

Pragmatic XML Access Control Enforcement Mechanism via Query Filtering and its Applications

Dongwon Lee

Dec. 20, 2004

Penn State University



Credits

- Joint work with
 - Padmapriya Ayyagari
 - Wang-Chien Lee
 - Cathy Li
 - Peng Liu
 - Robert Luo
 - Prasenjit Mitra
- From
 - XSym03, SDM04, CIKM04, etc

XML Research (IMHO)

- 1st Generation: **XML Model, QL**
 - Semi-structured/XML models and QLs
 - Eg, OEM, StrudQL, LOREL, ...
- 2nd Generation: **XML in RDBMS vs. Native XML**
 - XML-2-Relational Schema/Data Conversion
 - Relational-2-XML Schema/Data Publishing
 - XML encoding, indexing, storage, ...
 - Eg, STORED, Inlining, XRel, XTRACT, TIMBER, ...
- 3rd Generation: **Advanced Features of XML**
 - Building advanced features of XML databases
 - *XML-Something*: XML-Trigger, XML-Security, XML-Temporal, XML-Spatial, ...

XML Research (IMHO)

- In particular, in 3rd Generation,
- Building advanced features “XYZ” of XML using either **Vanilla XDBMS** or **RDBMS**
 - **Vanilla XDBMS**: Let middleware handle the feature XYZ so that underlying XDBMS can process the outputs
 - **RDBMS**: Convert *XML-XYZ* to *Relational-XYZ*

Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

Introduction

- **Relational Access Control** ensures only authorized users can access only authorized portion of relational data
 - Role-based security model (user << role)
 - GRANT / REVOKE
 - Can control table-level or column-level access
- Eg,
 - GRANT SELECT ON Foo to dlee
 - GRANT INSERT ON Bar(A, B) to admin

Introduction

- XML Access Control ensures only authorized users can access only authorized portion of XML data
- Emerging techniques and standards for XML Access Control
 - Model and semantics
 - Enforcement ⇔ Our Focus

Introduction

- Often, XML Access Control is represented as 5-tuple ACR
 - {subject, object, action, +/-, RC/LC}
 - Object is expressed by XPath
- Eg, “Manager can read employee’s project-related information, but not their salaries”

- (manager, //employee/proj, read, +, RC)
- (manager, //employee/@salary, read, -, LC)

ACR (Access Control Rule)

ACT (Access Control Table)

Introduction

- Relational AC vs. XML AC Comparison
 - Relational Model has table and column
 - XML has element and attribute
- In conversion, a node in an XML tree can be mapped to (in RDBMS)
 - Table
 - Column
 - Tuple
 - Cell (column X, tuple Y)
- Table and column level security is supported in commercial RDBMS, but not the other two cases



Introduction

XML Model

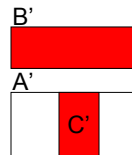
Relational Model

● $A \langle B^+, C \rangle, B \langle D, E \rangle \Leftrightarrow A'(C', FK), B'(D', E')$

● Control access to user "john"

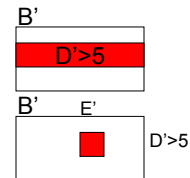
● Table: $/A/B \Leftrightarrow B'$

● Column: $/A/C \Leftrightarrow C'$



● Tuple: $/A/B[D > 5] \Leftrightarrow B' \text{ with } D' > 5$

● Cell: $/A/B[D > 5]/E \Leftrightarrow B'(E') \text{ with } D' > 5$



Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

Motivation

- Popular solution to enforce XML Access Control is to use “Materialized Views” (eg, [Dimiani, 2002; Bertino, 2002; Yu, 2002])
 - Construct a view per role/user
 - Once view is constructed, no more security check
 - Space cost / Maintenance issue
- Others rely on the support of security feature of XML database [Cho; 2002]
 - No XML databases have such features yet

Motivation

- Neither approach is fully satisfactory
- Our Approach is
 - **Framework-based**: devise and compare various approaches
 - **Practical solution**: can work with off-the-shelf XML database engine (ie, Vanilla XDBMS)

Outline

- Introduction
- Motivation
- **Framework**
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

Framework

Data

- XML data (document).
- Stored in XML database

Query

- Describes the information that users want in XPath
- Query has the same security role as the user who issues it

ACR

- Each ACR describes the access control policy of a role.
- Objects are specified in XPath

Scenarios

Scenario 1

Query

ACR

Data

Scenario 2

Query

ACR

Data

Scenario 3

Query

ACR

Data

Scenario 4

Query

Data

ACR

Scenario 5

Query

Data

ACR

Scenarios

Conventional

Query

ACR

Data

View-based

Query

ACR

Data

Pre-processing

Query

ACR

Data

Post-processing

Query

Data

ACR

Miscellaneous

Query

Data

ACR

Post-Processing

Query

Data

ACR

- Intermediate answers are computed as usual
- Then, ACR prunes out unsafe answers
- Suitable for role-based data delivery model, where the same data is delivered to different roles.
- Can be implemented by XML data filtering package (eg, YFilter [Diao, 2003])

Pre-Processing

Query

ACR

Data

- Query Q is **modified** to a safe one Q' by ACR
- Then, Q' is processed by regular XML engine
- “**Modified**” can be implemented by many ways
 - Primitive: $Q \cap \text{ACR} \Leftrightarrow Q'$
 - QFilter: $\text{QFilter}(Q, \text{ACR}) \Leftrightarrow Q'$

Pre-Processing: Primitive

- View Query and ACR as two **constraints** to satisfy
 - Q and + ACR: $Q \cap \text{ACR} \Leftrightarrow Q'$
 - Q and – ACR: $Q - \text{ACR} \Leftrightarrow Q'$
- Then, Q' is passed to regular XML engine that can handle XPath with set operator
 - Easy to implement
 - Performance is highly dependent on the capability of underlying XML engine (how it handles set operators, etc)

Pre-Processing: Primitive

- $Q: /dept[name='HR']//budget$
 - Manager John wants to access HR dept's budget
- ACR (for all manager/read)
 - R1: $/dept//salary, +, LC$
 - R2: $/dept/south, +, RC$
 - R3: $/dept[year=2003]//budget, -, LC$
- $Q' \Leftrightarrow Q \cap (+ \text{ rules}) - (- \text{ rules})$
 - $\Leftrightarrow Q \cap (R1 \cup R2) - R3$
 - $\Leftrightarrow /dept[name='HR']//budget \cap (/dept//salary \cup /dept/south) - /dept[year=2003]//budget$

Outline

- Introduction
- Motivation
- Framework
- **QFilter Details**
- QFilter Validation
- QFilter Applications
- Conclusion

Pre-Processing: QFilter

- Primitive Pre-Processing satisfies our two goals
 - Non-view based
 - Independent on underlying XML engine
- But, the re-written query Q' is not the most efficient form

Pre-Processing: QFilter

- Idea of QFilter: Improve Q' further for better performance
- **Contained Case** ($Q \cap ACR \Leftrightarrow Q$)
 - `/dept[year<2004]//budget` \cap `/dept//*`
 - \Leftrightarrow `/dept[year<2004]//budget`
- **Disjoint Case** ($Q \cap ACR \Leftrightarrow \{\}$)
 - `/dept[year<2004]//budget` \cap `/dept[year>2005]//budget` $\Leftrightarrow \{\}$
- **Overlapping Case** ($Q \cap ACR \Leftrightarrow Q'$)
 - `/dept[year<2004]//budget` \cap `//south/budget`
 - \Leftrightarrow `/dept/[year<2004]/south/budget`

Pre-Processing: QFilter

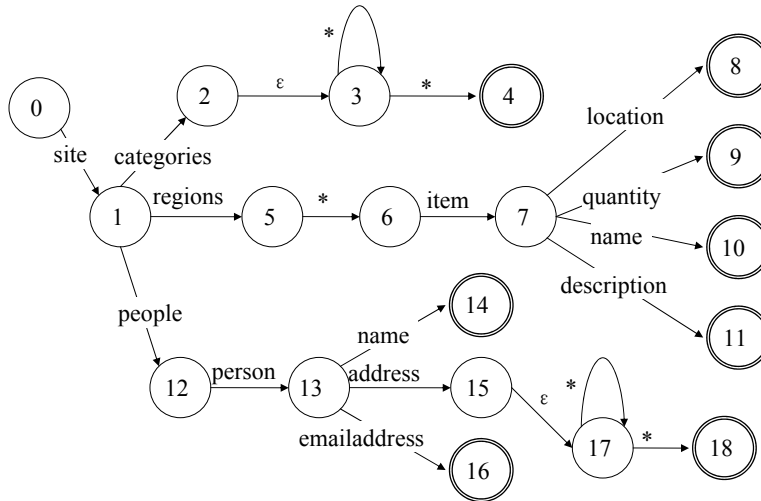
- QFilter captures ACR as NFA (Non-deterministic Finite Automata)
- Given Q, quickly determine if it is contained, disjoint or overlapping by traversing NFA
- When it's overlapping case, Q' is generated

- Cannot handle general case of XPath
- XPath containment is:
 - /, //, [], *: P [Wood, 2001] ⇔ QFilter supports this
 - =, NOT, <: undecidable [Neven, 2003]

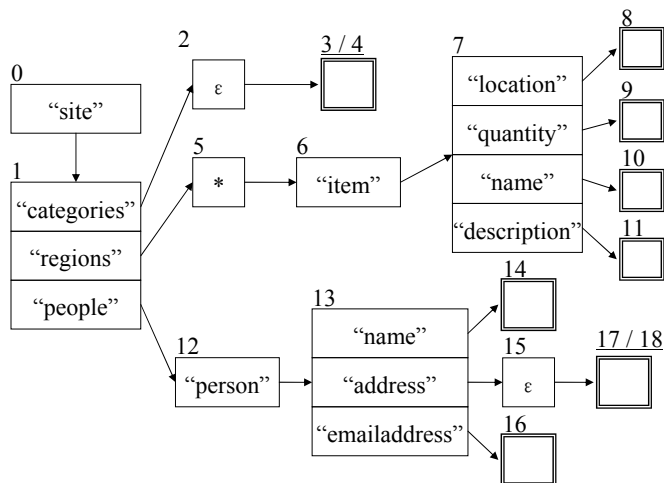
QFilter Example

- R1: /site/categories//*
- R2: /site/regions/*/item/location
- R3: /site/regions/*/item/quantity
- R4: /site/regions/*/item/name
- R5: /site/regions/*/item/description
- R6: /site/people/person/name
- R7: /site/people/person/address//*
- R8: /site/people/person/emailaddress

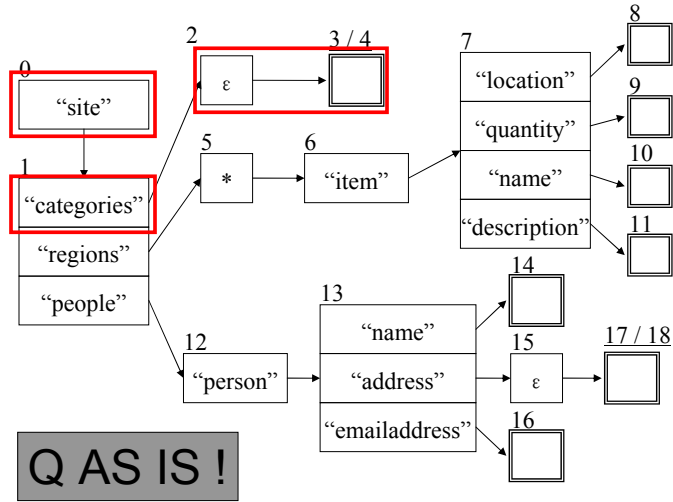
State Transition Map



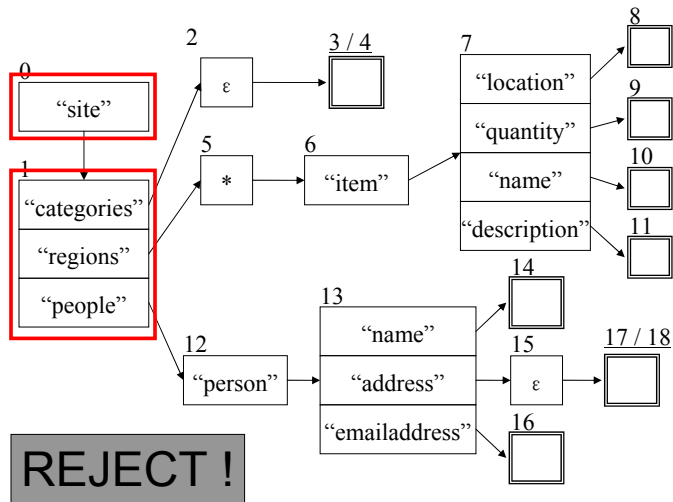
NFA



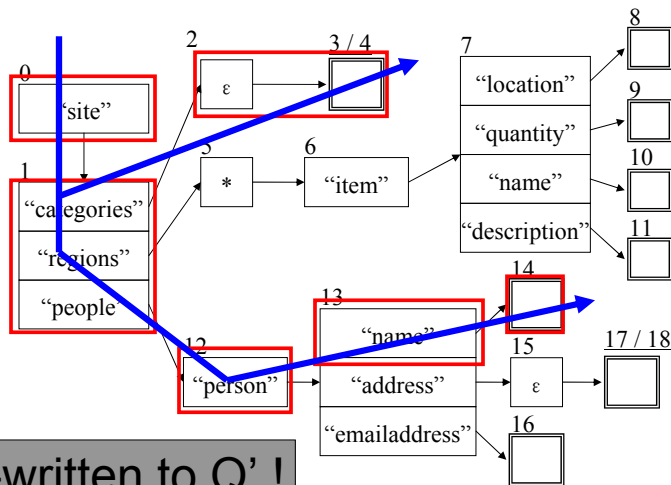
Q: /site/categories/NW/item



Q: /site/top//item



Q: /site/*/person/name

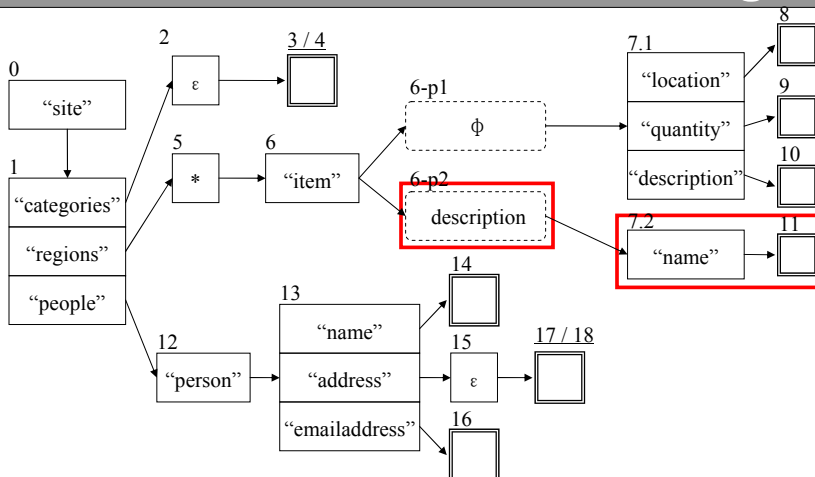


Re-written to Q' !



/site/categories/person/name \cup /site/people/person/name

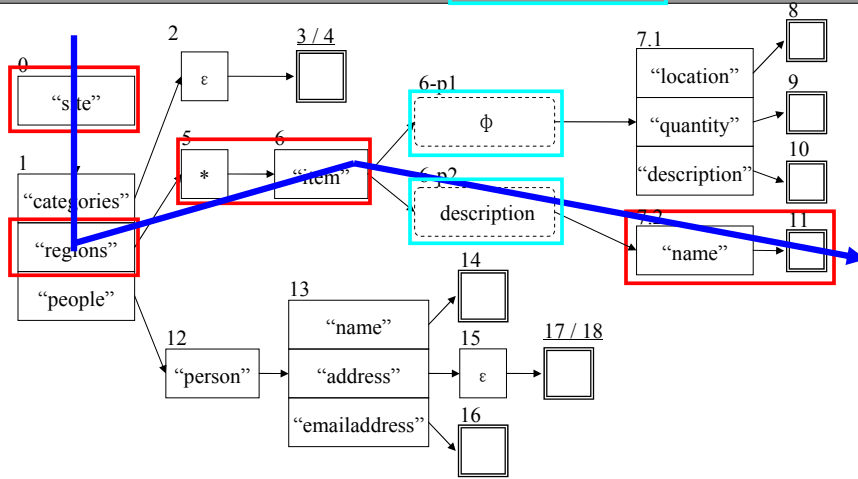
QFilter with Predicate Handling



R9: /site/regions/* /item[description]/name



Q: /site/regions/item[quantity]/name



Q' = /site/regions/item[quantity][description]/name

Dongwon Lee, 2004

33

QFilter Discussion

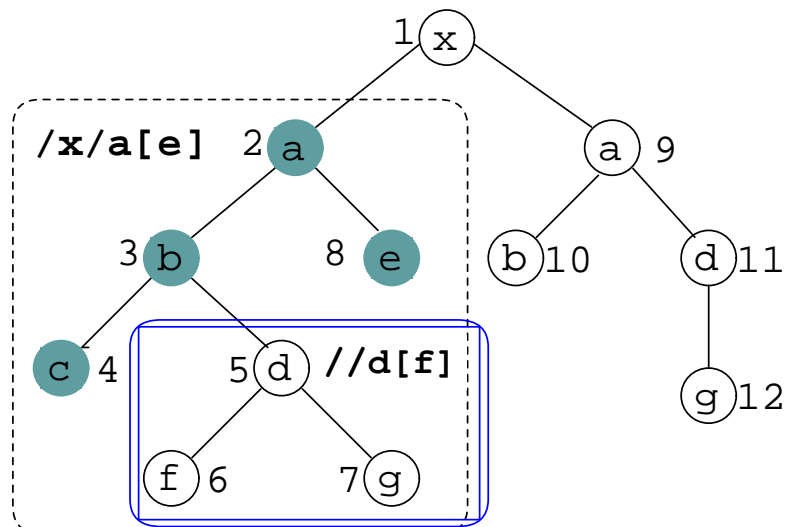
Theorem: Q' generated by QFilter never returns unauthorized answers to unauthorized users for XPath with /, //, *, [], []

- QFilter construction: $O(|ACR|)$
- QFilter execution
 - No *: $O(|Q|)$
 - *: $O(|NFA|)$
 - //: $O(|Q| * \Pi \text{ child for } i\text{-th } //)$
 - Worst case only occurs for a query $"/...//*/...//*"$

Deep-Set Operator

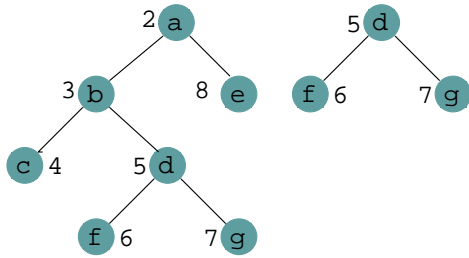
- QFilter often uses “set operators”
- However, standard semantics of XQuery/XPath set operators are too limited
- Eg,
 - $\text{/dept[year<2004]//budget} \cap \text{//south/budget} \Leftrightarrow \text{/dept[year<2004]/south/budget}$
 - $\text{/dept[year<2004]//budget} \cap \text{//south} \Leftrightarrow \{\}$

Deep-Set Example

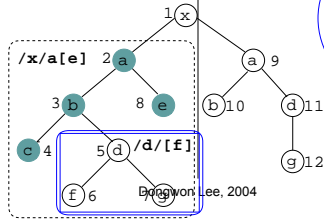
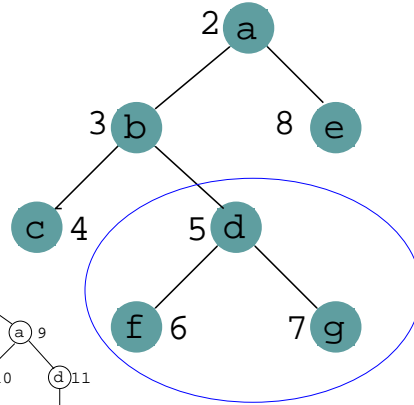


/x/a[e] UNION //d[f]

UNION



DEEP-UNION

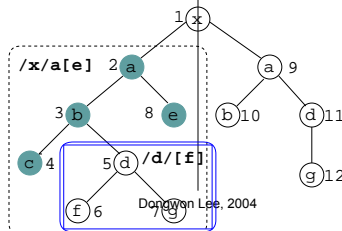
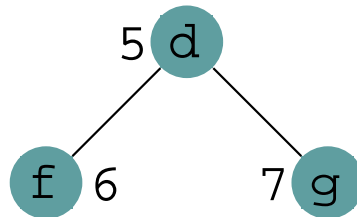


/x/a[e] INTERSECT //d[f]

INTERSECT

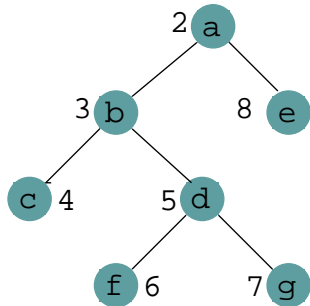
null

DEEP-INTERSECT

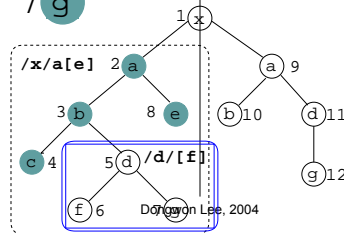
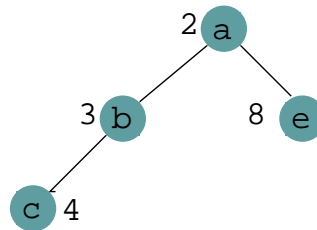


/x/a[e] EXCEPT //d[f]

EXCEPT



DEEP-EXCEPT



Eg, Deep-EXCEPT Implementation

// using XQuery's UDF

```

declare function local:DEEP-EXCEPT($P,$q) {
  for $pn in $p return
  if (empty($pn//* intersect $q) and empty($pn intersect
    $q) and not(empty($pn))) then $pn
  else if (not(empty($pn//* intersect $q))) then
    element{name($pn)}{
      $pn/@*, $pn/text(), for $pnn in $pn/*
        return local:DEEP-EXCEPT($pnn,$q)
    }
  else()
};
  
```

Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

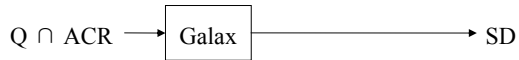
Evaluation Plan

- No Access Control



(a) No security support

- Pre-Processing Scenario



(b) Primitive

- Primitive

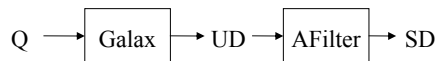
- QFilter

- Static Analysis [Murata; 2003]



(c) Pre-processing

- Post-Processing Scenario



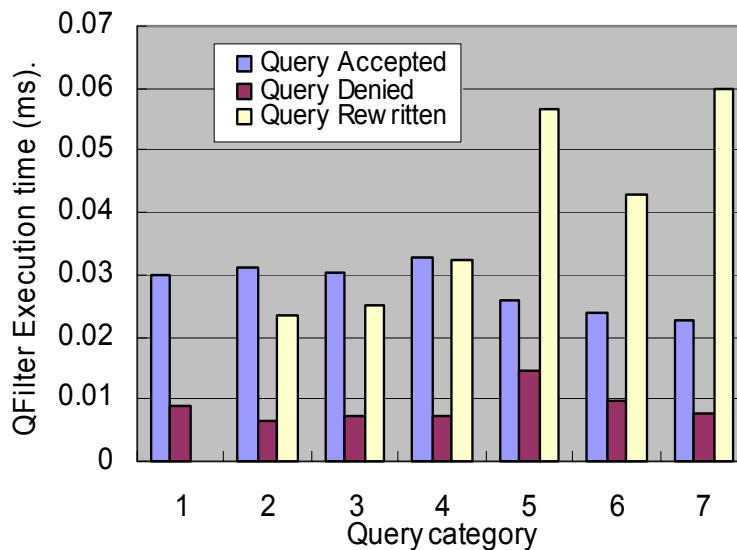
(d) Post-processing

Figure 1: Ways to support XML access control without using security features of DBMS

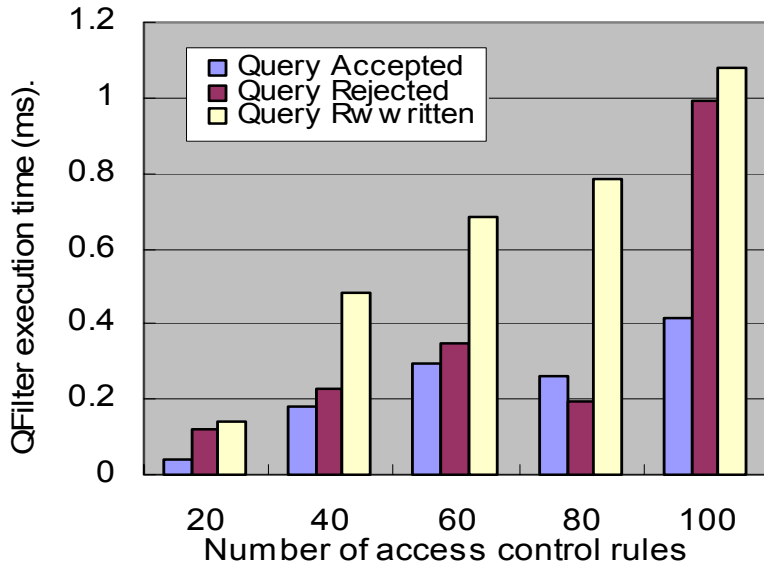
Set-Up

- XMark / Galax / YFilter
- XML data (1.5 MB)
- ACR
 - 550 Synthetic rules
 - 10 User-defined rules
- Q
 - 7 categories based on /, //, *, []
 - 100 Synthetic queries

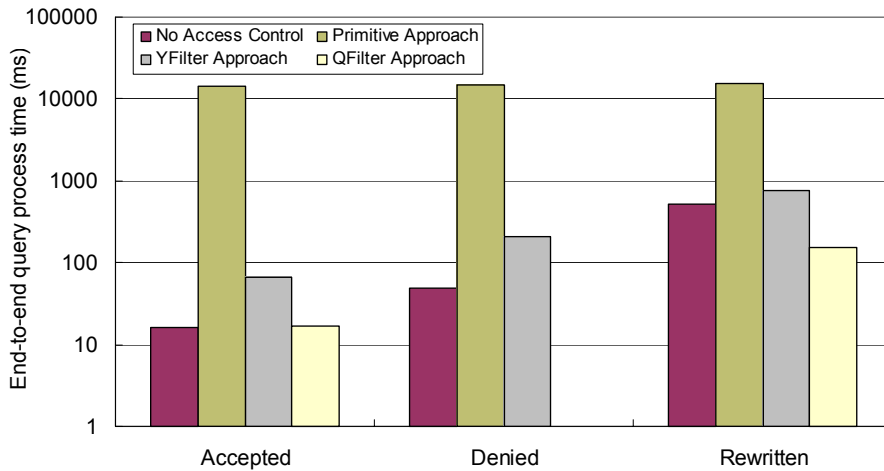
QFilter Performance



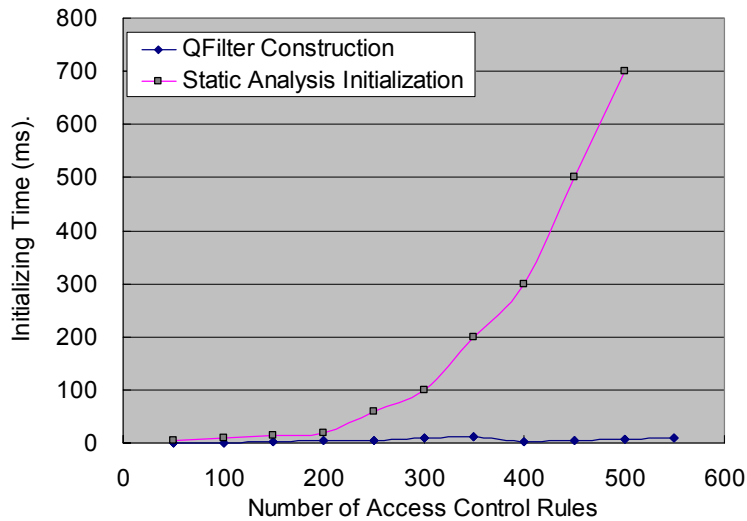
QFilter Performance



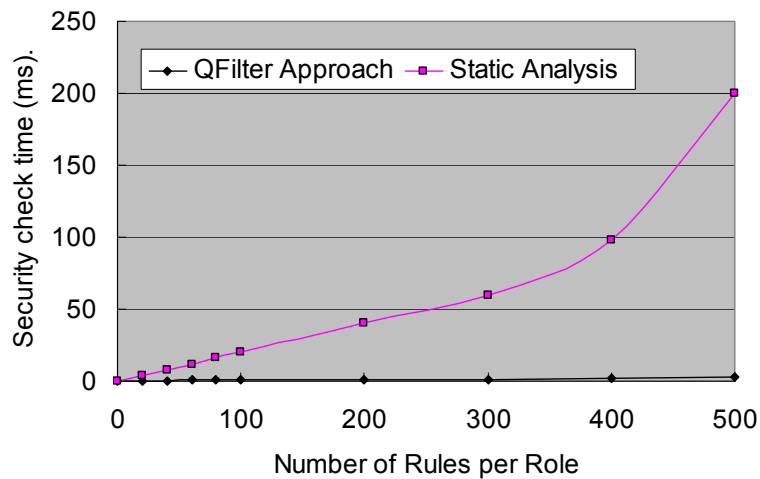
Comparison among Scenarios



Two Pre-Processing Methods



Two Pre-Processing Methods

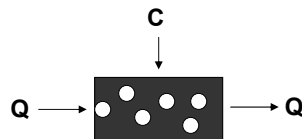


Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

QFilter Applications

- QFilter is in essence a black box that
 - Consists of *Constraints C*
 - Takes input Q
 - Outputs filtered result Q'



- Applications
 - ACR View Maintenance
 - Secure P2P

Scenarios Again

Conventional

Query

ACR

Data

View-based

Query

ACR

Data

Pre-processing

Query

ACR

Data

Post-processing

Query

Data

ACR

Miscellaneous

Query

Data

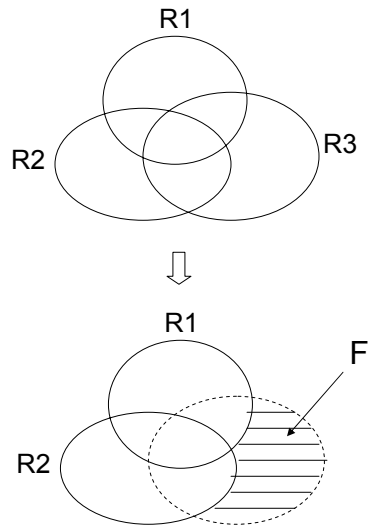
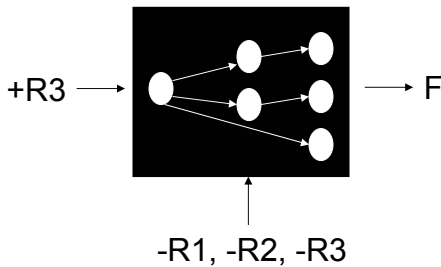
ACR

View-based XML Access Controls

- High maintenance cost when update occurs
- For an organization, frequent updates to ACR are possible
 - Rules are being added/removed
 - For each rule change, new view must be materialized
- Metric
 - Amount of data from DB to view
 - Time/Storage

View-based XML Access Controls

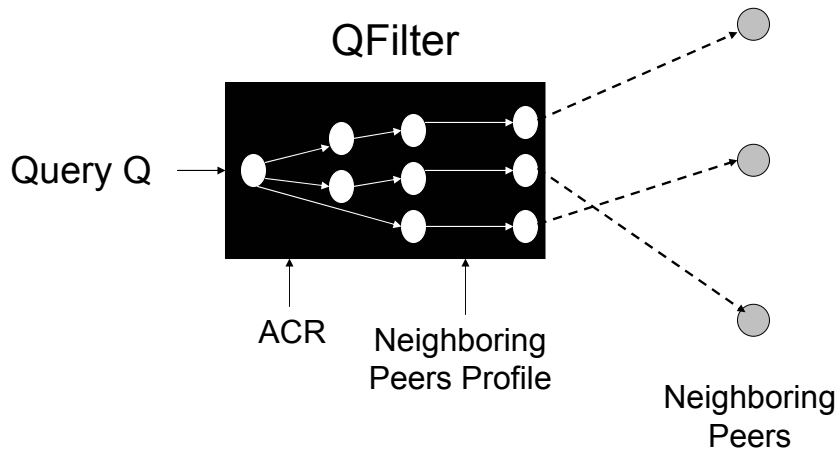
- T1: +R1, +R2, +R3
- T2: +R3 is removed



Secure P2P Overlay Network

- XML documents are stored on peer nodes (with possible duplication)
- Different peer nodes have different access controls
- When a node N gets a query Q , if N cannot handle Q , it has to forward Q to proper peers who have legitimate access controls.

Secure P2P Overlay Network



Conclusion

- Research on “XML-*Something*” in RDBMS or Vanilla XDBMS looks promising
- A framework and solution, QFilter, are presented in the context of XML Access Control
- Applications of QFilter

Thanks !