به نام خدا

Explainable Security for Relational Databases

Sajad Asadi

Seyed Mohammad Mehdi Ahmadpanah

Computer Eng. And IT Department

Amirkabir University of Technology

May 29, 2016

دانشکده مهندسی کامپیوتر
و فناوری اطلاعات

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

Reference

- Bender, Gabriel, Lucja Kot, and Johannes Gehrke. "Explainable security for relational databases." *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014.

Outline

- Introduction
- Goal
- Database and Security Views
- Disclosure Control System
- Policy Function
- Disclosure Orders
- Disclosure Lattice
- Policy Formulas
- Explanations
- Disclosure Control In Practice

Goal

- A novel security model
- **Formal Explanation** for security decisions
- Query is accepted or denied? Why?

Database & Security Views

User

UID	Name	Hobby
1	Babbage, Charles	math
2	Church, Alonzo	math
3	Lovelace, Ada	music
4	Turing, Alan	chess
5	von Neumann, John	history

Friend

UID1	UID2
1	3
3	1
2	4
4	2
4	5
5	4

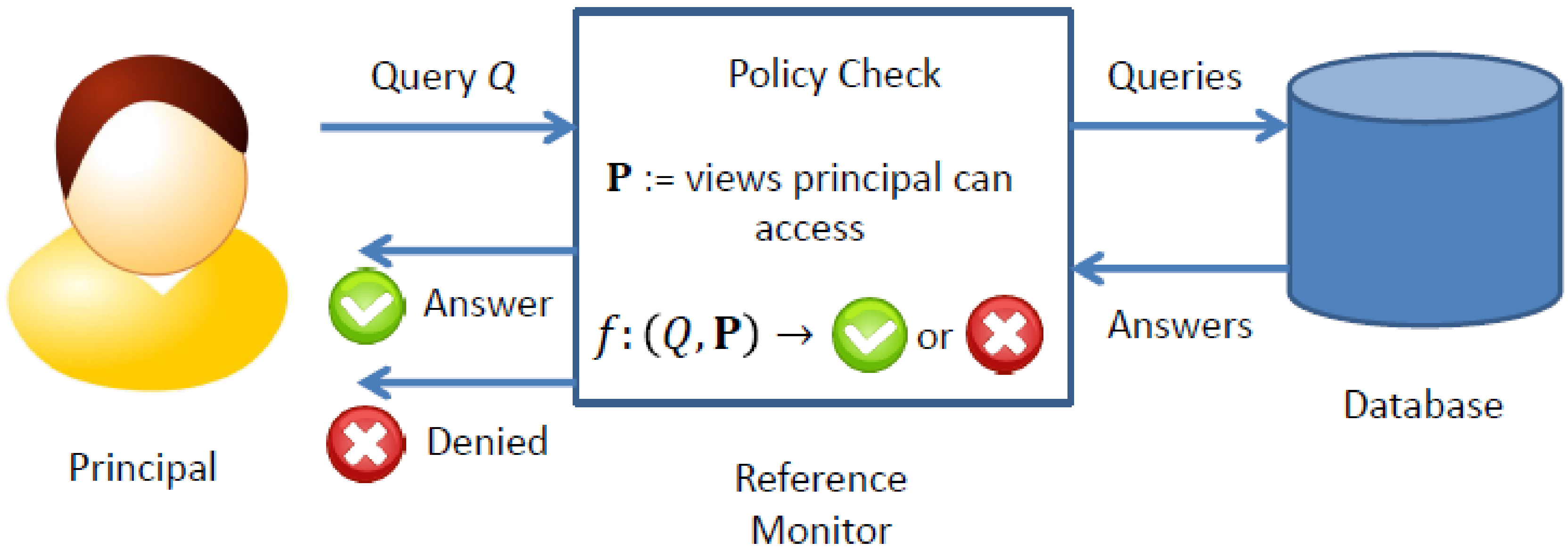
```
CREATE VIEW V1 AS
  (SELECT name, hobby FROM User WHERE uid = 1);

CREATE VIEW V2 AS
  (SELECT name FROM User WHERE uid = 1);

CREATE VIEW V3 AS
  (SELECT U.uid, U.name FROM User U, Friend F
    WHERE U.uid = F.uid2 AND F.uid1 = 1);
```

Written in Facebook Query Language

Disclosure Control System



Policy Function

- Variety of Policy Function:

  - Database-independent
  - Database-dependent

  - Not-data-derived
  - Data-derived

Example of not-data-derived

App has been granted to access V2 but not V1:

- SELECT name FROM V1;
- SELECT name FROM V2;

Two Different Semantics

In both relations and query answers:

- Set Semantics : No duplicates

- <u>Bag Semantics </u>: Can contain duplicate

Query Language: Conjunctive Query under bag Semantics

- H :- B

- H (Head of query) is a relational atom.
- B (Body of query) is conjunction of relational atoms.

- Atom is constant or variable.

Example

```
CREATE VIEW V1 AS
  (SELECT name, hobby FROM User WHERE uid = 1);

CREATE VIEW V2 AS
  (SELECT name FROM User WHERE uid = 1);

CREATE VIEW V3 AS
  (SELECT U.uid, U.name FROM User U, Friend F
    WHERE U.uid = F.uid2 AND F.uid1 = 1);
```

$$
\begin{aligned}
V_1(n,h) &:- \mathtt{U}(1,n,h) \\
V_2(n) &:- \mathtt{U}(1,n,h) \\
V_3(u,n) &:- \mathtt{U}(u,n,h), \mathtt{F}(1,u)
\end{aligned}
$$

Disclosure Orders

- Some set of views (V') reveal more information about data set than others (V).

DEFINITION 1. *A **disclosure order** is a binary relation on subsets of $\mathcal{U}$ such that*

(i) *If* $\mathbf{V} \subseteq \mathbf{V}'$ *then* $\mathbf{V} \preceq \mathbf{V}'$.

(ii) *If* $\mathbf{V} \preceq \mathbf{V}'$ *and* $\mathbf{V}' \preceq \mathbf{V}''$ *then* $\mathbf{V} \preceq \mathbf{V}''$.

(iii) *If* $\mathbf{V} \preceq \mathbf{V}''$ *and* $\mathbf{V}' \preceq \mathbf{V}''$ *then* $\mathbf{V} \cup \mathbf{V}' \preceq \mathbf{V}'$.

Example

(i) **Set containment**: $\mathbf{V} \preceq \mathbf{V}'$ if and only if $\mathbf{V} \subseteq \mathbf{V}'$.

(ii) **View determinacy**: $\mathbf{V} \preceq \mathbf{V}'$ if and only if the answers to the views in $\mathbf{V}'$ uniquely determine the answers to the views in $\mathbf{V}$ on every possible dataset.

(iii) **View rewriting**: $\mathbf{V} \preceq \mathbf{V}'$ if and only if every view in $\mathbf{V}$ has a rewriting using $\mathbf{V}'$.

Disclosure Orders
to
Disclosure Lattice Extension

We would like sets of views to form a lattice where the *greatest lower bound* (GLB) of $\mathbf{V}$ and $\mathbf{V}'$ (denoted $\mathbf{V} \sqcap \mathbf{V}'$) represents the information that is common to the two sets and the *least upper bound* (LUB) of $\mathbf{V}$ and $\mathbf{V}'$ (denoted $\mathbf{V} \sqcup \mathbf{V}'$) represents the combined information from the two sets.

Lattice Property

- Reflexivity
- Transitivity
- Not Antisymmetric!!!

$$(\Downarrow \mathbf{V}) = \{V \in \mathcal{U} : \{V\} \preceq \mathbf{V}\}$$

Theorem

THEOREM 1. *Let* $\mathcal{U}$ *be a set of views, and let* $\preceq$ *be a disclosure order for* $\mathcal{U}$. *Define* $\mathcal{I} = \{\Downarrow \mathbf{V} : \mathbf{V} \subseteq \mathcal{U}\}$. *Then* $(\mathcal{I}, \preceq)$ *is a lattice; details are as follows:*

- *LUB:* $(\Downarrow \mathbf{V}) \sqcup (\Downarrow \mathbf{V}') \;=\; \Downarrow (\mathbf{V} \cup \mathbf{V}')$.
- *GLB:* $(\Downarrow \mathbf{V}) \sqcap (\Downarrow \mathbf{V}') \;=\; (\Downarrow \mathbf{V}) \cap (\Downarrow \mathbf{V}')$.
- *Top element* $\top = (\Downarrow \mathcal{U}) = \mathcal{U}$, *bottom element* $\bot = (\Downarrow \emptyset)$.

Policy Function

- Data-derived policy functions can be expressed in terms of disclosure lattices.

f(Q;**P**) = 1 if and only if $\{Q\} \leq \mathbf{P}$

Where **P** is subset of security views.

$$\Downarrow \{Q\} \preceq \Downarrow \mathbf{P}$$

Policy Formulas

- Each view in V roughly corresponds to a single permission that a principal may or may not be granted.

$$\tau \quad ::= \quad 0 \quad | \quad 1 \quad | \quad V \quad | \quad (\tau \vee \tau) \quad | \quad (\tau \wedge \tau)$$

- $\mathbf{P} \vdash \varphi$ if and only if $f(\mathrm{Q};\mathbf{P}) = 1$

Means that φ is provable from **P**

Examples

- $\varphi = V1$
- $\varphi = V1 \vee V2$
- $\varphi = V1 \wedge V3$
- $V2 \vdash V1 \vee V2$

What φ tells?

- φ tells us precisely which combinations of the security views contain enough information to uniquely determine the answer to Q.

Rules of Reference Monitor

$\mathbf{P} \vdash 1$ and $\mathbf{P} \nvdash 0$

$\mathbf{P} \vdash V$ if and only if $V \in \mathbf{P}$

$\mathbf{P} \vdash (\varphi \vee \psi)$ if and only if $\mathbf{P} \vdash \varphi$ or $\mathbf{P} \vdash \psi$

$\mathbf{P} \vdash (\varphi \wedge \psi)$ if and only if $\mathbf{P} \vdash \varphi$ and $\mathbf{P} \vdash \psi$

Generating Data-Derived Formulas

procedure $\textsc{Policy}(\mathbf{Q}, \mathbf{V})$
    $\varphi \leftarrow 1$
    for $Q \in \mathbf{Q}$ **do**
        $\psi \leftarrow 0$
        for $V \in \mathbf{V}$ **do**
            if $\{Q\} \preceq \{V\}$ **then**
                $\psi \leftarrow (\psi \vee V)$
            end if
        end for
        $\varphi \leftarrow (\varphi \wedge \psi)$
    end for
    return φ
end procedure

Generating Data-Derived Formulas

$$\mathrm{policy}(\mathbf{Q}) = \bigwedge_{Q \in \mathbf{Q}} \left(\bigvee_{V \in \mathbf{V}:\ \{Q\} \preceq \{V\}} V \right)$$

Example

For example, let **Q** contain the queries

$$Q_{16}(1,n) \coloneq \mathtt{U}(1,n,h)$$
$$Q_{17}(h) \coloneq \mathtt{U}(u,n,h)$$

and let **V** consist of the views from Figure 3. Then

$$\text{policy}(\{Q_{16}\}) = V_9 \vee V_{10} \vee V_{11}$$
$$\text{policy}(\{Q_{17}\}) = V_9 \vee V_{12}$$

and therefore

$$\text{policy}(\{Q_{16}, Q_{17}\}) = (V_9 \vee V_{10} \vee V_{11}) \wedge (V_9 \vee V_{12})$$

Explanations

- Why-so: 0 for every view in policy formula that principal has not been granted access to.
- Why-not: 1 for every view in policy formula that principal has been granted access to.

Example

$$\text{policy}(\{Q_{16}, Q_{17}\}) = (V_9 \vee V_{10} \vee V_{11}) \wedge (V_9 \vee V_{12})$$

Why-so:

$$(V_9 \vee V_{10} \vee 0) \wedge (V_9 \vee 0) = (V_9 \vee V_{10}) \wedge (V_9) = V_9$$

Why-not:

$$(V_9 \vee 1 \vee 1) \wedge (V_9 \vee V_{12}) = 1 \wedge (V_9 \vee V_{12}) = (V_9 \vee V_{12})$$

Disclosure Control In Practice

- Reflective Policies
  - policies governing tuples in one part of database can depend on a different part of the same database
  - Frequently in Practice
    - e.g. Facebook's security policies

Running Example

SELECT U1.uid, U1.name FROM User U1 WHERE U1.uid IN

(SELECT F1.uid2 FROM Friend F1 WHERE F1.uid1 = 1)

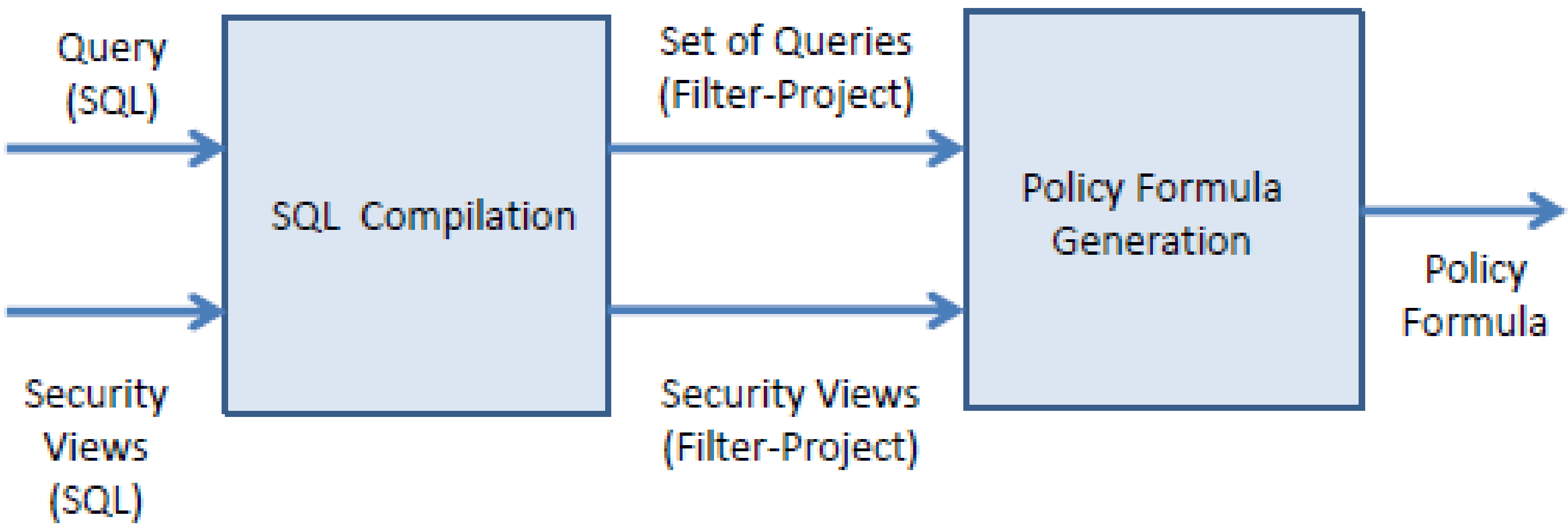
User

UID	Name	Hobby
1	Babbage, Charles	math
2	Church, Alonzo	math
3	Lovelace, Ada	music
4	Turing, Alan	chess
5	von Neumann, John	history

Friend

UID1	UID2
1	3
3	1
2	4
4	2
4	5
5	4

System Architecture



System Architecture (cont.)

- Each Security View: Filter-Project Query
  - SELECT-FROM-WHERE query with semi-joins but not inner or outer joins

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept

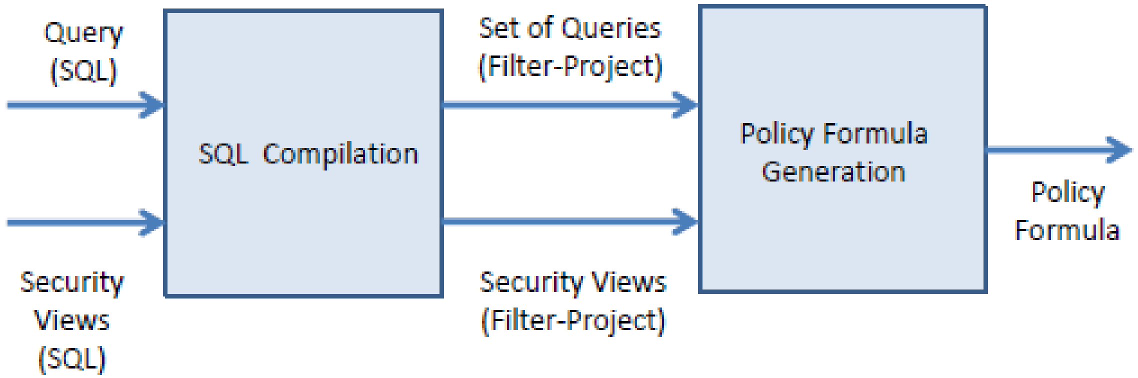
DeptName	Manager
Sales	Bob
Sales	Thomas
Production	Katie
Production	Mark

Employee ⋉ Dept

Name	EmpId	DeptName
Sally	2241	Sales
Harriet	2202	Production

System Architecture (cont.)

- Compilation is conservative
  - Policy Formula → Upper bound on the information needed to answer a query
- Database-independent
  - Policy formula and Explanations



Filter-Project Queries

- Tables vs. Table Instances
  - Query-independent vs. FROM clauses

SELECT U1.uid, U1.name FROM User U1, Friend F1
WHERE F1.uid1 = 1 AND F1.uid2 = U1.uid

- Which *columns* and *rows* in each table instance can affect the query's output?

Filter–Project Queries (cont.)

$$H :- B \ltimes F_1 \wedge F_2 \wedge \ldots$$

- A head atom H
- A main body atom B
- zero or more filter atoms F_1,F_2,…

Filter–Project Queries (cont.)

- Running Example:

SELECT U F1

WHERE F1

$$Q_{18}(u, n) :- \mathtt{U}(u, n, h) \ltimes \mathtt{F}(1, u)$$

```
SELECT U1.uid, U1.name FROM User U1 WHERE U1.uid IN
  (SELECT uid2 FROM Friend WHERE uid1 = 1)
```

$$h) \ltimes \mathtt{F}(1, u)$$

$$Q_{19}(u) :- \mathtt{F}(1, u) \ltimes \mathtt{U}(u, n, h)$$

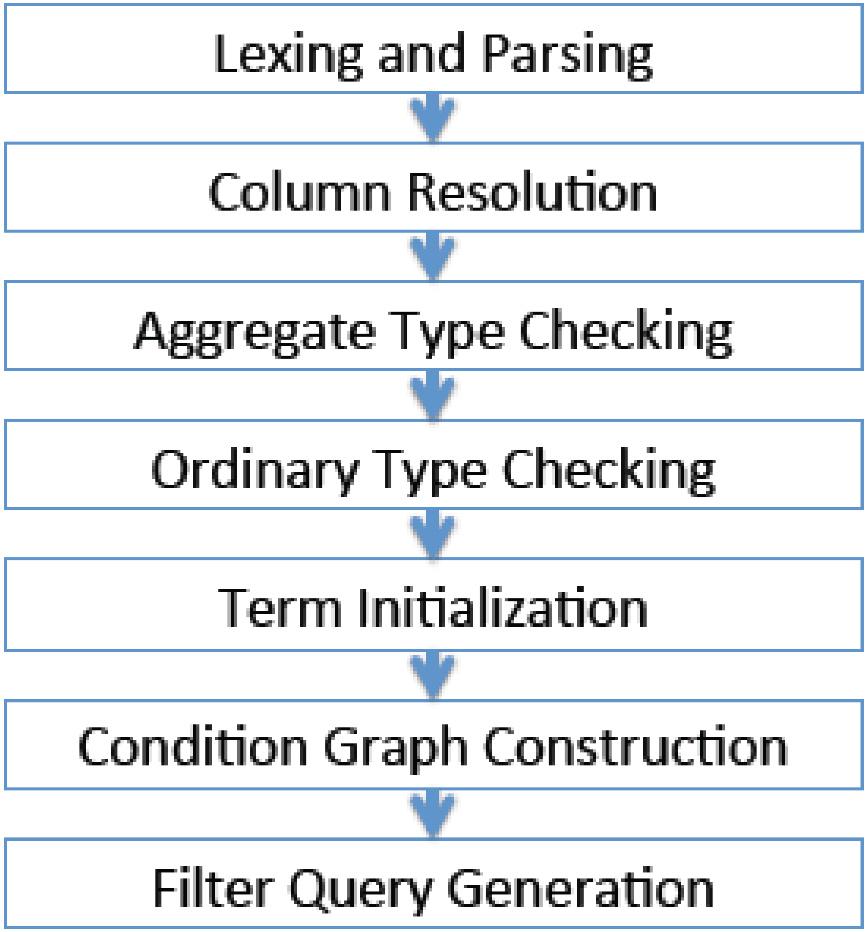
```
SELECT F1.uid2 FROM Friend F1 WHERE F1.uid1 = 1
  AND F1.uid2 IN (SELECT uid FROM User)
```

Filter-Project Queries (cont.)

- Filter atoms in the query language correspond to semi-joins in standard SQL

- Order filter-project queries according to the amount of information they disclose about their *main body atoms*
    - RM will preform exactly one policy check for each table instance in the original SQL query
    - The universe of filter-project queries is *decomposable*

Query Compilation Pipeline



Preprocessing and Column Resolution

- Lexing and Parsing using JSqlParser
- Execution a series of passes on the *Abstract Syntax Tree* (ATS)
- Identify table instances
- Preform column resolution

Type Checking

- Aggressive type-checking on the AST
  - *Aggregate type-checking*

    SELECT SUM(SUM(U1.uid)) FROM User U1

  - *Ordinary type-checking*

    SELECT * FROM User U1 WHERE U1.uid IN
    (SELECT F1.uid1, F1.uid2 FROM Friend F1)

Term Initialization

- Convert the AST for a SQL query Q into a collection of projection queries
  - Together reveal enough information to uniquely determine the answer to Q on any possible dataset

1. Generate one query for each table instance in Q
2. Determine the *existential* and *distinguished* variables in query

$$Q_{\mathtt{U1}}(u, n) \text{ :- } \mathtt{U}(u, n, h)$$

$$Q_{\mathtt{F1}}(u_1, u_2) \text{ :- } \mathtt{F}(u_1, u_2)$$

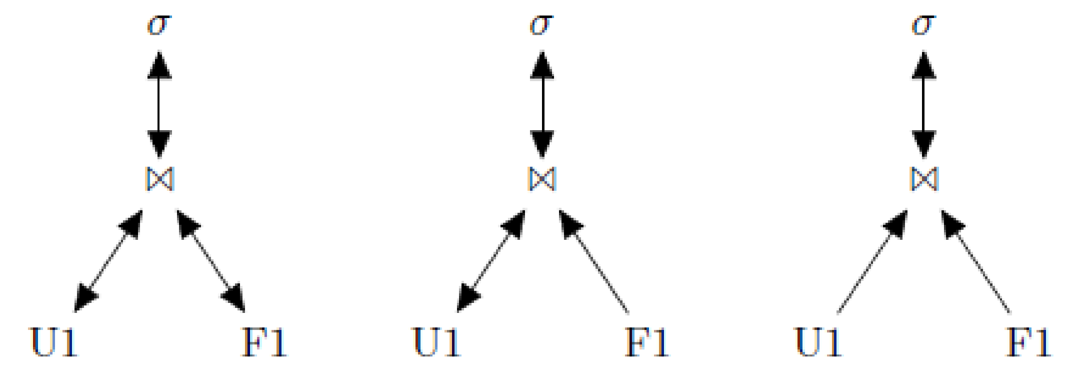
Generating Filter-Project Queries

3. Condition Graph
   - Determine equality constraints and filter atoms to be added to the queries of the previous step
     - Reachability in the condition graph

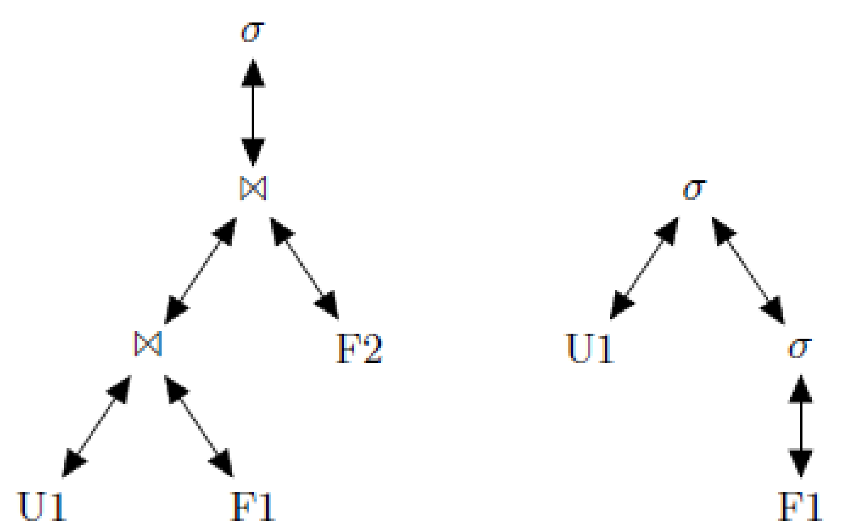
   - Similar to expression trees in the Relational Algebra

Generating Filter-Project Queries (cont.)

- Condition Graph



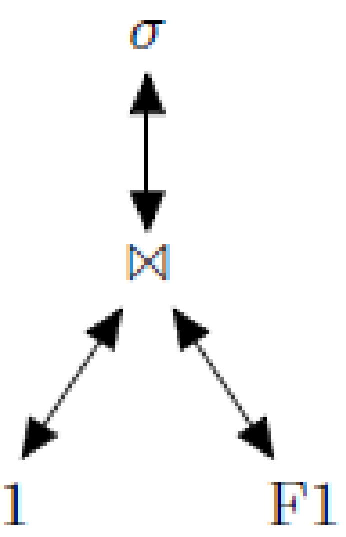
(a) Two-way join (b) Left outer join (c) Full outer join



(d) Three-way join (e) Nested subquery

Example

```
SELECT U1.uid, U1.name FROM User U1 WHERE U1.uid IN
  (SELECT F1.uid2 FROM Friend F1 WHERE F1.uid1 = 1)
```



$$
\begin{aligned}
Q_{\mathtt{U1}}(u, n) &:- \; \mathtt{U}(u, n, h) \ltimes \mathtt{F}(u_1, u_2) \\
Q_{\mathtt{F1}}(u_1, u_2) &:- \; \mathtt{F}(u_1, u_2) \ltimes \mathtt{U}(u, n, h)
\end{aligned}
$$

Generating Filter-Project Queries (cont.)

4. Find equality constraints to be added to the previous queries
   - Finding all SELECCT nodes that are reachable from the condition graph node associated with the table instances
     - Then check WHERE and HAVING clauses

Example

$$Q_{\mathtt{U1}}(u, n) \;:-\; \mathtt{U}(u, n, h) \ltimes \mathtt{F}(u_1, u_2)$$

$$Q_{\mathtt{F1}}(u_1, u_2) \;:-\; \mathtt{F}(u_1, u_2) \ltimes \mathtt{U}(u, n, h)$$

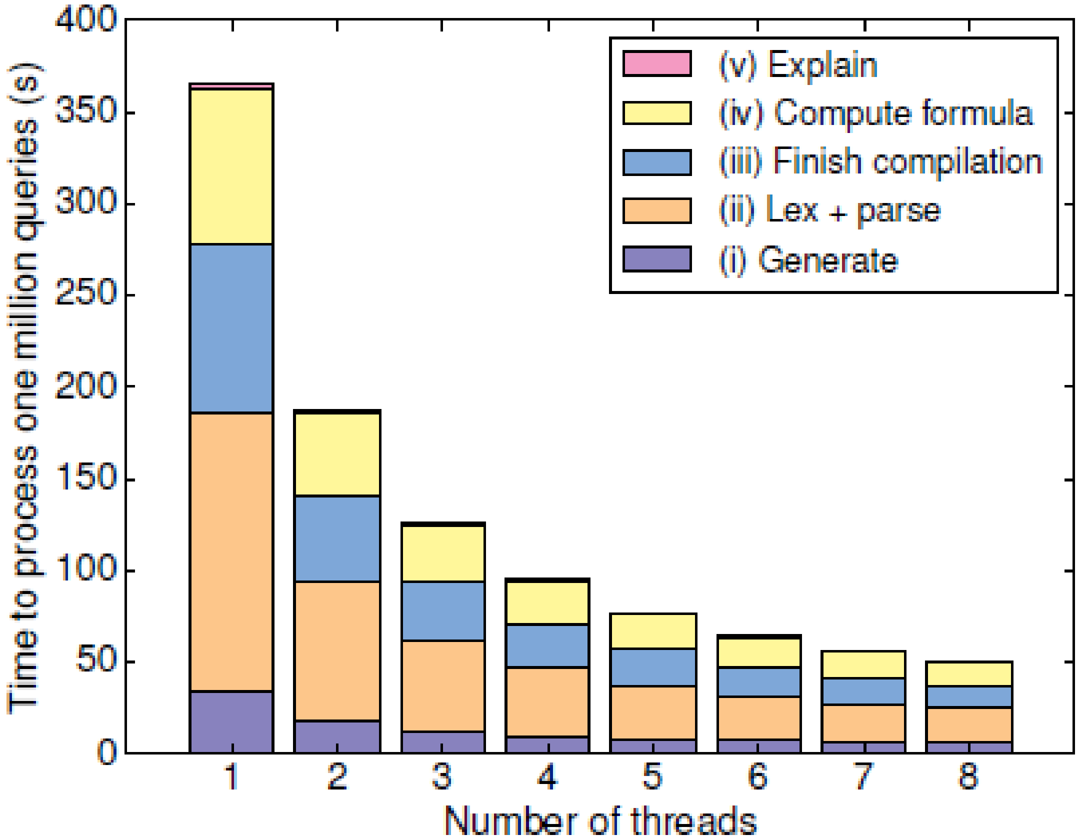
Constraints: (F1.uid1 = 1) and (F1.uid2 = U1.uid)

$$Q_{\mathtt{U1}}(u, n) \;:-\; \mathtt{U}(u, n, h) \ltimes \mathtt{F}(1, u)$$

$$Q_{\mathtt{F1}}(1, u_2) \;:-\; \mathtt{F}(1, u_2) \ltimes \mathtt{U}(u_2, n, h)$$

Experimental Evaluation

- Performance for ordinary queries
- Nearly linearly with the number of cores



Any Questions?


THANK
YOU!