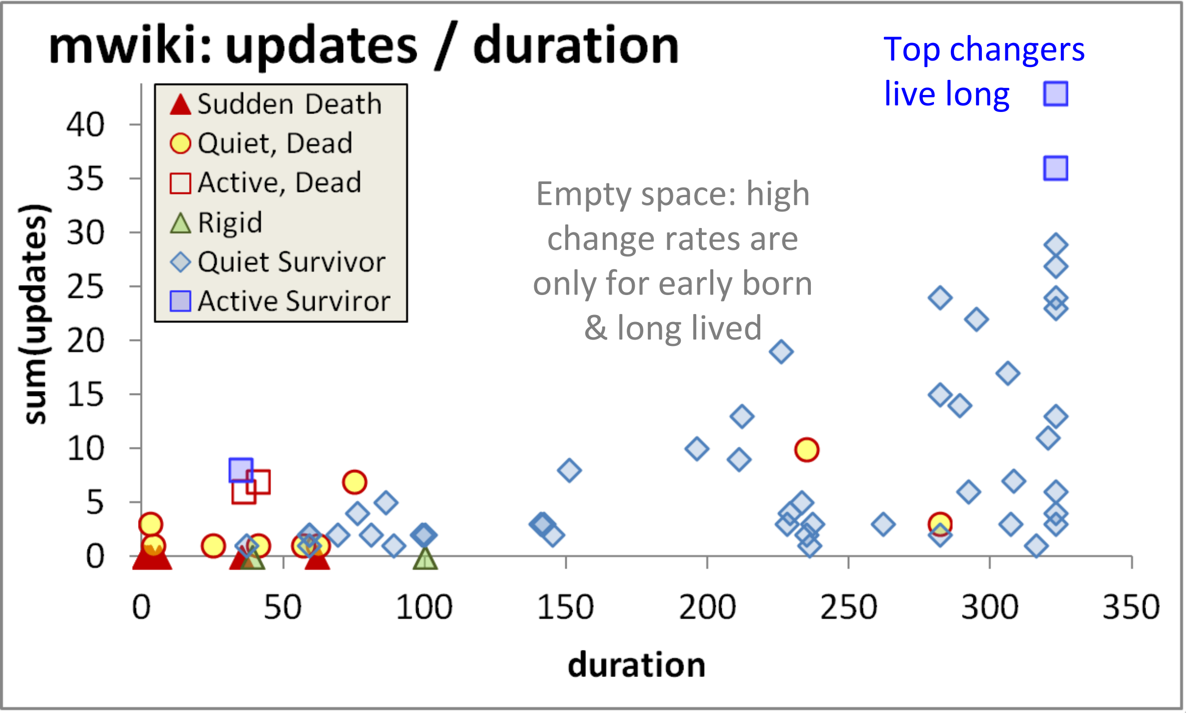
Too many top changers are born early

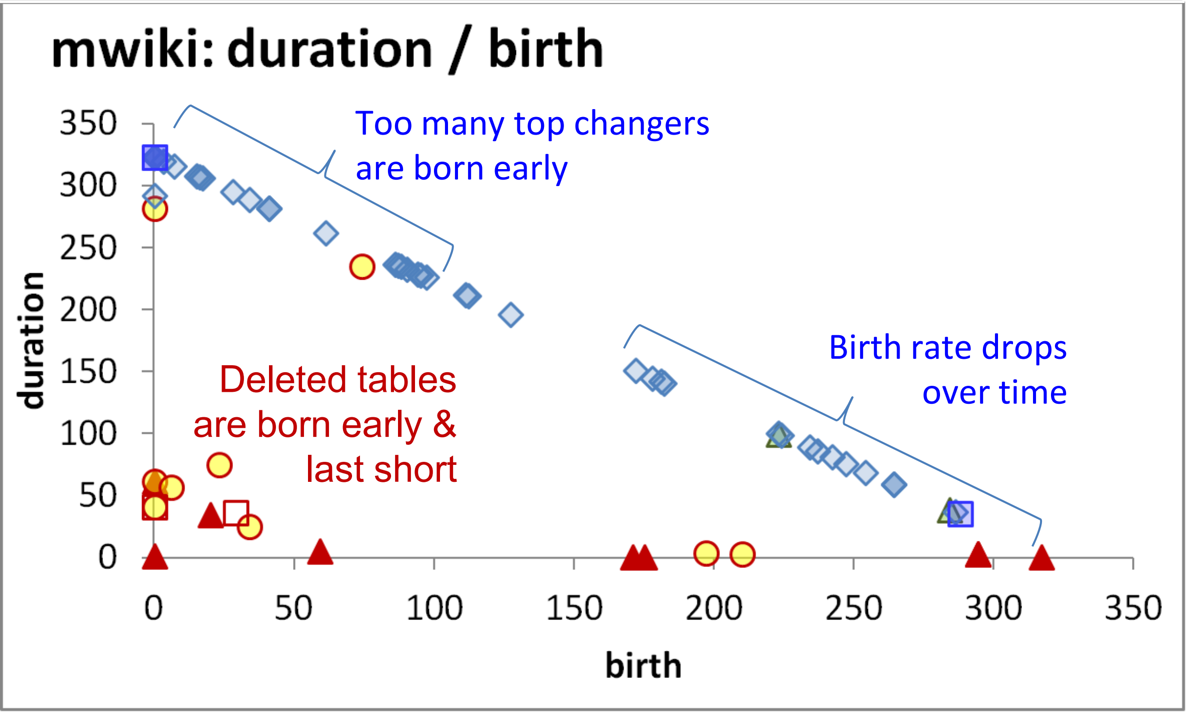
Birth rate drops over time

Deleted tables are born early & last short

Longevity and update activity correlate !!

- Remember: top changers are defined as such wrt ATU (AvgTrxnUpdate), not wrt sum(changes)
- Still, they dominate the sum(changes) too! (see top of inverse Γ)
- See also upper right blue part of diagonal: too many of them are born early and survive => live long!

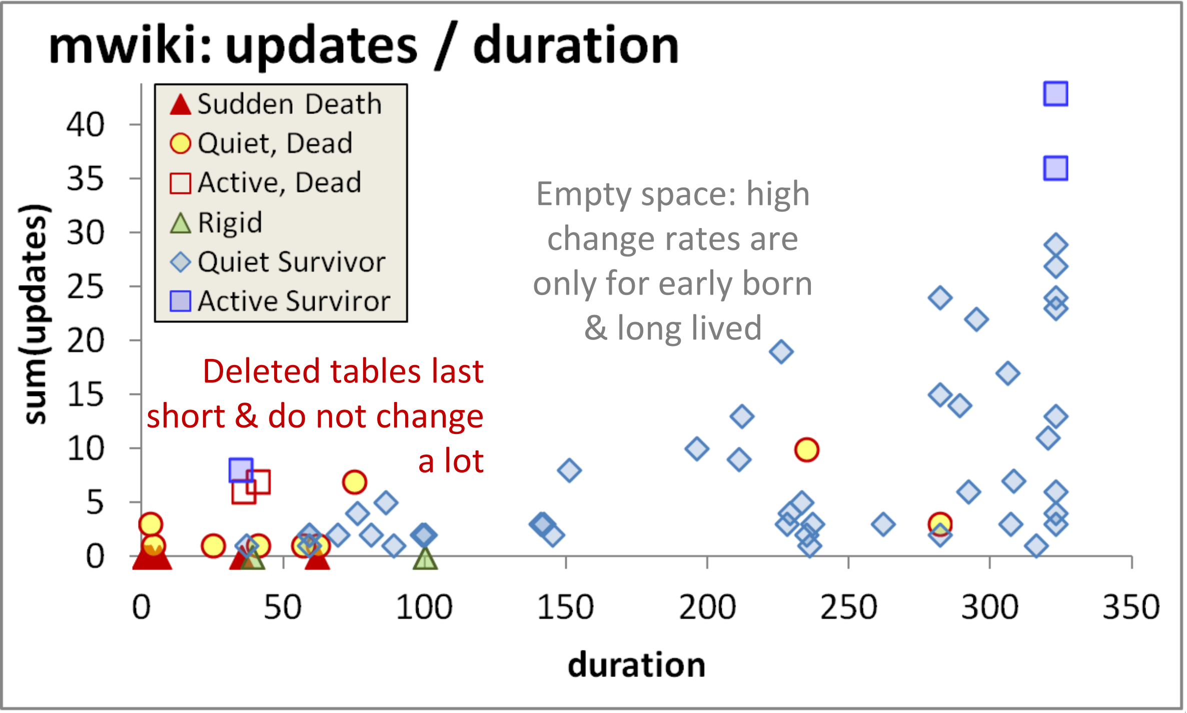


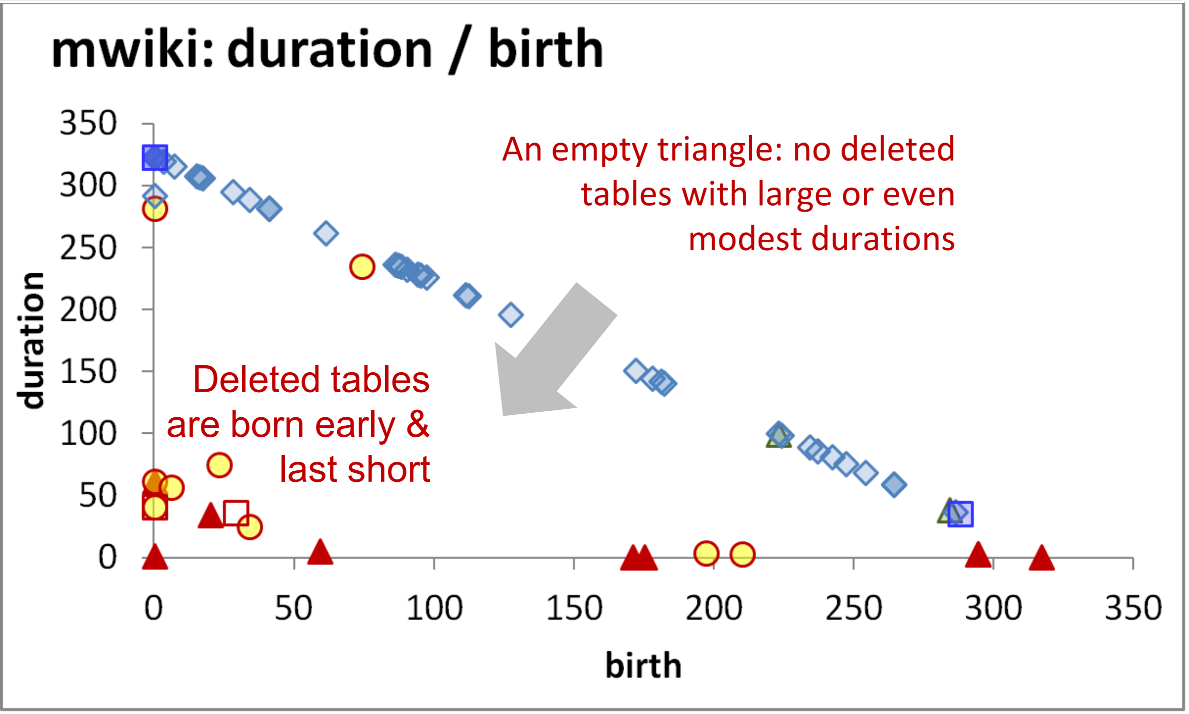


Longevity and update activity correlate !!

The few top-changers (in terms of ATU) ….

- are long lived,
- typically come from the early versions of the database
- due to the combination of high ATU and duration => they have high total amount of change, and,
- frequently start with medium schema sizes (not shown here)

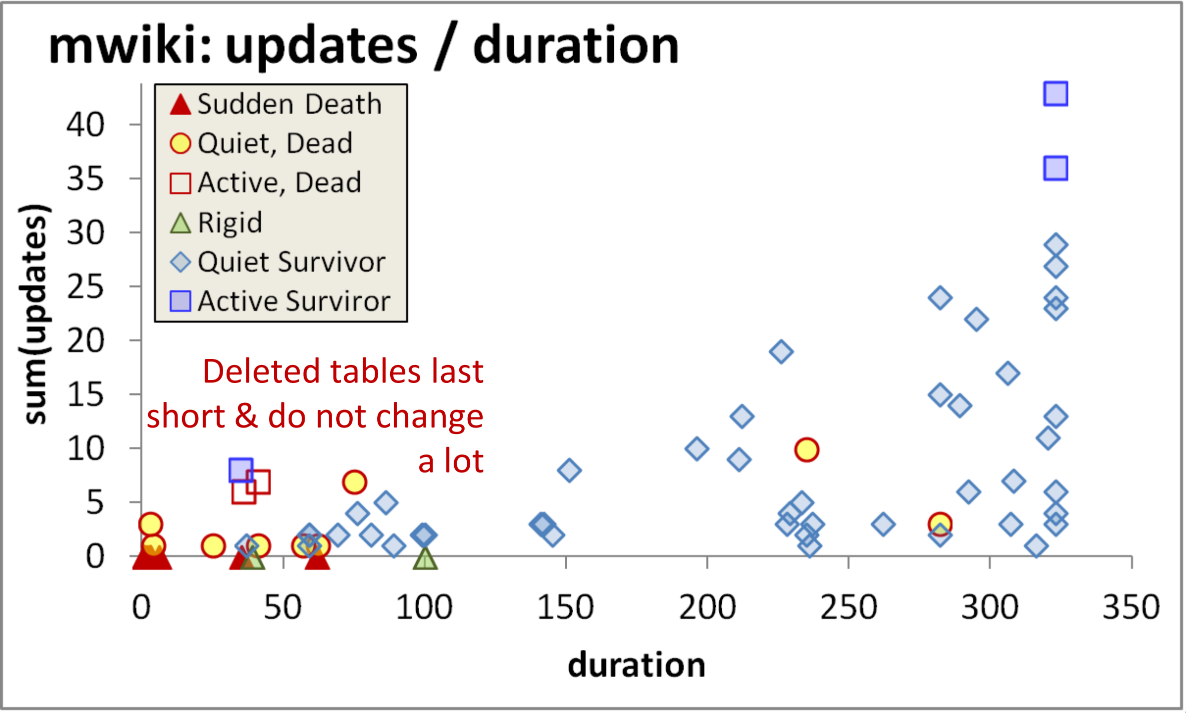




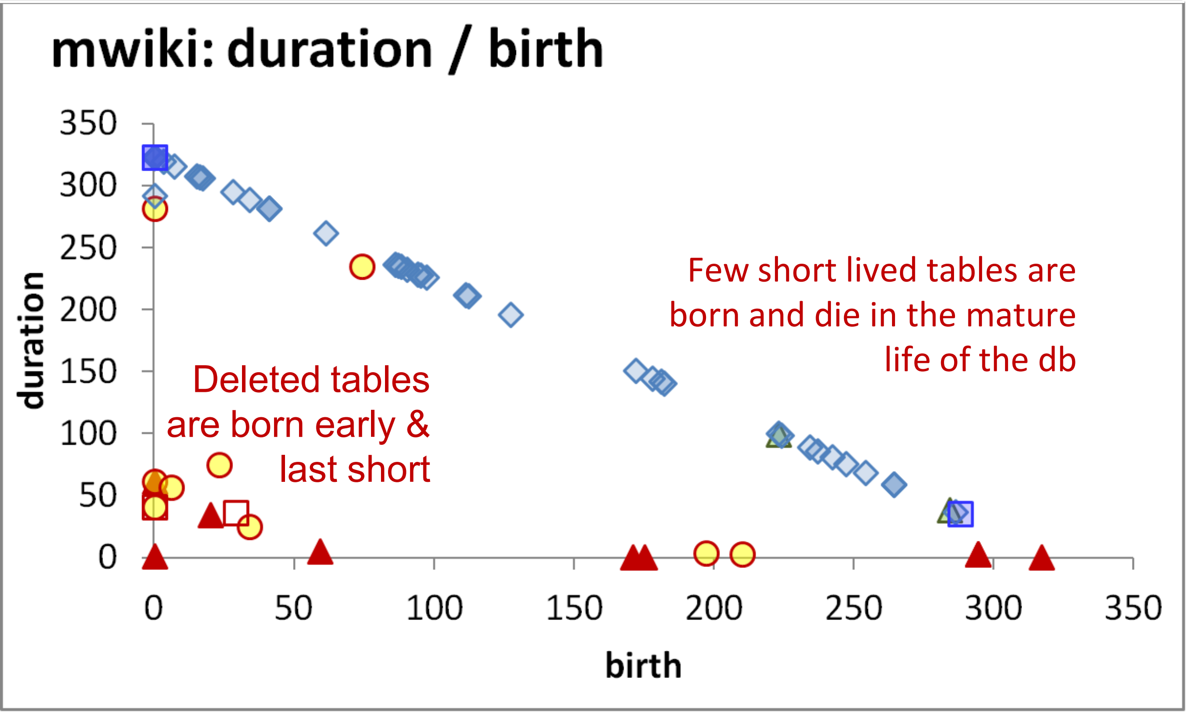
Die young and suddenly

- There is a very large concentration of the deleted tables in a small range of newly born, quickly removed, with few or no updates…
- …. resulting in very low numbers of removed tables with medium or long durations (empty triangle).

Die young and suddenly



Deleted tables last short & do not change a lot

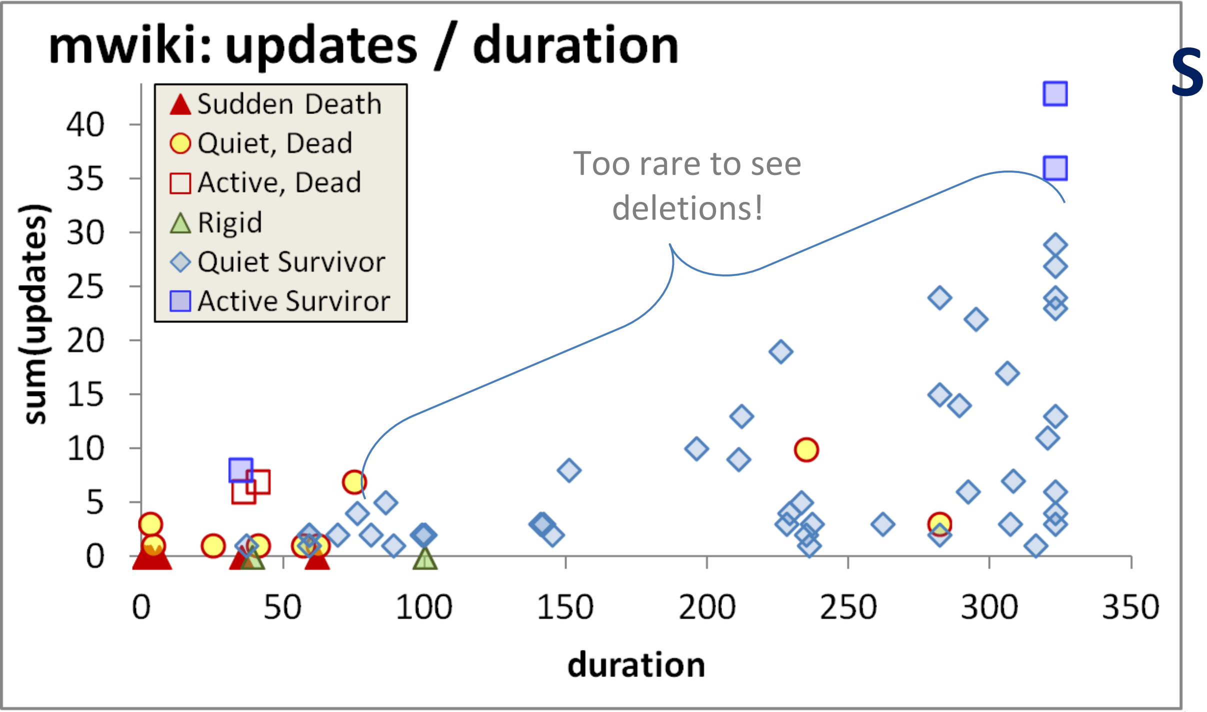


Few short lived tables are born and die in the mature life of the db

Deleted tables are born early & last short

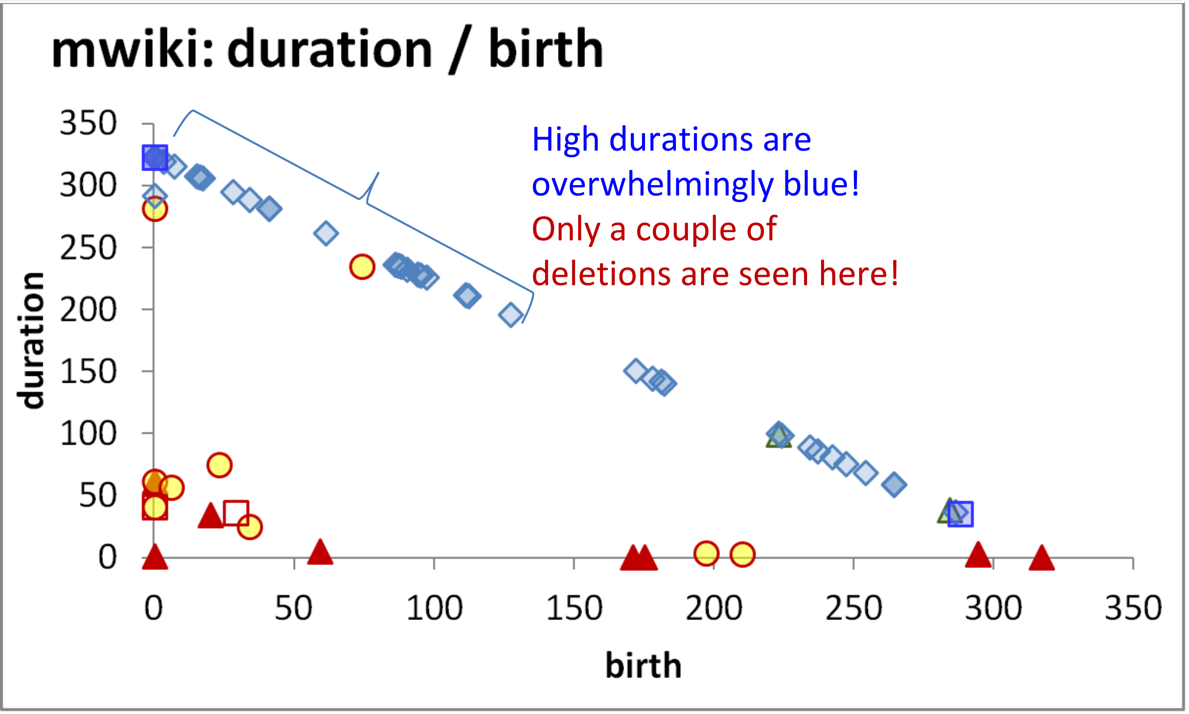
[Early life of the db] There is a very large concentration of the deleted tables in a small range of newly born, quickly removed, with few or no updates, resulting in very low numbers of removed tables with medium or long durations.

[**Mature db**] After the early stages of the databases, we see the birth of tables who eventually get deleted, but they mostly come with very small durations and sudden deaths.



mwiki: updates / duration

Too rare to see deletions!



mwiki: duration / birth

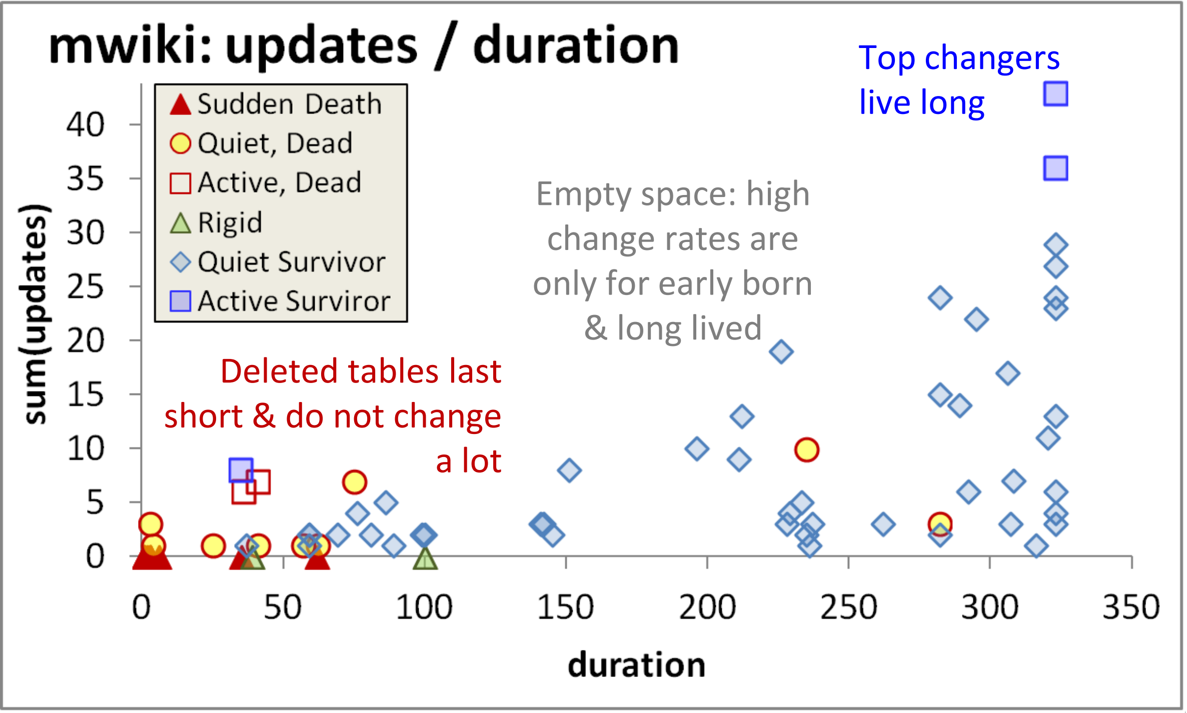
High durations are overwhelmingly blue!

Only a couple of deletions are seen here!

Survive long enough & you ‘re probably safe

It is quite rare to see tables being removed at old age

Typically, the area of high duration is overwhelmingly inhabited by survivors (although each data set comes with a few such cases)!

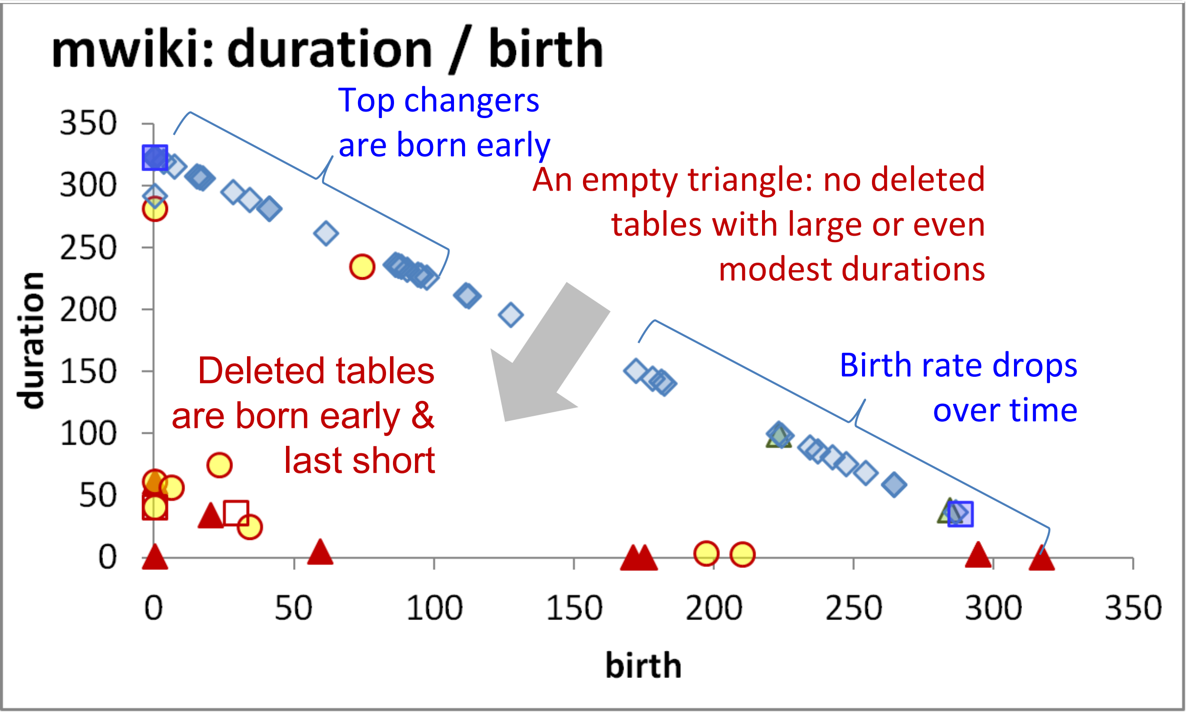


mwiki: updates / duration

Top changers live long

Empty space: high change rates are only for early born & long lived

Deleted tables last short & do not change a lot



mwiki: duration / birth

Top changers are born early

An empty triangle: no deleted tables with large or even modest durations

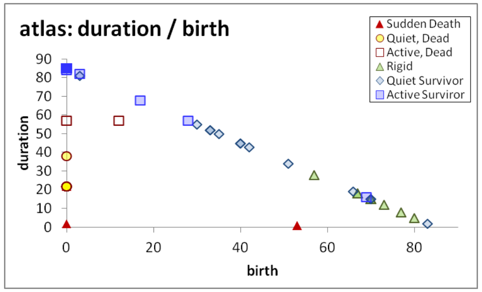
Deleted tables are born early & last short

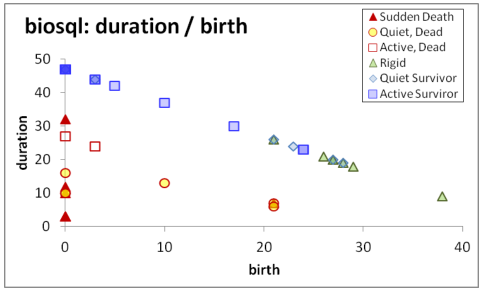
Birth rate drops over time

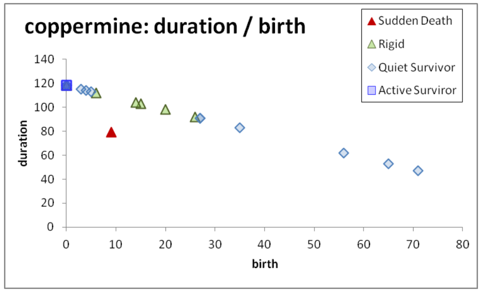
All in one

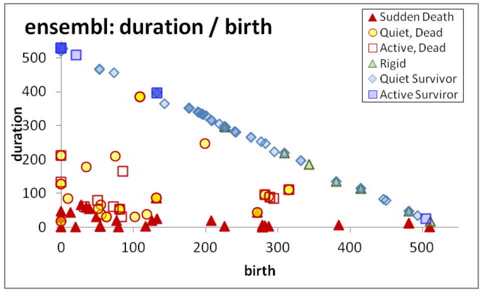
- Early stages of the database life are more "active" in terms of births, deaths and updates, and have higher chances of producing deleted tables.

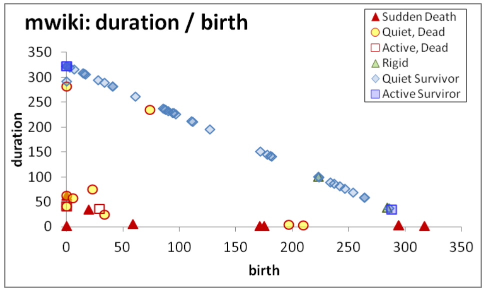
- After the first major restructuring, the database continues to grow; however, we see much less removals, and maintenance activity becomes more concentrated and focused.

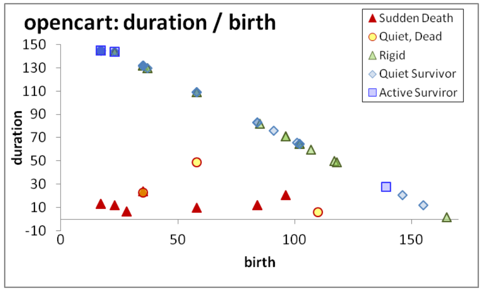

atlas: duration / birth
Sudden Death
Quiet, Dead
Active, Dead
Rigid
Quiet Survivor
Active Surviror
duration
90
80
70
60
50
40
30
20
10
0
0
20
40
60
80
birth

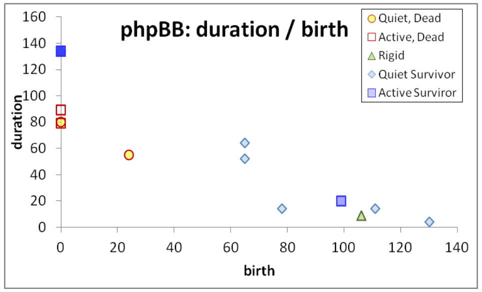

biosql: duration / birth
Sudden Death
Quiet, Dead
Active, Dead
Rigid
Quiet Survivor
Active Surviror
duration
50
40
30
20
10
0
0
10
20
30
40
birth

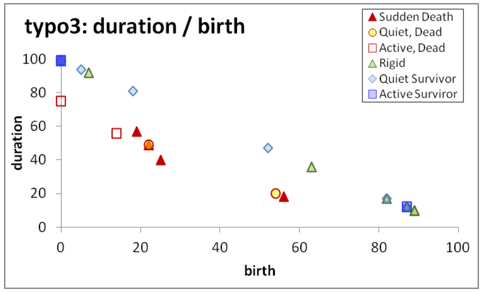

coppermine: duration / birth
Sudden Death
Rigid
Quiet Survivor
Active Surviror
duration
140
120
100
80
60
40
20
0
0
10
20
30
40
50
60
70
80
birth


ensembl: duration / birth
Sudden Death
Quiet, Dead
Active, Dead
Rigid
Quiet Survivor
Active Surviror
duration
500
400
300
200
100
0
0
100
200
300
400
500
birth

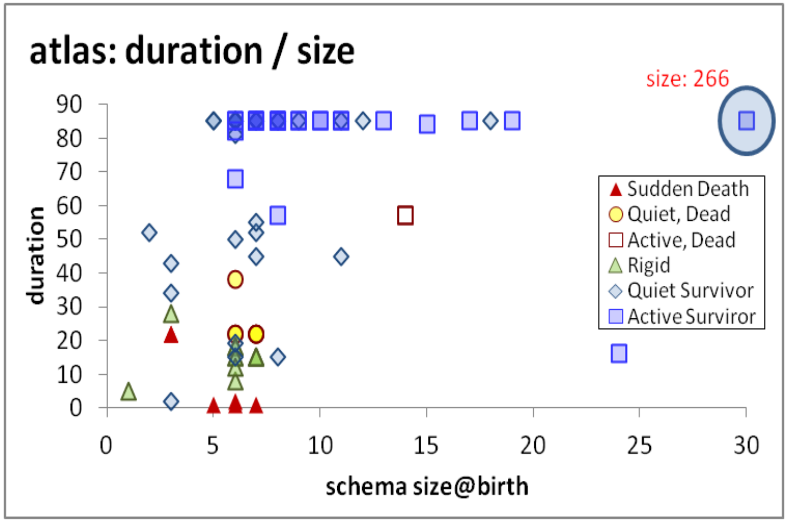

mwiki: duration / birth
Sudden Death
Quiet, Dead
Active, Dead
Rigid
Quiet Survivor
Active Surviror
duration
350
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
birth

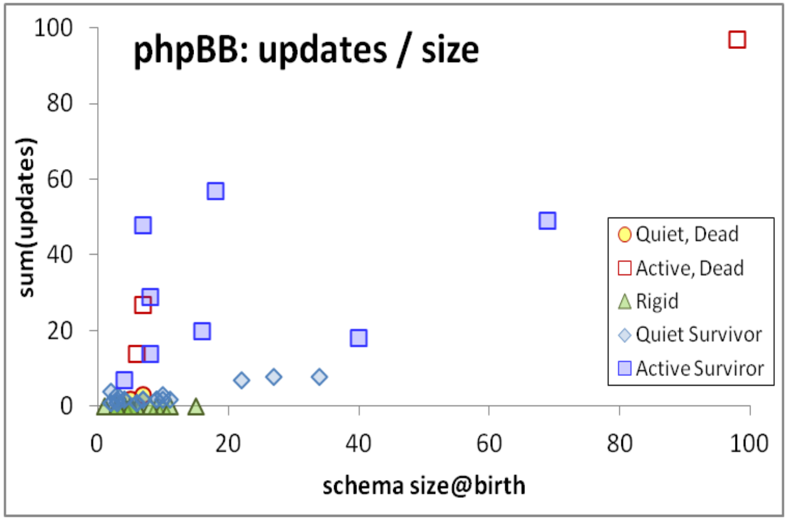

opencart: duration / birth
Sudden Death
Quiet, Dead
Rigid
Quiet Survivor
Active Surviror
duration
150
130
110
90
70
50
30
10
-10
0
50
100
150
birth


phpBB: duration / birth
Quiet, Dead
Active, Dead
Rigid
Quiet Survivor
Active Surviror
duration
160
140
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140
birth

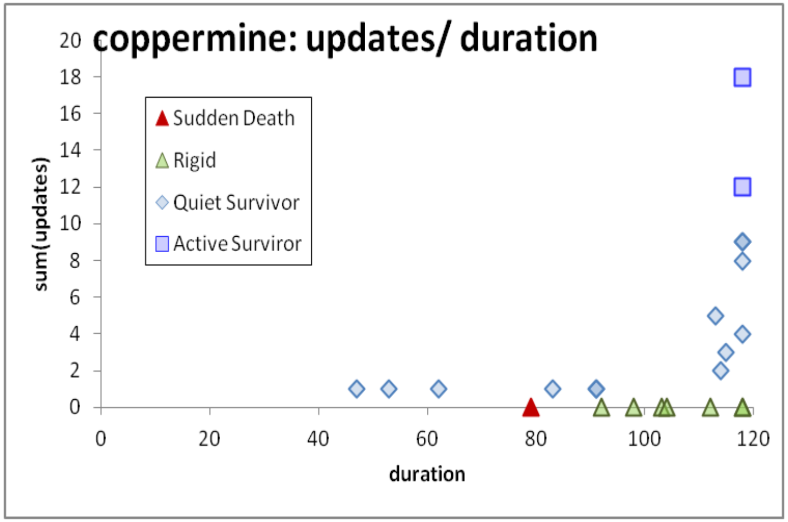

typo3: duration / birth
Sudden Death
Quiet, Dead
Active, Dead
Rigid
Quiet Survivor
Active Surviror
duration
100
80
60
40
20
0
0
20
40
60
80
100
birth

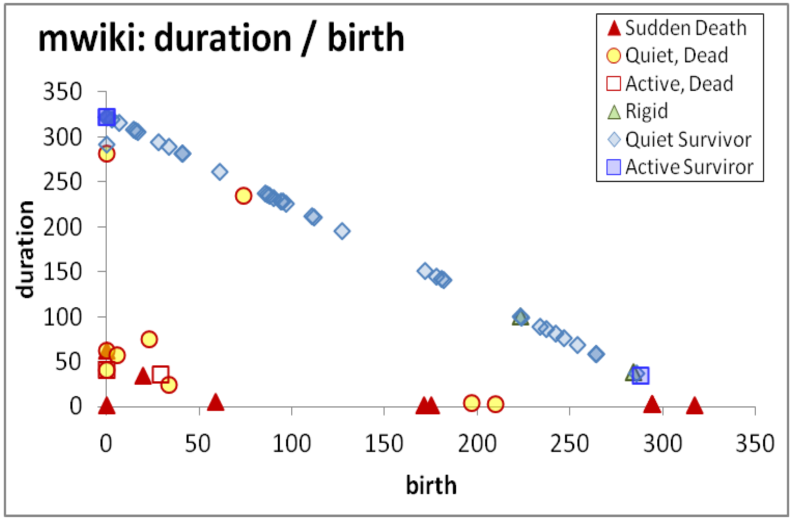
$Gamma$ Comet





Inv. $Gamma, Empty Triangle





Roadmap

- Evolution of views
- Data warehouse Evolution
- A case study (if time)
- Impact assessment in ecosystems
- Empirical studies concerning database evolution
- **Open Issues and discussions**

Where we stand

Open issues

… and discussions …

OPEN ISSUES

Where we stand

- We know to handle evolving materialized views
- And we also know to handle simple cases of DW evolution

- We have started some work on handling evolving ecosystems
- We have a first glimpse of the mechanics of database evolution

svn/git for db schemata

- The versioning tale says: keep the history of previous schemata available, as this can allow the automation of query/application migration/forward-engineering and the translation of old data to a new structure.
- When it comes to software, svn/git paradigm is the undisputed champion:
  - You make branches for concurrent development
  - Collisions are automatically detected
  - Different versions can be merged
  - You can refer to a particular version of the code easily
- How does this apply to databases and application development for databases?
- Is it really worth the trouble?

Schema curation and preservation

- Data curation and preservation is a very large topic on its own
- If we focus only at the schema part, and assuming we want to support history management for database schemata, how do we implement it?
- SMO’s can be the key for altering a db schema in a way that history can be replayed backwards / forward
  - Catch: meta information and functional dependencies are key to these methods. Need to pay the price for them.
- But how can we handle the data efficiently then?

Are there “laws” of schema evolution?

- Collect more test cases
- Tools for the automation of the process
  - Extract changes & verify their correctness (what happened)
  - Link changes to expressed user req’s / bugs / … (why it happened & by whom)
  - Extract sub-histories of focused maintenance (how it happened & when)
  - Co-change of schema and code (what is affected in the code)
  - Visualization
- Consolidate the fundamental laws that govern evolution && forecast it (what will change)

Management of ecosystems' evolution

- Can we find these constructs that are most sensitive to evolution?
  - Metrics for sensitivity to evolution?

- Automation of the reaction to changes
  - self-monitoring
  - impact prediction
  - auto-regulation (policy determination)
  - self-repairing

Current trends in data management

- How will the area of schema evolution be affected by the trends in the area of data management?

- First, we need to agree on how the future will look like…
- Open for discussion

What about DW evolution?

- How do current trends in DW technology relate to schema evolution?
- Largely depends on how different / unique DW's will be contrasted to
  - what they look like now
  - what databases will be in the future
- Open for discussion

Take Away Message

- Evolution is **viciously omnipresent**; due to its huge impact, **it is leading to non-evolvable (rigid) data & software structures**
- Practically:
  - **Plan for evolution**, well ahead of construction
  - So far, our solutions and tools help only so much
  - Industry not likely to help
- This is why **we can and have to do research**
  - We can do **pure scientific research** to find laws
  - We can do practical work for **tools and methods** that reduce the pain

… and don't forget to put everything in the git …

Thank you!

Q&A

https://github.com/DAINTINESS-Group/

http://www.cs.uoi.gr/~pvassil/

DB Schema Evolution

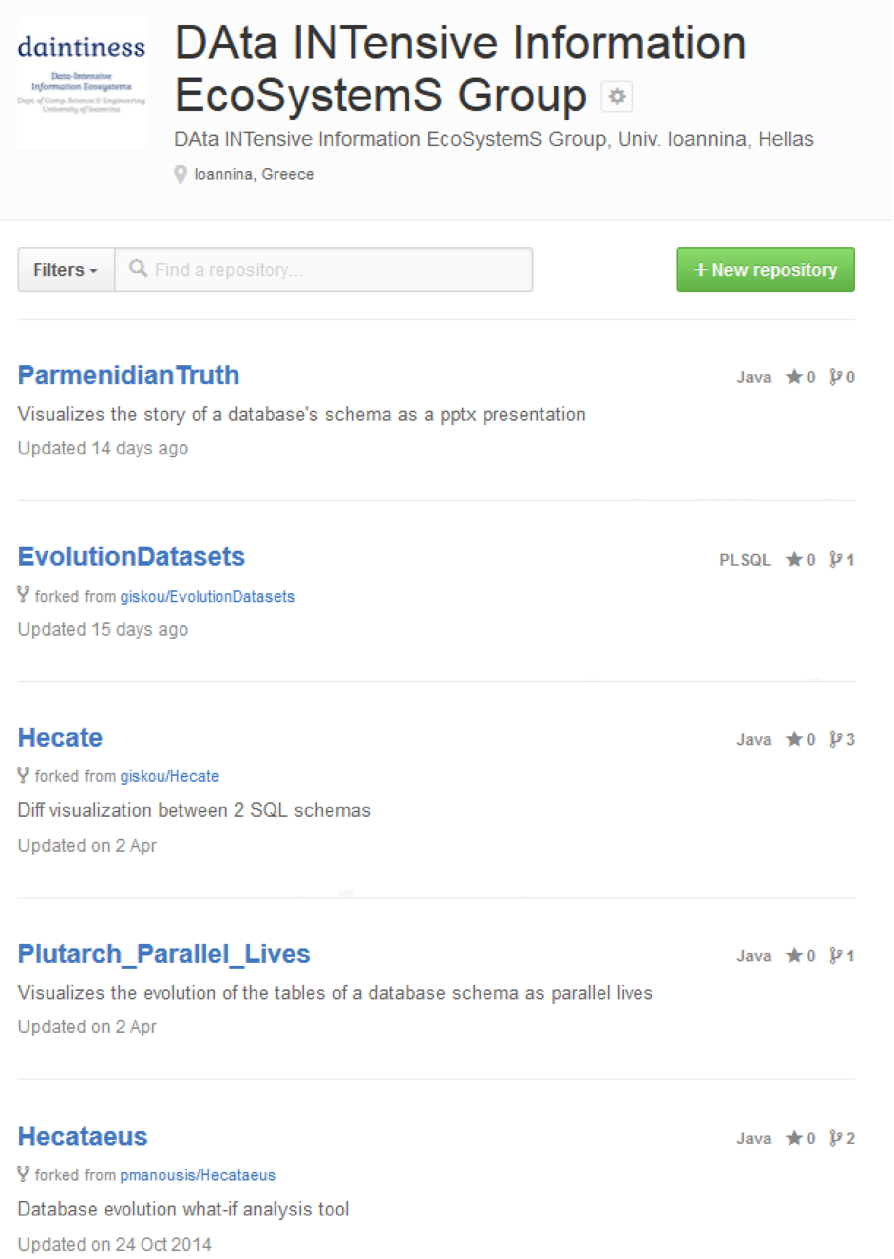
Data sets, Code, Results

publications/2014_CAiSE/

publications/2015_ER

Architecture Graphs && Hecataeus'

projects/hecataeus/



DAta INTensive Information EcoSystemS Group

DAta INTensive Information EcoSystemS Group, Univ. Ioannina, Hellas

Ioannina, Greece

ParmenidianTruth Java ★0 ⑂0

Visualizes the story of a database's schema as a pptx presentation

Updated 14 days ago

EvolutionDatasets PLSQL ★0 ⑂1

forked from giskou/EvolutionDatasets

Updated 15 days ago

Hecate Java ★0 ⑂3

forked from giskou/Hecate

Diff visualization between 2 SQL schemas

Updated on 2 Apr

Plutarch_Parallel_Lives Java ★0 ⑂1

Visualizes the evolution of the tables of a database schema as parallel lives

Updated on 2 Apr

Hecataeus Java ★0 ⑂2

forked from pmanousis/Hecataeus

Database evolution what-if analysis tool

Updated on 24 Oct 2014