Access control for data integration in presence of data dependencies

Mehdi Haddad, Mohand-Saïd Hacid

Outline

- Introduction
- Motivating example
- Related work
- Approach
  - Detection phase
  - (Re)configuration phase
- Conclusion

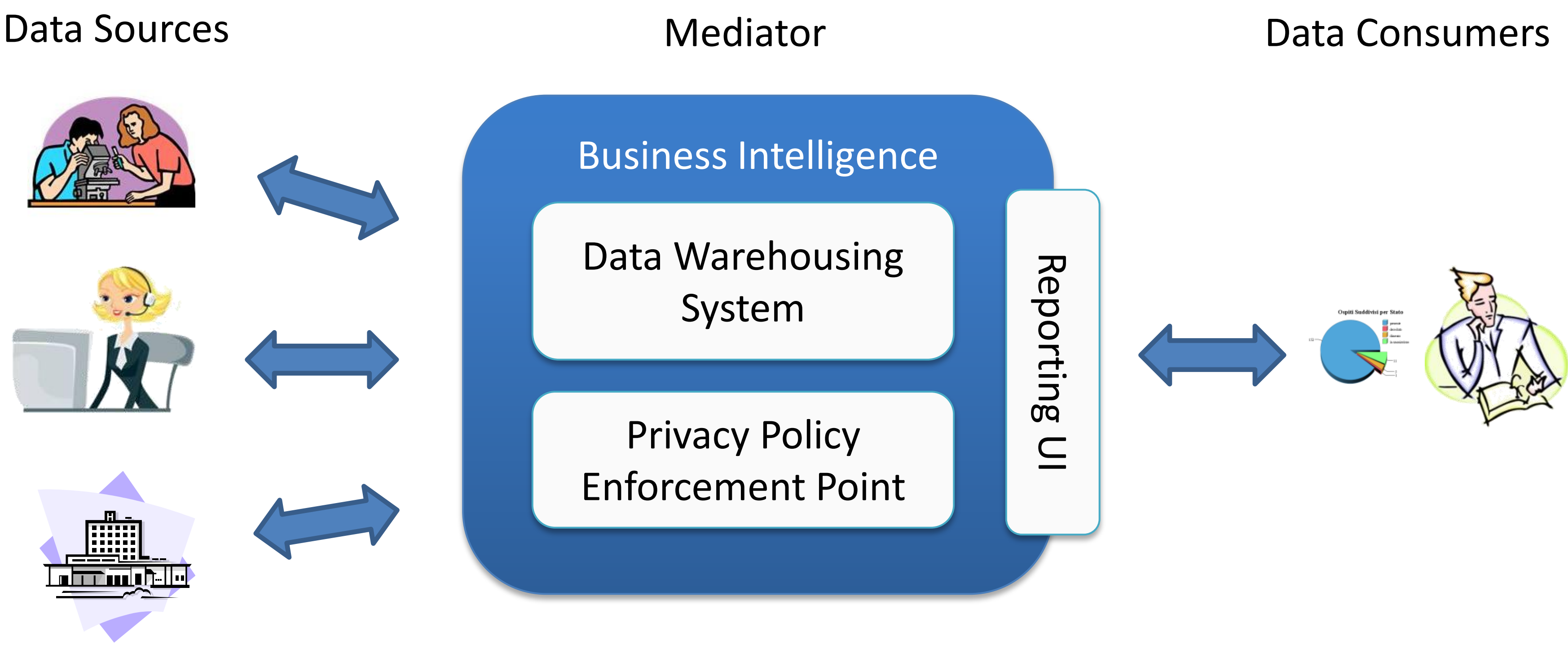
Introduction

- Access control aims at preventing unauthorized users from getting sensitive information.

- Access control protects data against unauthorized disclosure via direct access.

- Beyond access control: the inference problem
  - Preventing against indirect disclosure of data
  - Inferring sensitive information from non sensitive ones by resorting to semantic constraints

Context



- Many data sources.
- Each one with its own data schema.
- Each source has its own privacy policies defined on its own schema.
- Global As View (GAV) integration approach.

The inference problem [1]

- *The inference problem is the ability to deduce sensitive information from non sensitive one.*

- Two methods to make an inference :
  - Obtaining information about individuals from information about a population (e.g. statistics).
  - Combining non sensitive information with semantic constraints (e.g. metadata) to obtain sensitive information.

[1] Csilla Farkas, Sushil Jajodia: The Inference Problem: A Survey. SIGKDD Explorations 4(2): 6-11 (2002)

Access control of association

- Access to a set of attributes simultaneously is more sensitive than accessing each attribute individually.

- Example: consider the attributes SSN and Disease
  - The individual access to SSN or Disease could be allowed, whereas access to both attributes simultaneously is denied.
  - The association ***patient-disease*** is sensitive.

Motivating example

Sources
S1(SSN, Diagnosis, Doctor).
S2(SSN, AdmissionDate).
S3(SSN, Service).

Authorization policy at S1
Nurses are prohibited from accessing the association of SSN and Diagnosis.

Authorization rule
(SSN, Diagnosis) :- S1(SSN, Diagnosis, Doctor), role = nurse.

Motivating example

Mediator
M(SSN, Diagnosis, Doctor, AdmissionDate, Service) :-
S1(SSN, Diagnosis, Doctor) , S2(SSN, AdmissionDate),
S3(SSN, Service).

Functional dependencies
FD1 : AdmissionDate, Service ⟶ SSN
FD2 : AdmissionDate, Doctor⟶ Diagnosis

Authorization policy at the mediator (Propagation)
Nurses are prohibited from accessing the association of SSN and Diagnosis.
Authorization rule
(SSN, Diagnosis) :- M(SSN, Diagnosis, Doctor, AdmissionDate, Service),
role = nurse.

Motivating example

- A malicious user could execute the following queries :

Q1 (SSN, AdmissionDate, Service).
Q2(Diagnosis, AdmissionDate ,Service).

- Combining the results of the two queries by a join and taking advantage of FD1, a malicious user will obtain SSN and diagnosis, thus will violate the authorization policy

- Q3(SSN, Diagnosis) :- Q1 (SSN, AdmissionDate, Service),
  Q2(Diagnosis, AdmissionDate ,Service).

Motivating example

- The issue arises from the following
  - New semantic constraints appear at the mediator (e.g., FD1).
  - No source could have considered this new semantic constraints while defining its policy.
- Propagating and combining the sources' policies is not sufficient.

⇒ The need for a methodology that considers both combination and new semantic constraints that appear at the mediator.

Goal

- Help/advise the administrator defining the mediator’s policy such that:
  - Each source policy has to be preserved.
  - Prevent against illegal accesses
    - Direct access : ask for sensitive information.
    - Indirect access : infer sensitive information.
  - Maximize the availability at the mediator level.

State of the art

- To deal with the inference problem two main approaches have been proposed
  - At the design time
    - Modifies the schema or the policy in such a way that no inference could appear.
  - At the execution time
    - Keeps track of the previous queries and use them to make a decision about the current query.

State of the art

- At the design time [2]
  - Considers functional dependencies.
  - Assumes that if X $\longrightarrow$ Y then Y is "computable" from X.
  - Propagates the constraints of Y to X.
  - Does not consider association of information.

[2] Tzong-An Su, Gultekin Özsoyoglu: Data Dependencies and Inference Control in Multilevel Relational Database Systems. IEEE Symposium on Security and Privacy 1987: 202-211

State of the art

- At the execution time [3]
  - Considers past queries to make a decision about the current query.
  - Does not consider functional dependencies.
  - Does not consider access to associations.

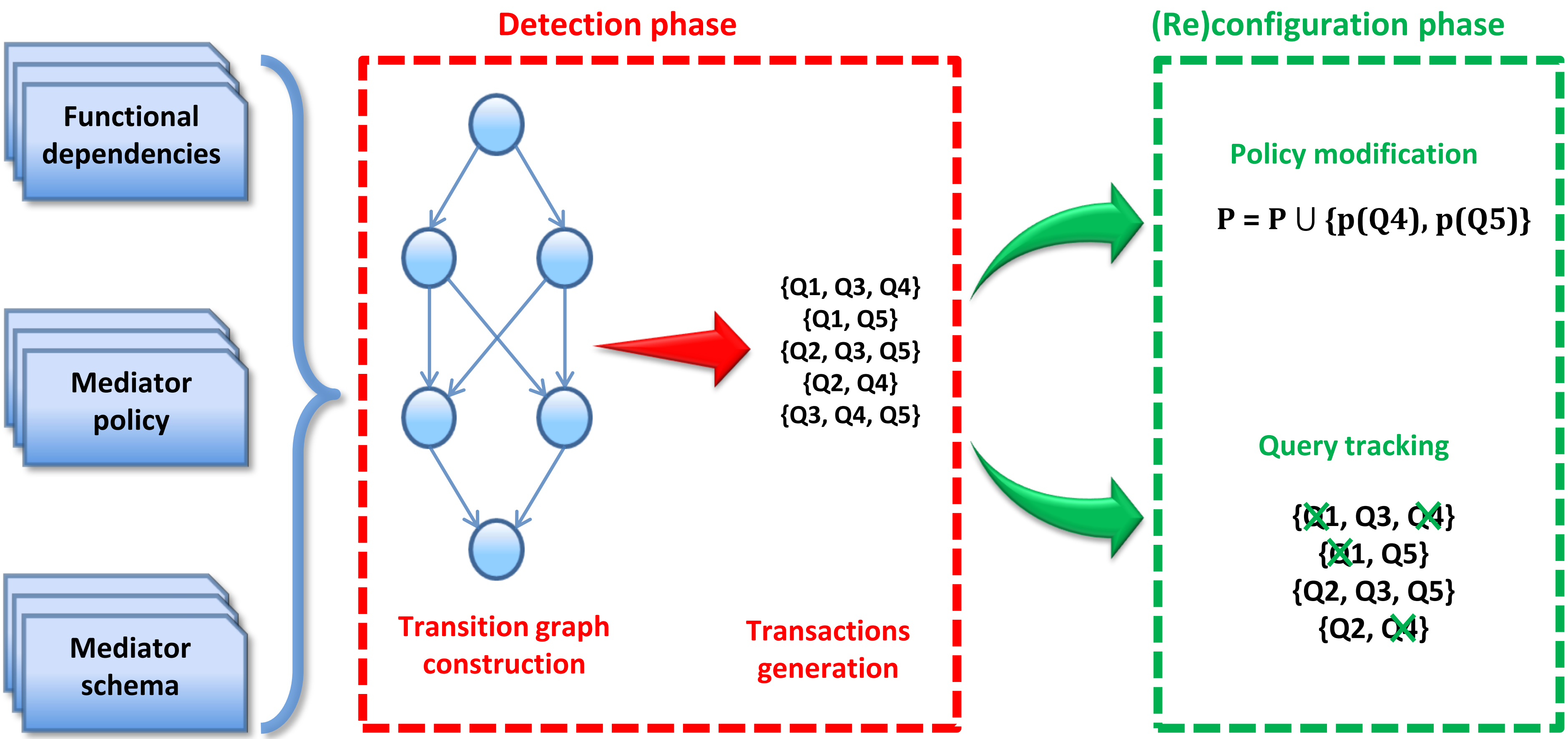
[3] MB Thuraisingham. Security checking in relational database management systems augmented with inference engines. Computers & Security, 6(6):479-492, 1987

Contribution

Assumptions

- Relational model & conjunctive queries.
- Global As View (GAV) integration approach
  - Each virtual relation of the mediator is constructed by a conjunctive query over the sources' relations.
  - e.g., M (SSN, Diagnosis, Doctor, AdmissionDate, Service) :- S1(SSN, Diagnosis, Doctor) , S2(SSN, AdmissionDate), S3(SSN, Service).
- Authorization rules expressing prohibition
  - e.g., (SSN, Diagnosis) :- S1(SSN, Diagnosis, Doctor), role = nurse.
- Semantic constraints : functional dependencies.

Methodology



Methodology

- Detection phase
  - Transition graph construction.
  - Violating transactions generation.
- (Re)configuration phase
  - Solution 1 : Policy revision.
  - Solution 2 : Query tracking.

Detection phase : problem definition

- Inputs
  - Sources' policies propagated to the mediator.
  - Functional dependencies that hold at the mediator level.

- Output
  - The set of all the transactions that could induce privacy violations.

Graph construction

Functional dependencies
FD1 : AdmissionDate, Service ⟶ SSN
FD2 : AdmissionDate, Doctor ⟶ Diagnosis

(SSN, Diagnosis)

Graph construction

Functional dependencies

FD1 : AdmissionDate, Service ⟶ SSN

FD2 : AdmissionDate, Doctor ⟶ Diagnosis

(SSN, Diagnosis)

FD1

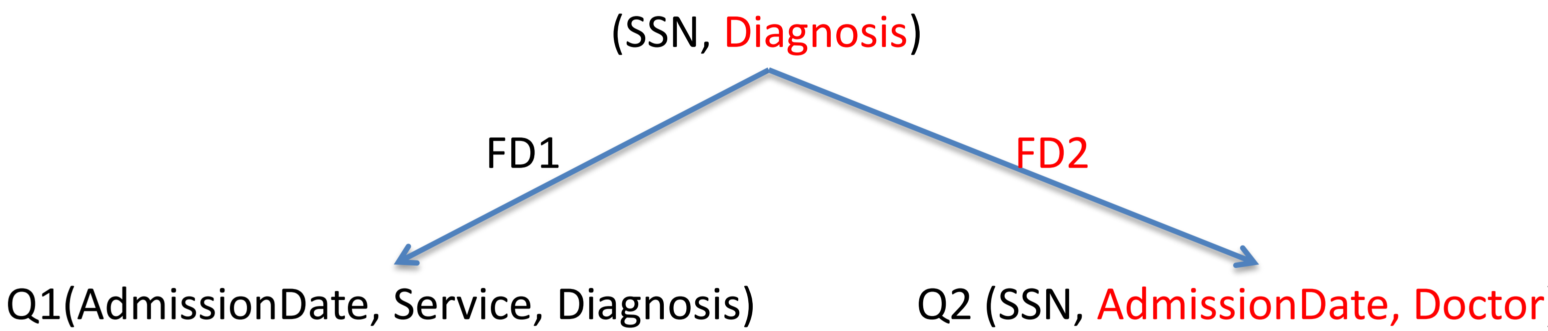
Q1 (AdmissionDate, Service, Diagnosis)

Graph construction

Functional dependencies

FD1 : AdmissionDate, Service ⟶ SSN

FD2 : AdmissionDate, Doctor ⟶ Diagnosis

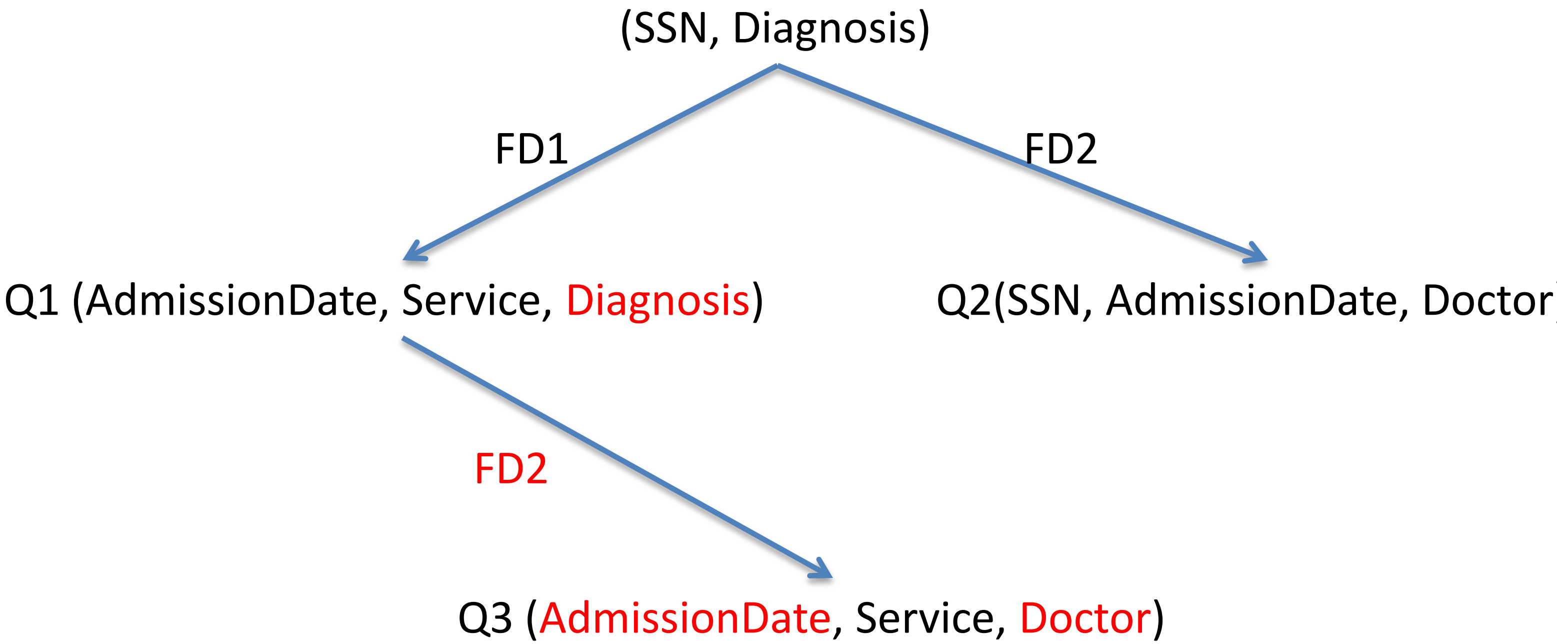


Graph construction

Functional dependencies

FD1 : AdmissionDate, Service ⟶ SSN

FD2 : AdmissionDate, Doctor ⟶ Diagnosis

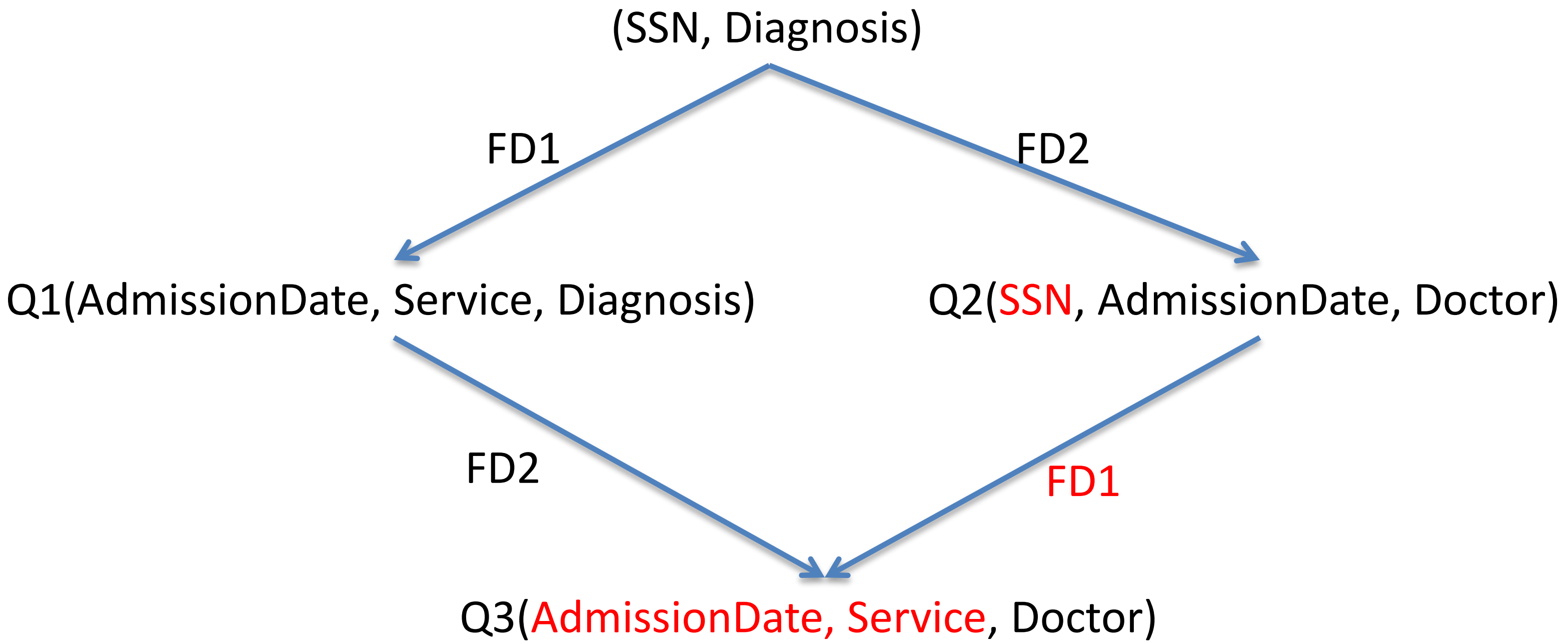


Graph construction

Functional dependencies

FD1 : AdmissionDate, Service ⟶ SSN

FD2 : AdmissionDate, Doctor ⟶ Diagnosis



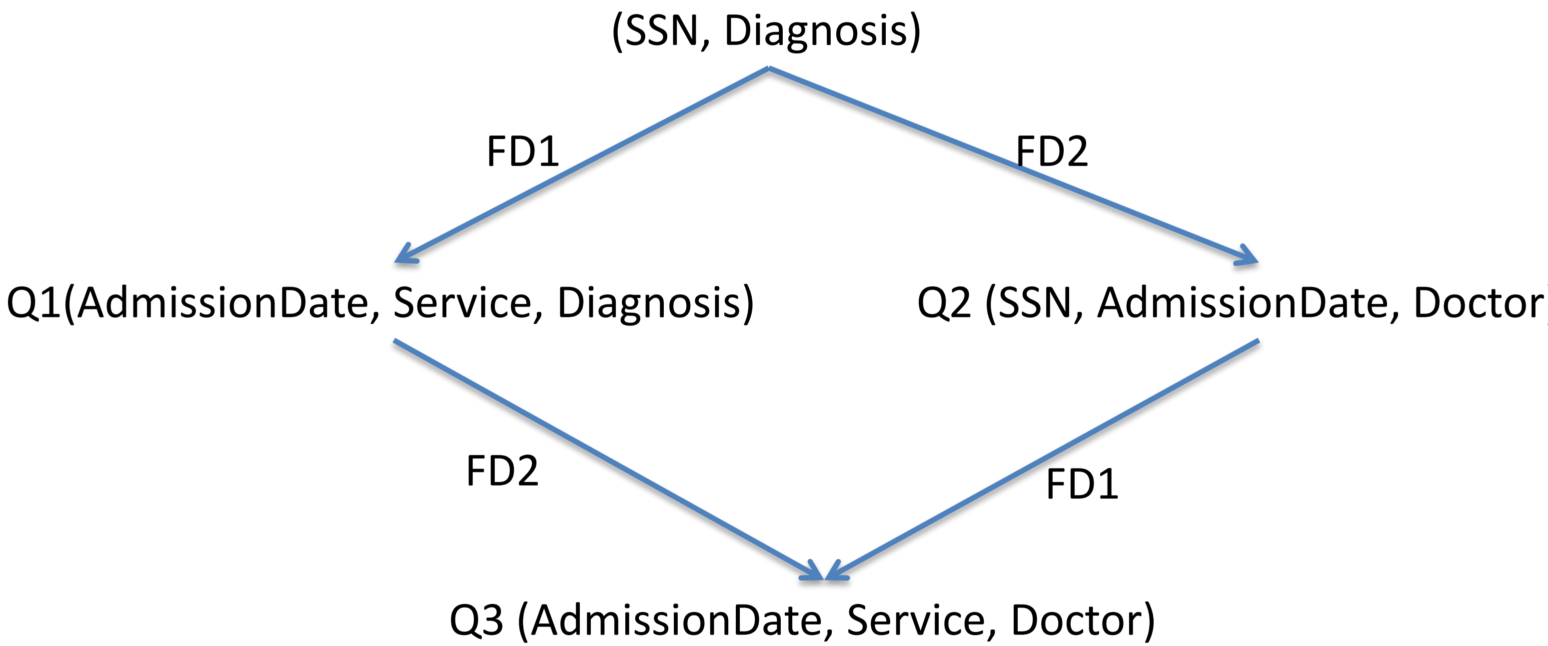
Upper bound & termination

- Assumption
  - WLOG, each FD has a RHS of one attribute.
- n: the number of attributes of the policy.
- m : the number of functional dependencies in FD^+ that have an attribute of the policy as RHS.
- The upper bound of the order (number of nodes) of the graph is :

$$\left(\frac{m}{n}\right)^n$$

⇒ The graph construction algorithm terminates.

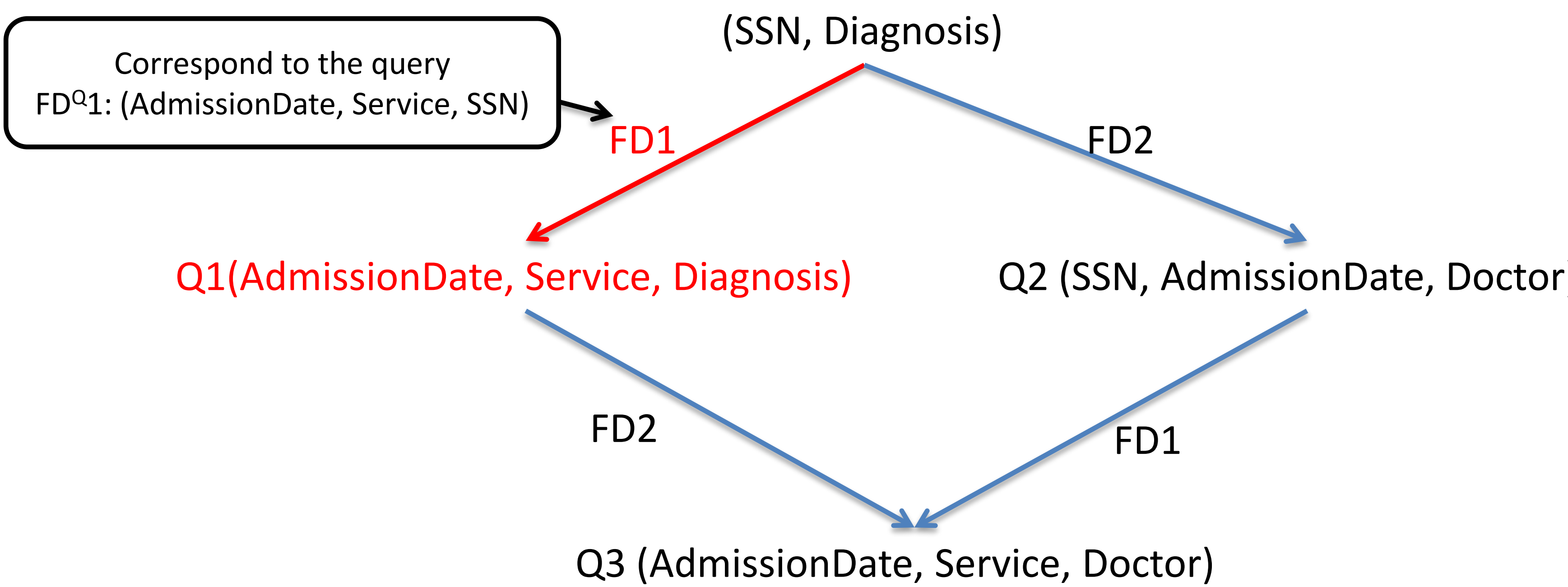
Generation of violating transactions (1/4)



How to generate the violating transactions?

- Each path between the initial node and a node Qi represents a transaction.
- A transaction is composed of all FDs on the path and the query of the node Qi.

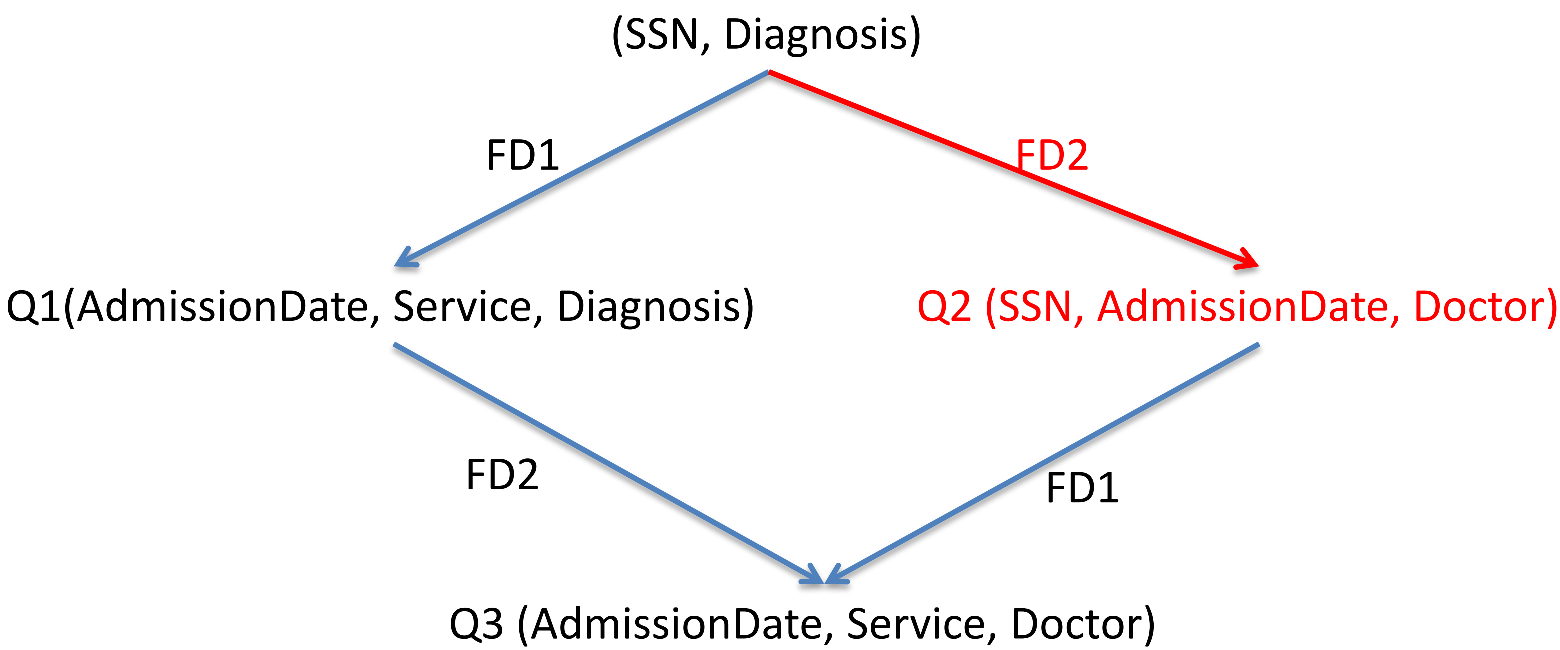
Generation of violating transactions (2/4)



Transactions

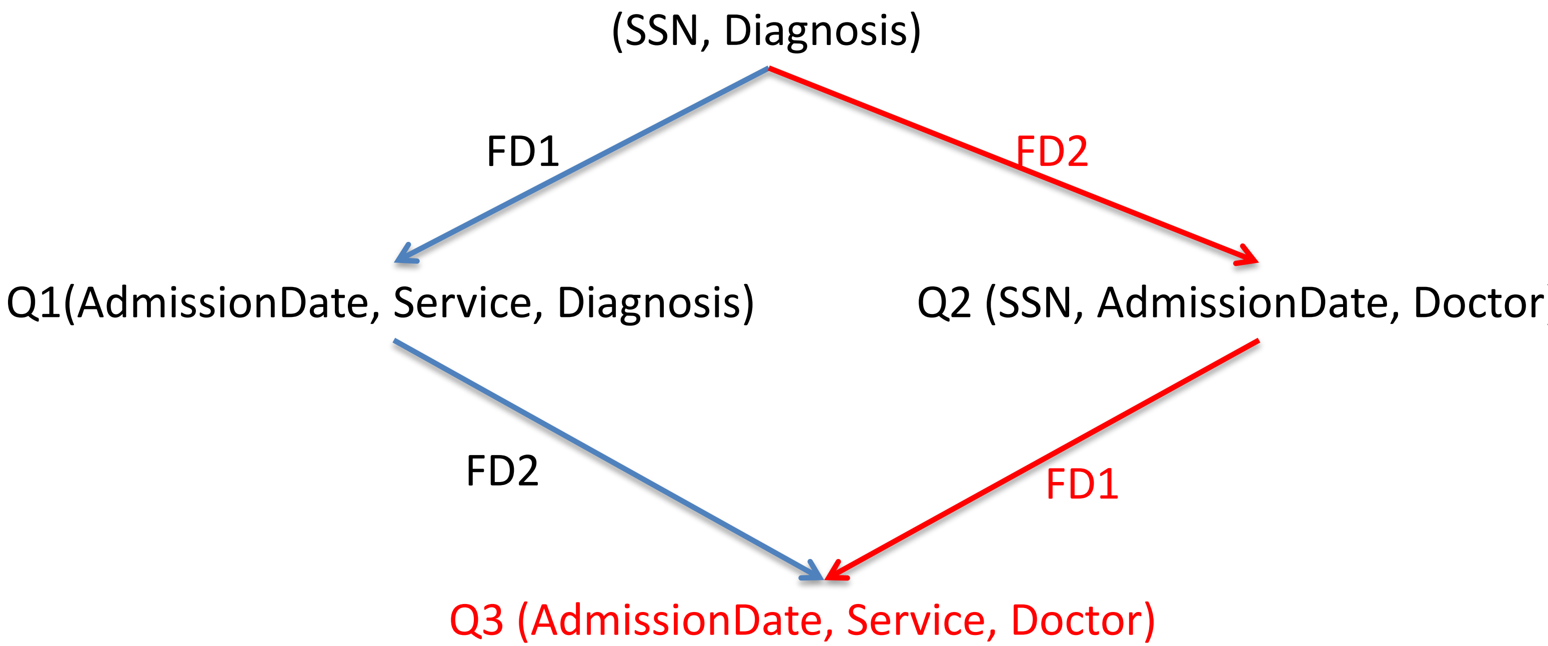
T1 ={$FD^{Q}1$, Q1}

Generation of violating transactions (3/4)



Transactions
T1 ={FD^Q1, Q1}
T2 ={FD^Q2, Q2}

Generation of violating transactions (4/4)



Transactions

T1 ={$FD^{Q}1$, Q1}

T2 ={$FD^{Q}2$, Q2}

T3 ={$FD^{Q}1$, $FD^{Q}2$, Q3}

(Re)configuration phase

- How to use these violating transactions?
  - At the design time : Policy revision
    - Add a new set of authorization rules.
    - No transaction could be completed.
  - At the execution time : Query tracking
    - Keep track of the user's queries.
    - Avoid the execution of the queries of a single transaction.

Solution 1 : Policy revision

- In the previous phase we have generated a set of transactions.

T1={Q1, Q2, Q3}
T2={Q3, Q4}
T3={Q5, Q6}
T4={Q7, Q6}

Q={Q3, Q6}

- If we add new authorization rules such that for any Ti at least one Qj is denied, then the policy will be preserved.
- Query cancellation problem : find the minimum set of Qj.

Query cancellation : problem definition

- Input : A set of violating transactions

$T1=\{Q^1_1, Q^1_2, \ldots Q^1_{n1}\}$
$T2=\{Q^2_1, Q^2_2, \ldots Q^2_{n2}\}$
…
$Tn=\{Q^n_1, Q^n_2, \ldots Q^n_{nn}\}$

- Output : a set Q of queries such that:
  - $\forall i, Ti \cap Q \neq \emptyset$
  - Q is minimal ($\nexists$ Q' st $\forall i, Ti \cap Q' \neq \emptyset$ and $|Q'|<|Q|$)

Complexity study

- Query cancelation problem is NP-complete.
  - Proof by reduction from the minimum dominating set problem.

- The associated optimization problem is NP-hard.

⇒ These results induce the use of exponential algorithm to obtain an exact solution.

Policy revision

- Find the minimum set of queries to be denied
  – Add a new rule for each query.
  – Ensure, at the design time, that no violating transaction could be completed.
- Finding the minimum set of queries increases the availability at the mediator level.

Solution 2 : Query tracking

- History based solution
  - Consider past queries to take a decision about the current query.
- Problem definition
  - Input
    - Past queries.
    - A set of violating transactions.
    - Current query.
  - Output
    - Decision about the current query (accept or deny).

Example

- Let T ={Q1, Q2, Q3} be a transaction.
- Let $Q^u=\{Q^u_1, Q^u_2, Q^u_3, Q^u_4\}$ be a sequence of user's queries.

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_4$

Example

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted

Example

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted
Q^u_2	T ={Q1, Q2, Q3}	Q^u_2 is accepted

Example

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted
Q^u_2	T ={Q1, Q2, Q3}	Q^u_2 is accepted
Q^u_3	T ={Q1, Q2, Q3}	Q^u_3 is accepted

Example

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted
Q^u_2	T ={Q1, Q2, Q3}	Q^u_2 is accepted
Q^u_3	T ={Q1, Q2, Q3}	Q^u_3 is accepted
Q^u_4	T ={Q1, Q2, Q3}	Q^u_4 is **denied**

Labeling method

- A query Qi could be simulated by a set of user's queries.
- If we modify the previous example as follows:

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_1 \bowtie Q^u_2 \bowtie Q^u_3$
$Q3 \subseteq Q^u_4$

Labeling method

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_1 \bowtie Q^u_2 \bowtie Q^u_3$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted

Labeling method

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_1 \bowtie Q^u_2 \bowtie Q^u_3$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted
Q^u_2	T ={Q1, Q2, Q3}	Q^u_2 is accepted

Labeling method

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_1 \bowtie Q^u_2 \bowtie Q^u_3$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted
Q^u_2	T ={Q1, Q2, Q3}	Q^u_2 is accepted
Q^u_3	T ={Q1, Q2, Q3}	Q^u_3 is **denied**

Labeling method

Relationship between Qi and Q^u_i
$Q1 \subseteq Q^u_1$
$Q2 \subseteq Q^u_2$
$Q3 \subseteq Q^u_1 \bowtie Q^u_2 \bowtie Q^u_3$
$Q3 \subseteq Q^u_4$

User's queries	Transaction	Evaluation
Q^u_1	T ={Q1, Q2, Q3}	Q^u_1 is accepted
Q^u_2	T ={Q1, Q2, Q3}	Q^u_2 is accepted
Q^u_3	T ={Q1, Q2, Q3}	Q^u_3 is **denied**
Q^u_4	T ={Q1, Q2, Q3}	Q^u_1 is denied

Query tracking

- Importance of the labeling method.
- Consider combination of user’s queries to simulate a query of a transaction.
- We have defined a specific operator that considers these combination while building the user history.

Comparison of the two solutions

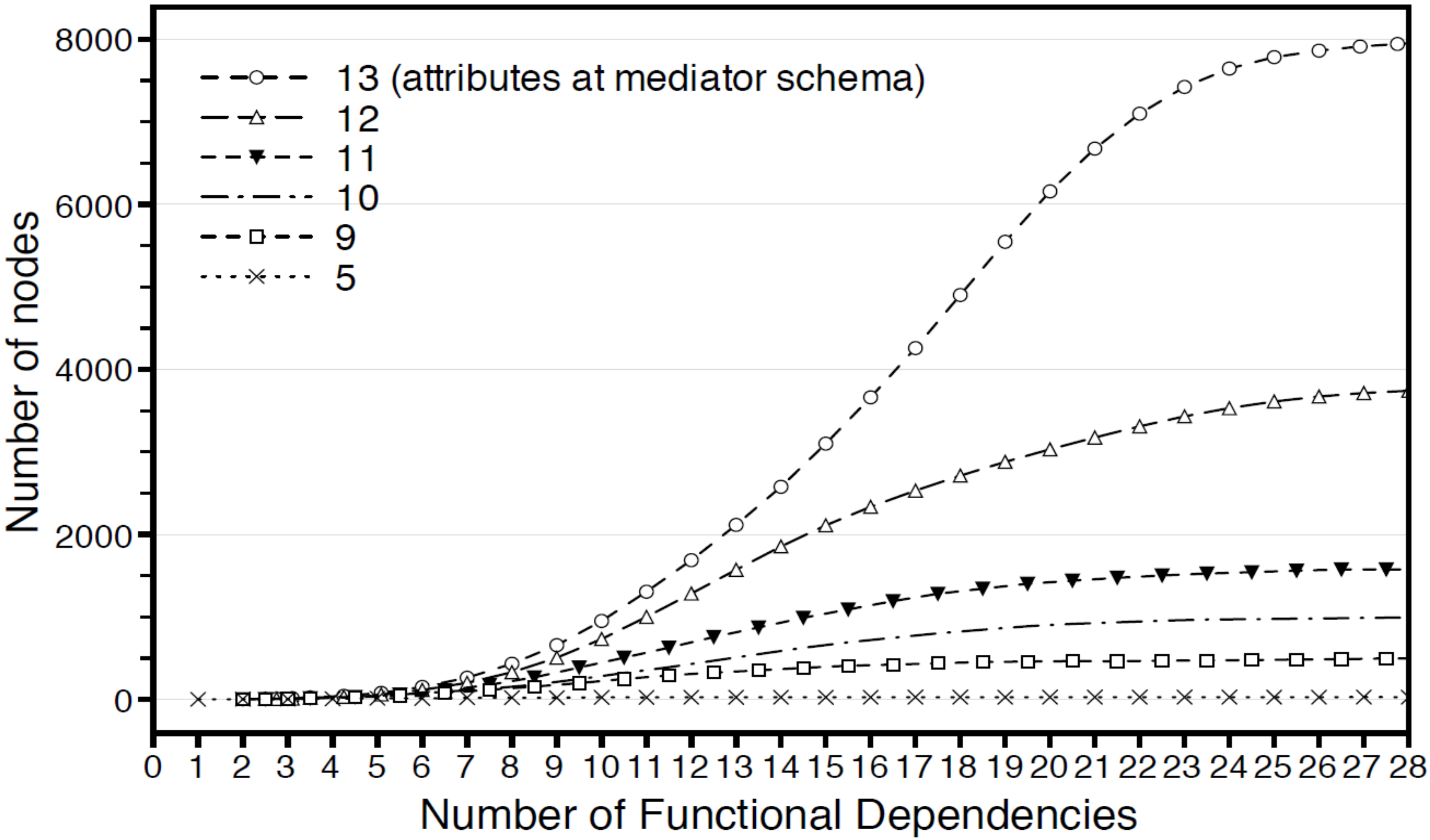
- Policy revision
  - Advantage : all the processing is achieved at design time.
  - Drawback : could be too restrictive.

- Query tracking
  - Advantage : maximizes the availability at the mediator level.
  - Drawback : maintaining the history of all users.

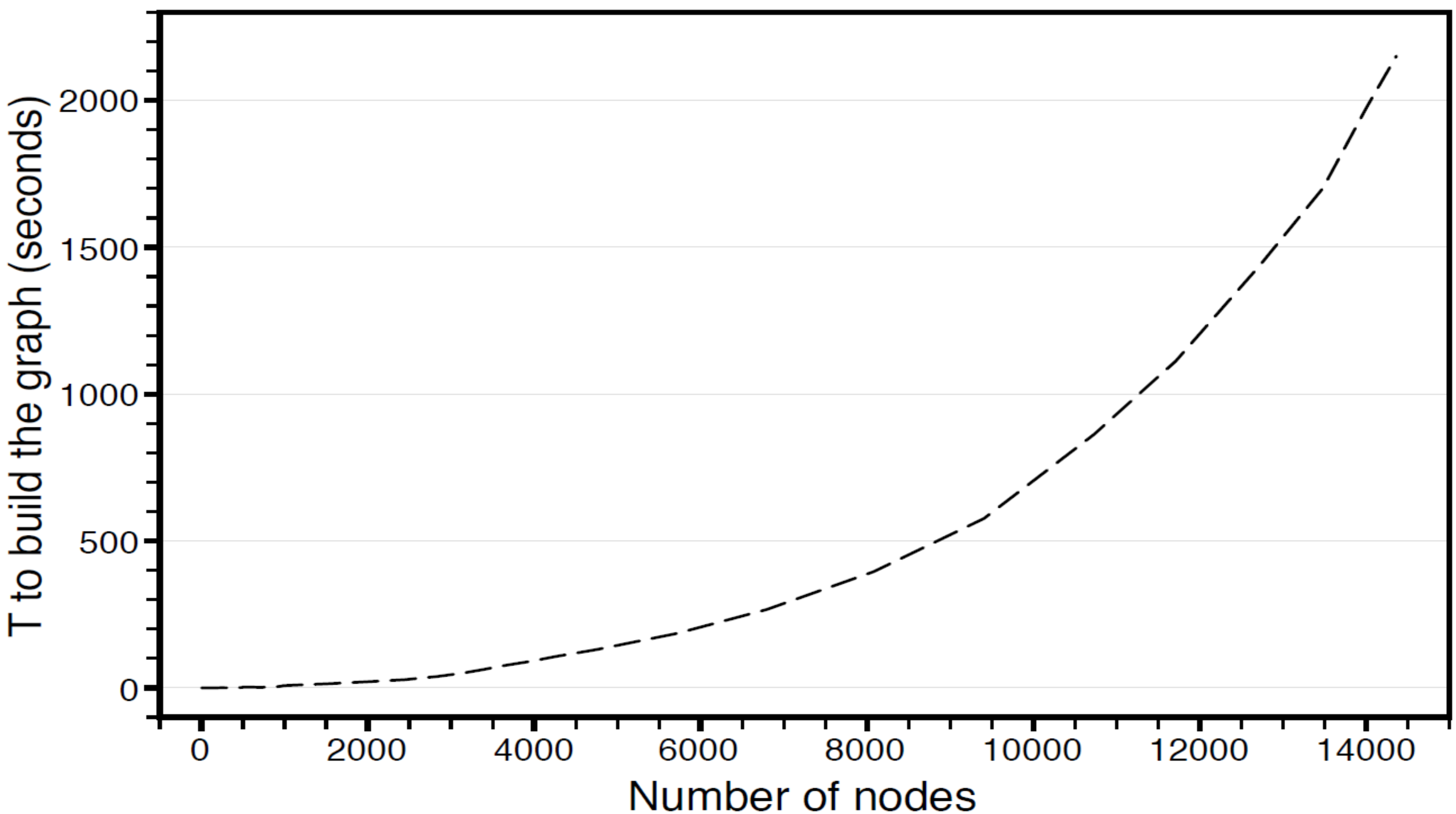
Experiments

- The proposed approach has been implemented and some experiments conducted:
  - We generated a mediator schema.
  - We generated a set of authorization rules.
  - We generated a set of functional dependencies.

Experiments



Experiments



Conclusion

- We have proposed a methodology that helps the administrator to define the mediator policy.

- We studied different theoretical aspects of the approach
  - Upper bound of the constructed graph.
  - NP-completness of the query cancellation problem.

- We conducted some experiments on synthetic

Perspectives

- Other kinds of dependencies
  - Inclusion dependencies.
  - Interaction between FDs and IDs.

- Other kinds of data integration (e.g., LAV).

- Mediator’s policy already defined
  - Consistency between the defined policy and the generated policy.

Thank you for your attention