Jenga and the art of data-intensive ecosystems maintenance

Panos Vassiliadis

in collaboration with

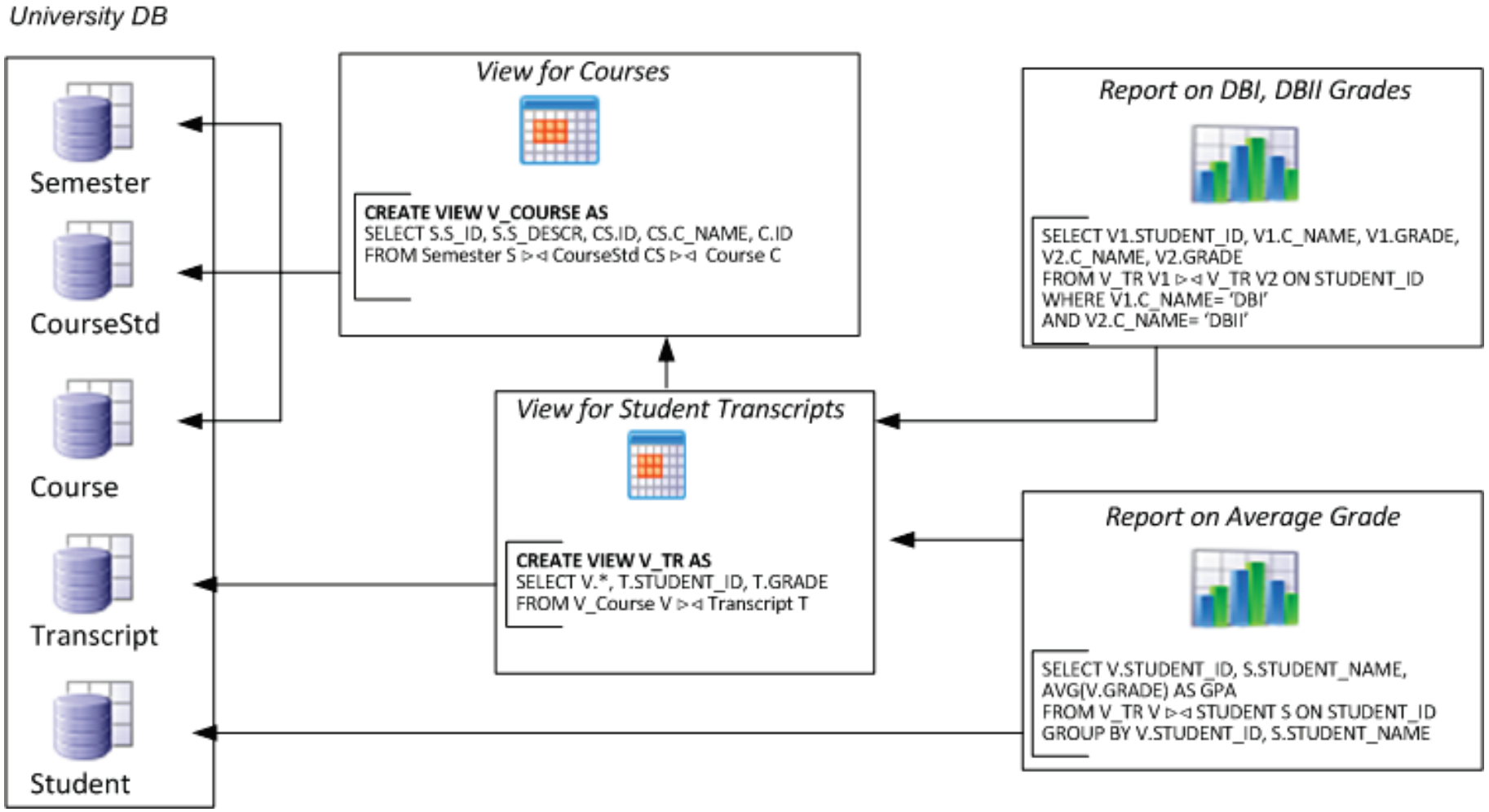
G. Papastefanatos, P. Manousis, A. Simitsis, Y. Vassiliou

University of Ioannina, Greece

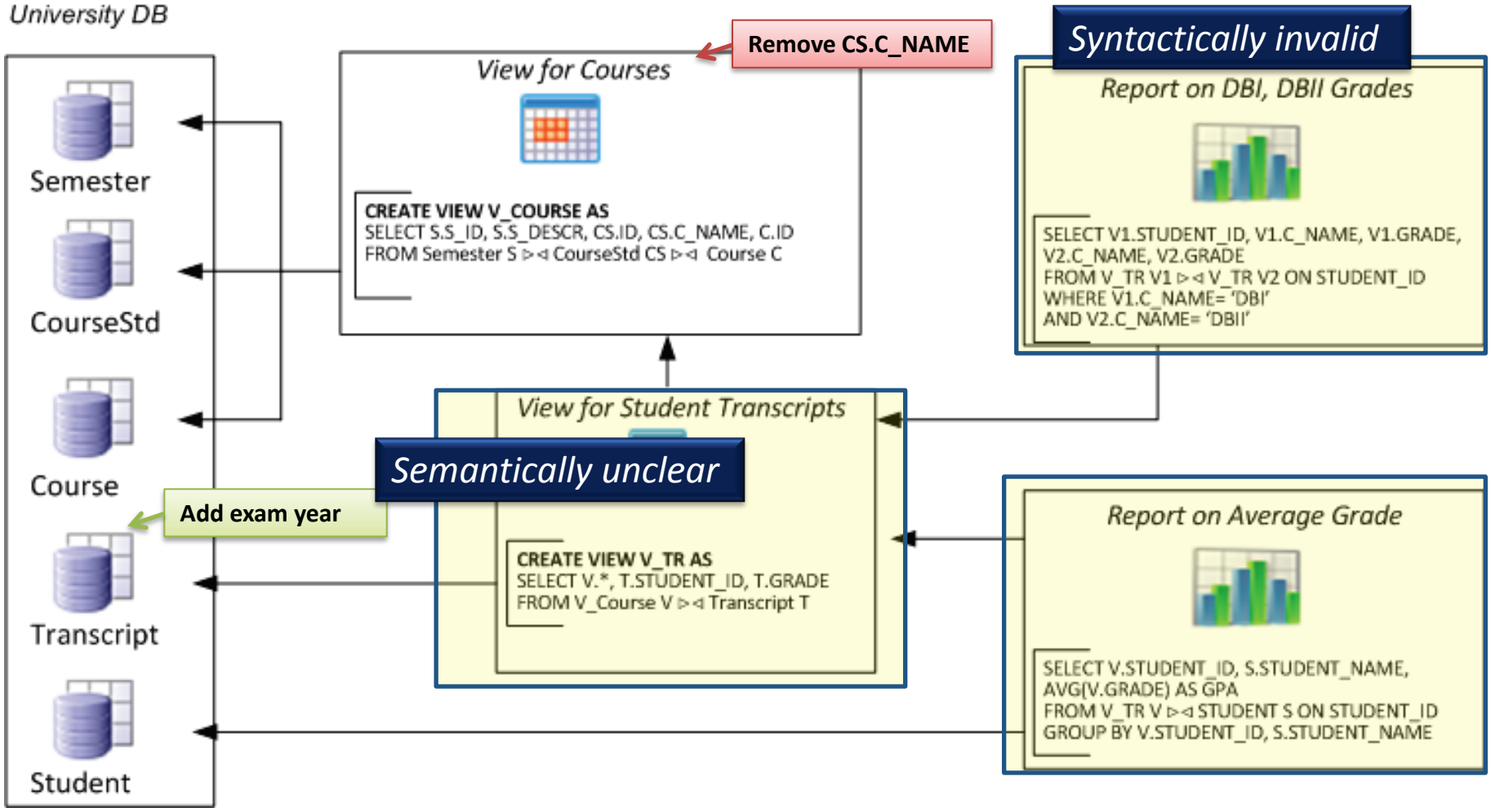
Software Evolution and ETL

- Software evolution causes at least as much as 60% of the costs for the entire software lifecycle
- ETL is not the exception:
  - Source database change their internal structure
  - Users require new structure and contents for their reports (and therefore for the collected DW data)
  - DBA and development teams do not synch well all the time
  - …

Evolving data-intensive ecosystem



Evolving data-intensive ecosystem



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

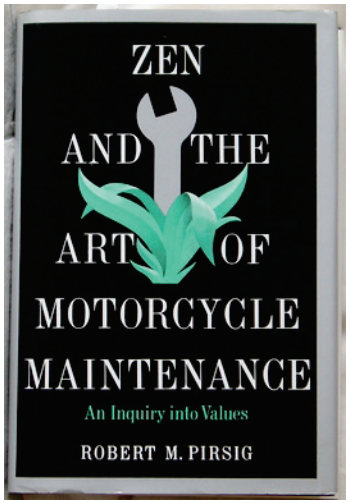
before changes happen

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

Research goals

- Part I: a case study for ETL evolution
  - Can we study ETL evolution and see in what ways do DW/ETL environments evolve?
  - Can we predict evolution in some way?
- Part II: regulating the evolution
  - Can we regulate the evolution?
  - Can we forbid unwanted changes?
  - Can we suggest adaptation to the code when the structure of data changes?

Zen and the Art of Motorcycle Maintenance, R. Pirsig



"Solve Problem: What is wrong with cycle?"

… By asking the right questions and choosing the right tests and drawing the right conclusions the mechanic works his way down the echelons of the motorcycle hierarchy until he has found the exact specific cause or causes of the engine failure, and then he changes them so that they no longer cause the failure…

Metrics for the Prediction of Evolution Impact in ETL Ecosystems: A Case Study

George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, Yannis Vassiliou

Journal on Data Semantics, August 2012, Volume 1, Issue 2, pp 75-97

Work conducted in the context of the "EICOS: foundations for perSOnalized Cooperative Information Ecosystems" project of the "Thales" Programme. The only source of funding for this research comes from the European Social Fund (ESF) -European Union (EU) and National Resources of the Greek State under the Operational Programme "Education and Lifelong Learning (EdLL).

Main goals of this effort

- We present a real-world case study of data warehouse evolution for exploring the behavior of a set of metrics that
  - monitor the vulnerability of warehouse modules to future changes and
  - assess the quality of various ETL designs with respect to their maintainability.
- We model the DW ecosystem as a graph and we employ a set of graph-theoretic metrics to see which ones fit best the series of actual evolution events
- We have used, Hecataeus, a publicly available, software tool, which allows us to monitor evolution and perform evolution scenarios in database-centric environments

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

Nothing is possible without a model

Architecture Graphs: the graph-based Model for data-Intensive ecosystems

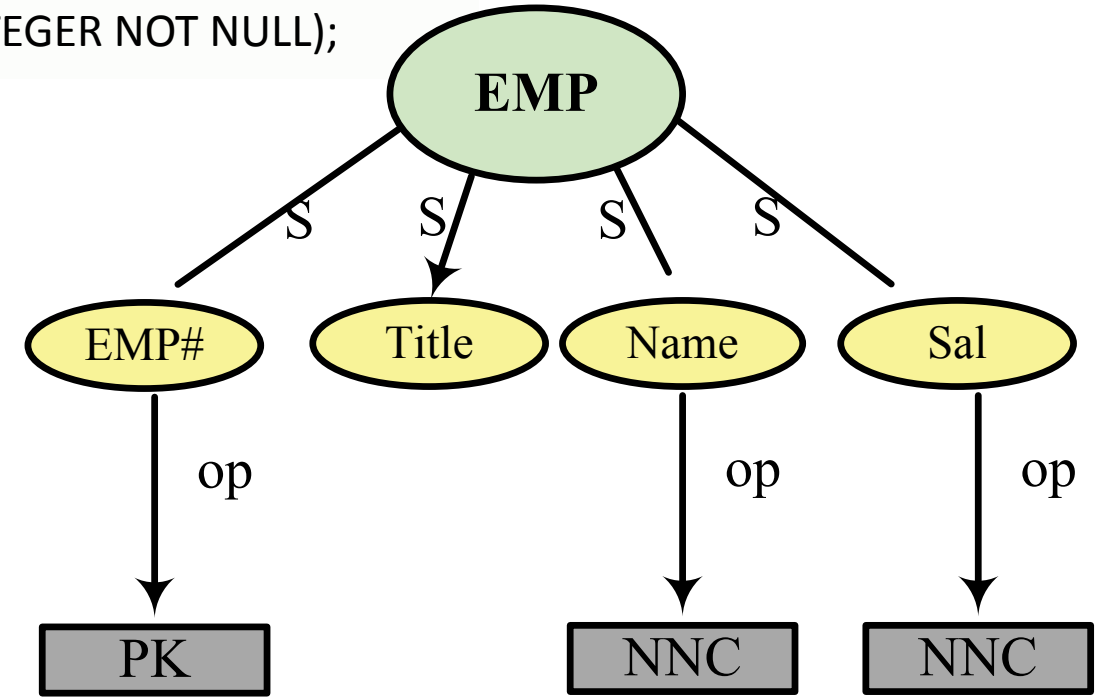
Graph modeling of a data-intensive ecosystem

- 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

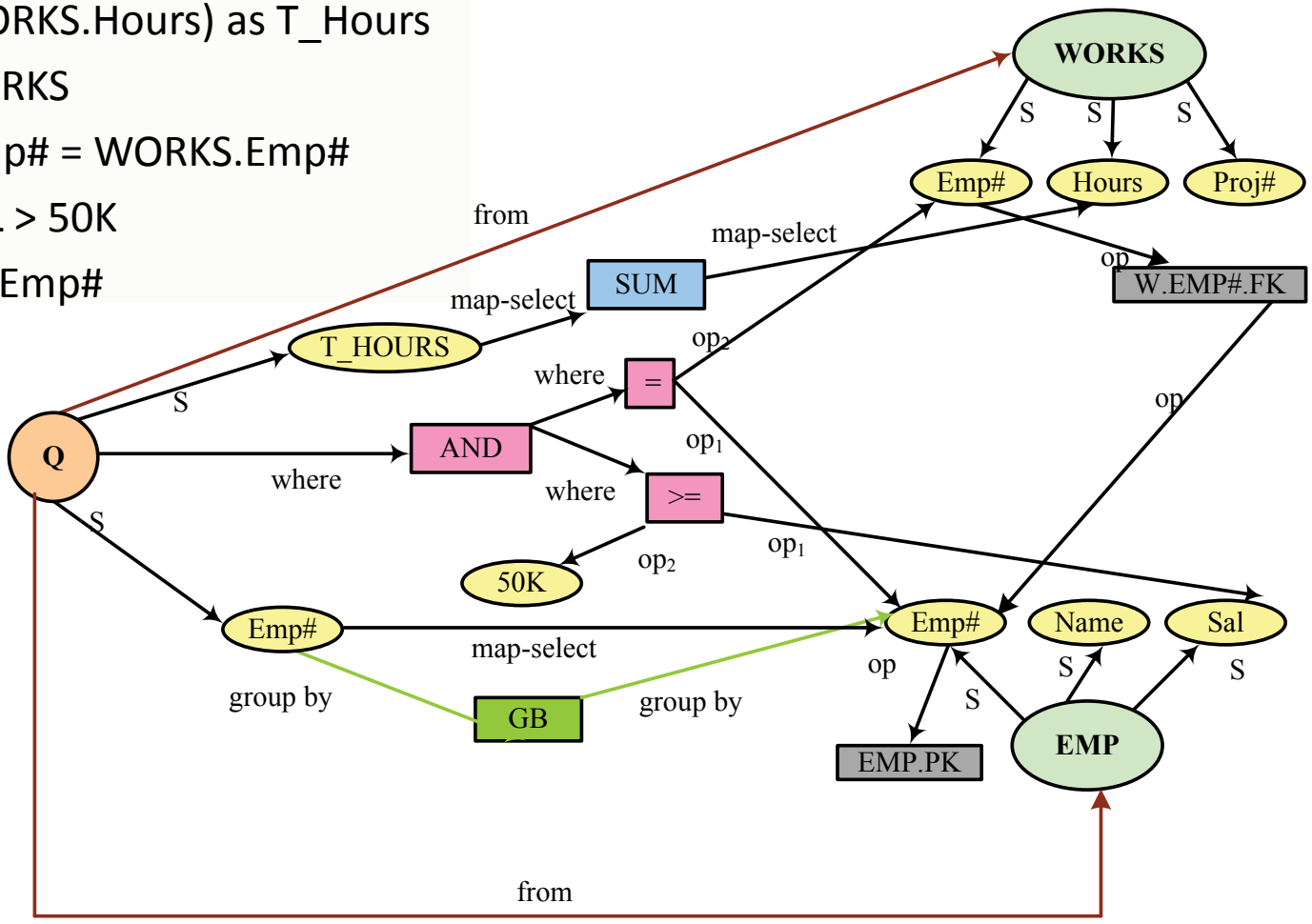
Relations – Attributes - Constraints

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

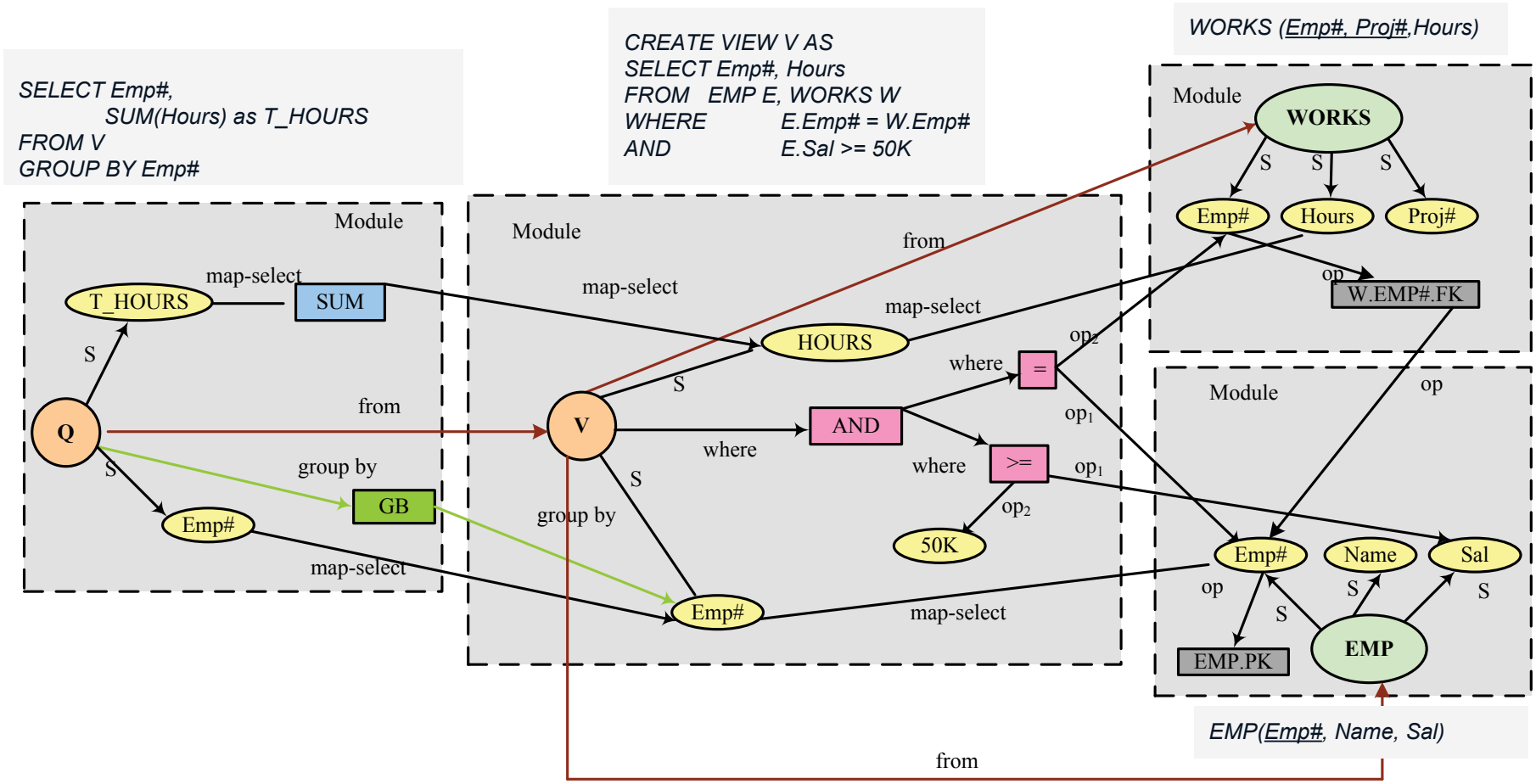


Queries & Views

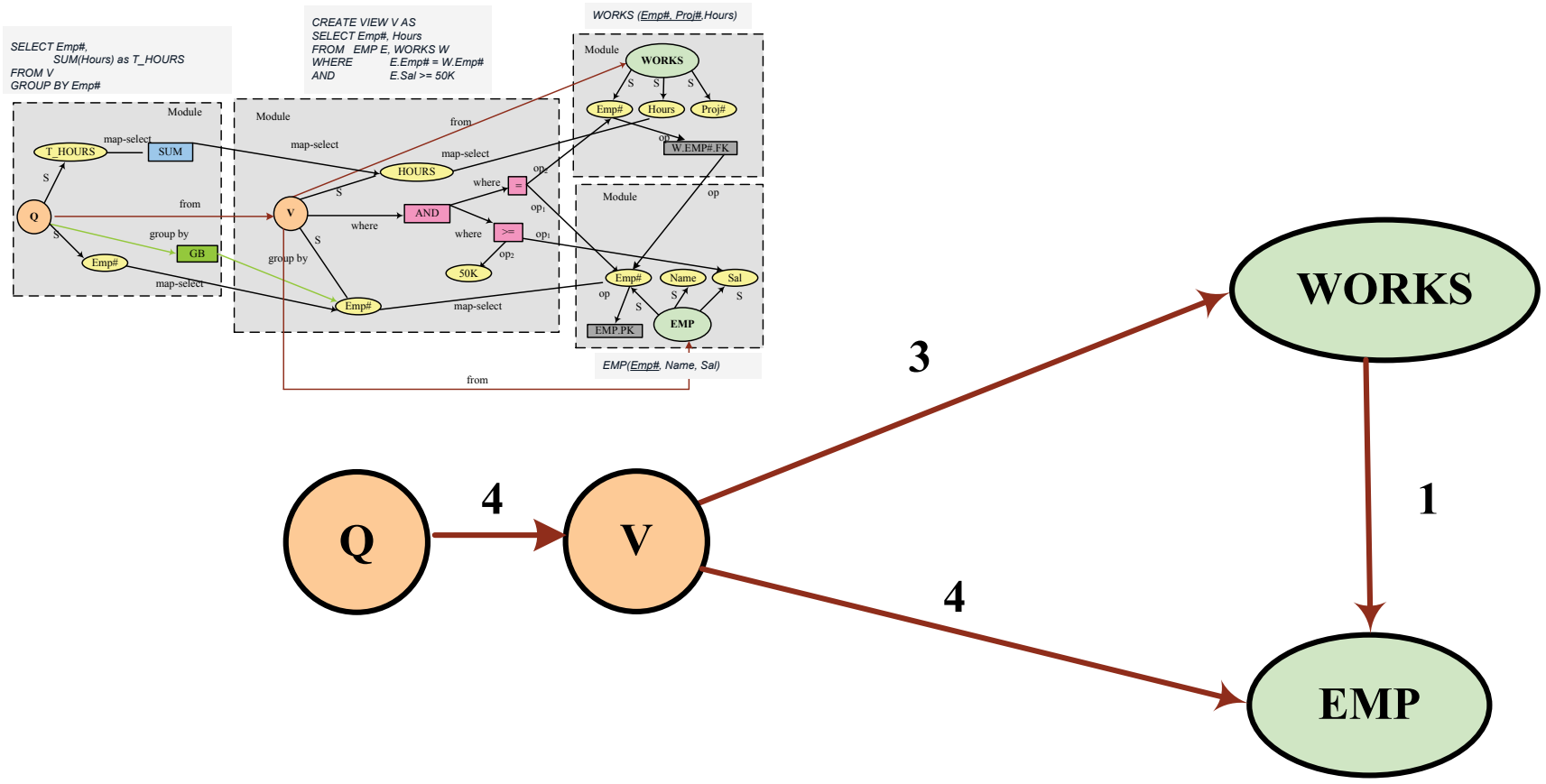
Q: SELECT EMP.Emp# as Emp#,
Sum(WORKS.Hours) as T_Hours
FROM EMP, WORKS
WHERE EMP.Emp# = WORKS.Emp#
AND EMP.SAL > 50K
GROUP BY EMP.Emp#



Modules: relations, queries, views



Zooming out to top-level nodes (modules)



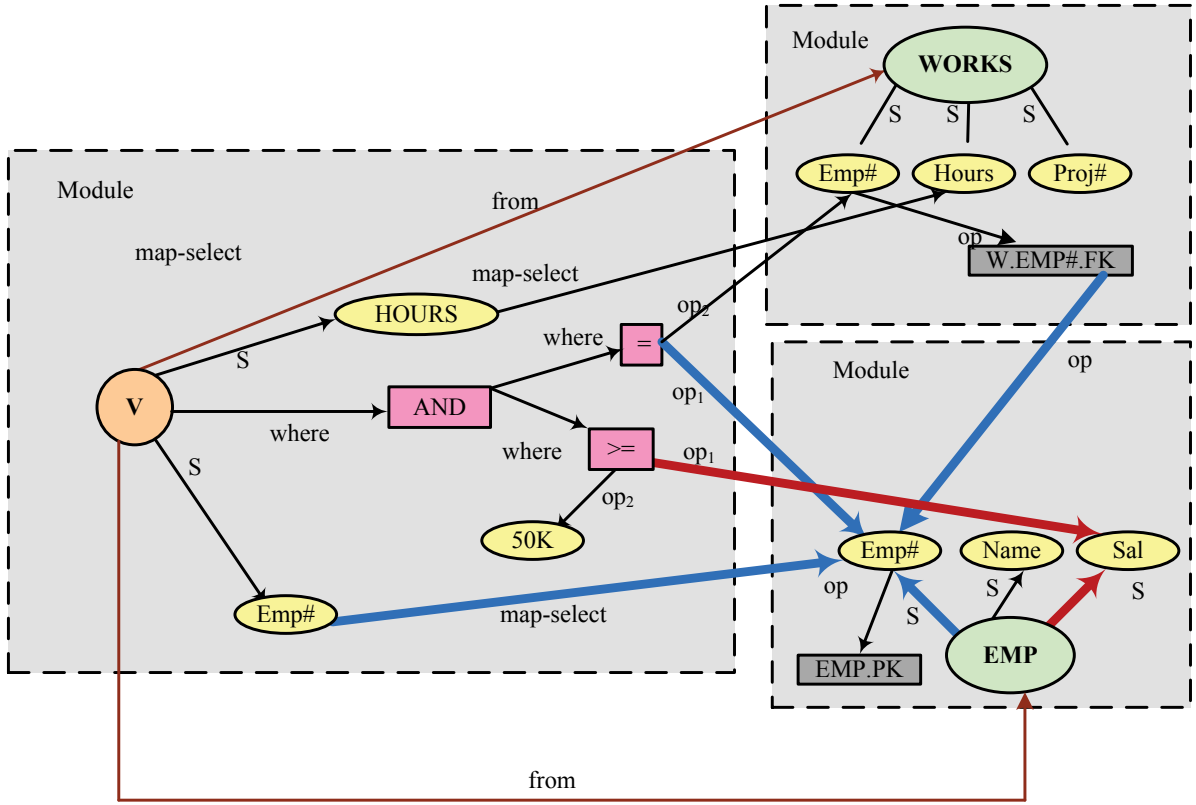
Can we relate graph-theoretic properties of nodes & modules to the probability of sustaining change?

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

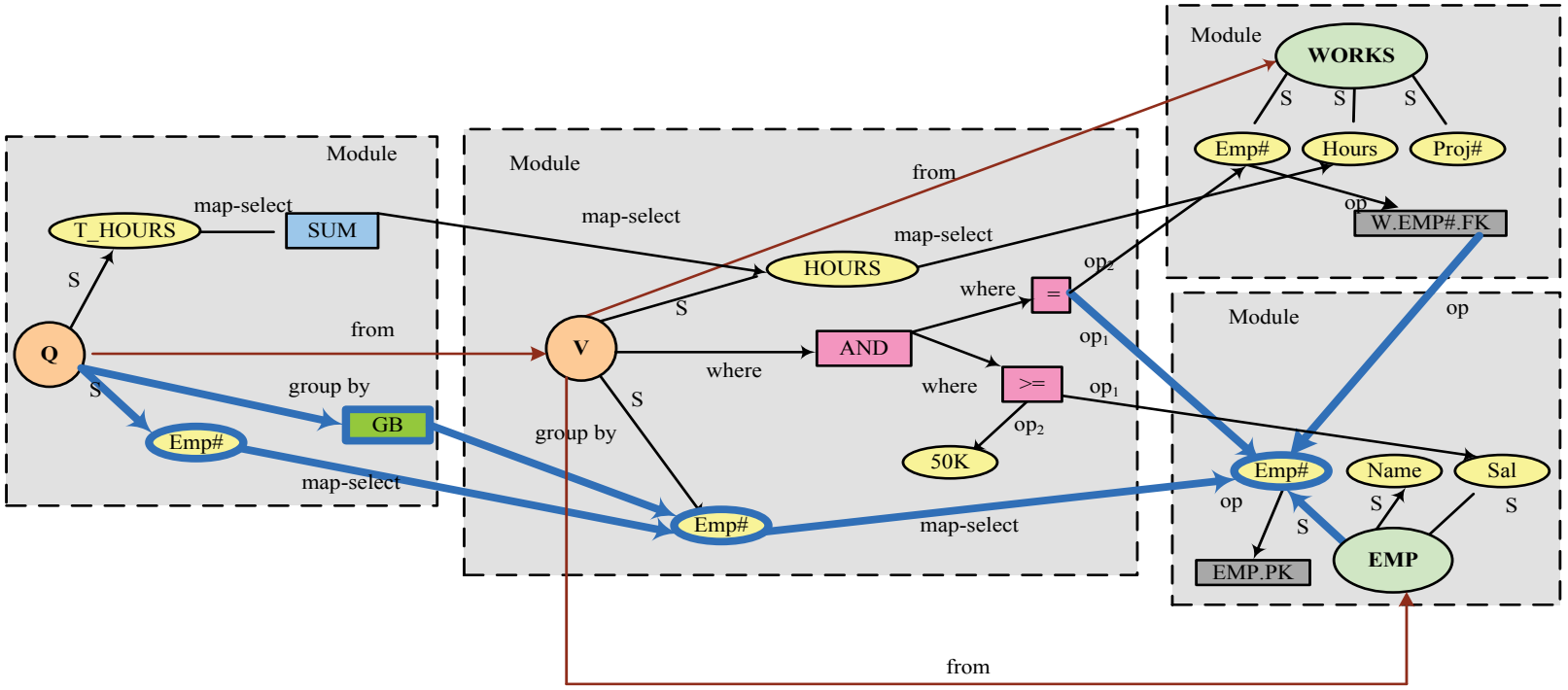


Edge direction:
from dependant
to depended upon

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

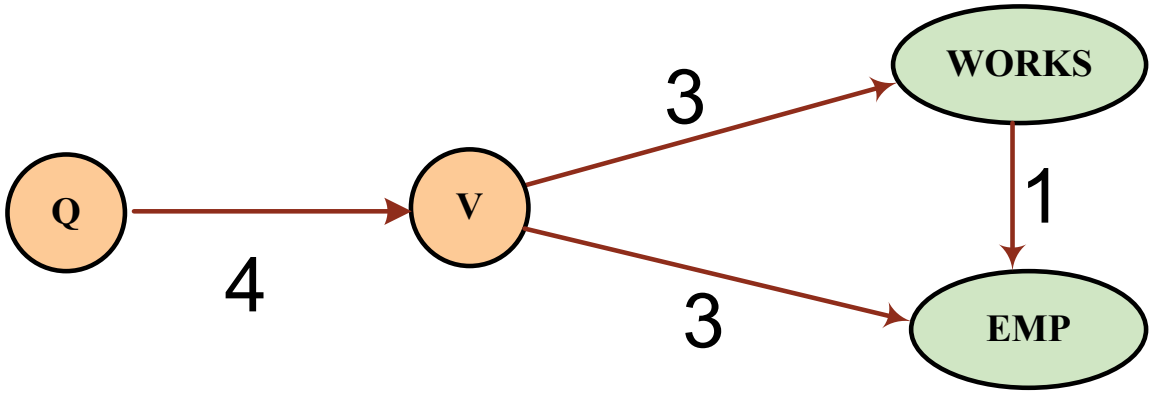


Strength: Zooming out to modules

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:

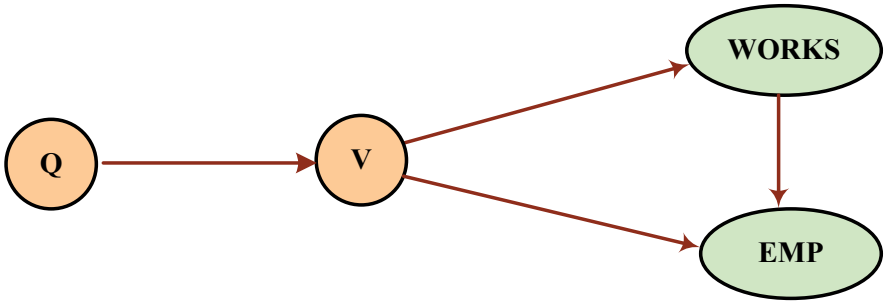
- Simple strength
- Transitive strength



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.

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

A case study

Experimental Assessment

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.

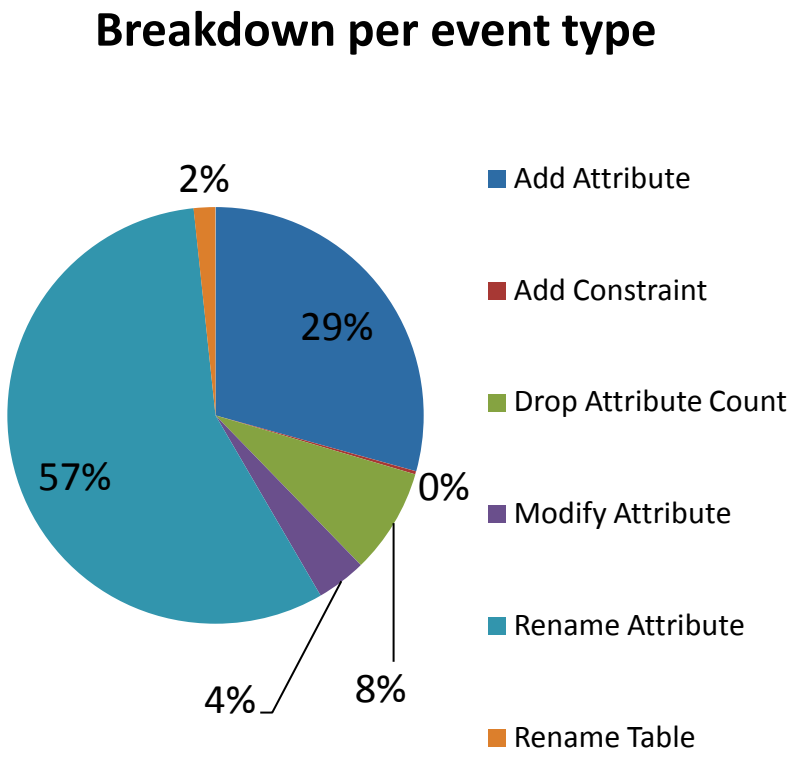
Method of assessment

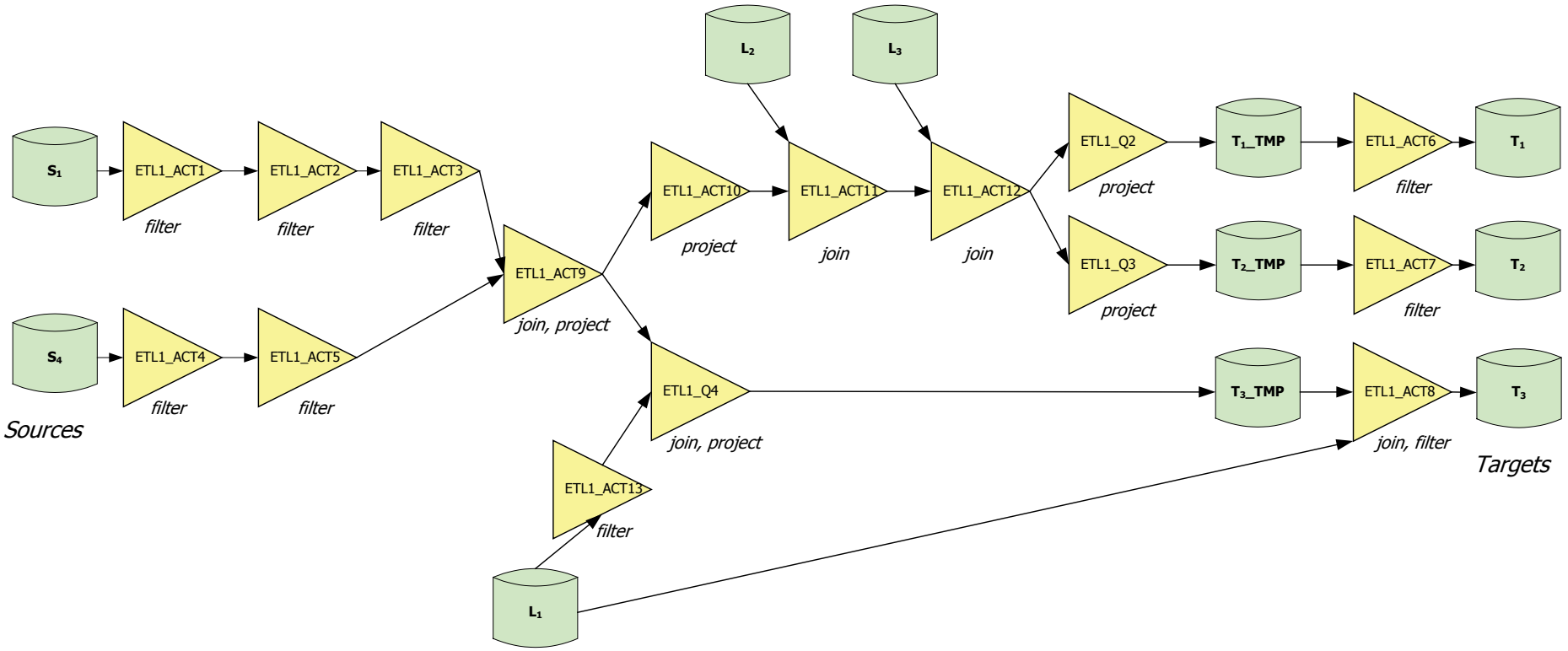
- We have represented the ETL workflows in our graph model
- We have recorded evolution events on the nodes of the source, lookup and temporary tables.
- We have applied each event sequentially on the graph and monitored the impact of the change towards the rest of the graph by recording the times that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

Macroscopic view

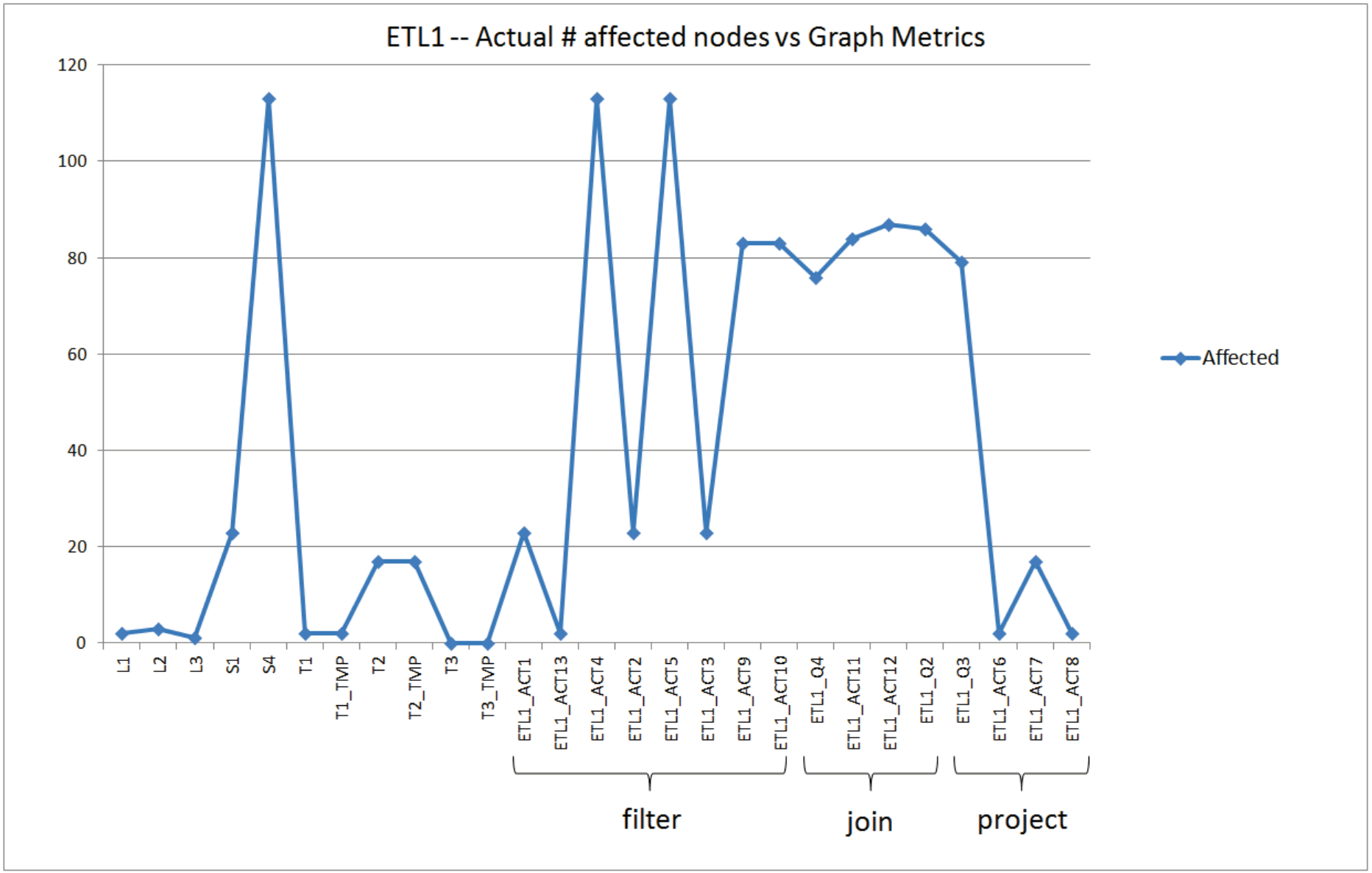
ATTN: change of requirements at the real world determines pct breakdown!!

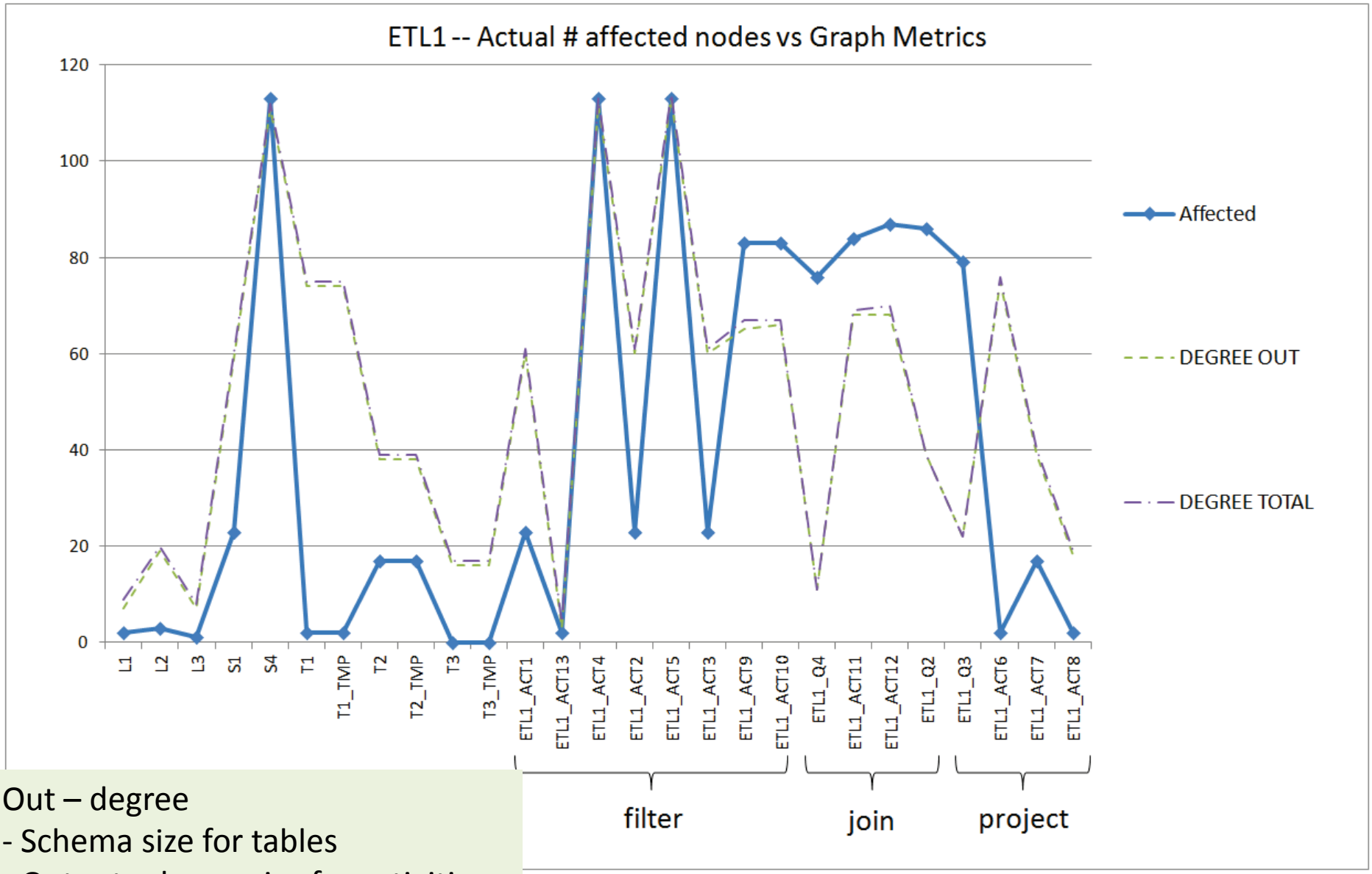
	# tables affected	Occurrences	pct
Add Attribute	8	122	**29%**
Add Constraint	1	1	0%
Drop Attribute Count	5	34	8%
Modify Attribute	9	16	4%
Rename Attribute	5	236	**57%**
Rename Table	7	7	2%
		416	



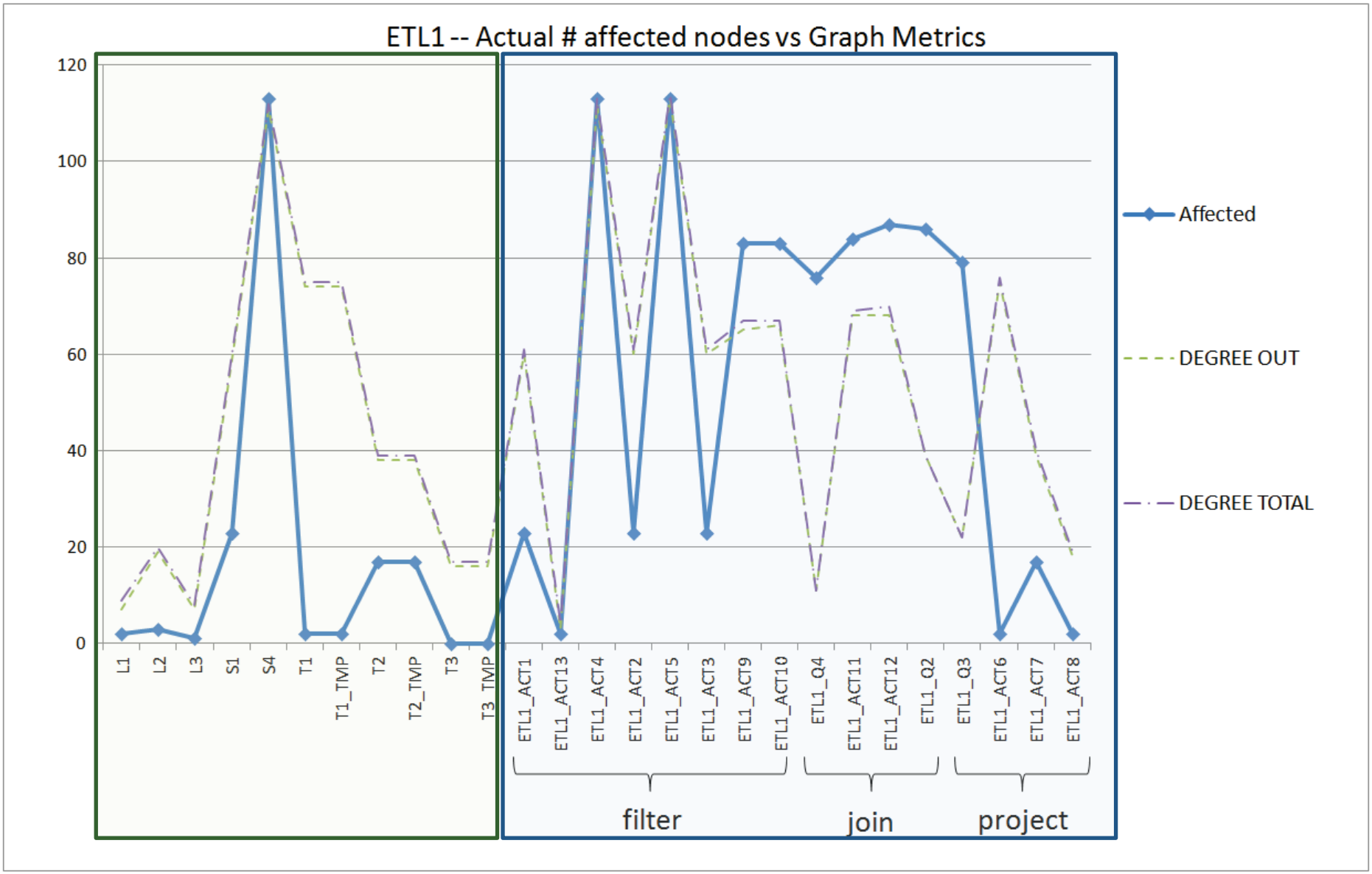


Workflow of the first ETL scenario, ETL1


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



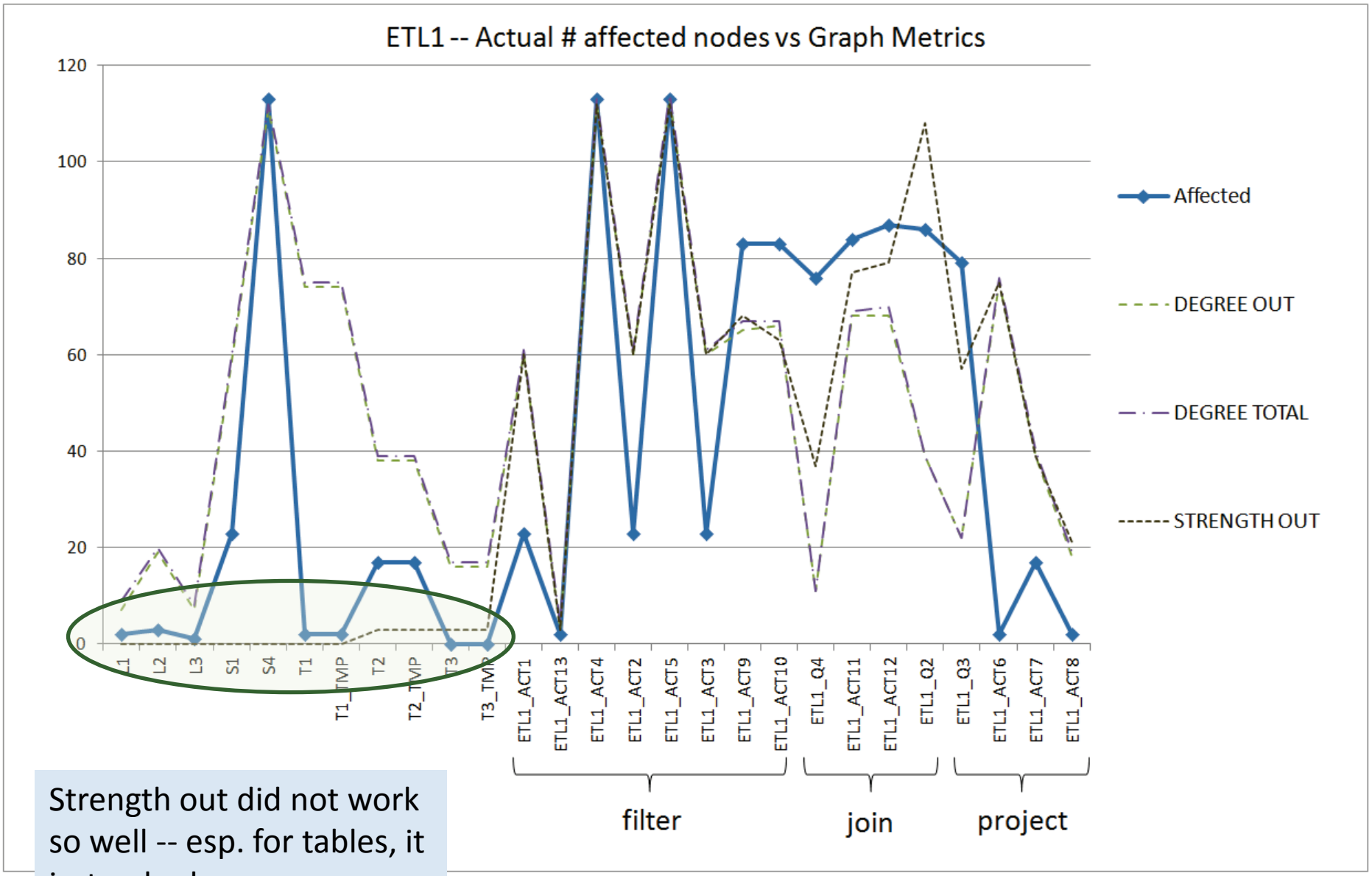
Out – degree
- Schema size for tables
- Output schema size for activities



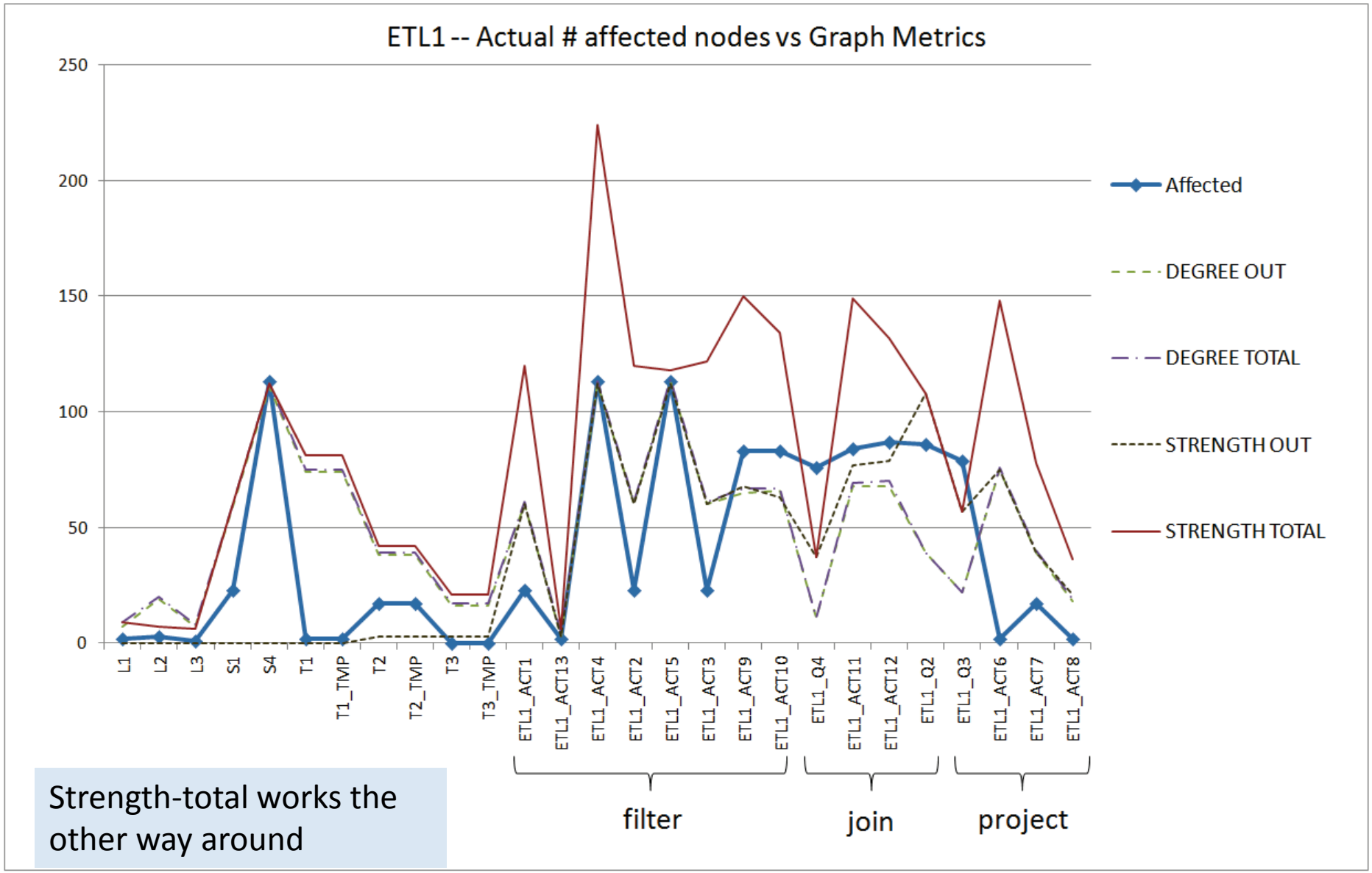
Pretty good job for tables

Decent job for filters and joins

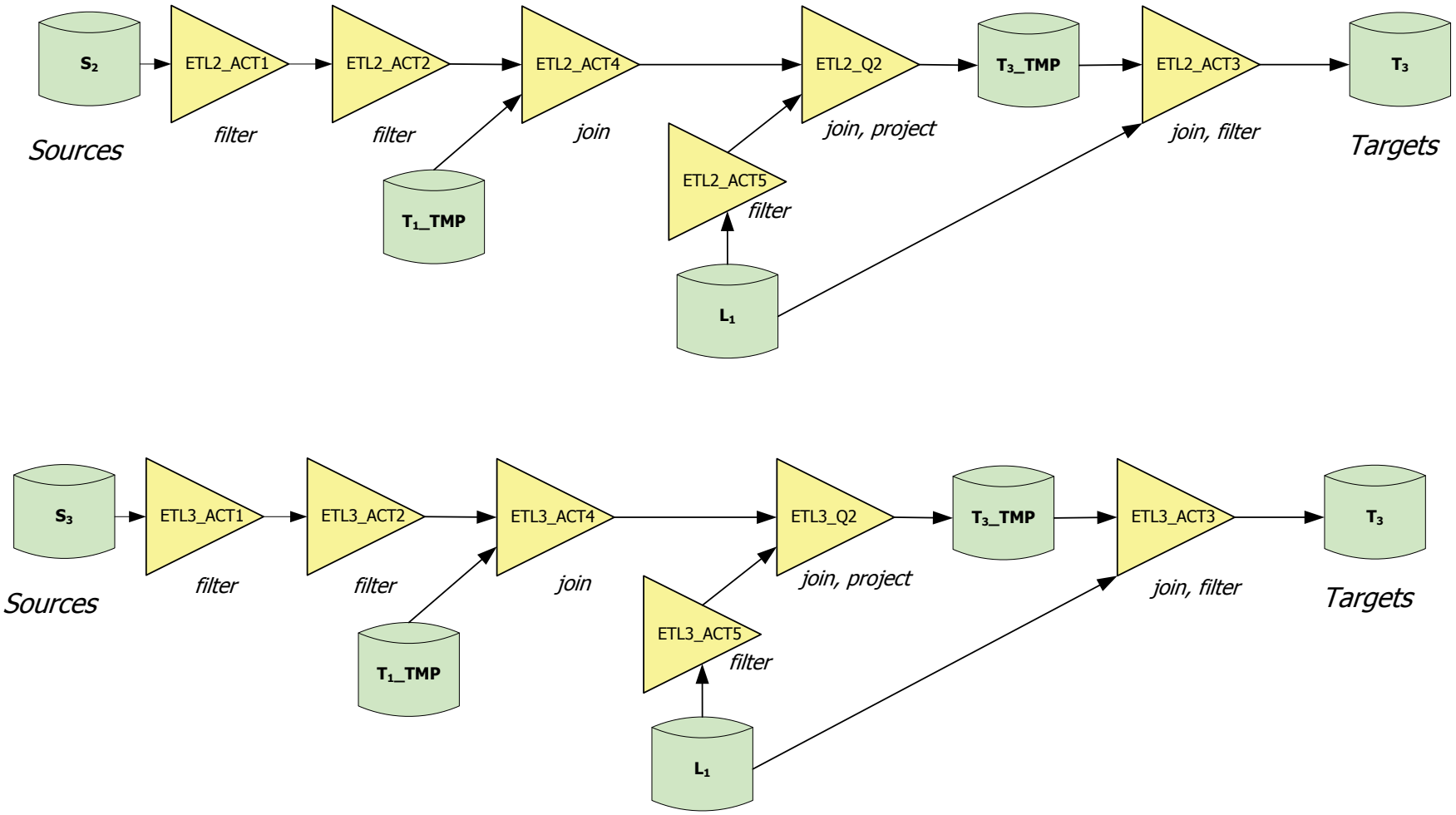
Not so good for projection activities



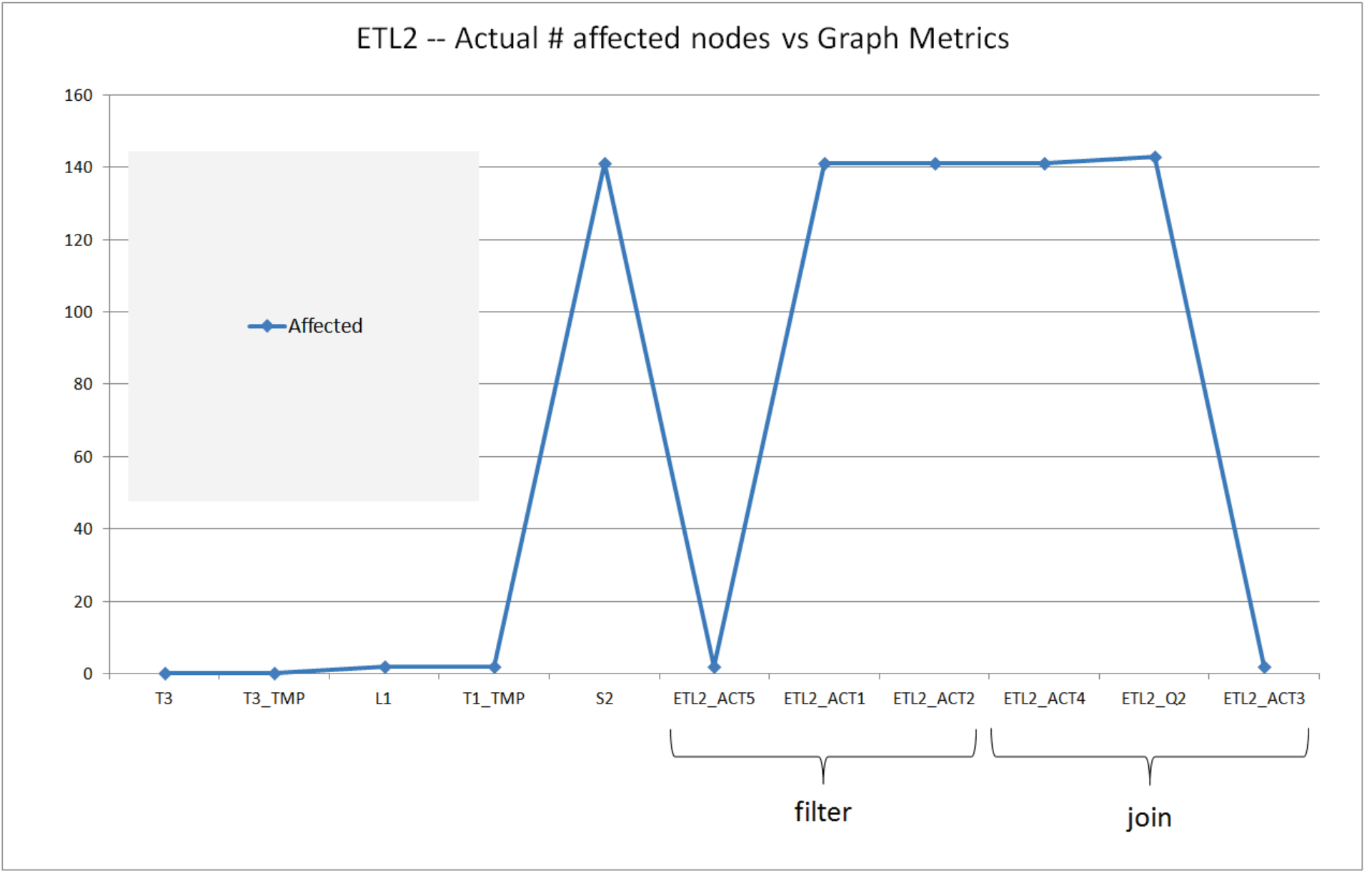
Strength out did not work so well -- esp. for tables, it is too bad

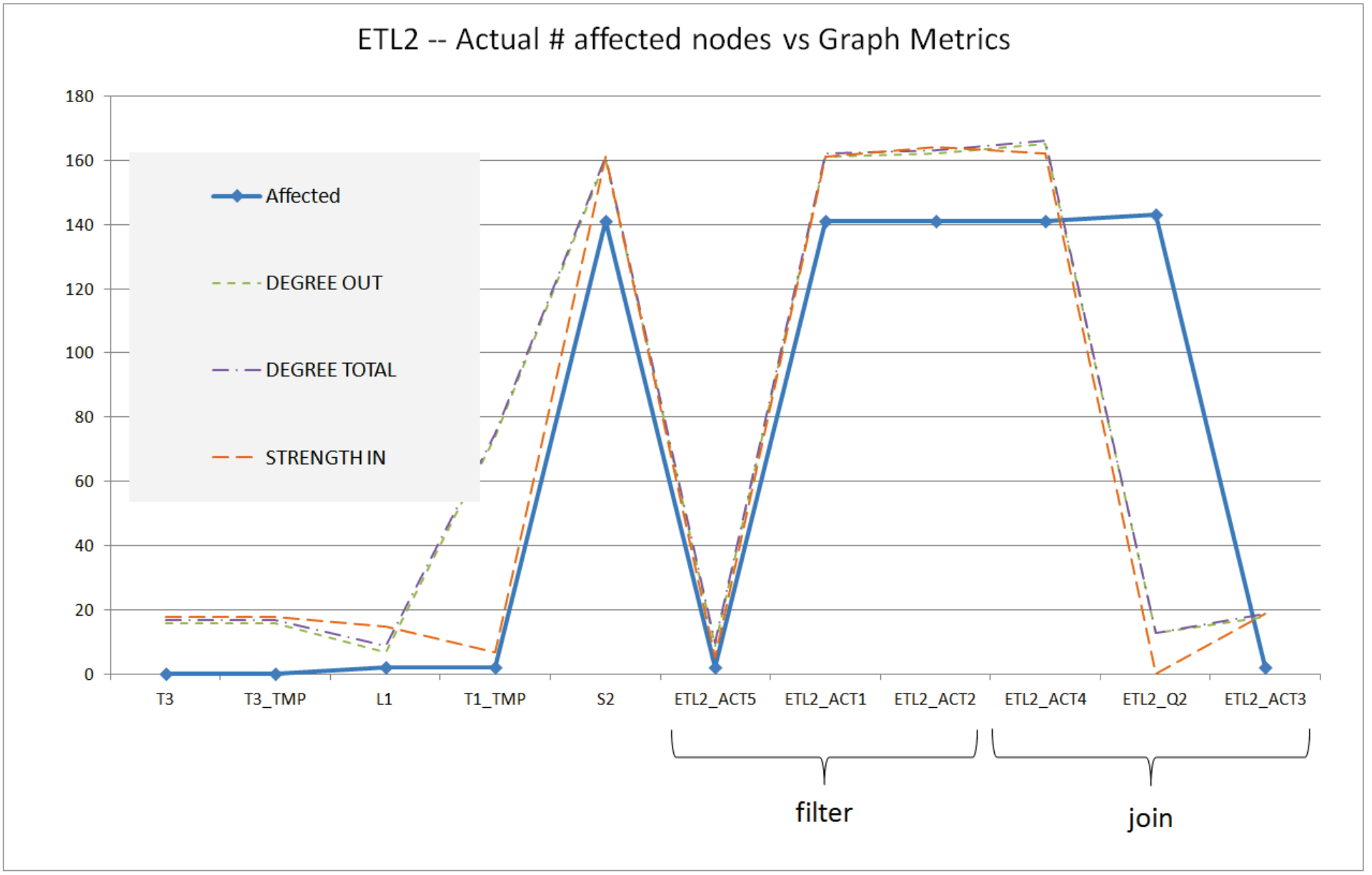


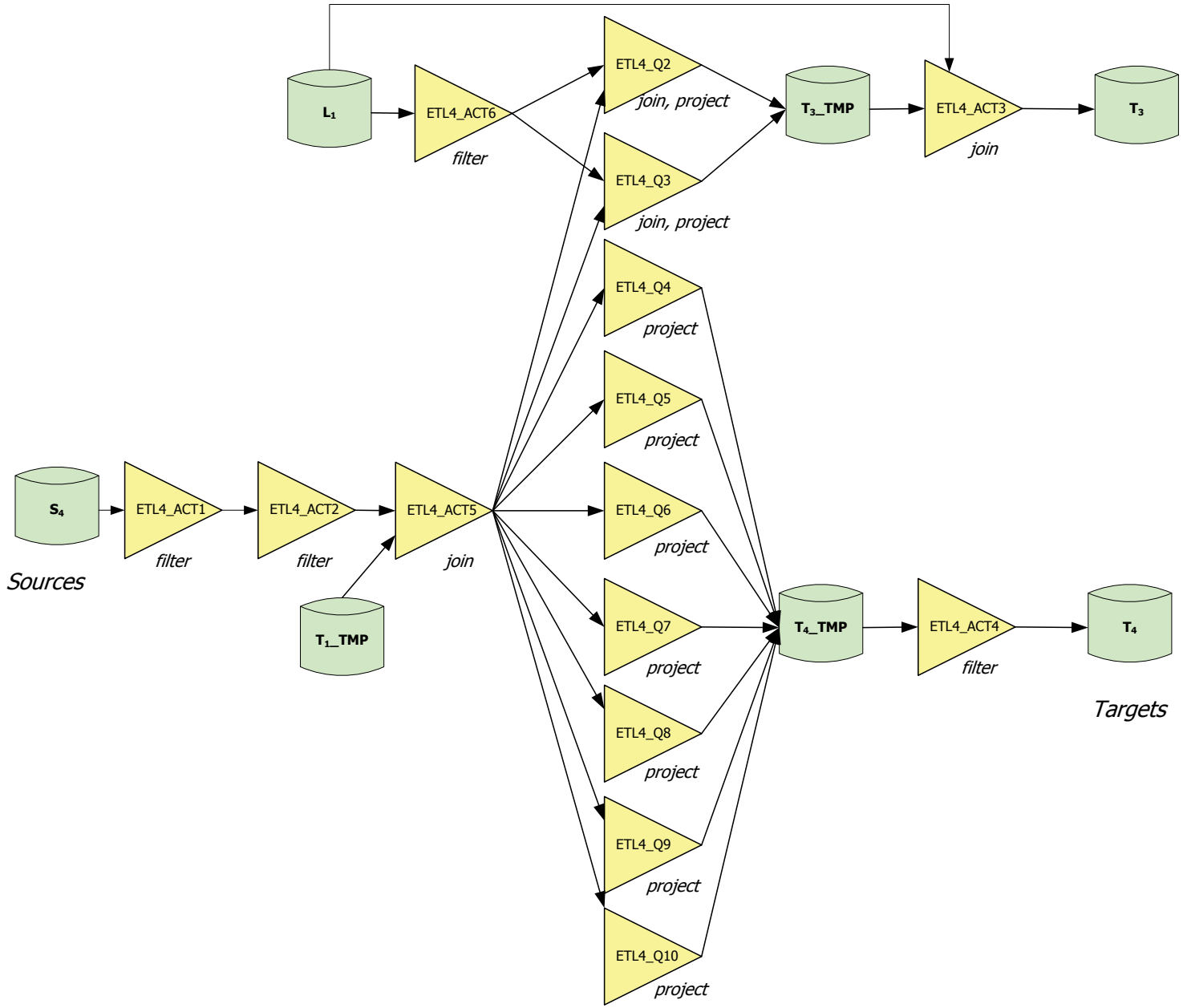
Strength-total works the other way around



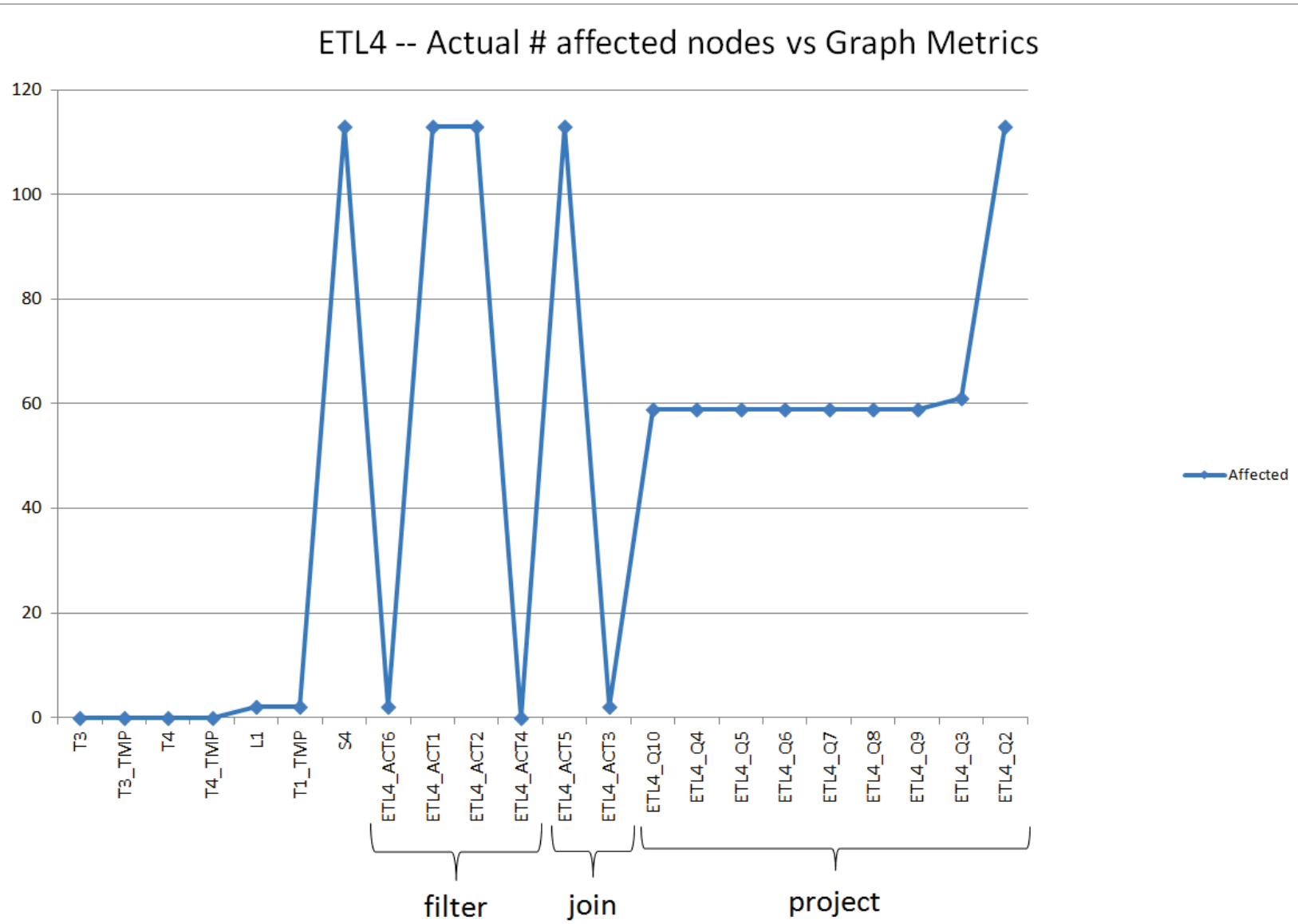
Workflows of the second & third ETL scenarios, ETL2 – ETL3

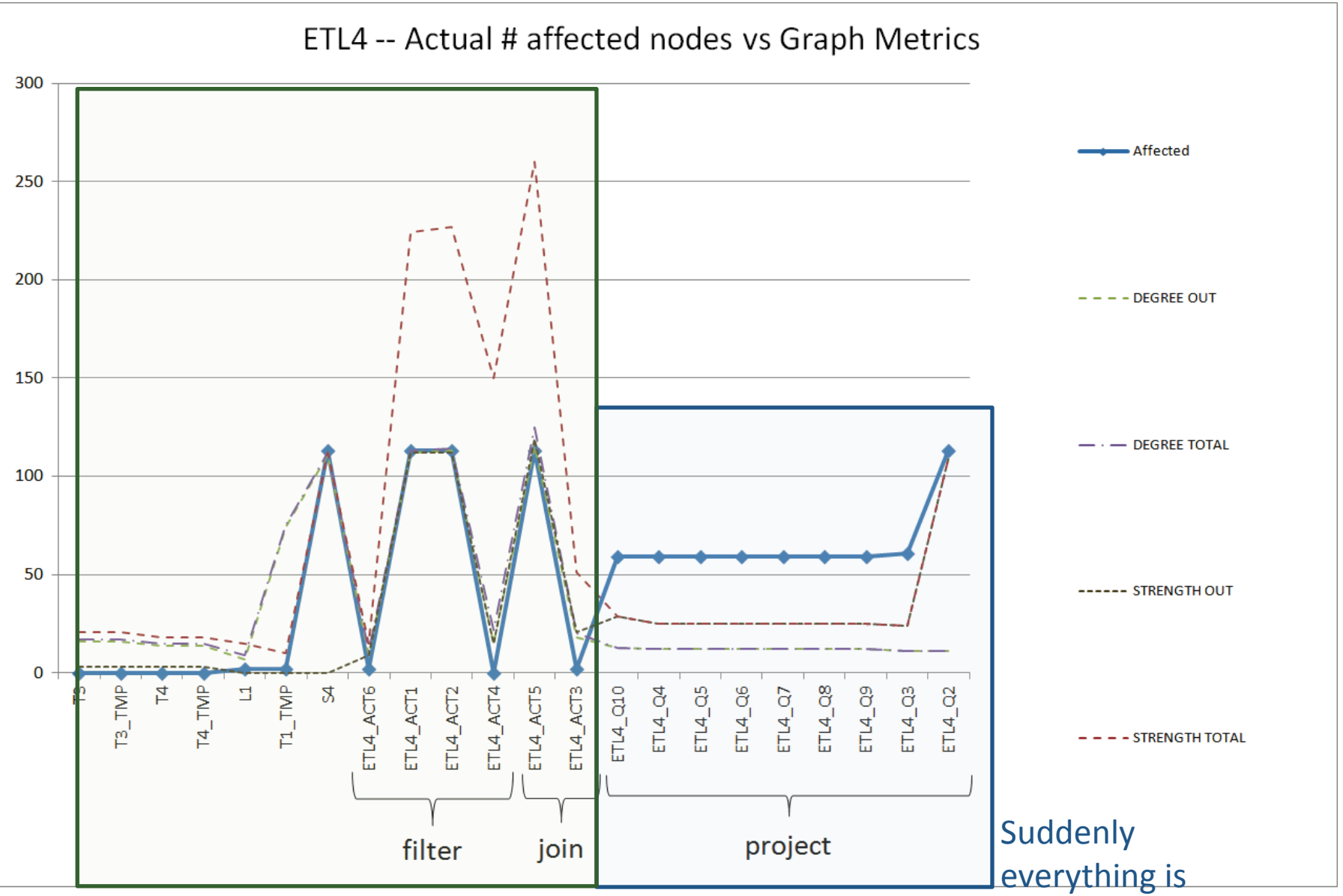

ETL2 -- Actual # affected nodes vs Graph Metrics
160
140
120
100
80
60
40
20
0
Affected
T3
T3_TMP
L1
T1_TMP
S2
ETL2_ACT5
ETL2_ACT1
ETL2_ACT2
ETL2_ACT4
ETL2_Q2
ETL2_ACT3
filter
join


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


L1
ETL4_ACT6
filter
ETL4_Q2
join, project
ETL4_Q3
join, project
T3_TMP
ETL4_ACT3
join
T3
ETL4_Q4
project
ETL4_Q5
project
ETL4_Q6
project
ETL4_Q7
project
ETL4_Q8
project
ETL4_Q9
project
ETL4_Q10
project
S4
ETL4_ACT1
filter
ETL4_ACT2
filter
T1_TMP
ETL4_ACT5
join
Sources
T4_TMP
ETL4_ACT4
filter
T4
Targets

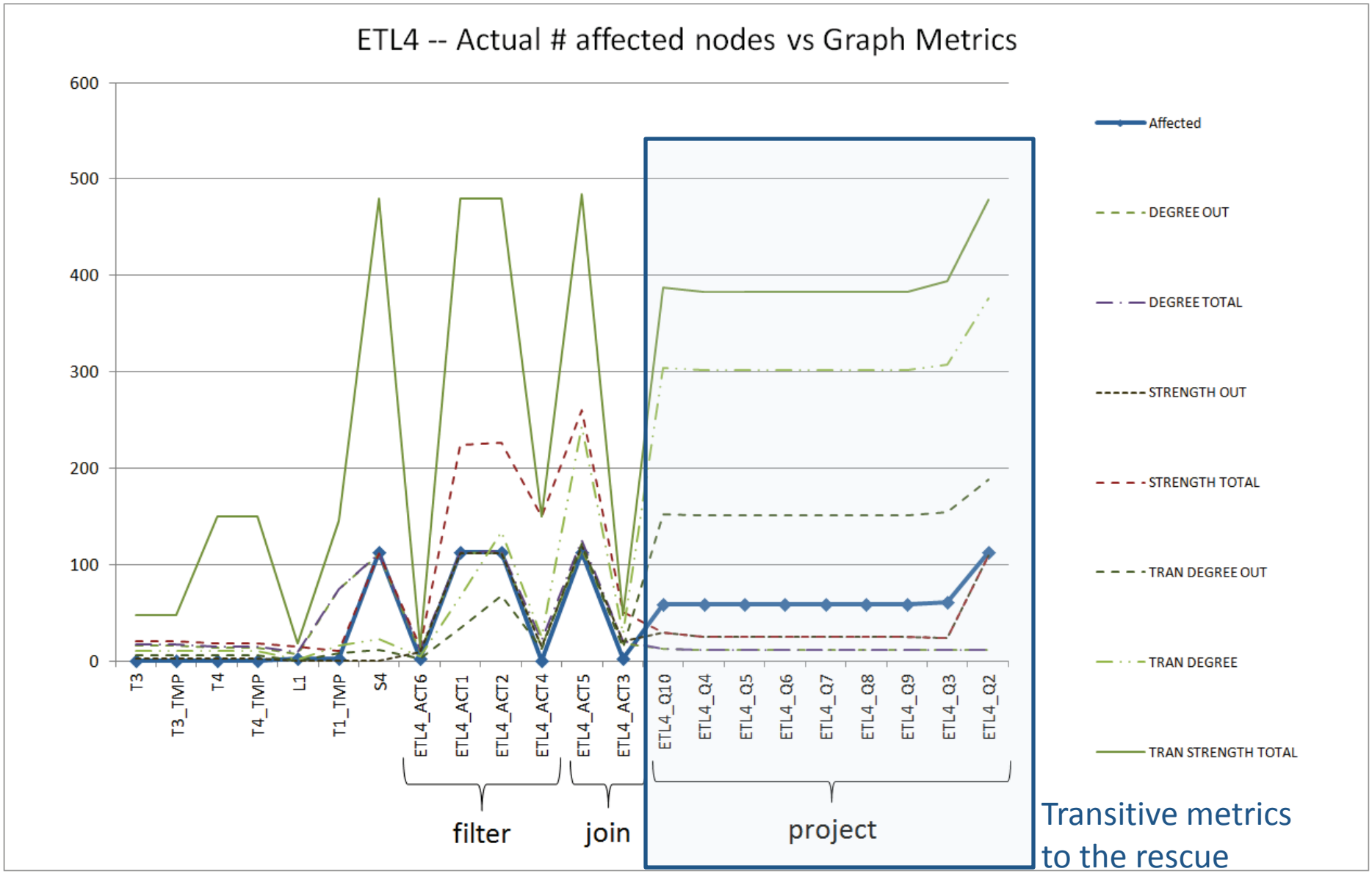
ETL 4


ETL4 -- Actual # affected nodes vs Graph Metrics
120
100
80
60
40
20
0
T3
T3_TMP
T4
T4_TMP
L1
T1_TMP
S4
ETL4_ACT6
ETL4_ACT1
ETL4_ACT2
ETL4_ACT4
ETL4_ACT5
ETL4_ACT3
ETL4_Q10
ETL4_Q4
ETL4_Q5
ETL4_Q6
ETL4_Q7
ETL4_Q8
ETL4_Q9
ETL4_Q3
ETL4_Q2
filter
join
project
Affected

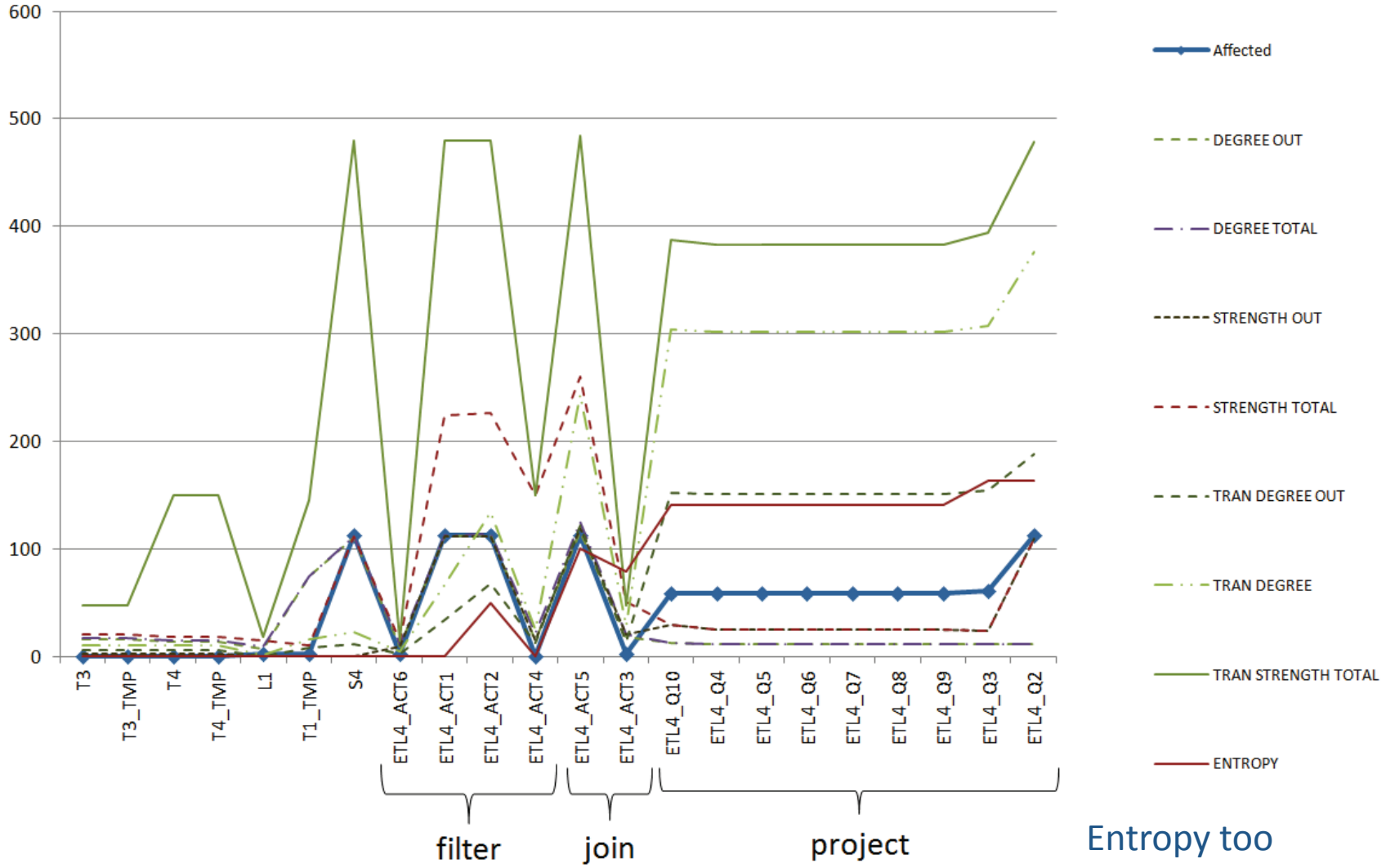


Pretty good job in the left part

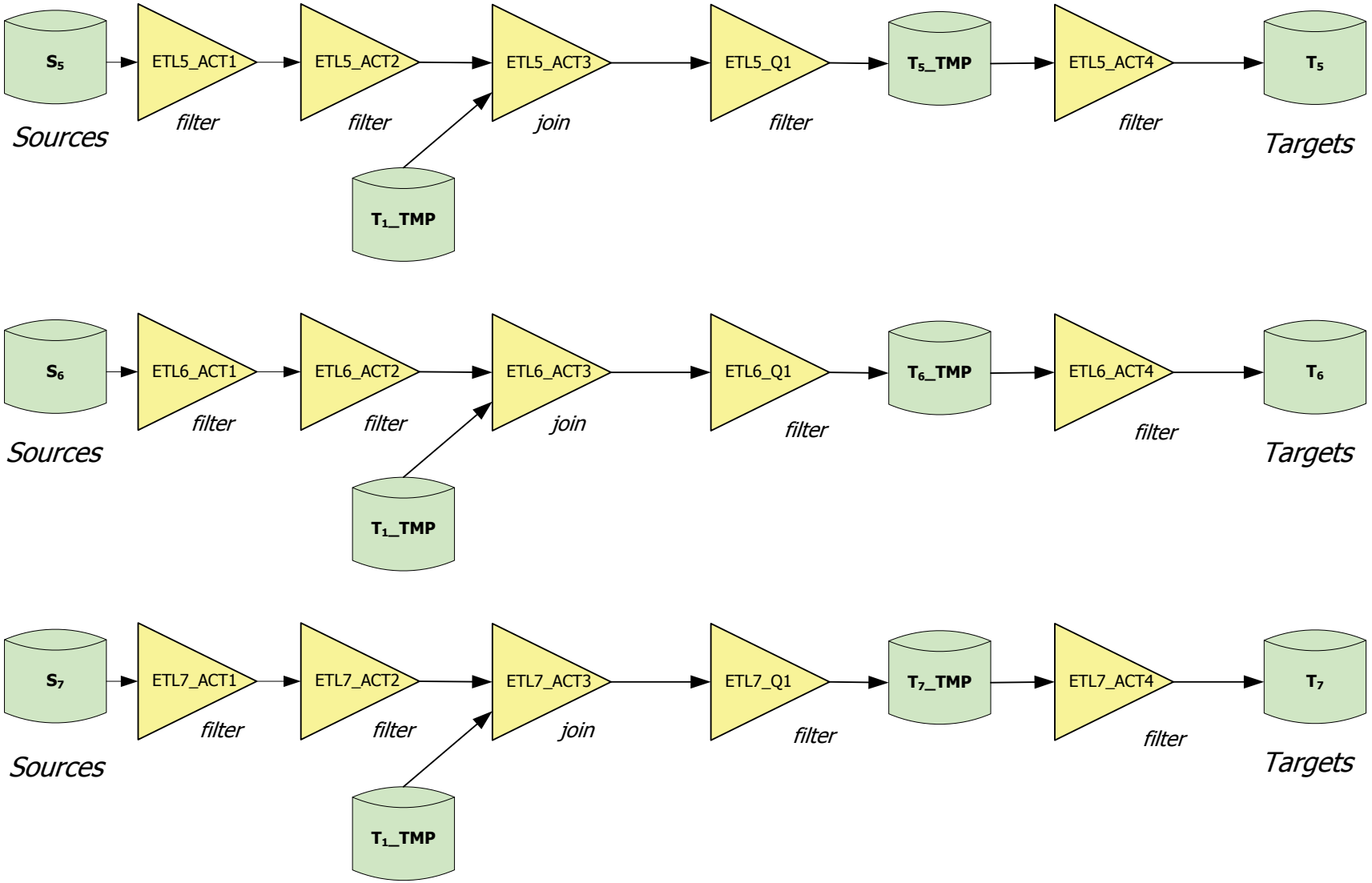
Suddenly everything is underestimated



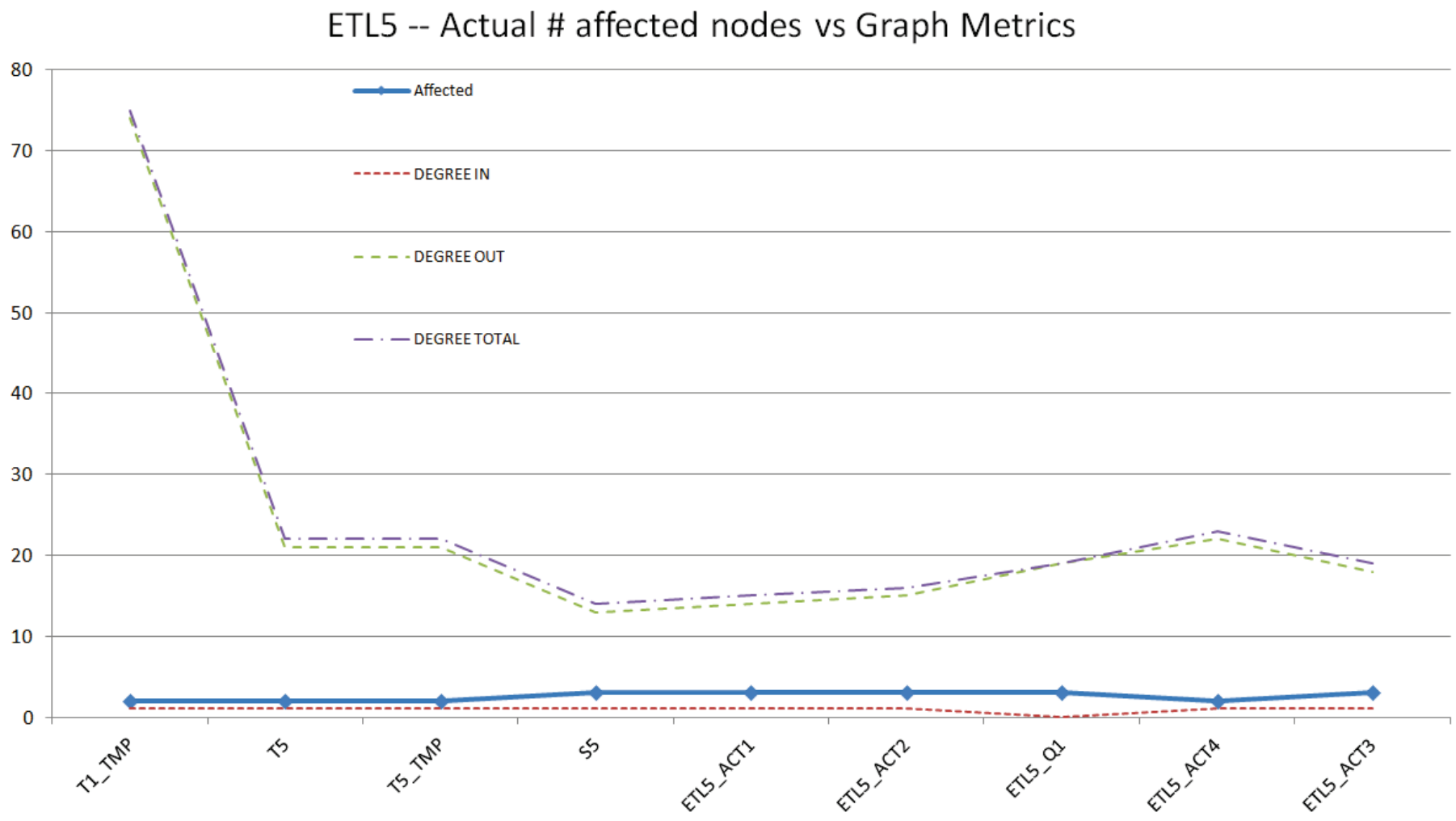
Transitive metrics to the rescue

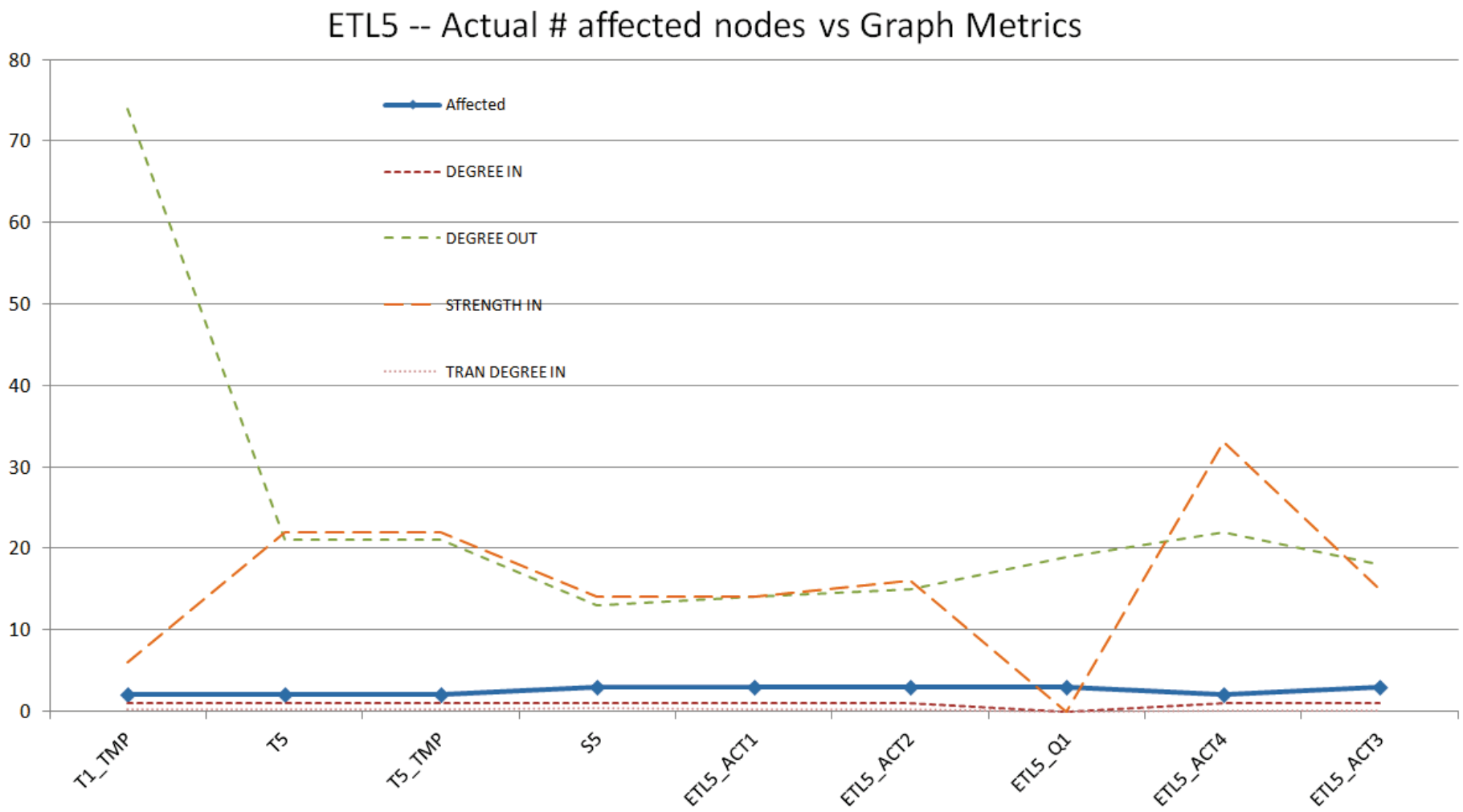

ETL4 -- Actual # affected nodes vs Graph Metrics
600
500
400
300
200
100
0
T3
T3_TMP
T4
T4_TMP
L1
T1_TMP
S4
ETL4_ACT6
ETL4_ACT1
ETL4_ACT2
ETL4_ACT4
ETL4_ACT5
ETL4_ACT3
ETL4_Q10
ETL4_Q4
ETL4_Q5
ETL4_Q6
ETL4_Q7
ETL4_Q8
ETL4_Q9
ETL4_Q3
ETL4_Q2
filter
join
project
Affected
DEGREE OUT
DEGREE TOTAL
STRENGTH OUT
STRENGTH TOTAL
TRAN DEGREE OUT
TRAN DEGREE
TRAN STRENGTH TOTAL
ENTROPY

Entropy too


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
80
70
60
50
40
30
20
10
0
Affected
DEGREE IN
DEGREE OUT
DEGREE TOTAL
T1_TMP
T5
T5_TMP
S5
ETL5_ACT1
ETL5_ACT2
ETL5_Q1
ETL5_ACT4
ETL5_ACT3


ETL5 -- Actual # affected nodes vs Graph Metrics
80
70
60
50
40
30
20
10
0
Affected
DEGREE IN
DEGREE OUT
STRENGTH IN
TRAN DEGREE IN
T1_TMP
T5
T5_TMP
S5
ETL5_ACT1
ETL5_ACT2
ETL5_Q1
ETL5_ACT4
ETL5_ACT3

Lessons Learned

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*

This was just a first step

- … we need many more studies to establish a firm knowledge of the mechanics of evolution
- … and we have not answered yet the core question:

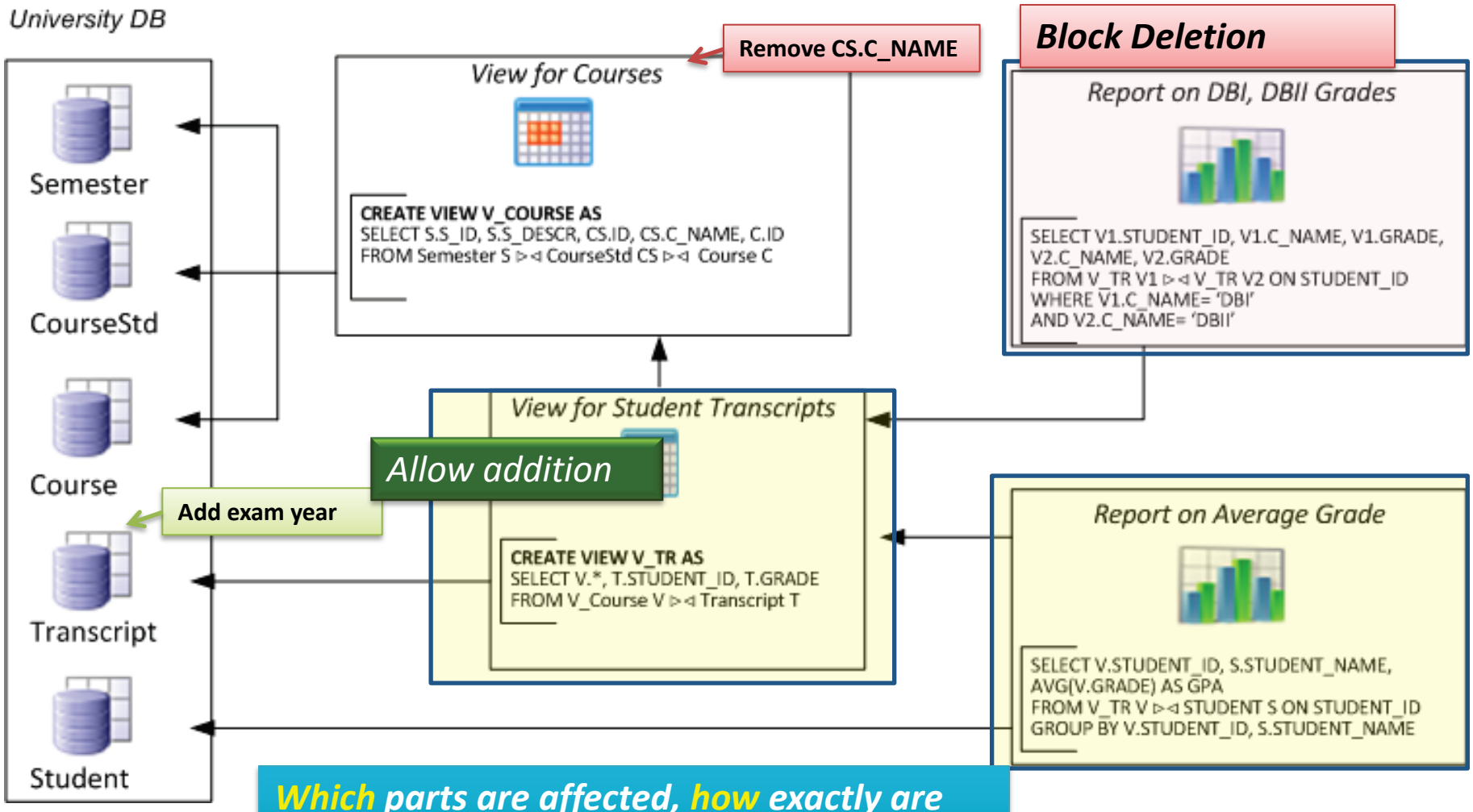
Are we helpless in managing evolution with predictability?

Automating the adaptation of evolving data-intensive ecosystems

Petros Manousis, Panos Vassiliadis, and George Papastefanatos

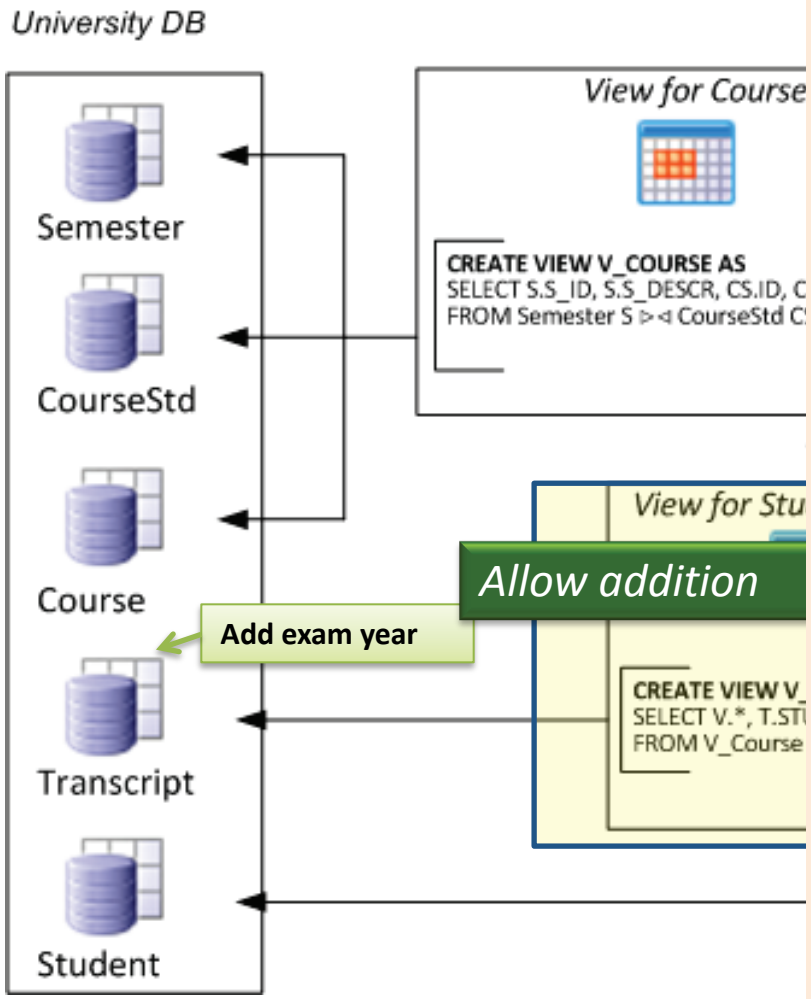
Mainly based on the work of the MSc P. Manousis, currently under submission

Evolving data-intensive ecosystem



***Which** parts are affected, **how** exactly are they affected and, how can we intervene and **predetermine their reaction**?*

Policies to predetermine reactions




```
DATABASE: ON ADD_ATTRIBUTE              TO RELATION THEN PROPAGATE;
DATABASE: ON ADD_CONDITION              TO RELATION THEN PROPAGATE;
DATABASE: ON ADD_RELATION               TO RELATION THEN PROPAGATE;
DATABASE: ON DELETE_ATTRIBUTE           TO RELATION THEN PROPAGATE;
DATABASE: ON DELETE_CONDITION           TO RELATION THEN PROPAGATE;
DATABASE: ON DELETE_RELATION            TO RELATION THEN PROPAGATE;
DATABASE: ON MODIFYDOMAIN_ATTRIBUTE     TO RELATION THEN PROPAGATE;
DATABASE: ON RENAME_ATTRIBUTE           TO RELATION THEN PROPAGATE;
DATABASE: ON MODIFY_CONDITION           TO RELATION THEN PROPAGATE;
DATABASE: ON RENAME_RELATION            TO RELATION THEN PROPAGATE;


DATABASE: ON ADD_ATTRIBUTE              TO VIEW THEN BLOCK;
DATABASE: ON ADD_CONDITION              TO VIEW THEN BLOCK;
DATABASE: ON ADD_RELATION               TO VIEW THEN BLOCK;
DATABASE: ON DELETE_ATTRIBUTE           TO VIEW THEN BLOCK;
DATABASE: ON DELETE_CONDITION           TO VIEW THEN BLOCK;
DATABASE: ON DELETE_RELATION            TO VIEW THEN BLOCK;
DATABASE: ON MODIFYDOMAIN_ATTRIBUTE     TO VIEW THEN BLOCK;
DATABASE: ON RENAME_ATTRIBUTE           TO VIEW THEN BLOCK;
DATABASE: ON MODIFY_CONDITION           TO VIEW THEN BLOCK;
DATABASE: ON RENAME_RELATION            TO VIEW THEN BLOCK;

DATABASE: ON ADD_ATTRIBUTE              TO QUERY THEN BLOCK;
DATABASE: ON ADD_CONDITION              TO QUERY THEN BLOCK;
DATABASE: ON ADD_RELATION               TO QUERY THEN BLOCK;
DATABASE: ON DELETE_ATTRIBUTE           TO QUERY THEN BLOCK;
DATABASE: ON DELETE_CONDITION           TO QUERY THEN BLOCK;
DATABASE: ON DELETE_RELATION            TO QUERY THEN BLOCK;
DATABASE: ON MODIFYDOMAIN_ATTRIBUTE     TO QUERY THEN BLOCK;
DATABASE: ON RENAME_ATTRIBUTE           TO QUERY THEN BLOCK;
DATABASE: ON MODIFY_CONDITION           TO QUERY THEN BLOCK;
DATABASE: ON RENAME_RELATION            TO QUERY THEN BLOCK;
```

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

Overview of solution

- 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

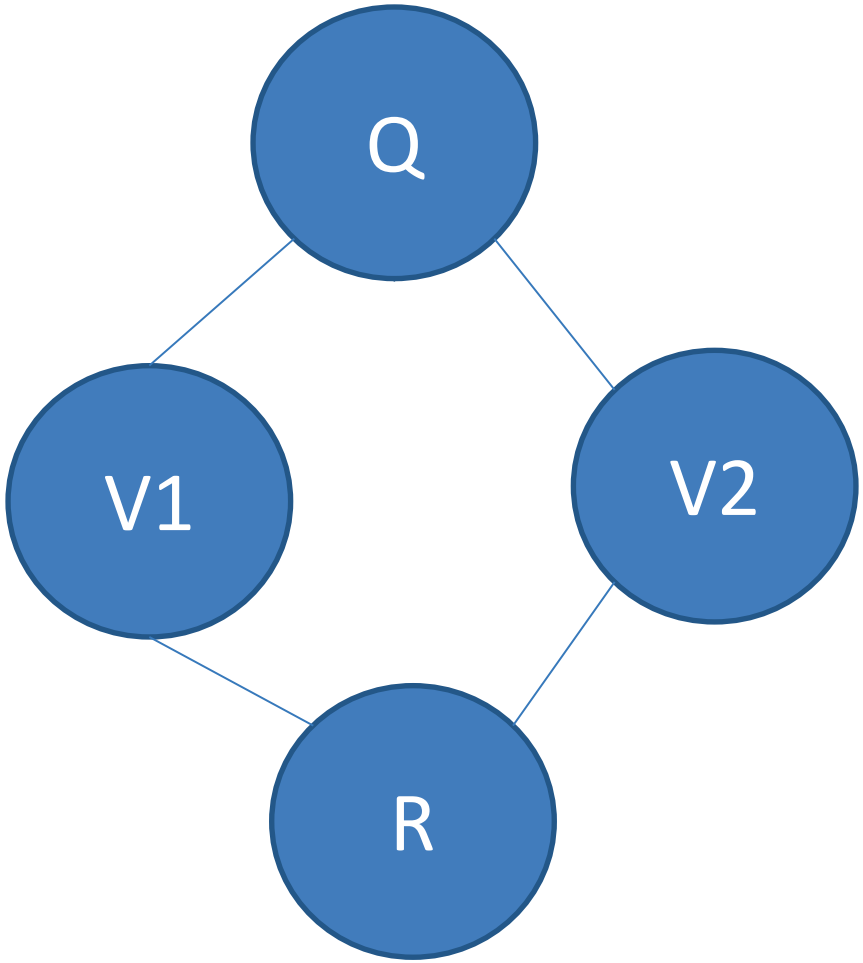
Background
Status Determination
Path check
Rewriting
Experiments and Results

Status Determination: who is affected and how

Correctness of “event flooding”

How do we **guarantee** that when a change occurs at the source nodes of the AG, this is **correctly** propagated to the end nodes of the graph?

- We notify exactly the nodes that should be notified
- The status of a node is determined independently of how messages arrive at the node
- Without infinite looping – i.e., termination

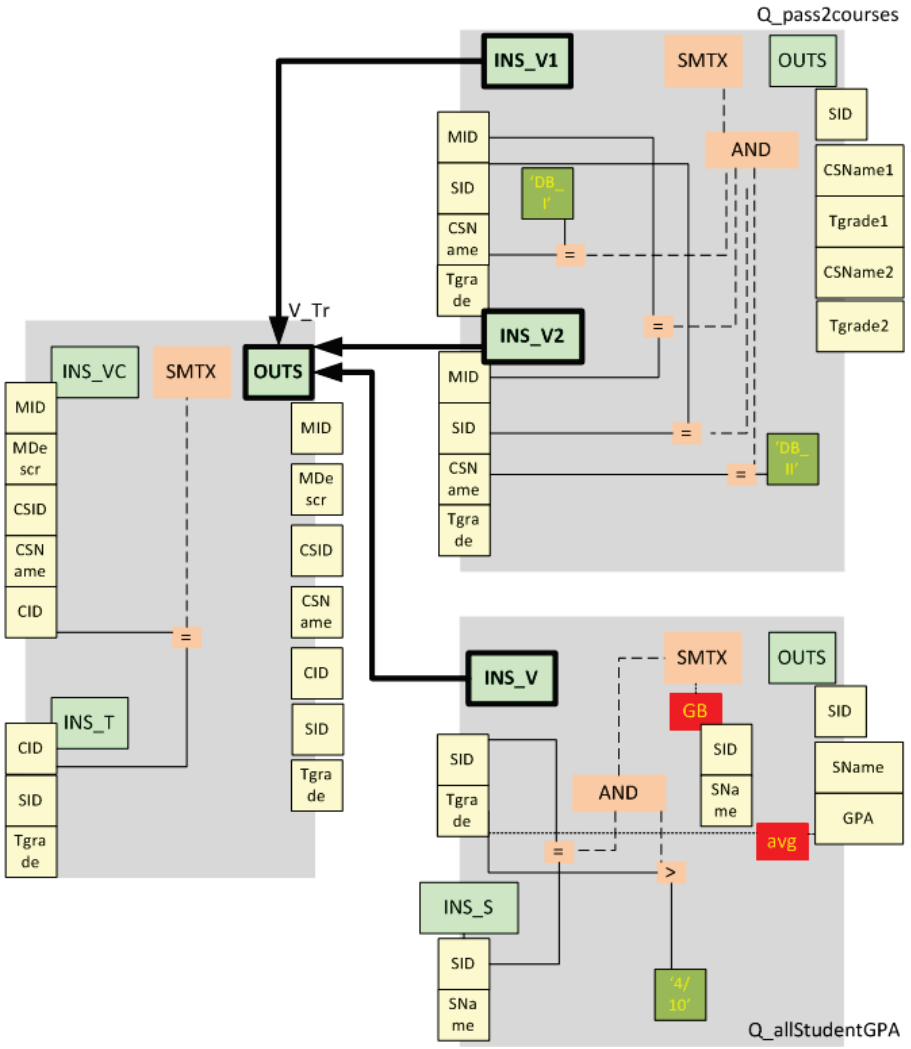


Method at a glance

1. Topologically sort the graph
2. Visit affected modules with its topological order and process its incoming messages for it.
3. Principle of locality: process locally the incoming messages and make sure that within each module
   - Affected internal nodes are appropriately highlighted
   - The reaction to the event is determined correctly
   - If the final status is not a veto, notify appropriately the next modules

Propagation mechanism

- Modules communicate with each other via a single means: the schema of a provider module notifies the input schema of a consumer module when this is necessary
- Two levels of propagation:
  - Graph level: At the module level, we need to determine the order and mechanism to visit each module
  - Intra-module level: within each module, we need to determine the order and mechanism to visit the module's components and decide who is affected and how it reacts + notify consumers



Algorithm 2 Status determination algorithm

Input: A topologically sorted architecture graph summary $\mathbf{G_s}(\mathbf{V_s}, \mathbf{E_s})$ (output of algorithm 1), a global queue Q that facilitates the exchange of messages between modules

Ouput: A list of modules $Affected\ Modules \subseteq \mathbf{V_s}$ that were affected by the event and acquire a status other than NO_STATUS

```
1:  function SetStatus(Module, Messages)
2:      Consumers Messages = ∅;
3:      for all Message ∈ Messages do
4:          decide status of Module;
5:          put messages for Module's consumers in Consumers Messages;
6:      end for
7:  end function
8:  Begin
9:      for all node ∈ G_s(V_s, E_s) do
10:         node.status = NO_STATUS;
11:     end for
12:     while size(Q) > 0 do
13:         visit module (node) in head of Q;
14:         insert node in Affected Modules list;
15:         get all messages, Messages, that refer to node;
16:         SetStatus(node, Messages);
17:         if node.status == PROPAGATE then
18:             insert node.Consumers Messages to the Q;
19:         end if
20:     end while
21:     return Affected Modules;
22: End
```

Theoretical Guarantees

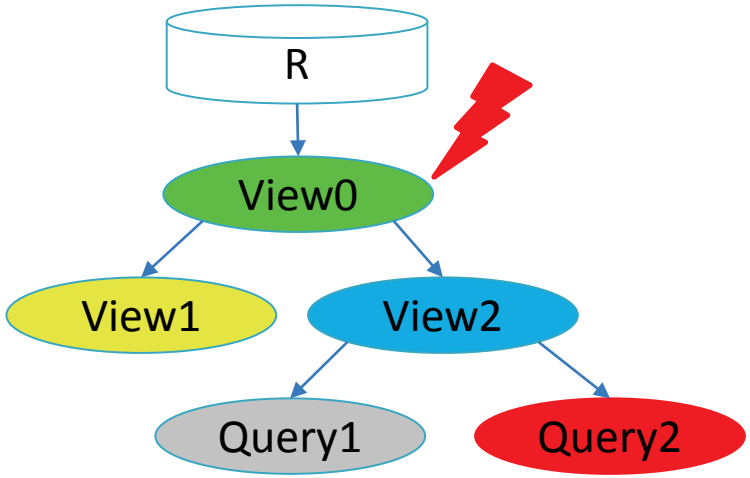
- At the inter-module level
  - Theorem 1 (termination). The message propagation at the inter-module level terminates.
  - Theorem 2 (unique status). Each module in the graph will assume a unique status once the message propagation terminates.
  - Theorem 3 (correctness). Messages are correctly propagated to the modules of the graph
- At the intra-module level
  - Theorem 4 (termination and correctness). The message propagation at the intra-module level terminates and each node assumes a status.

Background
Status Determination
Path check
Rewriting
Experiments and Results

Path Check: handling policy conflicts

Conflicts: what they are and how to handle them

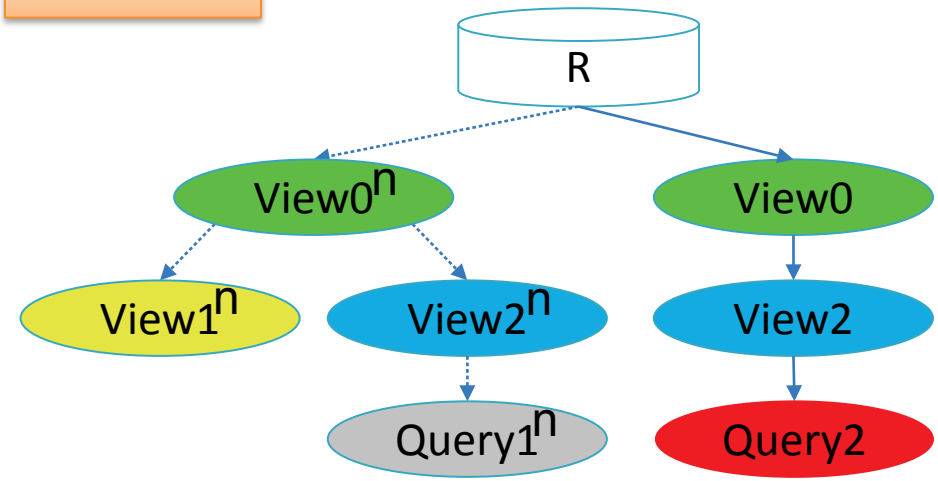
BEFORE



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

- Query2 rejects the change
- Query1 accepts the change

AFTER



- 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

Path Check algorithm

Algorithm 3 Path check algorithm

Input: A summary of an architecture graph $\mathbf{G_s}(\mathbf{V_s}, \mathbf{E_s})$, a list of modules $Affected\ modules$, that were affected by the event (output of algorithm 2)

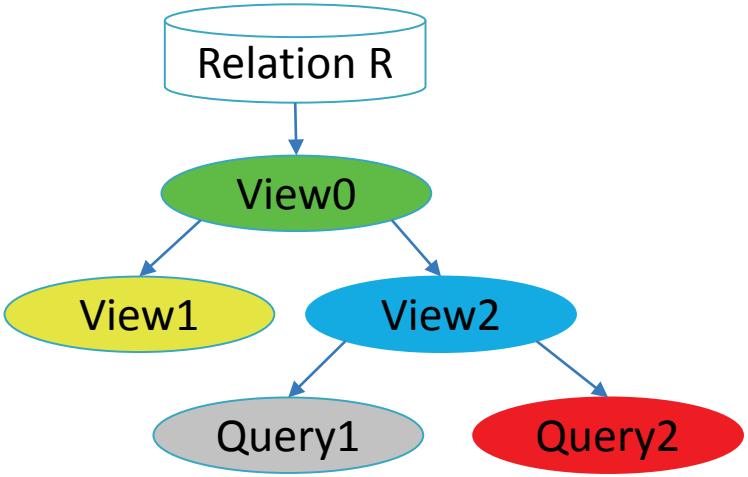
Ouput: Annotation of the modules of $Affected\ modules$ on the action needed to take, and specifically whether we have to make a new version of it, or, implement the change the user asked on the current version

```
1:  function CheckModule(Module, Affected modules)
2:      if Module has been marked then
3:          return;                                   ▷ notified by previous block path
4:      end if
5:      mark Module to keep current version and apply the change on a clone;
6:      for all New module ∈ Affected modules feeding Module do
7:          CheckModule(New module, Affected modules);          ▷ notify path
8:      end for
9:  end function
10: Begin
11:     for all Module ∈ Affected modules do
12:         if Module.status == BLOCK then
13:             CheckModule(Module, Affected modules);
14:             mark Module not to change;          ▷ blockers keep only current version
15:         end if
16:     end for
17: End
```

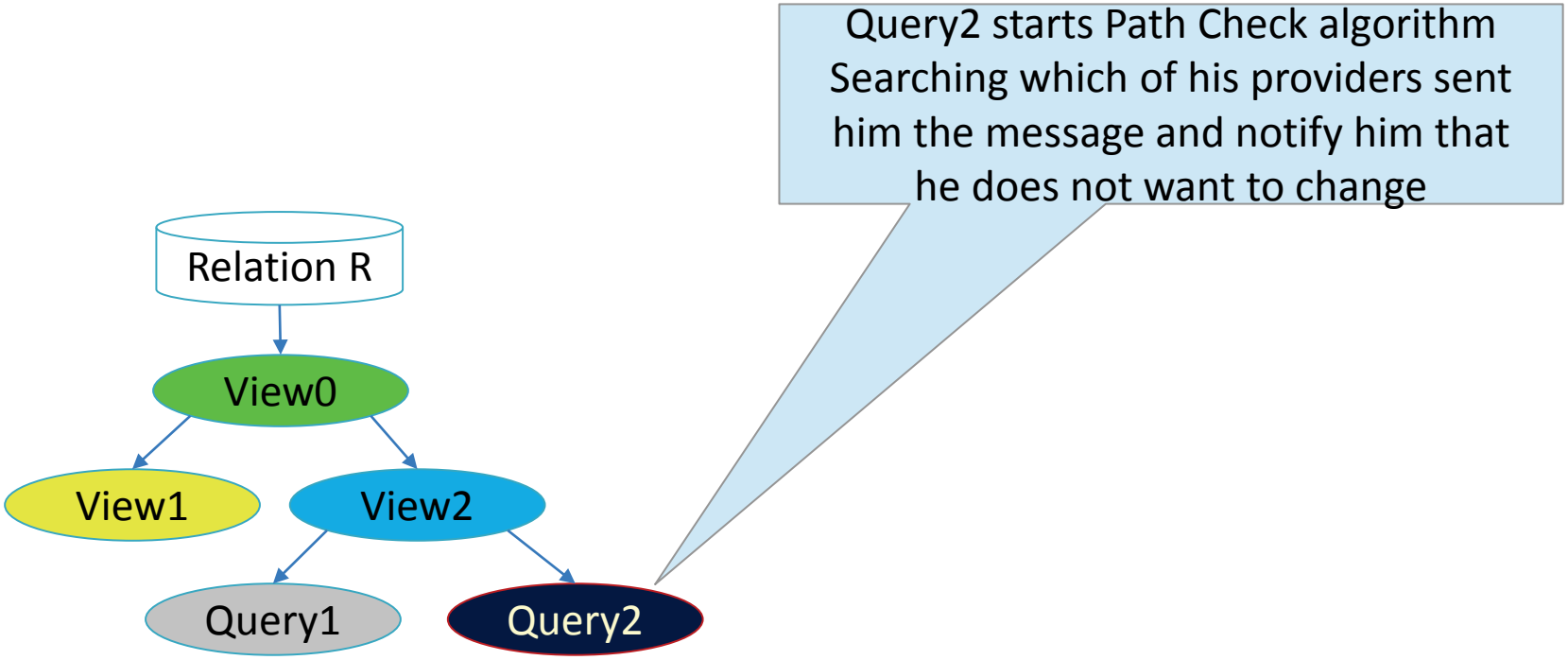
Path Check

- If there exists any Block Module we travel in reverse the Architecture Graph from blocker node to initiator of change
- In each step we inform the Module to keep current version and produce a new one adapting to the change
- We inform the blocker node that it should not change at all.

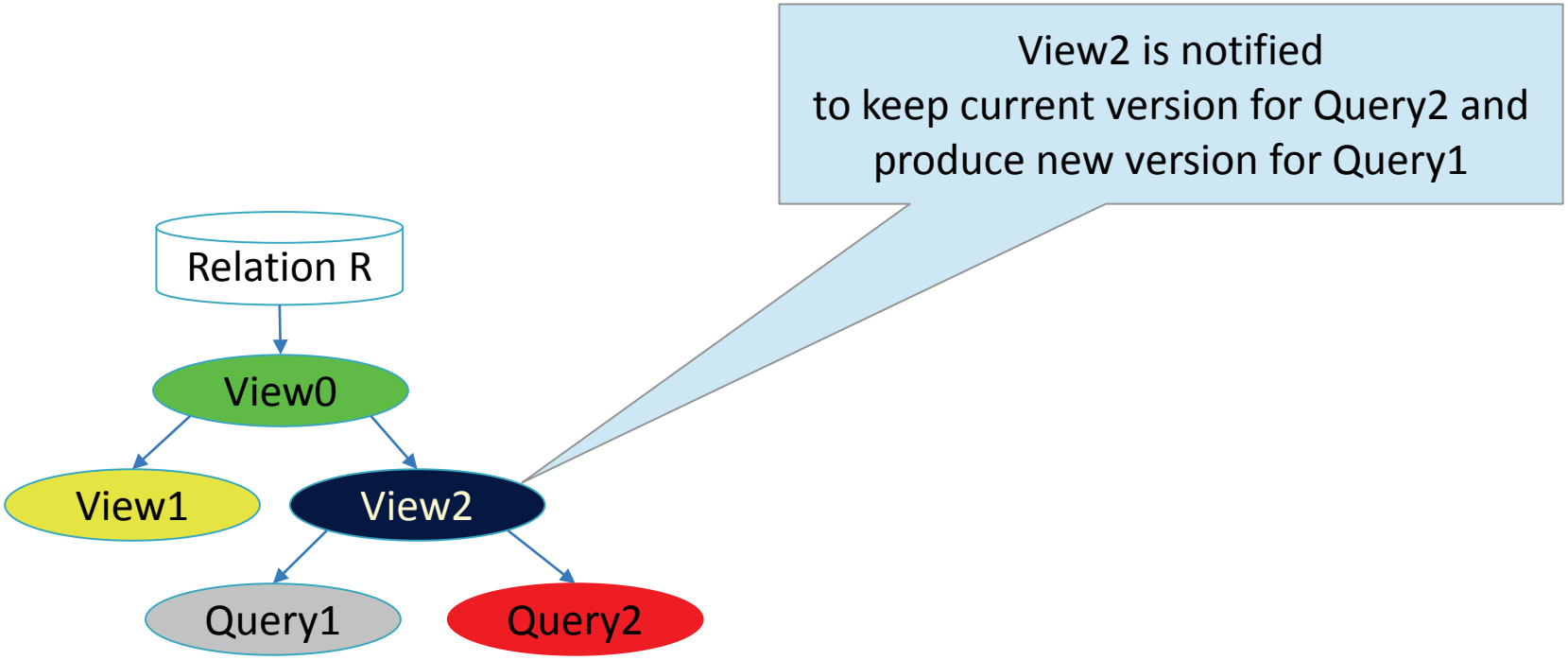
Path Check



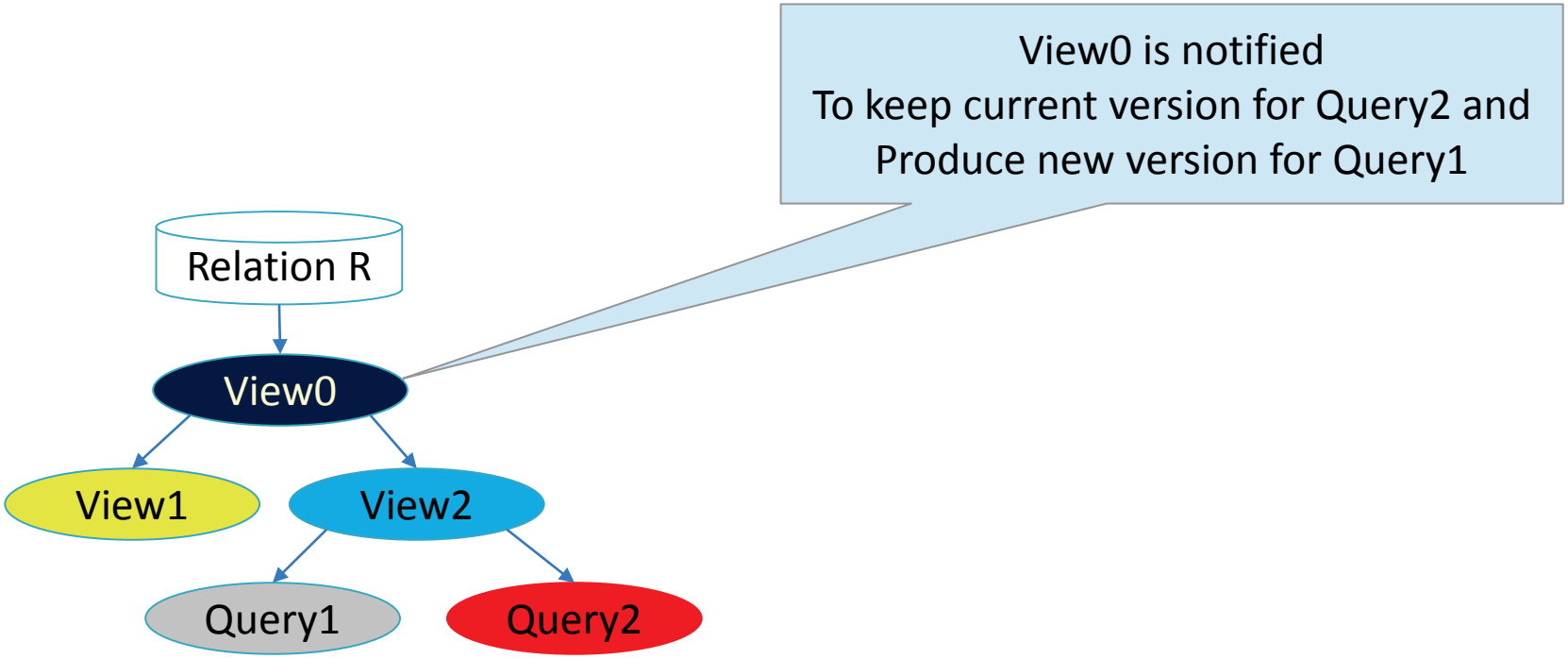
Path Check


Query2 starts Path Check algorithm
Searching which of his providers sent him the message and notify him that he does not want to change
Relation R
View0
View1
View2
Query1
Query2

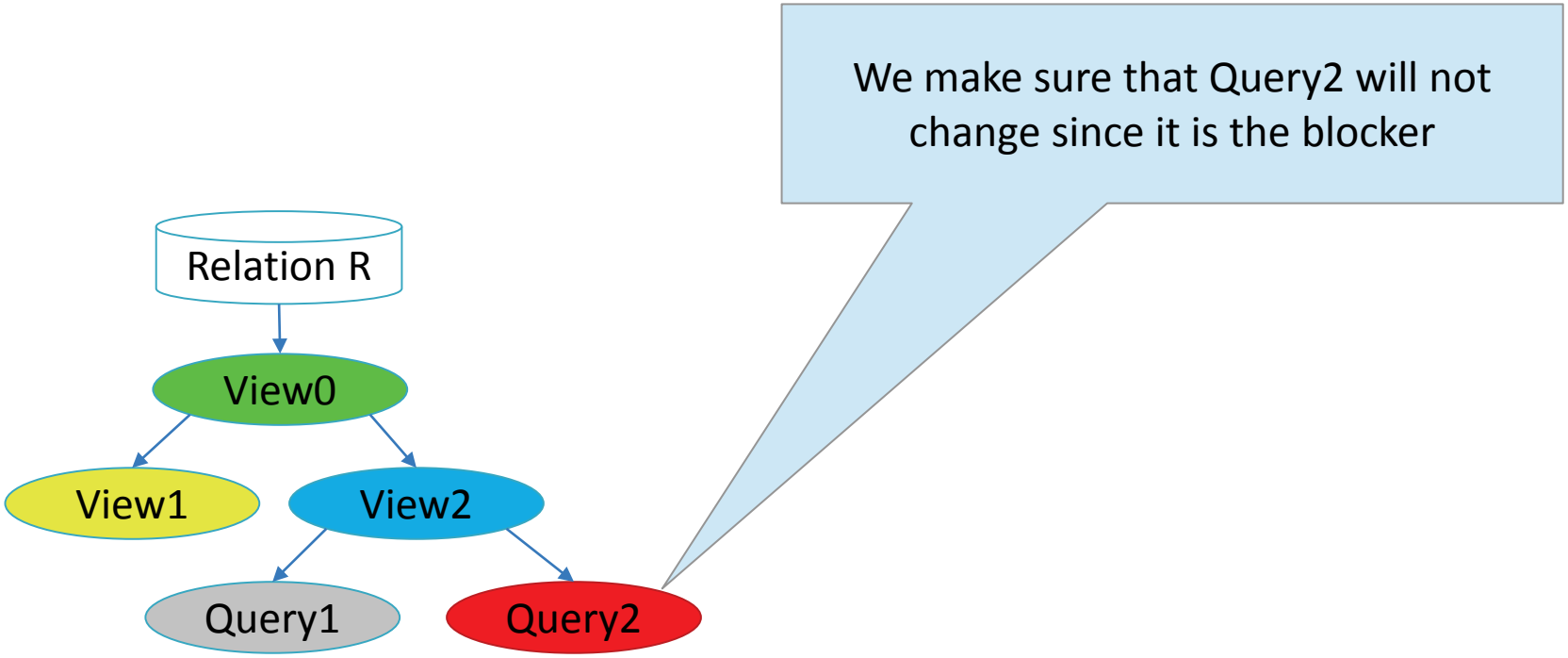
Path Check


View2 is notified
to keep current version for Query2 and
produce new version for Query1
Relation R
View0
View1
View2
Query1
Query2

Path Check


View0 is notified
To keep current version for Query2 and
Produce new version for Query1
Relation R
View0
View1
View2
Query1
Query2

Path Check


We make sure that Query2 will not change since it is the blocker
Relation R
View0
View1
View2
Query1
Query2

Background
Status Determination
Path check
Rewriting
Experiments and Results

Rewriting: once we identified affected parts and resolved conflicts, how will the ecosystem look like?

Rewriting algorithm

Algorithm 4 Rewriting algorithm

Input: A list of modules $Affected\ modules$, knowing the number of versions they have to retain (output of algorithm 3), initial messages of $Affected\ modules$

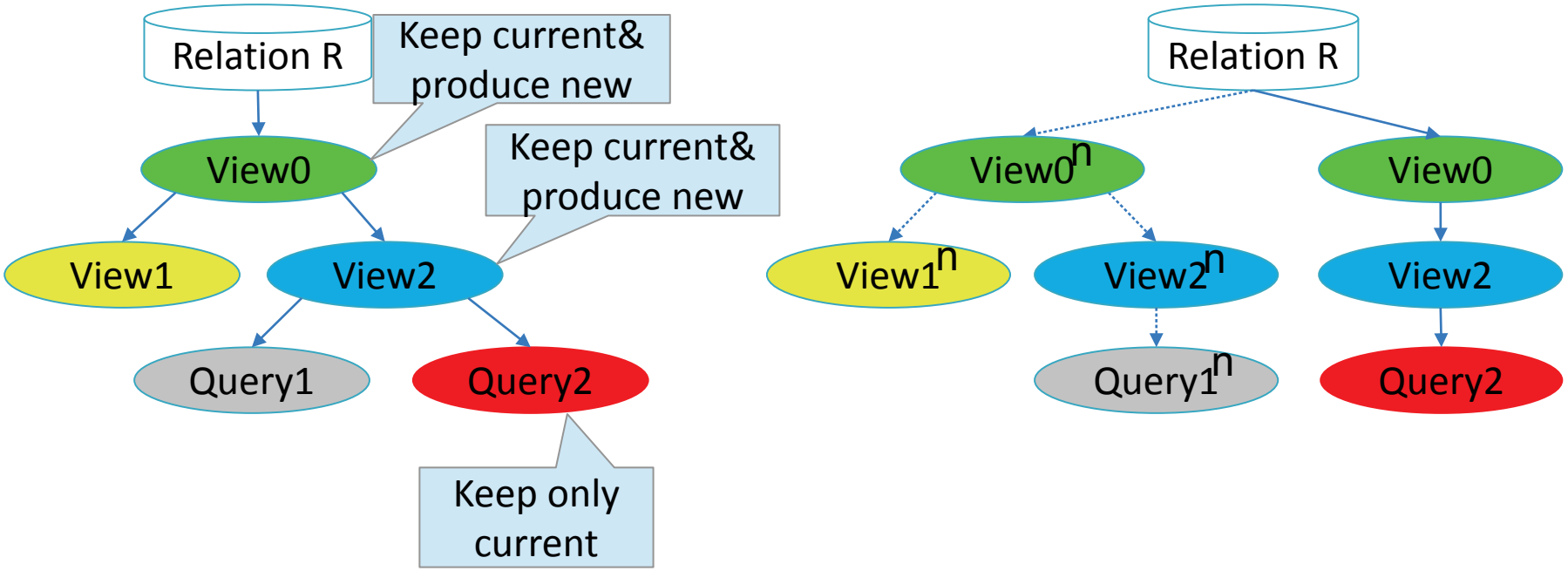
Ouput: Architecture graph after the implementation of the change the user asked

```
1:  Begin
2:      if any of Affected modules has status BLOCK then
3:          if initial message started from Relation module type then
4:              return ;                                  ▷ Relations do not change at all
5:          else
6:              for all Module ∈ Affected modules do
7:                  if Module needs only new version then
8:                      proceed with rewriting of Module;
9:                      connect Module to new providers;      ▷ new version goes to new path
10:                 else
11:                     clone Module;                  ▷ clone module, to keep both versions
12:                     connect cloned Module to new providers;   ▷ clone is the new version
13:                     proceed with rewriting of cloned Module;
14:                 end if
15:             end for
16:         end if
17:     else
18:         for all Module ∈ Affected modules do
19:             proceed with rewriting of Module                  ▷ no blocker node;
20:         end for
21:     end if
22: End
```

Rewriting

- If there is no Block, we perform the rewriting.
- If there is Block
  - If the change initiator is a relation we stop further processing.
  - Otherwise:
    - We clone the Modules that are part of a block path and were informed by Path Check and we perform the rewrite on the clones
    - We perform the rewrite on the Module if it is not part of a block path.
- Within each module, all its internals are appropriately adjusted (attribute / selection conditions / … additions and removals)

Rewriting


Relation R
Keep current& produce new
View0
Keep current& produce new
View1
View2
Query1
Query2
Keep only current
Relation R
View0n
View0
View1n
View2n
View2
Query1n
Query2

Background
Status Determination
Path check
Rewriting
Experiments and Results

Experiments and results

Experimental setup

- University database ecosystem (the one of we used in previous slides, consisting of 5 relations, 2 views and 2 queries)
- TPC-DS ecosystem (consisting of 15 relations, 5 views and 27 queries) where we used two workloads of events
  - WL1 with changes mainly at tables
  - WL2 with changes mainly at views

- Policies used (for both ecosystems):
  - propagate all policy and
  - mixture policy (20% blockers)

- Measurements: effectiveness & cost

Impact & adaptation assessment for TPC-DS

	Impact assessment				Adaptation assessment		
Event:Node	**AM**	**% AM**	**AI**	**% AI**	**NM**	**ERM**	**RM**
DS:WEB_SALES	0	**100**	6	99.59	0	1	**1**
RS:CUSTOMER_DEMOGRAPHICS.CD_DEMO_SK	3	**90.63**	8	99.46	0	4	**4**
RS:VIEW38.C_LAST_NAME	2	**93.75**	4	99.73	1	1	**2**
RS:CUSTOMER_TOTAL_RET.CTR_TOTAL_RETURN	2	**93.94**	4	99.73	1	1	**2**
RS:CUSTOMER_TOTAL_RETRN.CTR_TOTAL_RETURN	2	**94.12**	6	99.6	1	1	**2**
AS:VIEW38	2	**94.29**	2	99.87	1	1	**2**
AS:CUSTOMER_TOTAL_RET	3	**91.67**	3	99.81	1	2	**3**
AS:CUSTOMER_TOTAL_RETRN	3	**91.89**	3	99.81	1	2	**3**
AA:VIEW38	2	**94.74**	4	99.75	1	1	**2**
AA:Q18	1	**97.44**	1	99.94	0	1	**1**
DS:Q18	1	**97.44**	3	99.82	0	1	**1**
DS:CUSTOMER_DEMOGRAPHICS	3	**92.11**	34	97.9	0	4	**4**
RS:ITEM	10	**73.68**	11	99.32	0	0	**0**
RS:PROMOTION	2	**94.74**	3	99.81	0	3	**3**

Impact & adaptation assessment

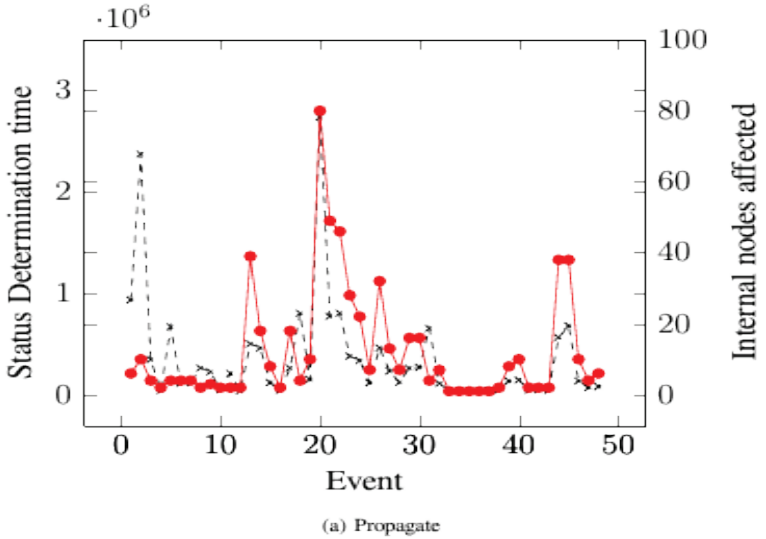
		Impact assessment				Adaptation assessment		
		AM	**% AM**	**AI**	**% AI**	**NM**	**ERM**	**RM**
Minimum	University ecosystem propagate all	1	**0**	2	66.25	0	2	**2**
Maximum		4	**75**	54	98.37	0	5	**5**
Average		2.79	**30.36**	12.14	91.47	0	3.57	**3.57**
Minimum	University ecosystem mixture	0	**0**	1	66.14	0	0	**0**
Maximum		7	**100**	64	99.31	2	7	**7**
Average		3.86	**28.01**	15.86	90.19	0.21	1.64	**1.86**
Minimum	TPC-DS workload 1 propagate all	0	**22.58**	1	94.44	0	1	**1**
Maximum		24	**100**	80	99.94	0	25	**25**
Average		3.88	**87.51**	12.46	99.19	0	4.56	**4.56**
Minimum	TPC-DS workload 1 mixture	0	**21.21**	1	94.2	0	0	**0**
Maximum		26	**100**	86	99.94	1	4	**4**
Average		3.92	**88.22**	12.63	99.15	0.13	1.02	**1.15**
Minimum	TPC-DS workload 2 propagate	0	**67.74**	1	97.68	0	1	**1**
Maximum		10	**100**	34	99.93	0	11	**11**
Average		2.57	**91.86**	6.57	99.55	0	2.93	**2.93**
Minimum	TPC-DS workload 2 mixture	0	**73.68**	1	97.9	0	0	**0**
Maximum		10	**100**	34	99.94	1	4	**4**
Average		2.57	**92.89**	6.57	99.58	0.5	1.64	**2.14**

Cost analysis

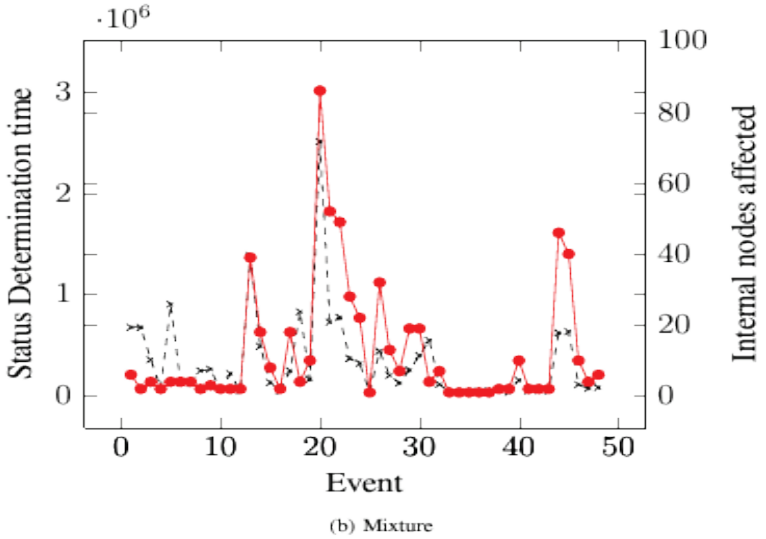
	Average time (nanosecs)			
	Status Determination	Path Check	Rewriting	Total
Propagate all	358161	4947	367071	730179
Mixture	327488	18340	341735	687563
	Percentage Breakdown			
	Status Determination	Path Check	Rewriting	
Propagate all	49%	1%	50%	
Mixture	48%	2%	50%	

- The results of TPC-DS ecosystem in workload 1
- Path check nearly no cost at all, but in 20% blockers doubled its value

Status Determination Cost



(a) Propagate

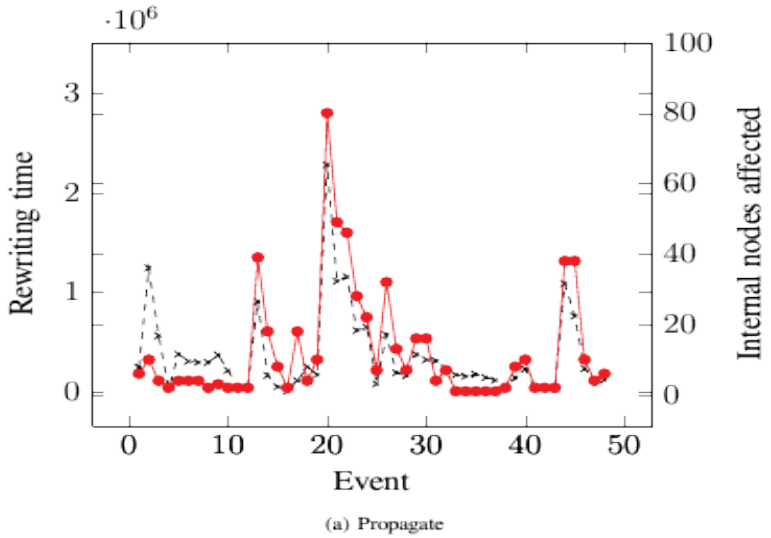


(b) Mixture

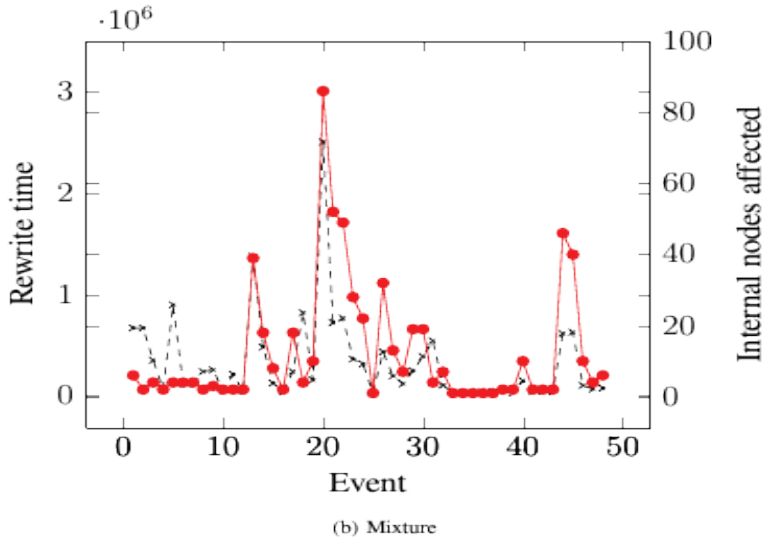
Blue line: time
Red line: affected nodes

Slightly slower time in mixture mode due to blockers.

Rewrite Cost



(a) Propagate

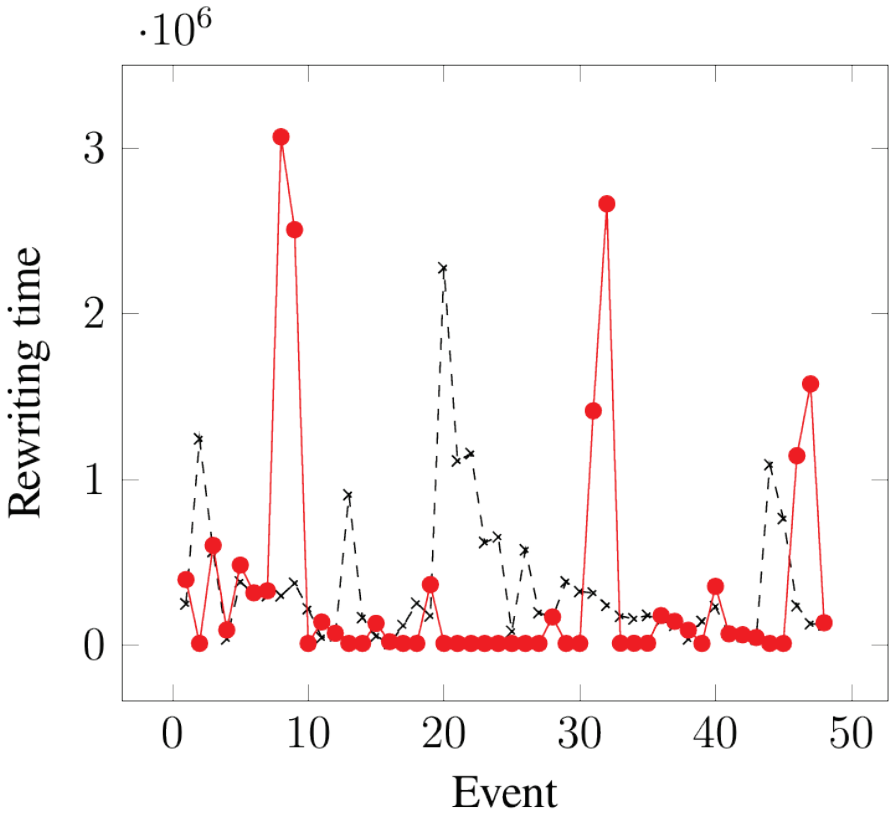


(b) Mixture

Blue line: time
Red line: affected nodes

Due to blockers and workload containing mostly relation changes, we have no rewrites in mixture mode in a set of events

Rewrite time comparison



- Peaks of red are due to cloning of modules.
- Valleys of red are due to blockers at a relation related event.

Lessons Learned #1

- Users gain up to 90% of effort.
- Even in really cohesive environments, users gain at least 25% of effort.
- When all modules propagate changes, on average there are 3.5 modules that rewrite themselves.

Lessons Learned #2

- “Popular” modules need more time to process compared to unpopular ones.
- Module-cloning costs more than other tasks
- But since the time is measured in nanoseconds this is not big deal

Wrapping things up

In a nutshell

- Studying the evolution of ecosystems is important
  - Not just the database; the surrounding applications too!
  - Case studies are important (and very rare!!)
  - Reducing unnecessary schema elements can help us reduce the impact of maintaining applications in the presence of changes
- Managing the evolution of ecosystems is possible
  - We need to model the ecosystem and annotate it with evolution management techniques that dictate its reaction to future events
  - We can highlight who is impacted and if there is a veto or not.
  - We can handle conflicts, suggest automated rewritings and guarantee correctness
  - We can do it fast and gain effort for all involved stakeholders

Selected readings

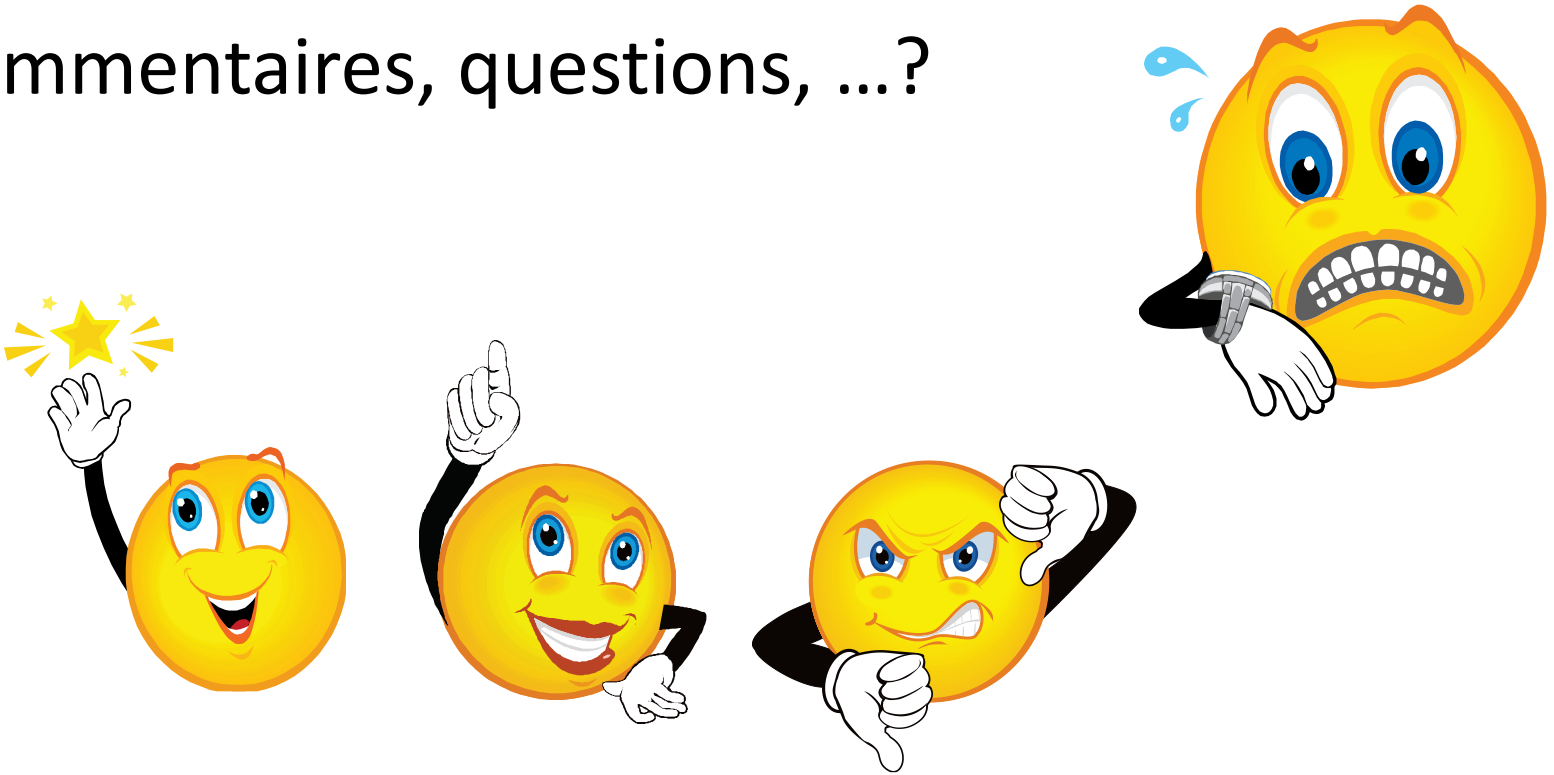
- Matteo Golfarelli, Stefano Rizzi: A Survey on Temporal Data Warehousing. International Journal of Data Warehousing and Mining, Volume 5, Number 1, 2009, p. 1-17
- Robert Wrembel: A Survey of Managing the Evolution of Data Warehouses. International Journal of Data Warehousing and Mining, Volume 5, Number 2, 2009, p. 24-56
- Michael Hartung,James Terwilliger,Erhard Rahm: Recent Advances in Schema and Ontology Evolution. In: Zohra Bellahsene, Angela Bonifati, Erhard Rahm (Eds.): Schema Matching and Mapping. Springer 2011, ISBN 978-3-642-16517-7

Some thoughts for future work

- Vision: come up with laws (i.e., recurring patterns) that govern the evolution of data-intensive ecosystems
  - More (a lot more) case studies needed!
- Visualization: graph modeling results in large graphs that are really hard to use interactively
- Coupling applications with the underlying databases (e.g., via plugging A.G. + policies inside db's or other repositories)
  - Useful to avoid unexpected crashes
  - Not without problems (too much coupling can hurt)
  - Data warehouses pose a nice opportunity

Merci bien pour votre attention!

Commentaires, questions, …?



Auxiliary slides

Detailed experimental results

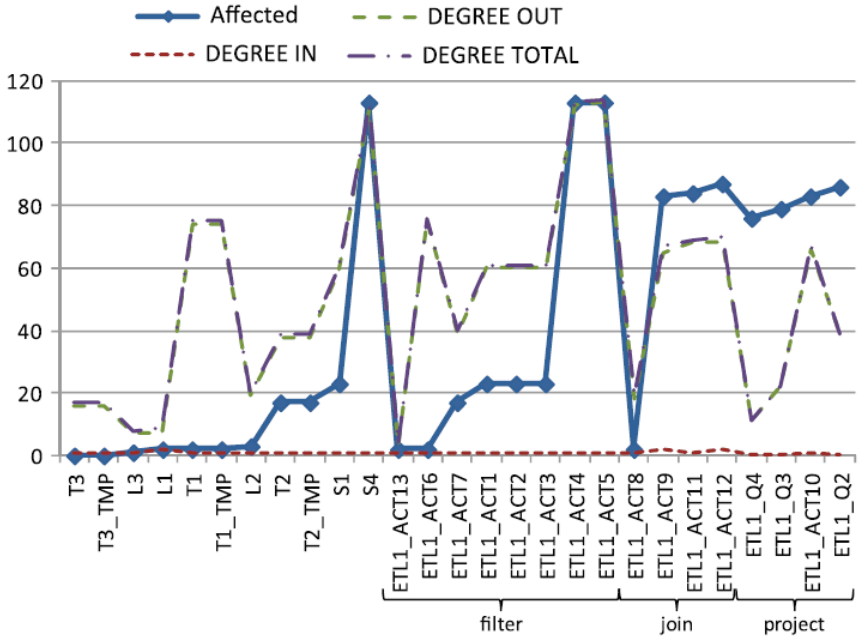


Fig. 4 Results for degree metrics for ETL1

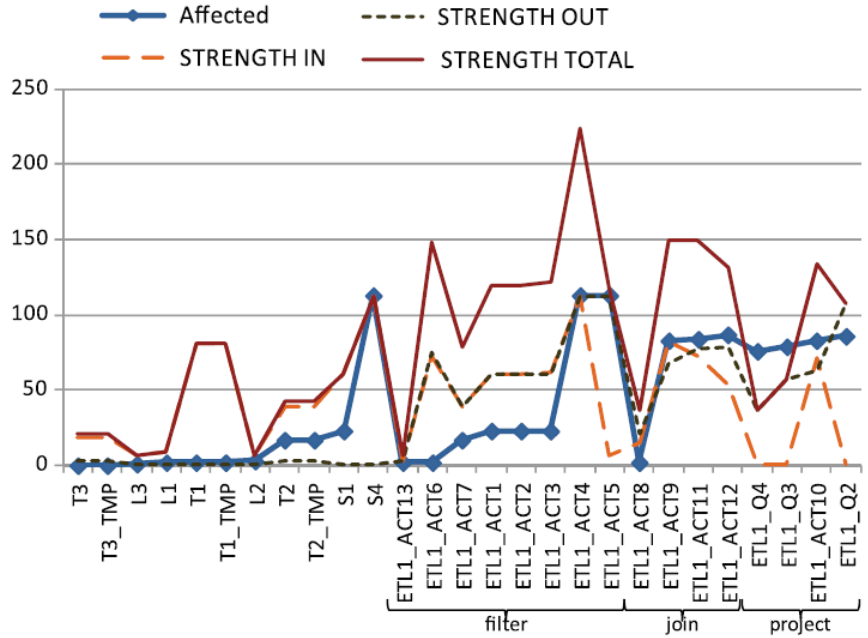


Fig. 5 Results for strength metrics for ETL1

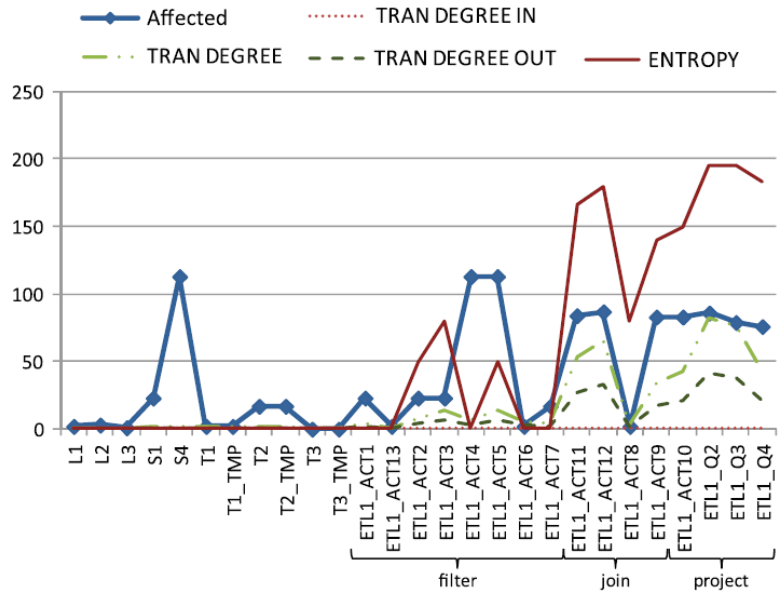


Fig. 6 Results for transitive degrees and entropy metrics for ETL1

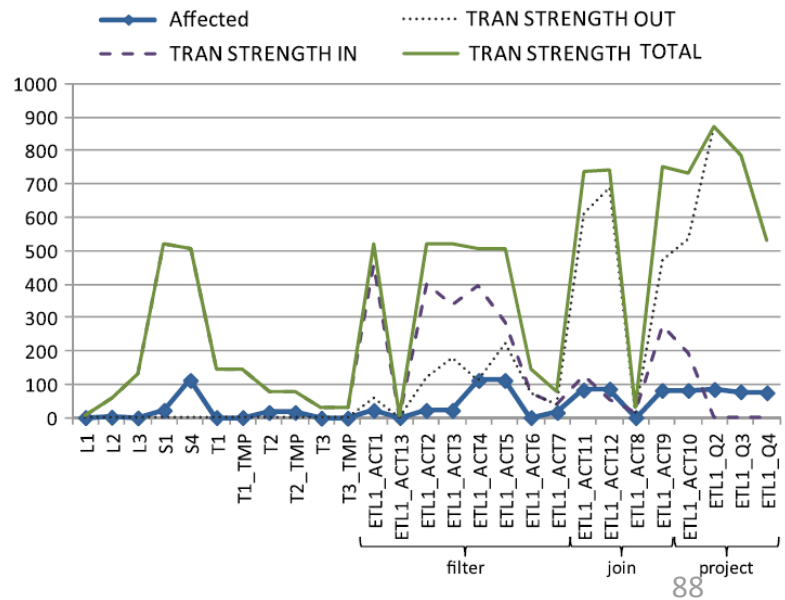
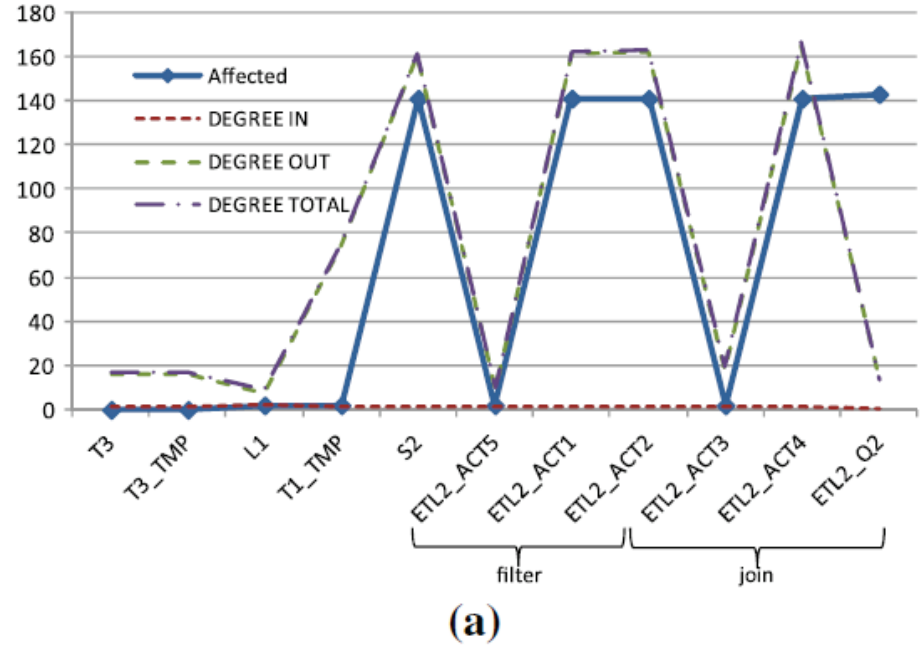
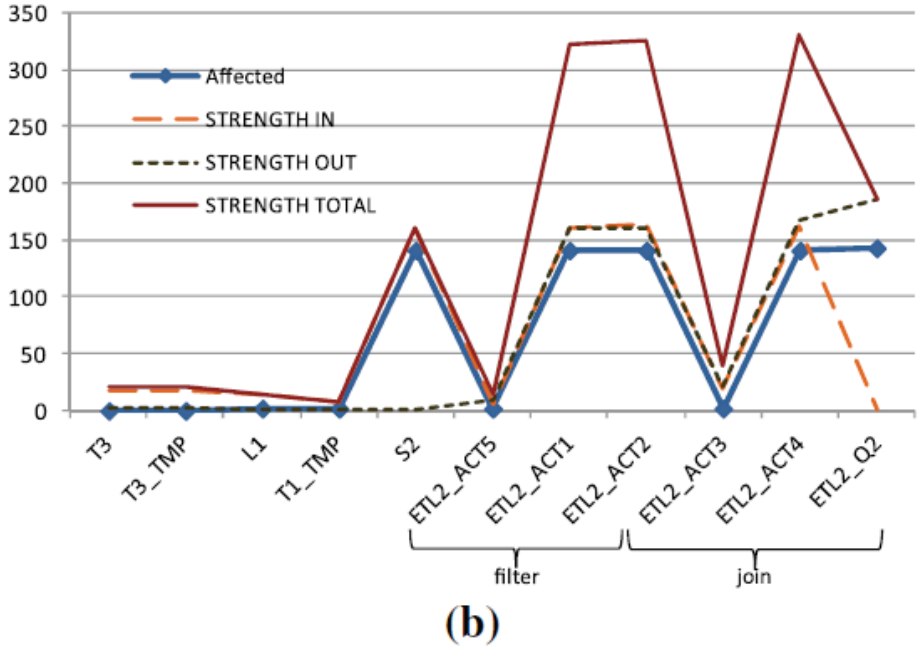


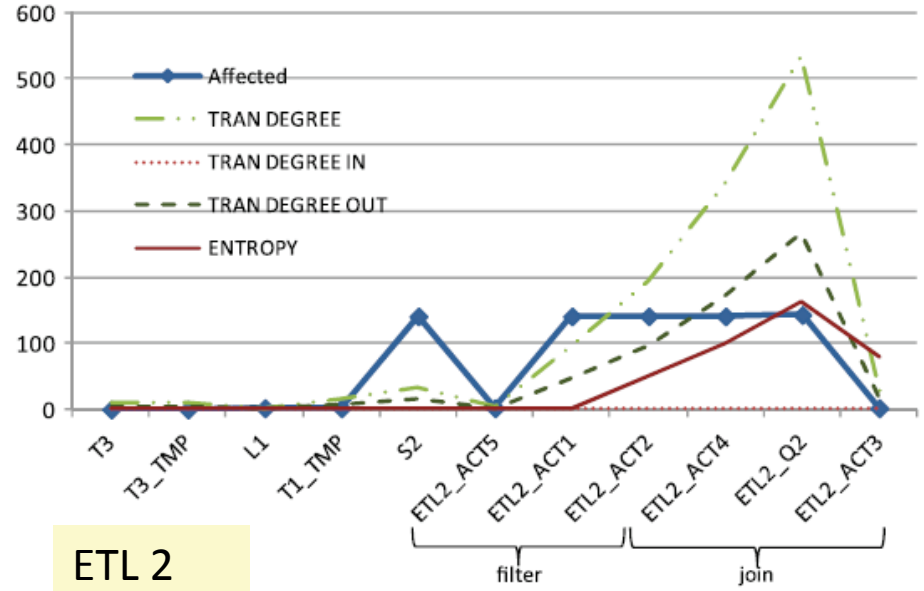
Fig. 7 Results for transitive strength metrics for ETL1


Affected
DEGREE IN
DEGREE OUT
DEGREE TOTAL
T3
T3_TMP
L1
T1_TMP
S2
ETL2_ACT5
ETL2_ACT1
ETL2_ACT2
ETL2_ACT3
ETL2_ACT4
ETL2_Q2
filter
join

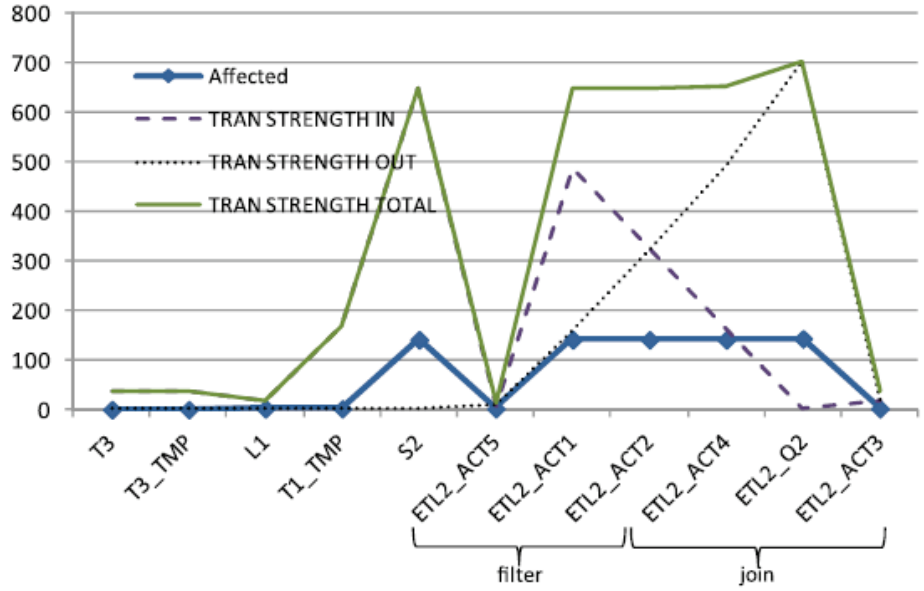
(a)


Affected
STRENGTH IN
STRENGTH OUT
STRENGTH TOTAL
T3
T3_TMP
L1
T1_TMP
S2
ETL2_ACT5
ETL2_ACT1
ETL2_ACT2
ETL2_ACT3
ETL2_ACT4
ETL2_Q2
filter
join

(b)

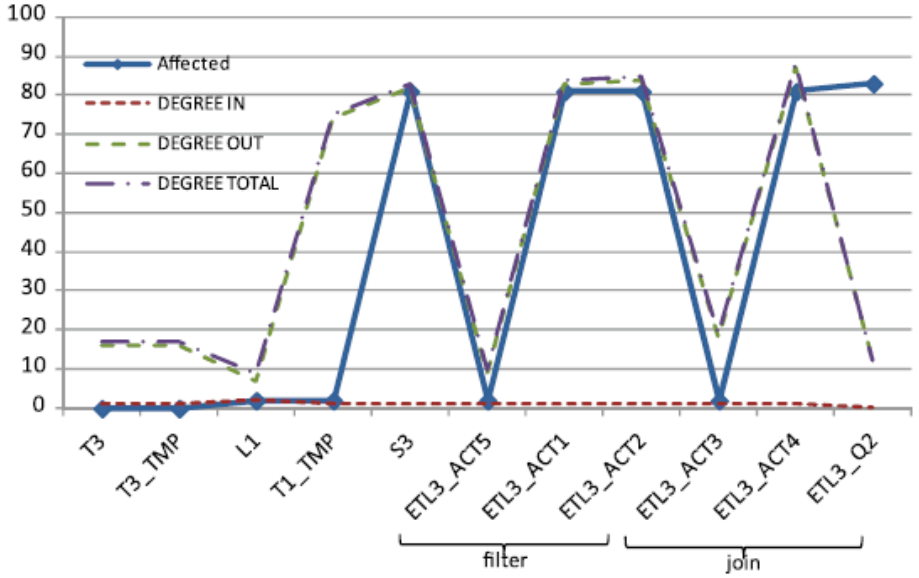

Affected
TRAN DEGREE
TRAN DEGREE IN
TRAN DEGREE OUT
ENTROPY
T3
T3_TMP
L1
T1_TMP
S2
ETL2_ACT5
ETL2_ACT1
ETL2_ACT2
ETL2_ACT4
ETL2_Q2
ETL2_ACT3
filter
join

(c)

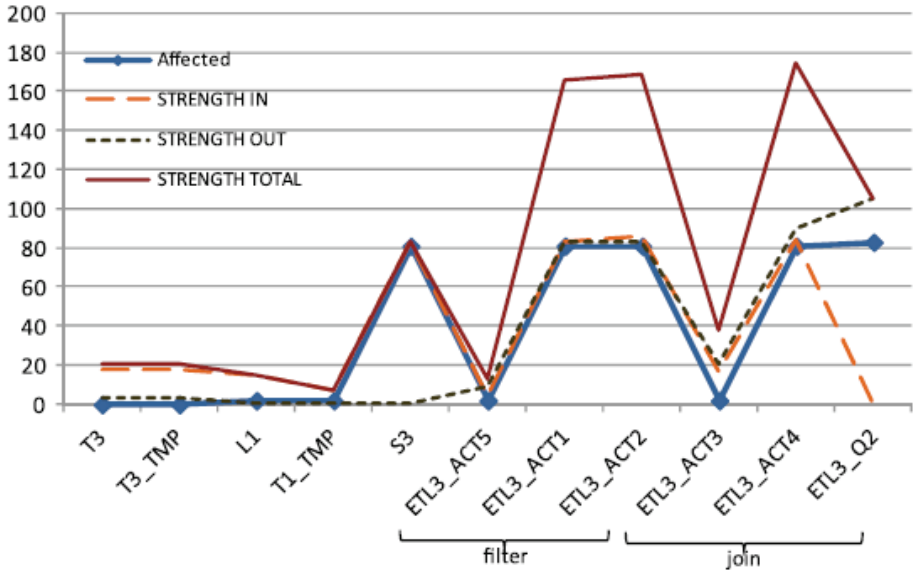

Affected
TRAN STRENGTH IN
TRAN STRENGTH OUT
TRAN STRENGTH TOTAL
T3
T3_TMP
L1
T1_TMP
S2
ETL2_ACT5
ETL2_ACT1
ETL2_ACT2
ETL2_ACT4
ETL2_Q2
ETL2_ACT3
filter
join

(d)

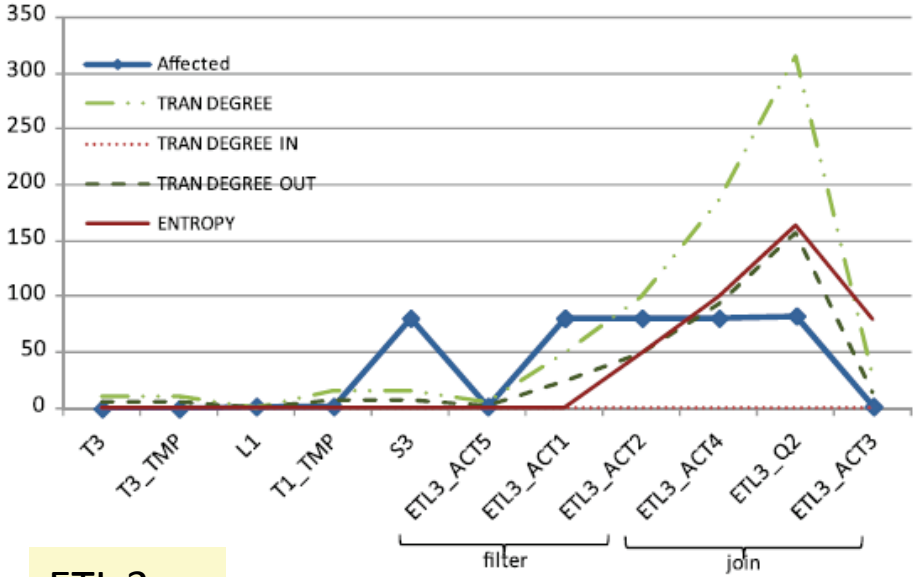
ETL 2


Affected
DEGREE IN
DEGREE OUT
DEGREE TOTAL
T3
T3_TMP
L1
T1_TMP
S3
ETL3_ACT5
ETL3_ACT1
ETL3_ACT2
ETL3_ACT3
ETL3_ACT4
ETL3_Q2
filter
join

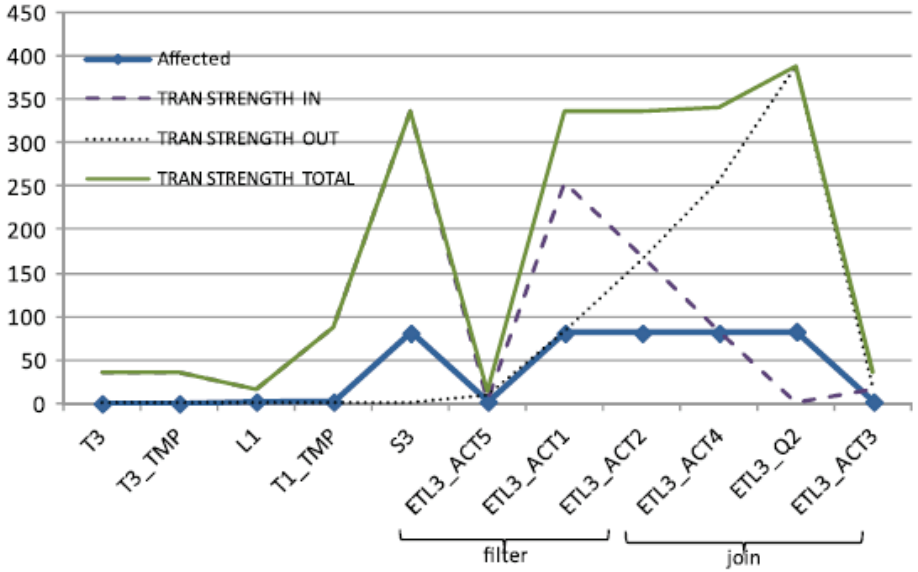
(a)


Affected
STRENGTH IN
STRENGTH OUT
STRENGTH TOTAL
T3
T3_TMP
L1
T1_TMP
S3
ETL3_ACT5
ETL3_ACT1
ETL3_ACT2
ETL3_ACT3
ETL3_ACT4
ETL3_Q2
filter
join

(b)

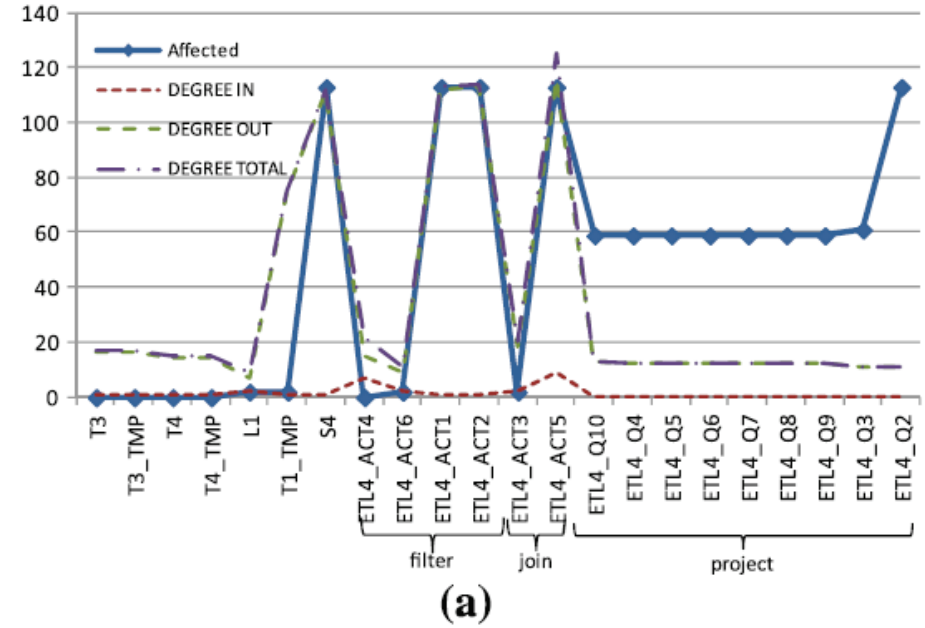

Affected
TRAN DEGREE
TRAN DEGREE IN
TRAN DEGREE OUT
ENTROPY
T3
T3_TMP
L1
T1_TMP
S3
ETL3_ACT5
ETL3_ACT1
ETL3_ACT2
ETL3_ACT4
ETL3_Q2
ETL3_ACT3
filter
join

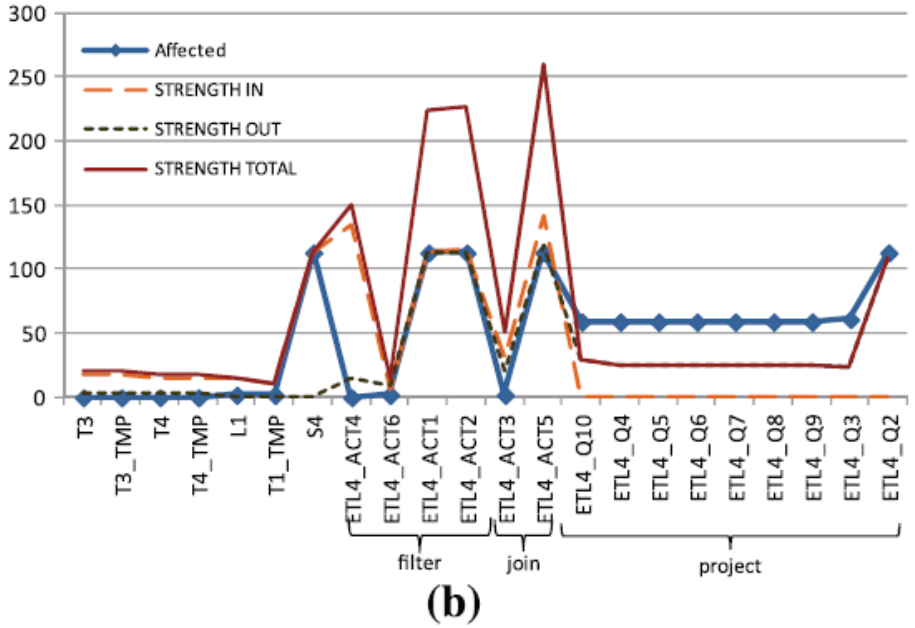
(c)

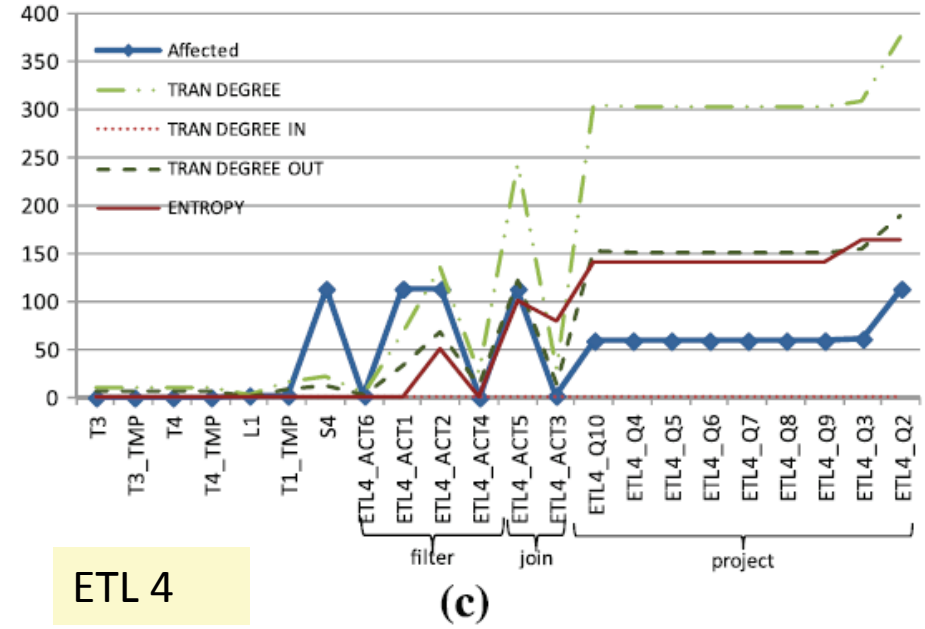

Affected
TRAN STRENGTH IN
TRAN STRENGTH OUT
TRAN STRENGTH TOTAL
T3
T3_TMP
L1
T1_TMP
S3
ETL3_ACT5
ETL3_ACT1
ETL3_ACT2
ETL3_ACT4
ETL3_Q2
ETL3_ACT3
filter
join

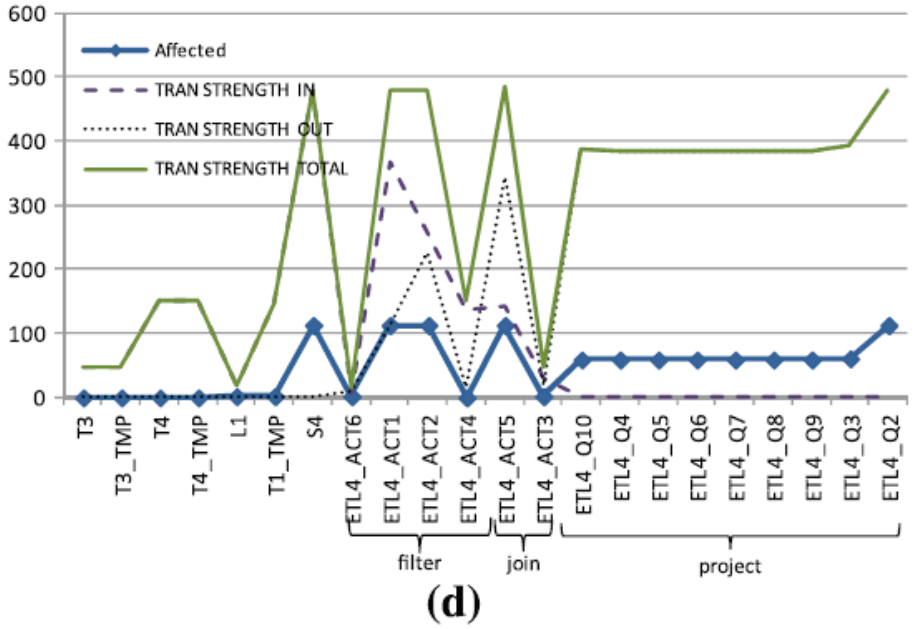
(d)

ETL 3

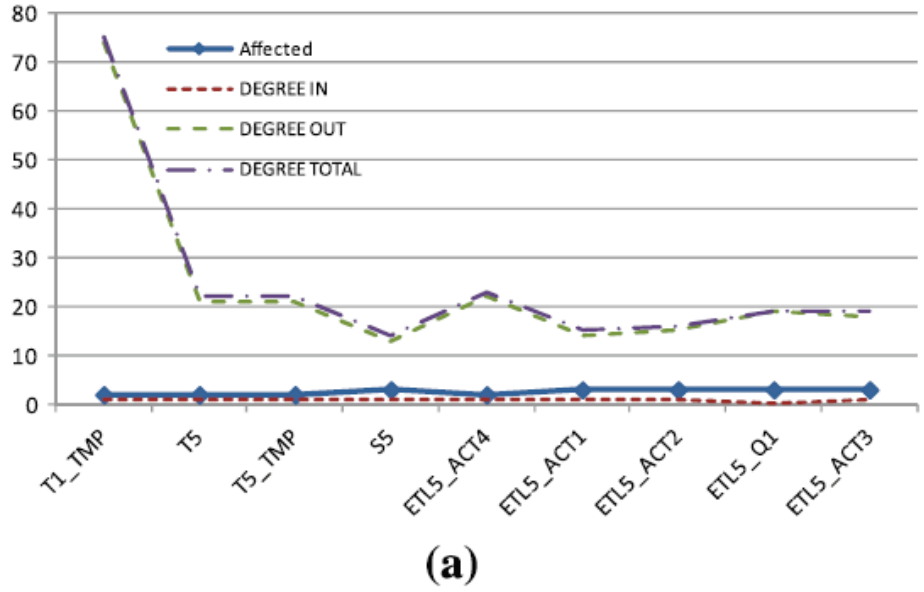

Affected
DEGREE IN
DEGREE OUT
DEGREE TOTAL
T3
T3_TMP
T4
T4_TMP
L1
T1_TMP
S4
ETL4_ACT4
ETL4_ACT6
ETL4_ACT1
ETL4_ACT2
ETL4_ACT3
ETL4_ACT5
ETL4_Q10
ETL4_Q4
ETL4_Q5
ETL4_Q6
ETL4_Q7
ETL4_Q8
ETL4_Q9
ETL4_Q3
ETL4_Q2
filter
join
project
(a)


Affected
STRENGTH IN
STRENGTH OUT
STRENGTH TOTAL
filter
join
project
(b)

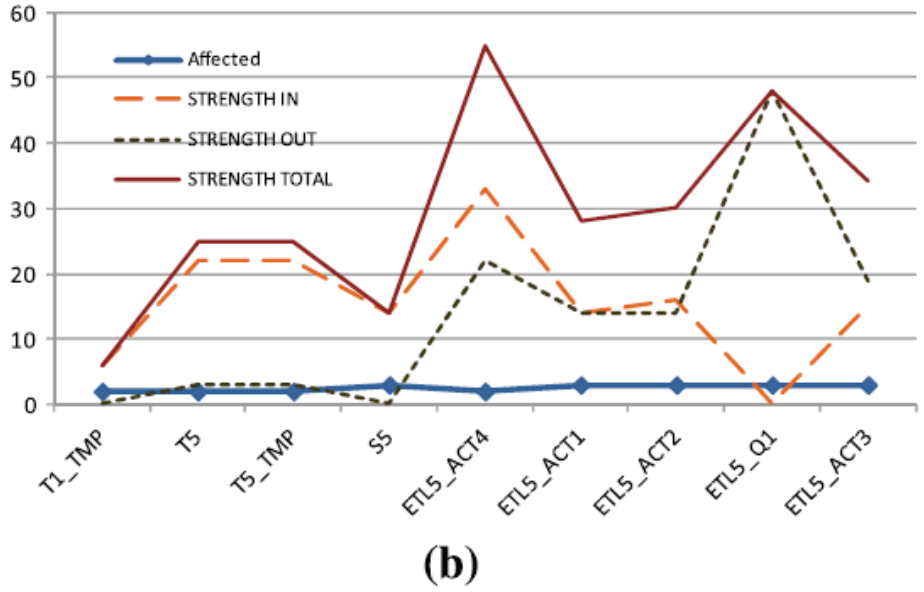

Affected
TRAN DEGREE
TRAN DEGREE IN
TRAN DEGREE OUT
ENTROPY
filter
join
project
(c)


Affected
TRAN STRENGTH IN
TRAN STRENGTH OUT
TRAN STRENGTH TOTAL
filter
join
project
(d)

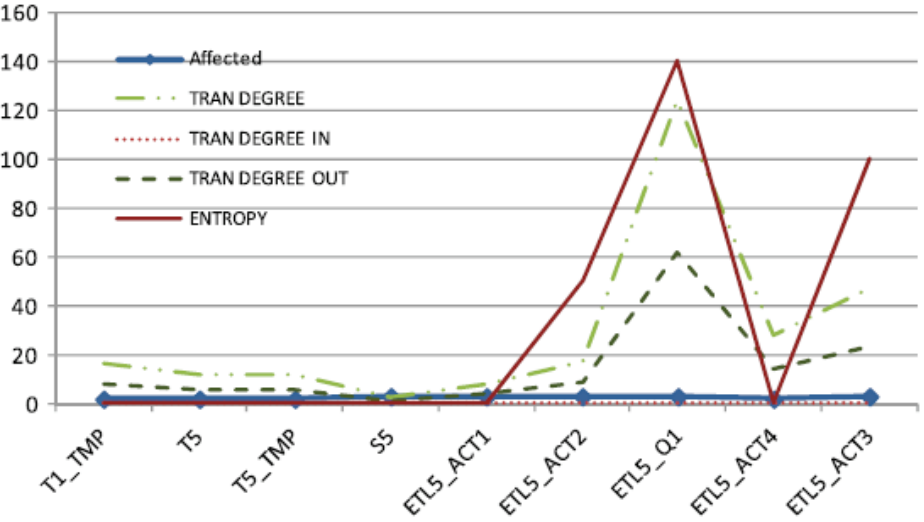
ETL 4


Affected
DEGREE IN
DEGREE OUT
DEGREE TOTAL
T1_TMP
T5
T5_TMP
S5
ETL5_ACT4
ETL5_ACT1
ETL5_ACT2
ETL5_Q1
ETL5_ACT3

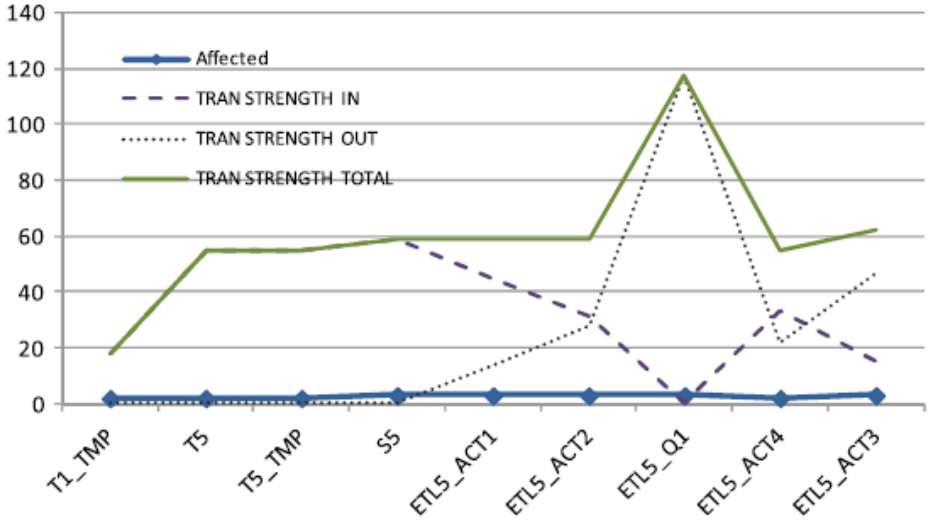
(a)


Affected
STRENGTH IN
STRENGTH OUT
STRENGTH TOTAL
T1_TMP
T5
T5_TMP
S5
ETL5_ACT4
ETL5_ACT1
ETL5_ACT2
ETL5_Q1
ETL5_ACT3

(b)

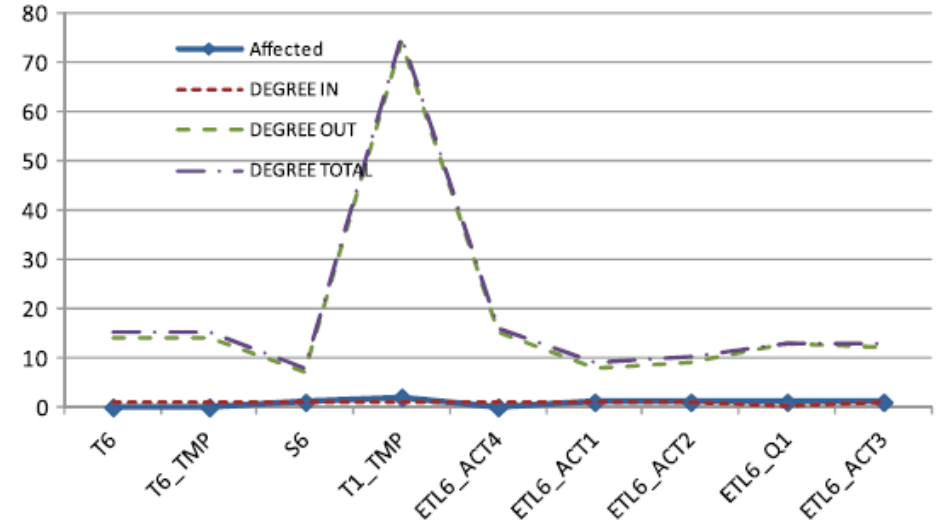

Affected
TRAN DEGREE
TRAN DEGREE IN
TRAN DEGREE OUT
ENTROPY
T1_TMP
T5
T5_TMP
S5
ETL5_ACT1
ETL5_ACT2
ETL5_Q1
ETL5_ACT4
ETL5_ACT3

(c)

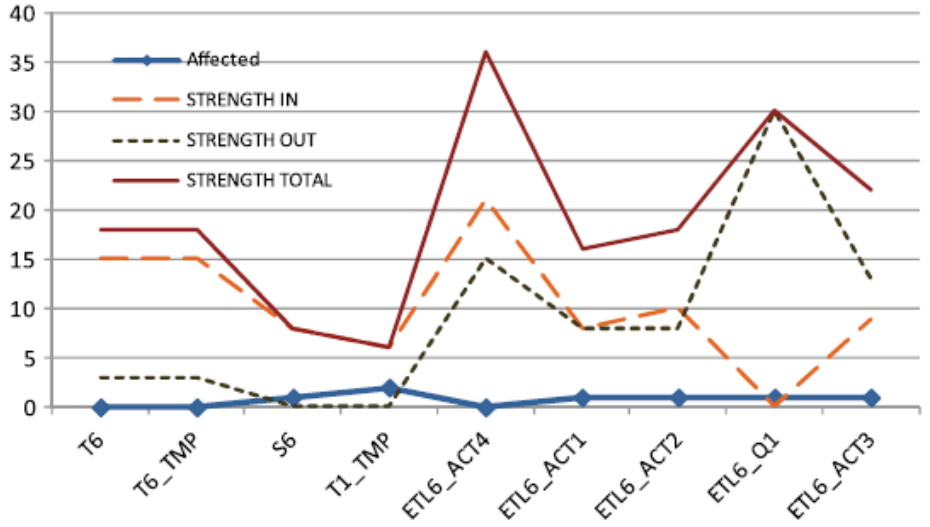

Affected
TRAN STRENGTH IN
TRAN STRENGTH OUT
TRAN STRENGTH TOTAL
T1_TMP
T5
T5_TMP
S5
ETL5_ACT1
ETL5_ACT2
ETL5_Q1
ETL5_ACT4
ETL5_ACT3

(d)

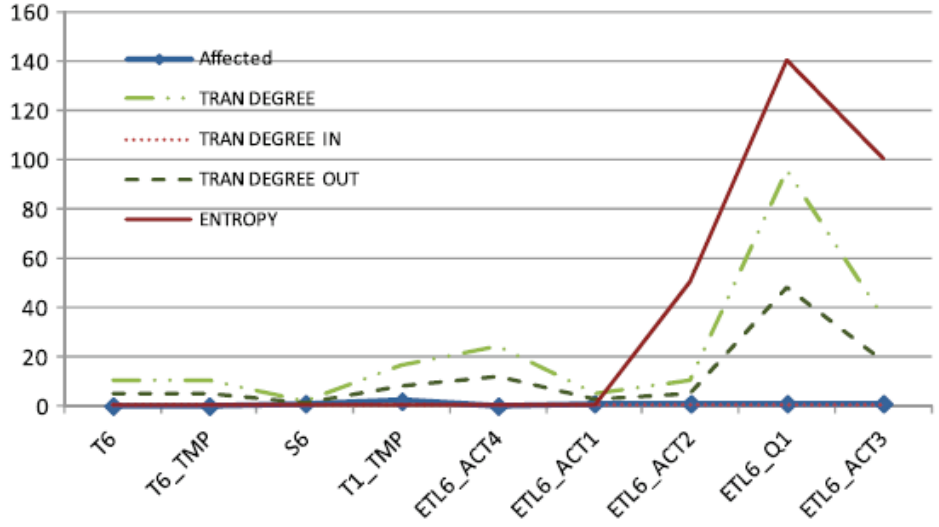
ETL 5


80
70
60
50
40
30
20
10
0
Affected
DEGREE IN
DEGREE OUT
DEGREE TOTAL
T6
T6_TMP
S6
T1_TMP
ETL6_ACT4
ETL6_ACT1
ETL6_ACT2
ETL6_Q1
ETL6_ACT3

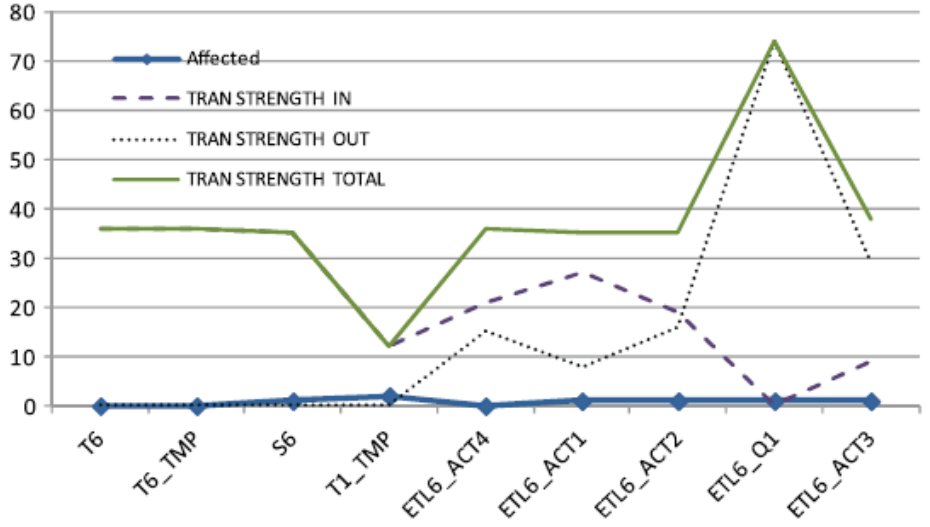
(a)


40
35
30
25
20
15
10
5
0
Affected
STRENGTH IN
STRENGTH OUT
STRENGTH TOTAL
T6
T6_TMP
S6
T1_TMP
ETL6_ACT4
ETL6_ACT1
ETL6_ACT2
ETL6_Q1
ETL6_ACT3

(b)

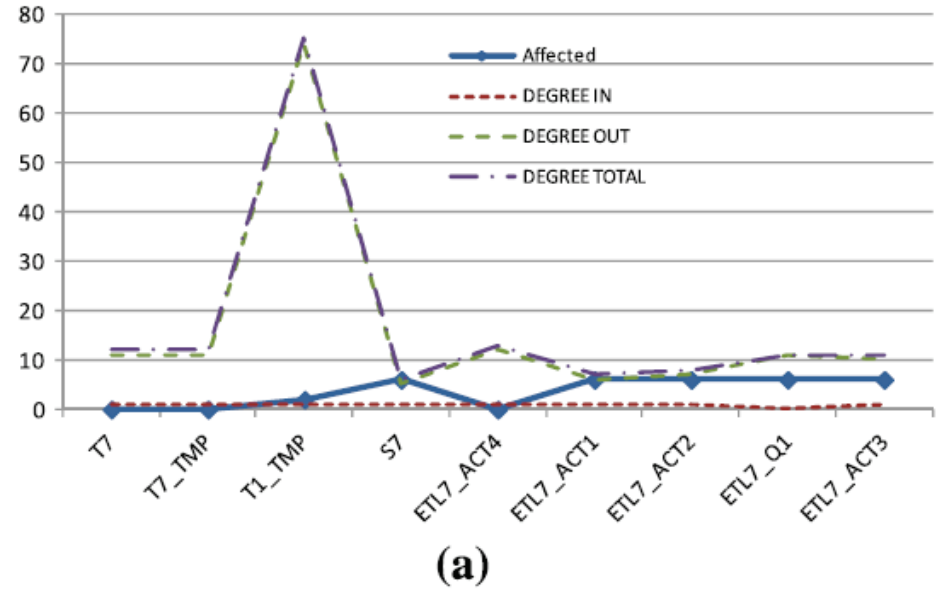

160
140
120
100
80
60
40
20
0
Affected
TRAN DEGREE
TRAN DEGREE IN
TRAN DEGREE OUT
ENTROPY
T6
T6_TMP
S6
T1_TMP
ETL6_ACT4
ETL6_ACT1
ETL6_ACT2
ETL6_Q1
ETL6_ACT3

(c)

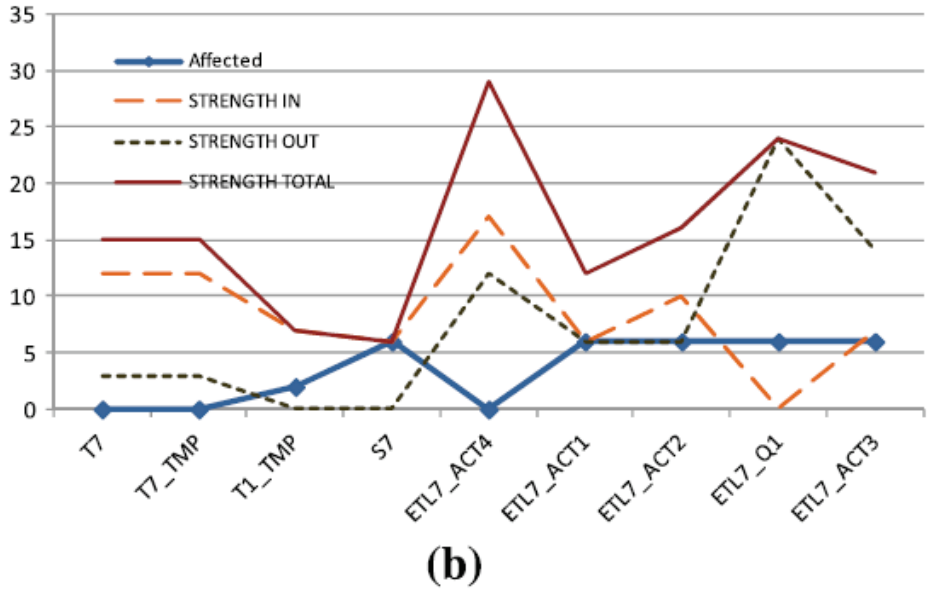

80
70
60
50
40
30
20
10
0
Affected
TRAN STRENGTH IN
TRAN STRENGTH OUT
TRAN STRENGTH TOTAL
T6
T6_TMP
S6
T1_TMP
ETL6_ACT4
ETL6_ACT1
ETL6_ACT2
ETL6_Q1
ETL6_ACT3

(d)

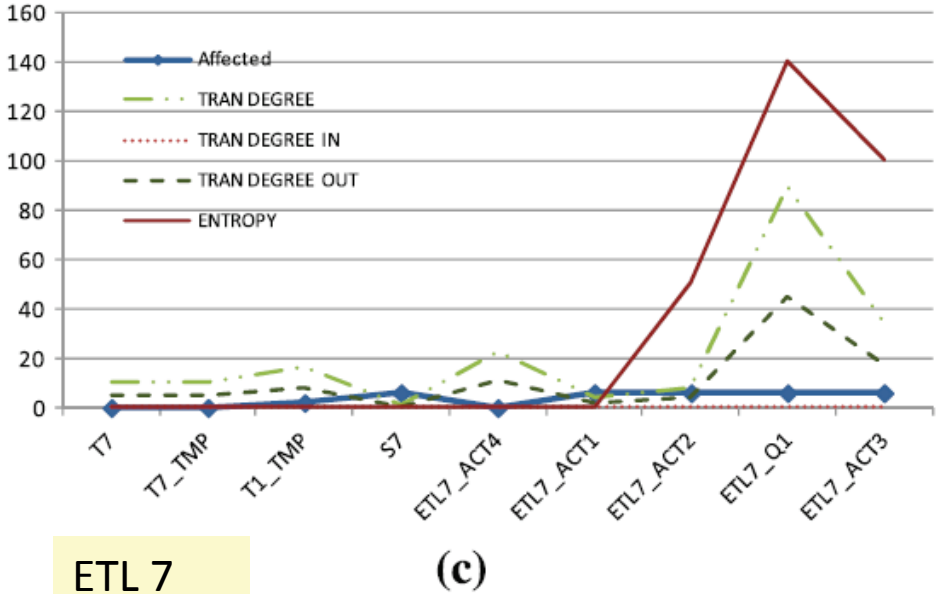
ETL 6


80
70
60
50
40
30
20
10
0
Affected
DEGREE IN
DEGREE OUT
DEGREE TOTAL
T7
T7_TMP
T1_TMP
S7
ETL7_ACT4
ETL7_ACT1
ETL7_ACT2
ETL7_Q1
ETL7_ACT3

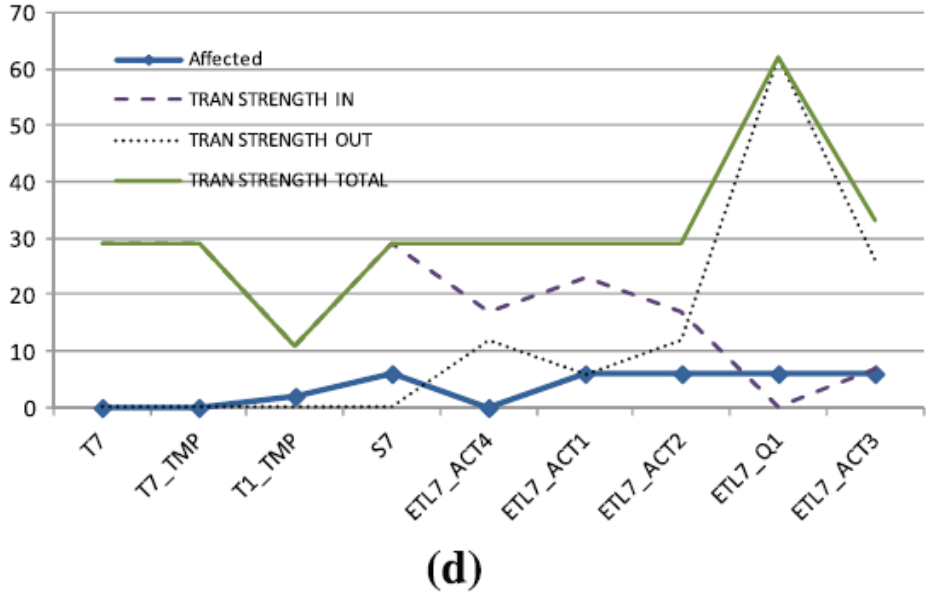
(a)


35
30
25
20
15
10
5
0
Affected
STRENGTH IN
STRENGTH OUT
STRENGTH TOTAL
T7
T7_TMP
T1_TMP
S7
ETL7_ACT4
ETL7_ACT1
ETL7_ACT2
ETL7_Q1
ETL7_ACT3

(b)


160
140
120
100
80
60
40
20
0
Affected
TRAN DEGREE
TRAN DEGREE IN
TRAN DEGREE OUT
ENTROPY
T7
T7_TMP
T1_TMP
S7
ETL7_ACT4
ETL7_ACT1
ETL7_ACT2
ETL7_Q1
ETL7_ACT3

(c)


70
60
50
40
30
20
10
0
Affected
TRAN STRENGTH IN
TRAN STRENGTH OUT
TRAN STRENGTH TOTAL
T7
T7_TMP
T1_TMP
S7
ETL7_ACT4
ETL7_ACT1
ETL7_ACT2
ETL7_Q1
ETL7_ACT3

(d)

ETL 7

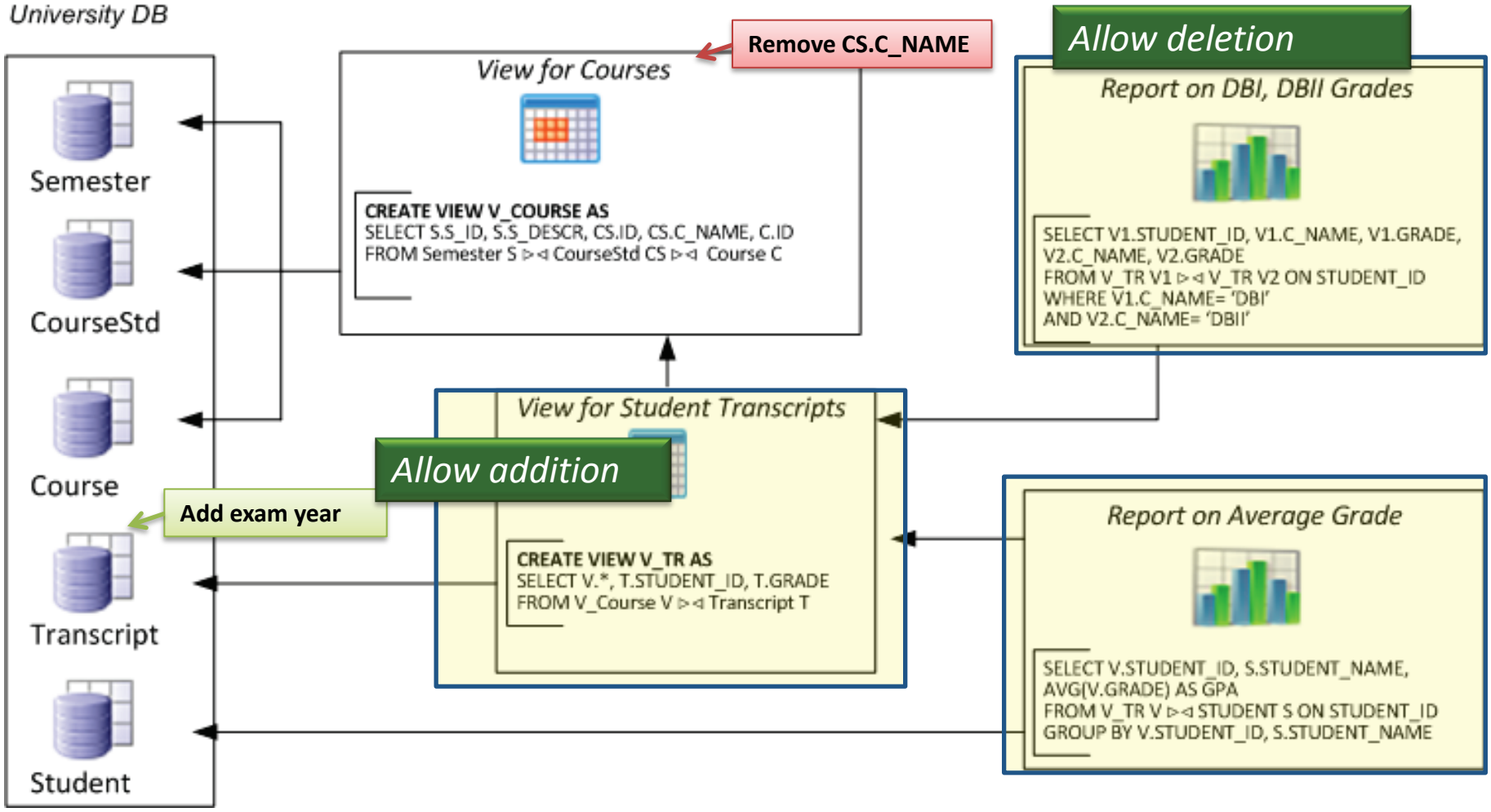
Events and Policies

Modeling tools

How to regulate evolution of ecosystems

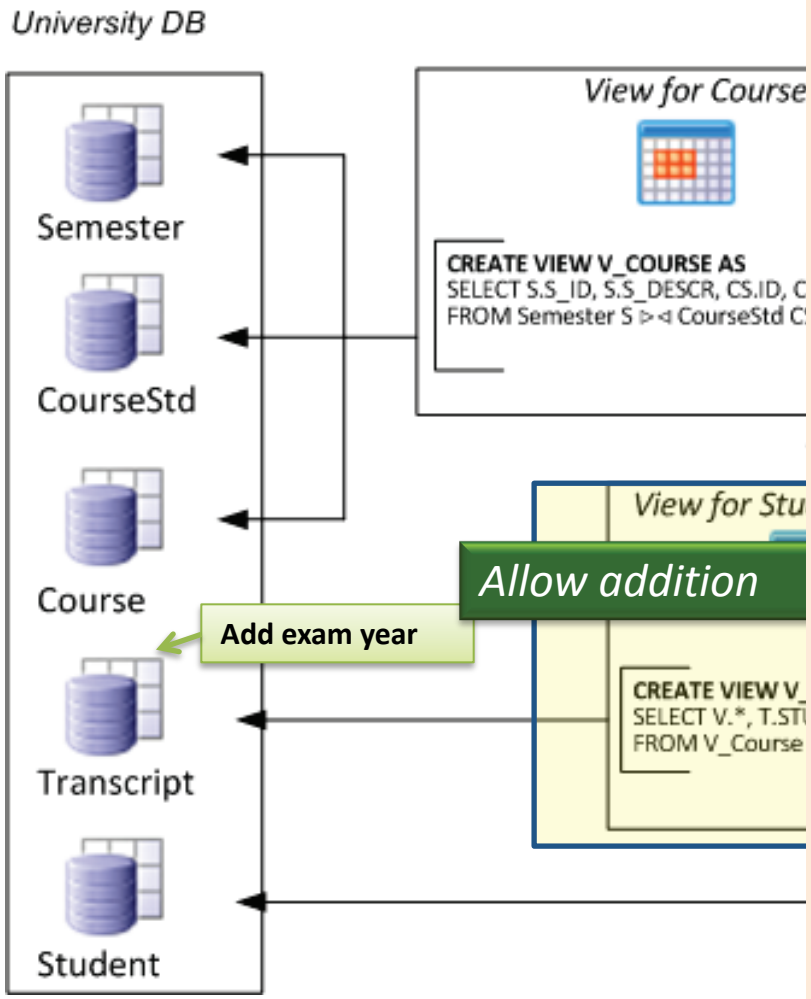
- Impact Analysis
  - We employ evolution events to model how data-intensive ecosystems change
  - We apply a hypothetical event and propagate it over the Architecture Graph to assess which modules are affected by it, and how (i.e., in which parts of their internal structure)
  - This way, we can visualize and measure the impact of a potential change to the entire ecosystem
- Impact Regulation
  - We employ evolution policies to pre-determine how modules should react to incoming events
  - Whenever a notification on an event “arrives” at a module, the module knows what to do: adapt to the incoming event, or block it and require to retain its previous structure and semantics
  - This Blocking restricts the flooding of events to the entire Architecture Graph and can allow developers “fix contracts” with the underlying database

Evolving data-centric ecosystem



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

A language for policies




```
DATABASE: ON ADD_ATTRIBUTE              TO RELATION THEN PROPAGATE;
DATABASE: ON ADD_CONDITION              TO RELATION THEN PROPAGATE;
DATABASE: ON ADD_RELATION               TO RELATION THEN PROPAGATE;
DATABASE: ON DELETE_ATTRIBUTE           TO RELATION THEN PROPAGATE;
DATABASE: ON DELETE_CONDITION           TO RELATION THEN PROPAGATE;
DATABASE: ON DELETE_RELATION            TO RELATION THEN PROPAGATE;
DATABASE: ON MODIFYDOMAIN_ATTRIBUTE     TO RELATION THEN PROPAGATE;
DATABASE: ON RENAME_ATTRIBUTE           TO RELATION THEN PROPAGATE;
DATABASE: ON MODIFY_CONDITION           TO RELATION THEN PROPAGATE;
DATABASE: ON RENAME_RELATION            TO RELATION THEN PROPAGATE;


DATABASE: ON ADD_ATTRIBUTE              TO VIEW THEN BLOCK;
DATABASE: ON ADD_CONDITION              TO VIEW THEN BLOCK;
DATABASE: ON ADD_RELATION               TO VIEW THEN BLOCK;
DATABASE: ON DELETE_ATTRIBUTE           TO VIEW THEN BLOCK;
DATABASE: ON DELETE_CONDITION           TO VIEW THEN BLOCK;
DATABASE: ON DELETE_RELATION            TO VIEW THEN BLOCK;
DATABASE: ON MODIFYDOMAIN_ATTRIBUTE     TO VIEW THEN BLOCK;
DATABASE: ON RENAME_ATTRIBUTE           TO VIEW THEN BLOCK;
DATABASE: ON MODIFY_CONDITION           TO VIEW THEN BLOCK;
DATABASE: ON RENAME_RELATION            TO VIEW THEN BLOCK;

DATABASE: ON ADD_ATTRIBUTE              TO QUERY THEN BLOCK;
DATABASE: ON ADD_CONDITION              TO QUERY THEN BLOCK;
DATABASE: ON ADD_RELATION               TO QUERY THEN BLOCK;
DATABASE: ON DELETE_ATTRIBUTE           TO QUERY THEN BLOCK;
DATABASE: ON DELETE_CONDITION           TO QUERY THEN BLOCK;
DATABASE: ON DELETE_RELATION            TO QUERY THEN BLOCK;
DATABASE: ON MODIFYDOMAIN_ATTRIBUTE     TO QUERY THEN BLOCK;
DATABASE: ON RENAME_ATTRIBUTE           TO QUERY THEN BLOCK;
DATABASE: ON MODIFY_CONDITION           TO QUERY THEN BLOCK;
DATABASE: ON RENAME_RELATION            TO QUERY THEN BLOCK;
```

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

A language for policies

- For all possible events, we define **rules**

```
DATABASE: ON <event> TO <module type> THEN <reaction policy>
```

- Module types: relations, views, queries
- Events:

```
{add, delete, rename}  X {module internals}
```

- Policies:
  - **Propagate**: adapt to the incoming notification for change, willing to modify structure and semantics
  - **Block**: resist change; require to retain the previous structure and semantics
  - Prompt: indecisive for the moment, prompt the user at runtime (never implemented)

A language for policies

- Language requirements:
  - Completeness
  - Conciseness
  - Customizability

The two first req's are covered by a complete set of default policies that have to be defined for the entire ecosystem

- We can override default policies, to allow module parts to differentiate their behavior from the default

DATABASE:	ON	DELETE_ATTRIBUTE	TO	RELATION	THEN	PROPAGATE
TRANSICRIPT:	ON	DELETE_ATTRIBUTE			THEN	BLOCK
TRANSCRIPT.STUDENT_ID:	ON	DELETE_ATTRIBUTE			THEN	PROPAGATE

For the metrics part

Other Useful: metrics

Evolution Variants

- Data Evolution: INS/DEL/UPD of data without affecting the structure of the db.
- Schema Evolution: the structure of the db changes, without loss of existing data, but without retaining historical information on previous snapshots of the db.
- Schema versioning: schema evolution + the ability to answer historical queries (practically being able to restore a snapshot of the db at a given timepoint).

Node Entropy

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

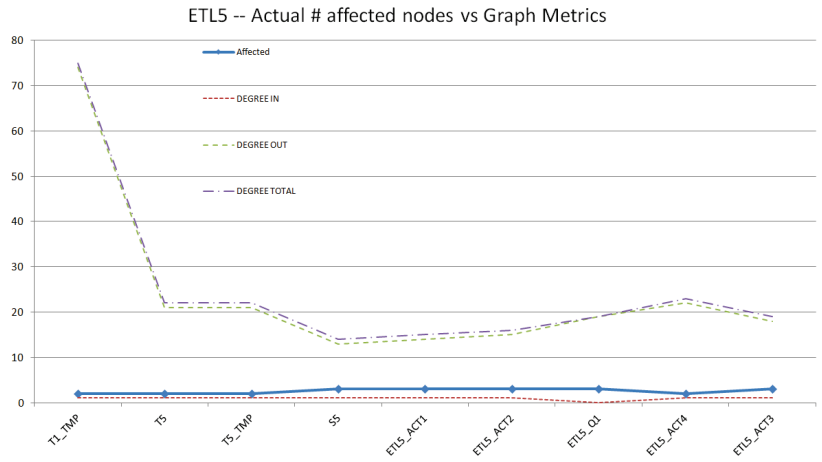
$$H(v) = -\sum_{y_i \in V} P(v \mid y_i) \log_2 P(v \mid y_i), \text{ for all nodes } y_i \in V.$$

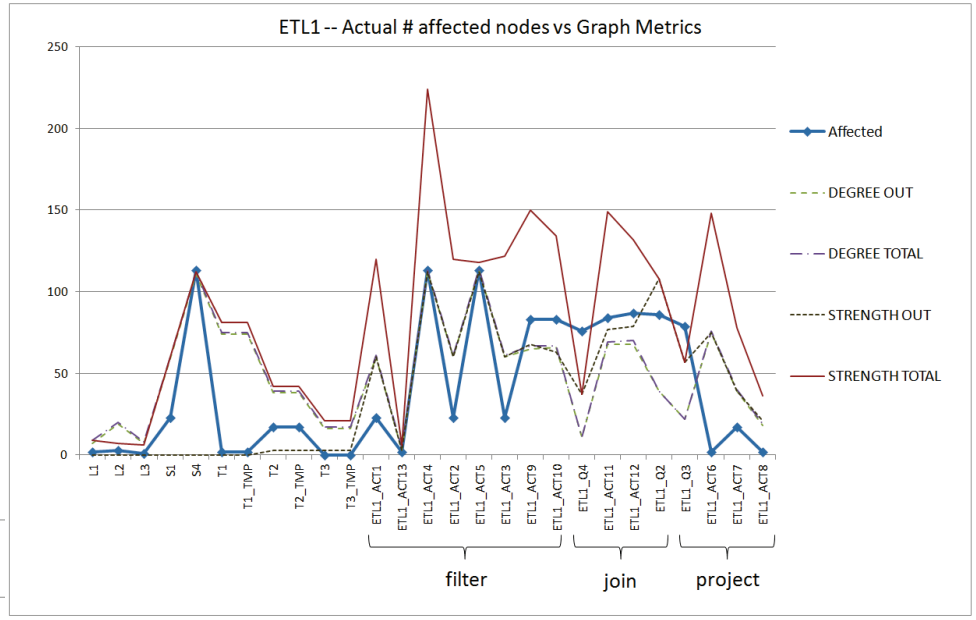
High values of entropy are assigned to the nodes of the graph with high level of dependence to other nodes, either directly or transitively.

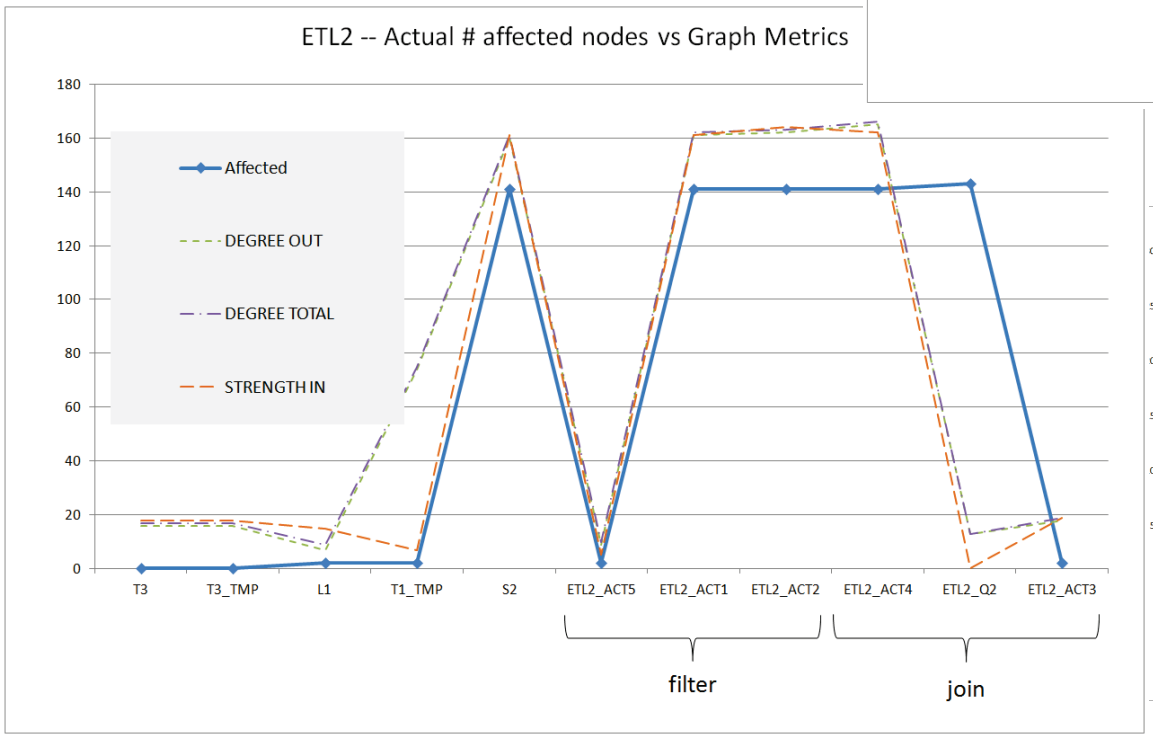
Who is likely to undergo change?

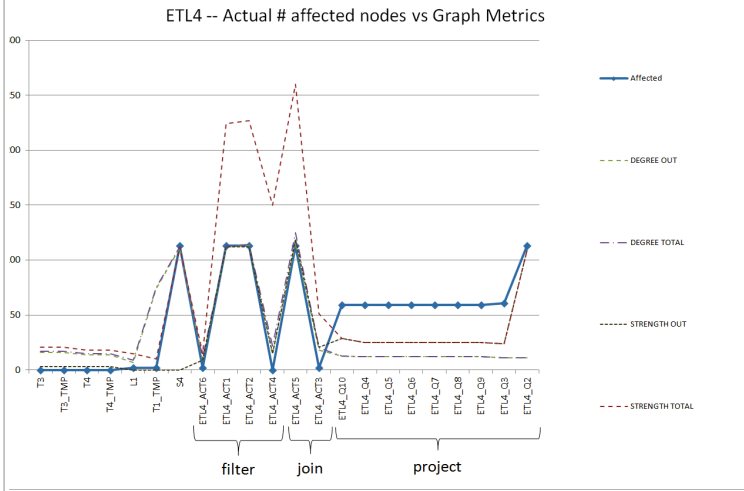
- Schema size is (quite expectedly) the most important factor for a relation's vulnerability to change
- The same holds for activities, too!

Most accurate predictors: *out-degree* and *out-strength*









Internal structure of activities

- Activities with high *out-degree* and *out-strength* tend to be more vulnerable to evolution. 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.

- Activities with joins between many sources tend to be more affected than activities sourced by only one provider, but still, the most decisive factor seems to be the activity size.
  - Thus, activities that perform an attribute reduction on the workflow through either a group-by operation or a projection of a small number of attributes are in general, less vulnerable to evolution events and propagate the impact of evolution further away on the workflow (e.g., Q4 in ETL1 or Q2 – Q10 in ETL4).
  - In contrast, activities that perform join and selection operations on many sources and result in attribute preservation or generation on the workflow have a higher potential to be affected by evolution events (e.g., observe the activities ETL1_ACT10 - ETL1_ACT12 or the activity ETL4_ACT5).

Transitive degrees

- Transitive degree 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.
- For cases where the out-degree metrics *do not* provide a clear view of the evolution potential of two or more modules, the out-transitive degree and entropy metrics may offer a more adequate prediction (as for example ETL4_Q3 and ETL4_Q2).

Context and internals of evolution

- As already mentioned, source S1 stores the constant data of the surveys and did not change a lot. The rest of the source tables (S2-S7), on the other hand, sustained maintenance.
- The recorded changes in these tables mainly involve restructuring, additions and renaming of the questions comprising each survey, which are furthermore captured as changes in the source attributes names and types.
- The set of evolution events includes renaming of relations and attributes, deletion of attributes, modification of their domain, and lastly addition of primary key constraints. We have recorded a total number of 416 evolution events (see next table for a breakdown).
- The majority of evolution changes concerns attribute renaming and attribute additions.

Some numbers

ETL scenarios configuration

Scenario	# Activ.	Sources	Tmp Tables	Targets
ETL 1	16	L1,L2,L3,S1,S4	T1_Tmp, T2_Tmp, T3_Tmp	T1, T2, T3
ETL 2	6	L1,S2	T1_Tmp, T3_Tmp	T3
ETL 3	6	L1,S3	T1_Tmp, T3_Tmp	T3
ETL 4	15	L1,S4	T1_Tmp, T3_Tmp, T4_Tmp	T3, T4
ETL 5	5	S5	T1_Tmp, T5_Tmp	T5
ETL 6	5	S6	T1_Tmp, T6_Tmp	T6
ETL 7	5	S7	T1_Tmp, T7_Tmp	T7
Total	58			

Number of Attributes in ETL Source Tables

Table	S1	S2	S3	S4	S5	S6	S7	L1	L2	L3
# Attributes	59	160	82	111	13	7	5	7	19	7

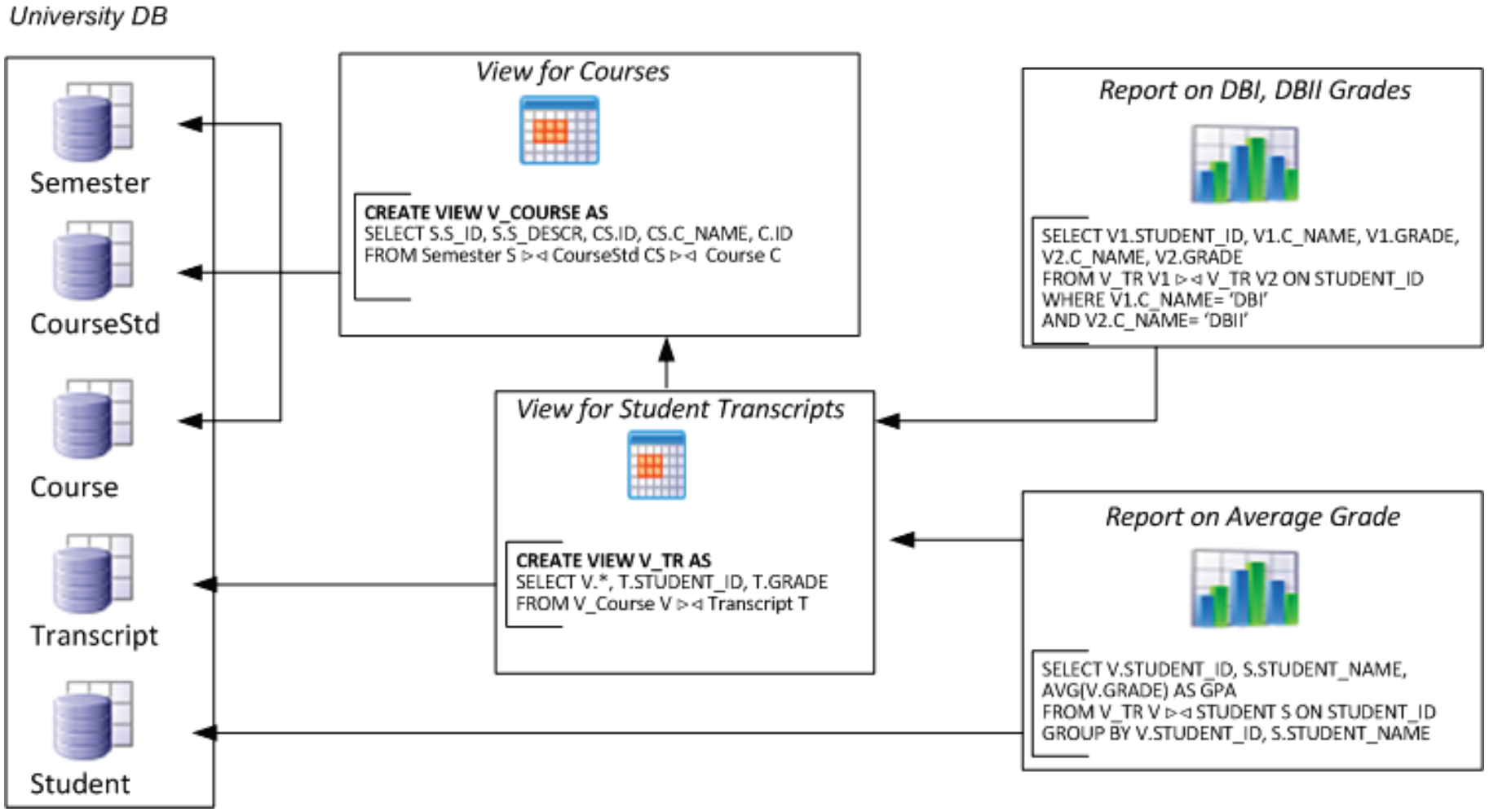
Source	Change Type	Occurrence	Affected ETL
L1	Add Attribute	1	ETL 1, 2, 3, 4
L1	Add Constraint	1	ETL 1, 2, 3, 4
L2	Add Attribute	3	ETL 1
L3	Add Attribute	1	ETL 1
S1	Add Attribute	14	ETL 1
S1	Drop Attribute	2	ETL 1
S1	Modify Attribute	3	ETL 1
S1	Rename Attribute	3	ETL 1
S1	Rename Table	1	ETL 1
S2	Add Attribute	15	ETL 2
S2	Drop Attribute	4	ETL 2
S2	Rename Attribute	121	ETL 2
S2	Rename Table	1	ETL 2
S3	Rename Attribute	80	ETL 3
S3	Rename Table	1	ETL 3
S4	Add Attribute	58	ETL 1, 4
S4	Drop Attribute	26	ETL 1, 4
S4	Modify Attribute	1	ETL 1, 4
S4	Rename Attribute	27	ETL 1, 4
S4	Rename Table	1	ETL 1, 4
S5	Modify Attribute	2	ETL 5
S5	Rename Table	1	ETL 6
S6	Rename Table	1	ETL 6
S7	Rename Attribute	5	ETL 7
S7	Rename Table	1	ETL 7
T1	Drop Attribute	1	ETL 1
T1	Modify Attribute	1	ETL 1
T1_tmp	Drop Attribute	1	ETL1-7
T1_tmp	Modify Attribute	1	ETL1-7
T2	Add Attribute	15	ETL 1
T2	Modify Attribute	2	ETL 1
T2_tmp	Add Attribute	15	ETL 1
T2_tmp	Modify Attribute	2	ETL 1
T5	Modify Attribute	2	ETL5
T5_tmp	Modify Attribute	2	ETL5
Total		416	

Distribution of events at the ETL tables

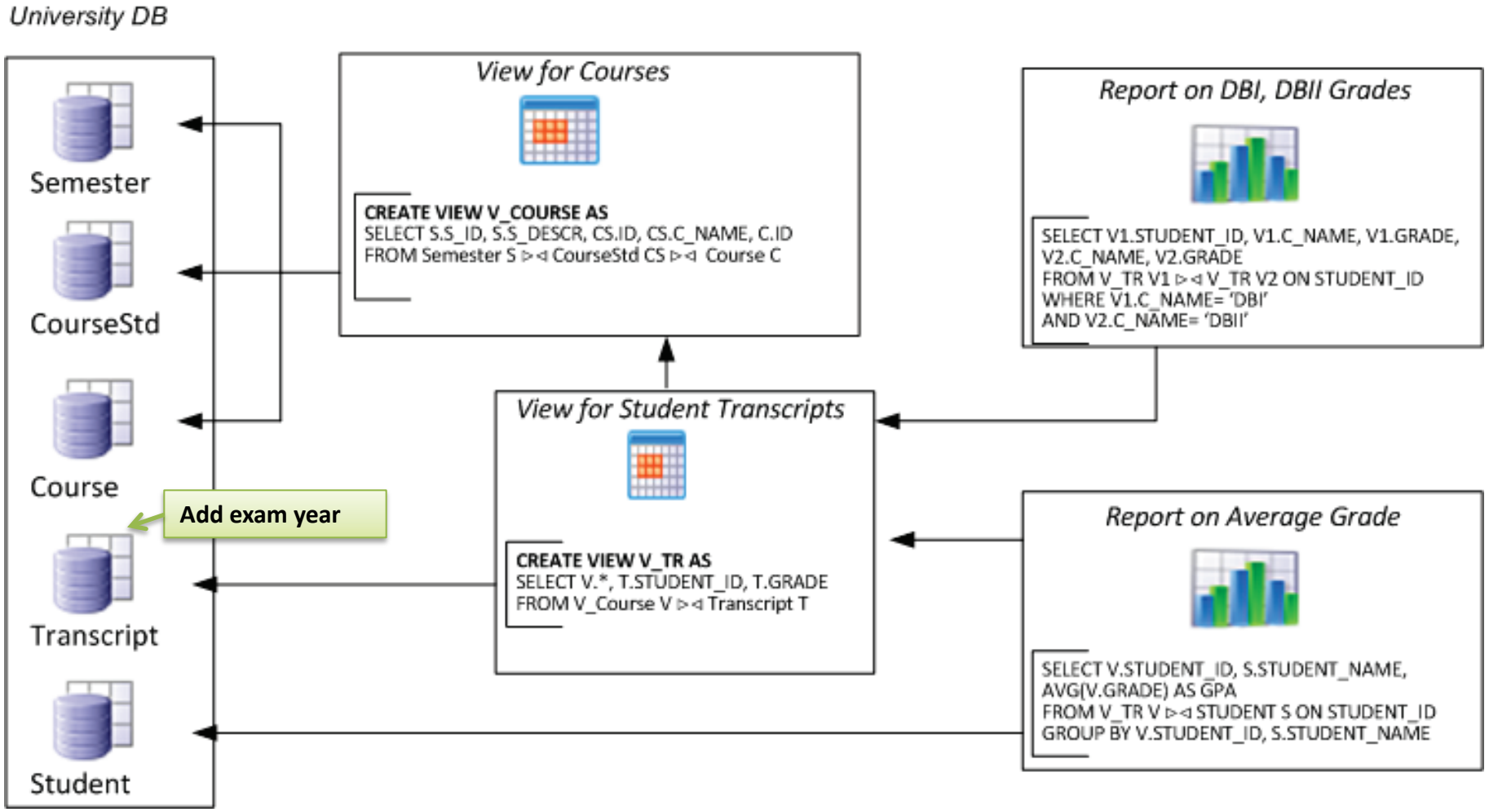
How to design a scenario

- When persistent data stores are involved, the generic guideline is to retain their schema as small as possible.
- Since the schema size affects a lot the propagation of evolution events, it is advisable to reduce schema sizes across the ETL flow, so activities that help in that direction should be considered first.
- Since attribute reduction activities (e.g., projections, group by queries) are less likely to be affected by evolution actions than other activities that retain or increase the number of attributes in the workflow (many projections with joins), the ETL designer should attempt placing the attribute reduction activities in the early stages of the workflow in order to restrain the flooding of evolution events.

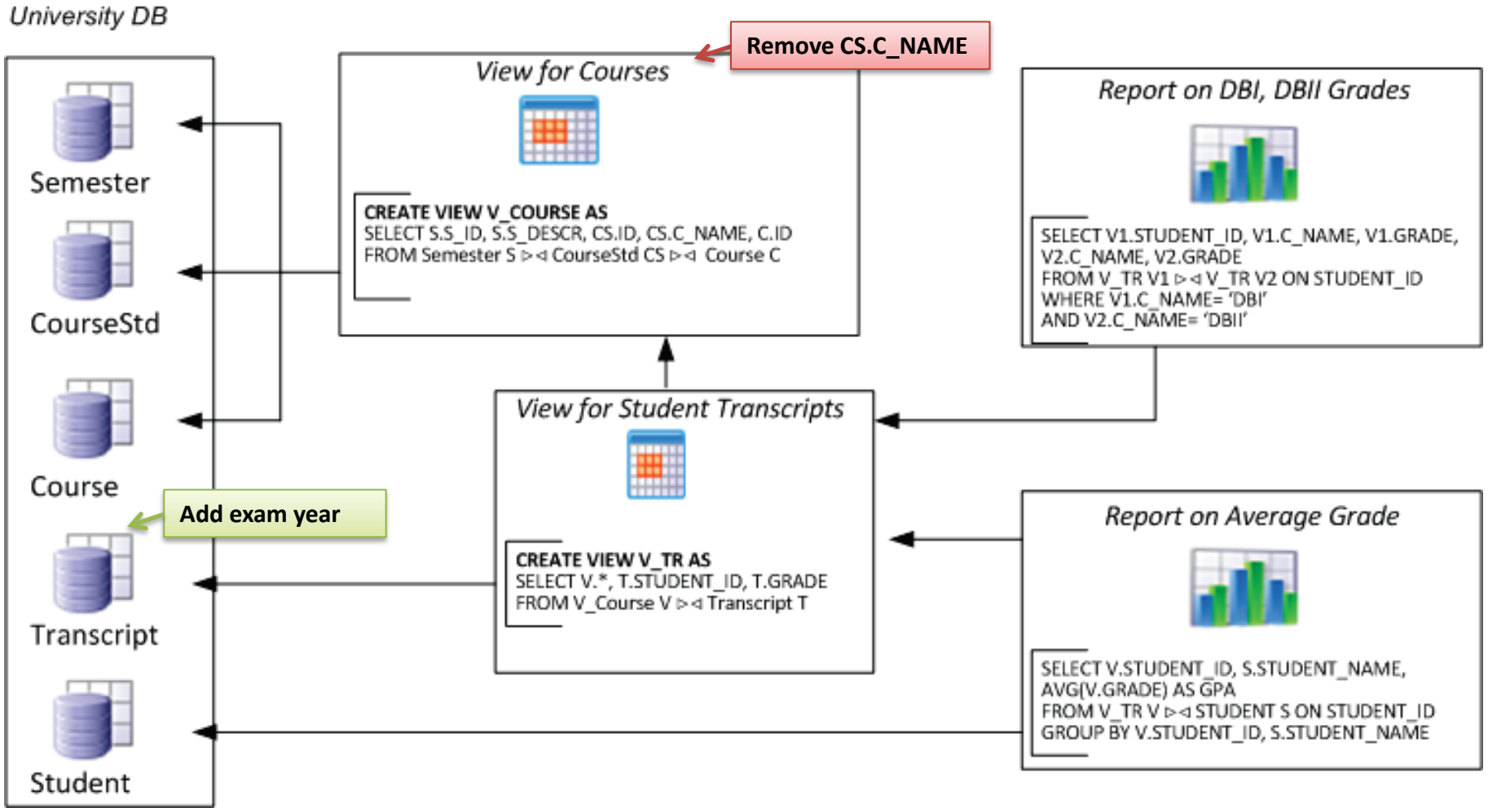
Evolving data-intensive ecosystem



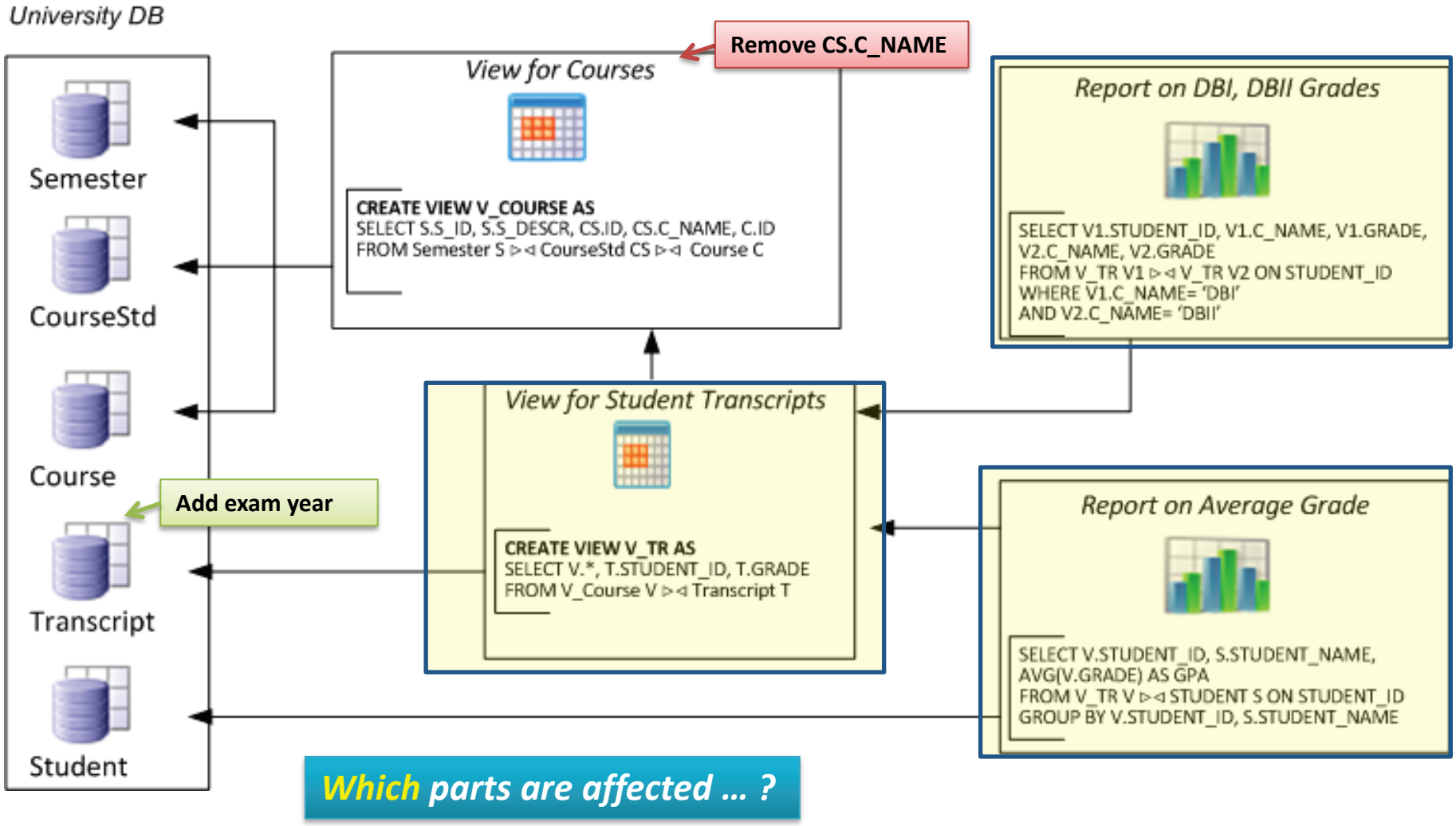
Evolving data-intensive ecosystem



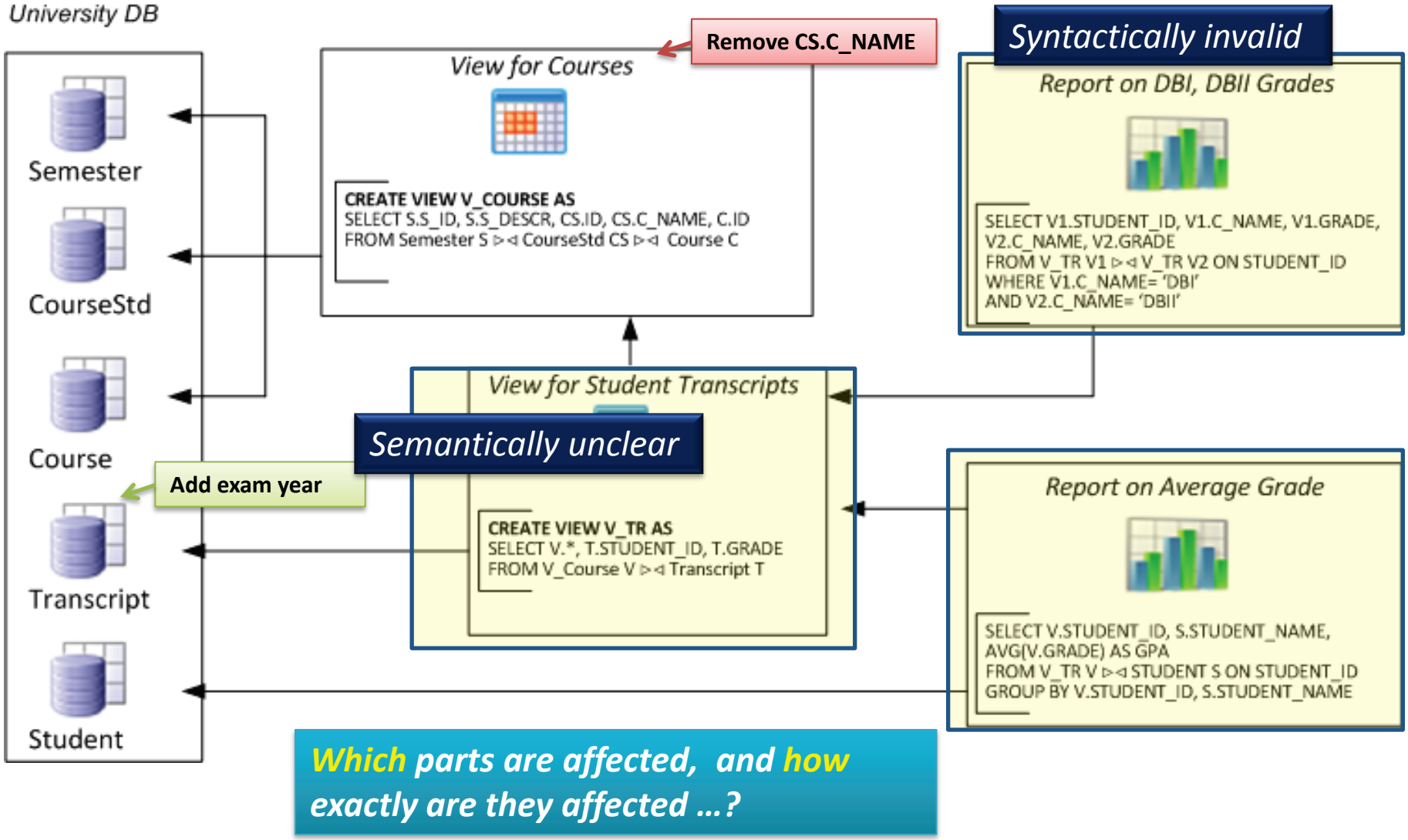
Evolving data-intensive ecosystem



Evolving data-intensive ecosystem



Evolving data-intensive ecosystem



For the regulation part

Other Useful: Regulation

Problem definition

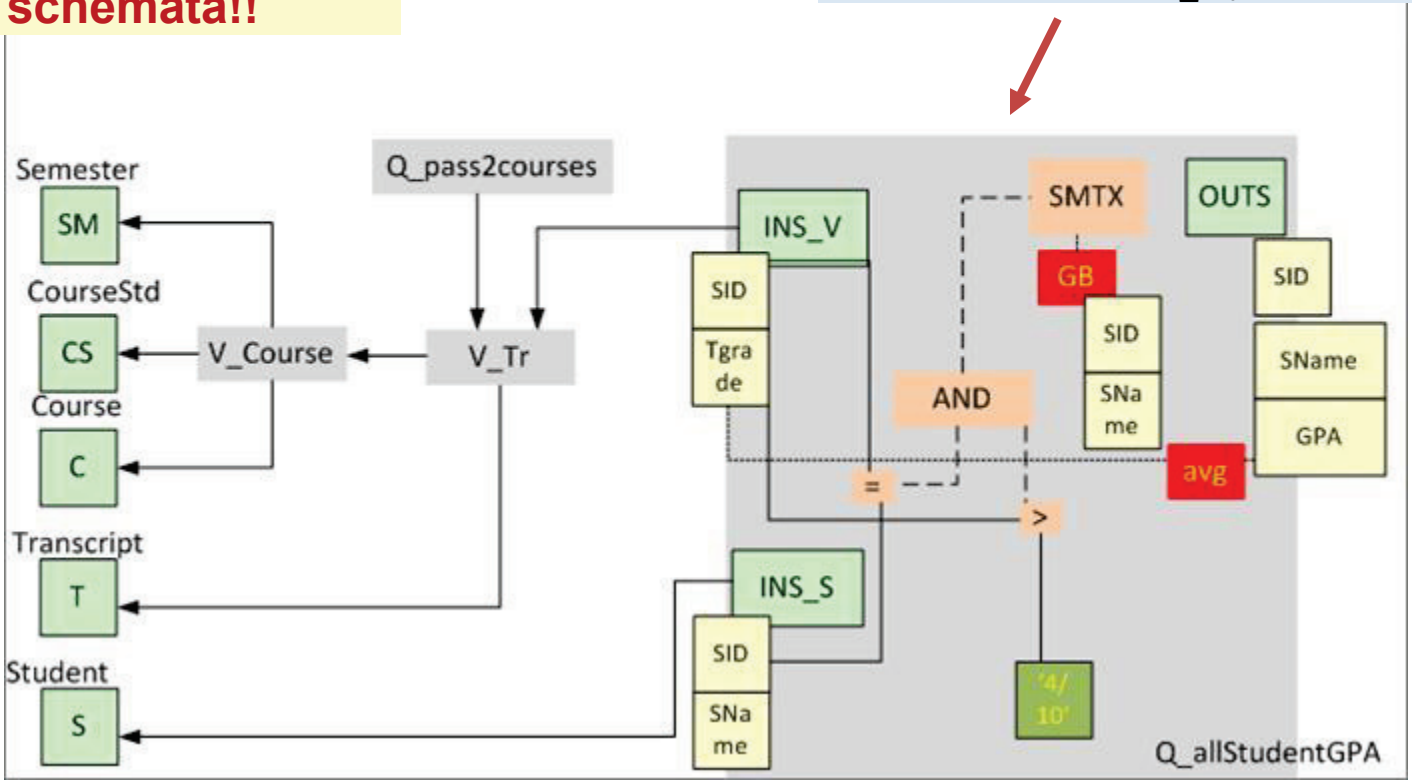
- *Changes on a database schema may cause syntactic or semantic inconsistency in its surrounding applications; is there a way to regulate the evolution of the database in a way that application needs are taken into account?*
- *If there are conflicts between the applications' needs on the acceptance or rejection of a change in the database, is there a possibility of satisfying all the different constraints?*
- *If conflicts are eventually resolved and, for every affected module we know whether to accept or reject a change, how can we rewrite the ecosystem to reflect the new status?*

Architecture Graph

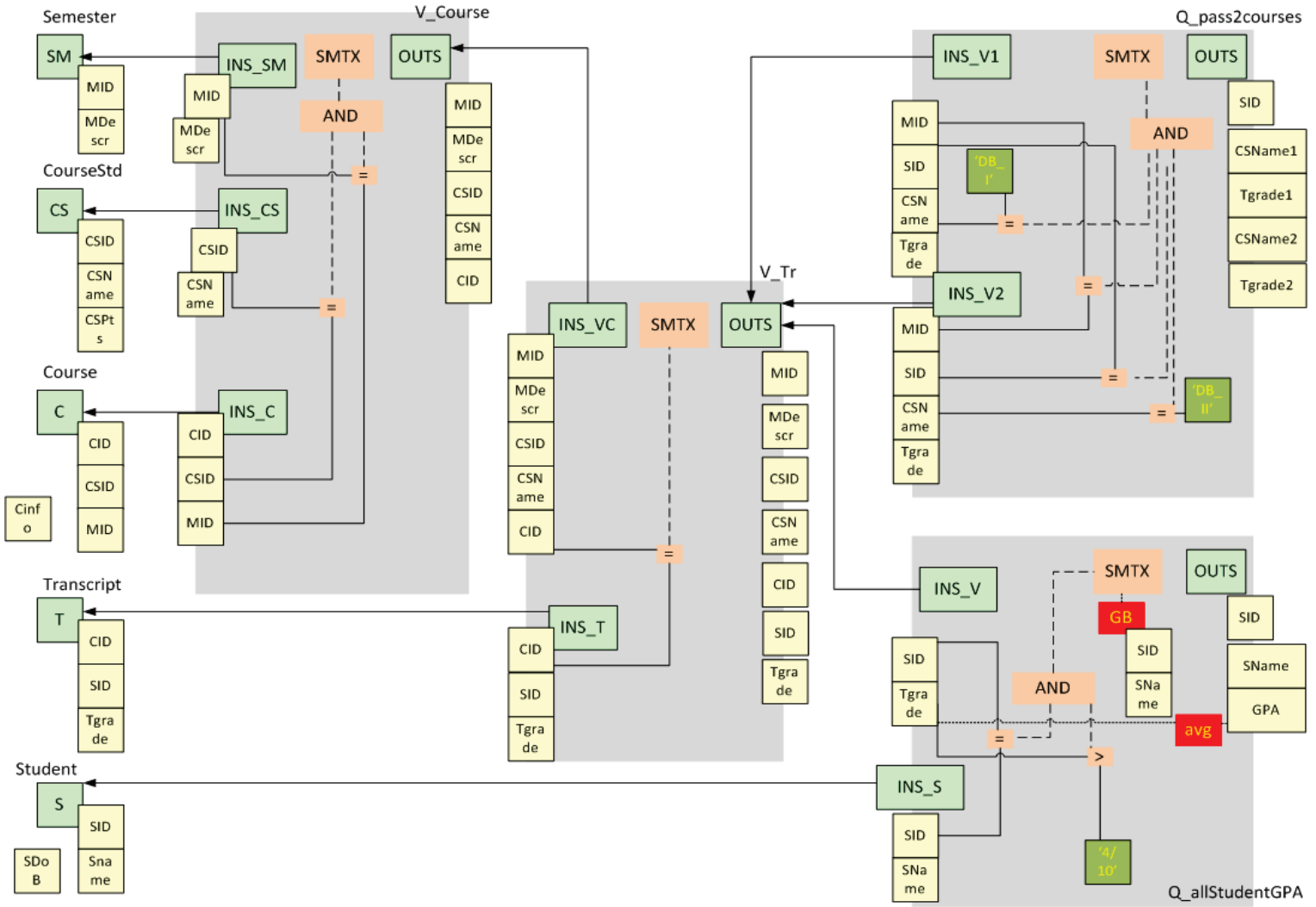
Modules and Module Encapsulation

Observe the input and output schemata!!

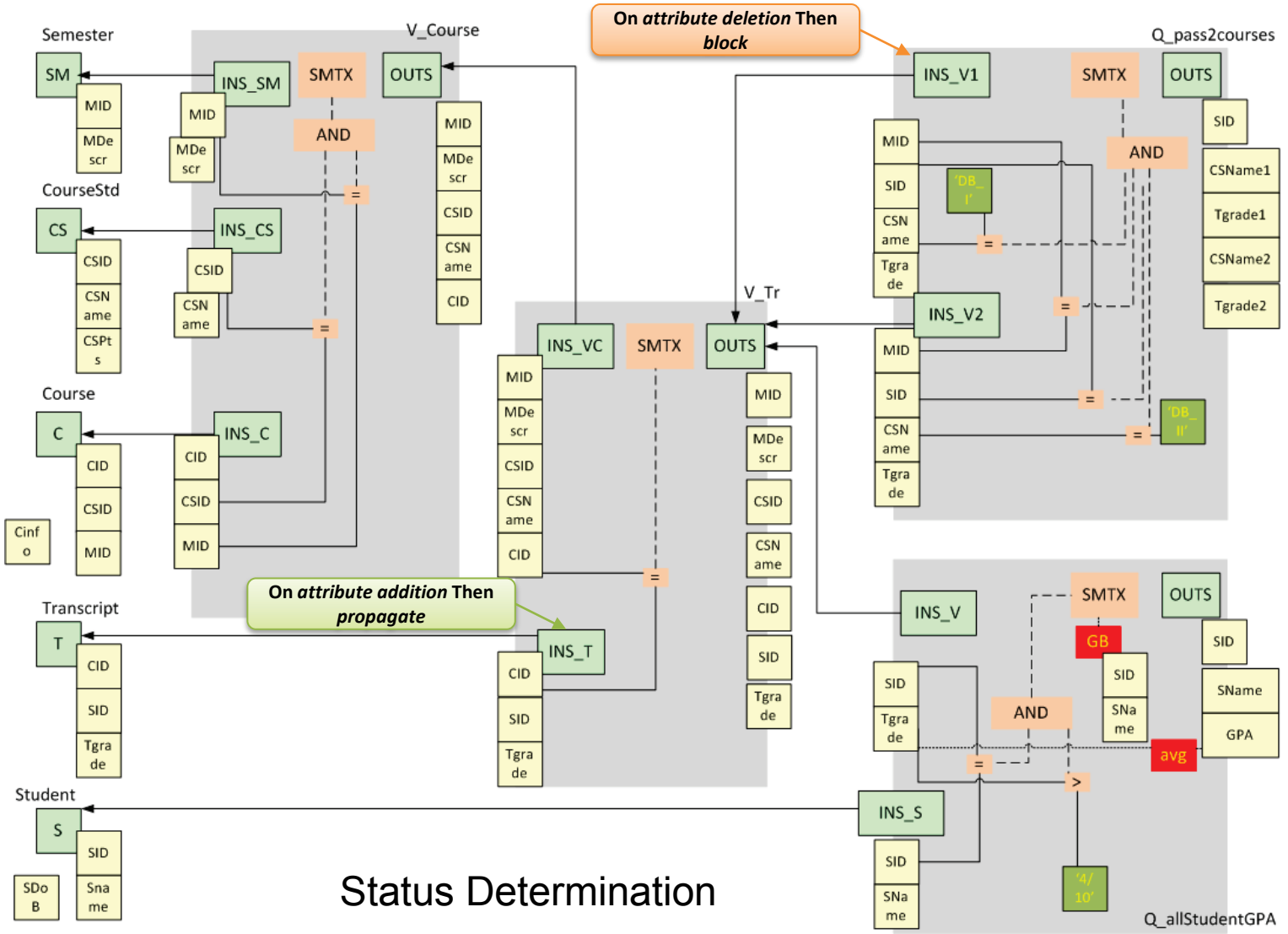
```
SELECT  V.STUDENT_ID, S.STUDENT_NAME,
                AVG(V.TGRADE) AS GPA
FROM V_TR V |><| STUDENT S ON STUDENT_ID
WHERE V.TGRADE > 4 / 10
GROUP BY V.STUDENT_ID, S.STUDENT_NAME
```



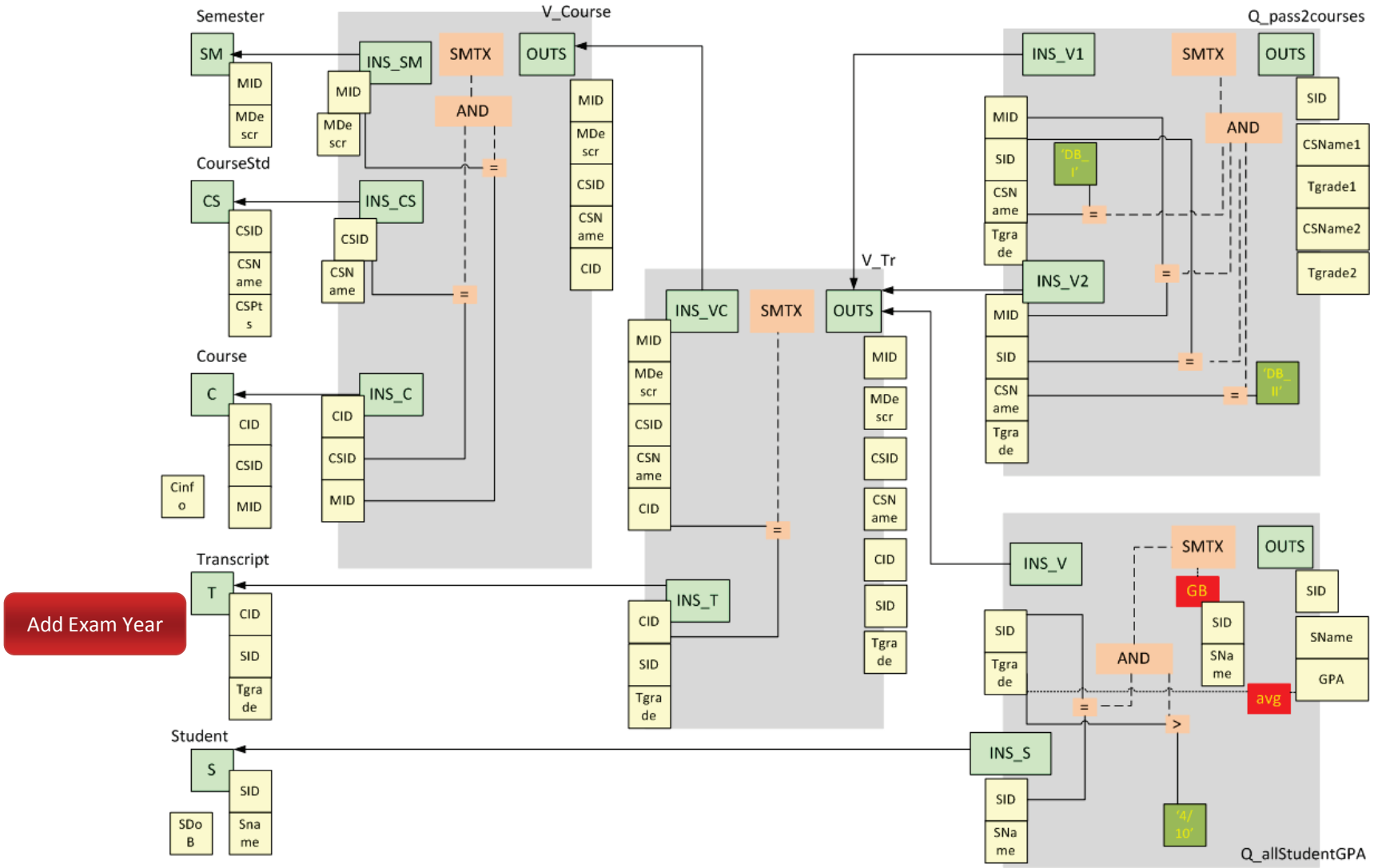
University E/S Architecture Graph



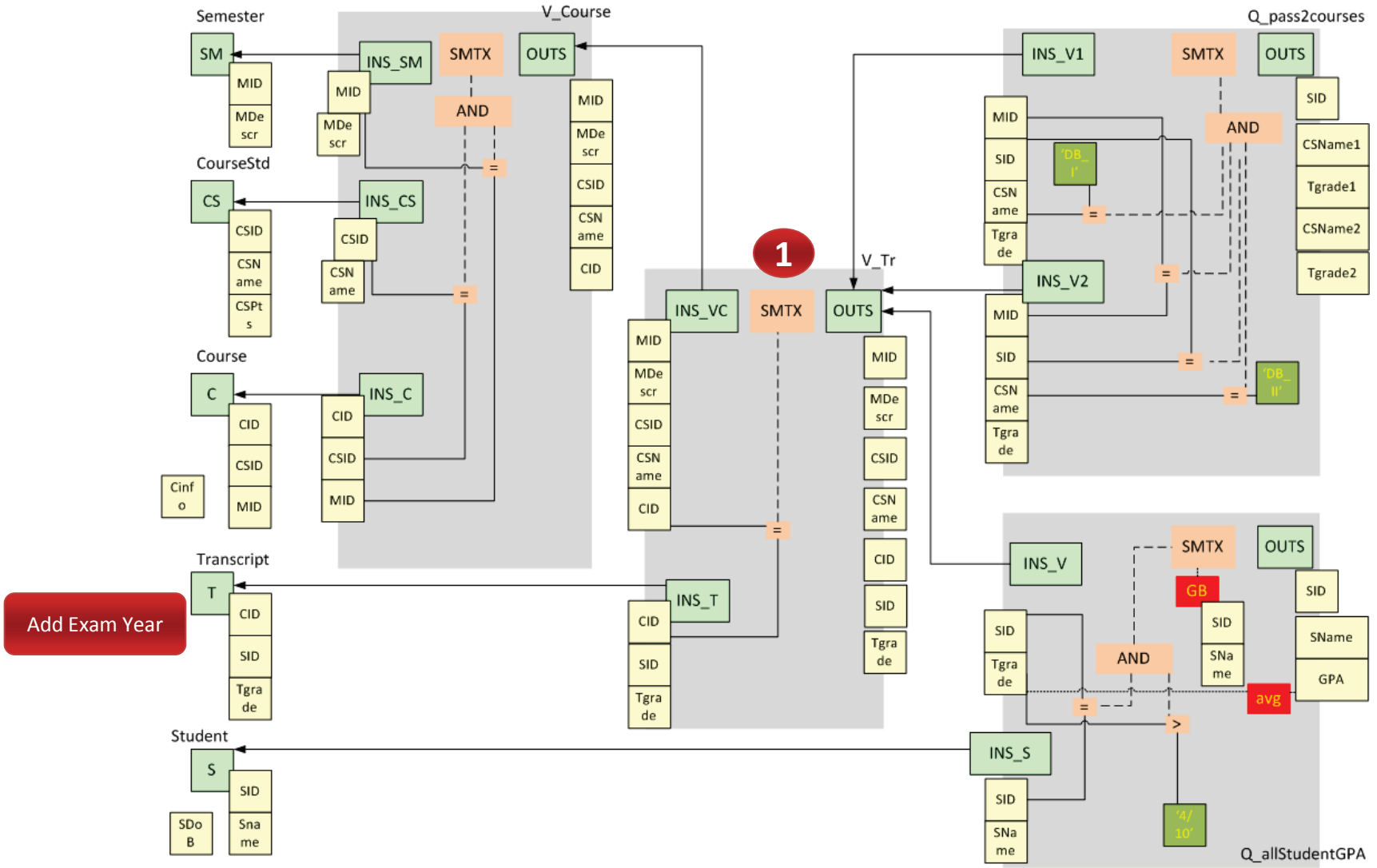
Annotation with Policies



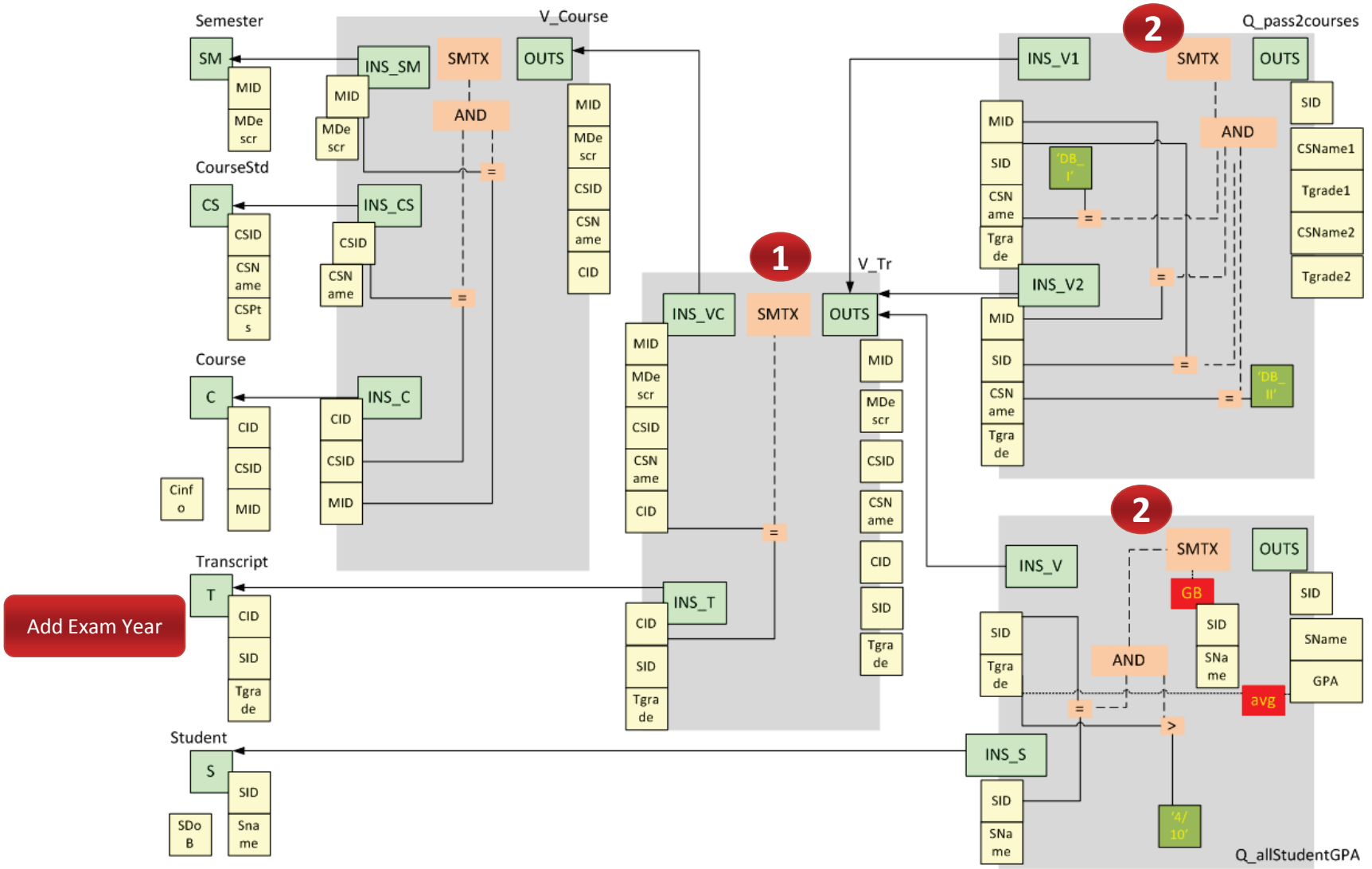
Module Level Propagation



Module Level Propagation



Module Level Propagation



Message initiation

- The Message is initiated in one of the following schemata:
  - Output schema and its attributes if the user wants to change the output of a module (add / delete / rename attribute).
  - Semantics schema if the user wants to change the semantics tree of the module.

Intra-module processing

- When a Message arrives at a module via the propagation mechanism, these steps describe the module's way of handling it:

1) Input schema and its attributes if applicable, are probed.
2) If the parameter of the Message has any kind of connection with the semantics tree, then the Semantics schema is probed.
3) Likewise if the parameter of the Message has any kind of connection with the output schema, then the Output schema and its attributes (if applicable) is probed.

- Finally, Messages are produced within the module for its consumers.