the Semantic SQL Transducer

seamless knowledge graphs and databases interoperation

Enrico Franconi, KRDB - Free University of Bozen-Bolzano

Agenda

The Semantic SQL Transducer *FOR THE* RDF *CROWD*

- Start from a relational database
- Find its core (i.e., lossless) conceptual schema
- Express the core conceptual schema in RDFS + SHACL
- Materialise the relational database in the core conceptual schema,
  maintaining the bidirectional connection
- The relational database can now be queried and updated via SPARQL
  (and dually the RDF graph can be queried and updated via SQL)
- Non-trivial technical issues, but the transducer is generated automatically from the database
- Implemented in SQL

ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

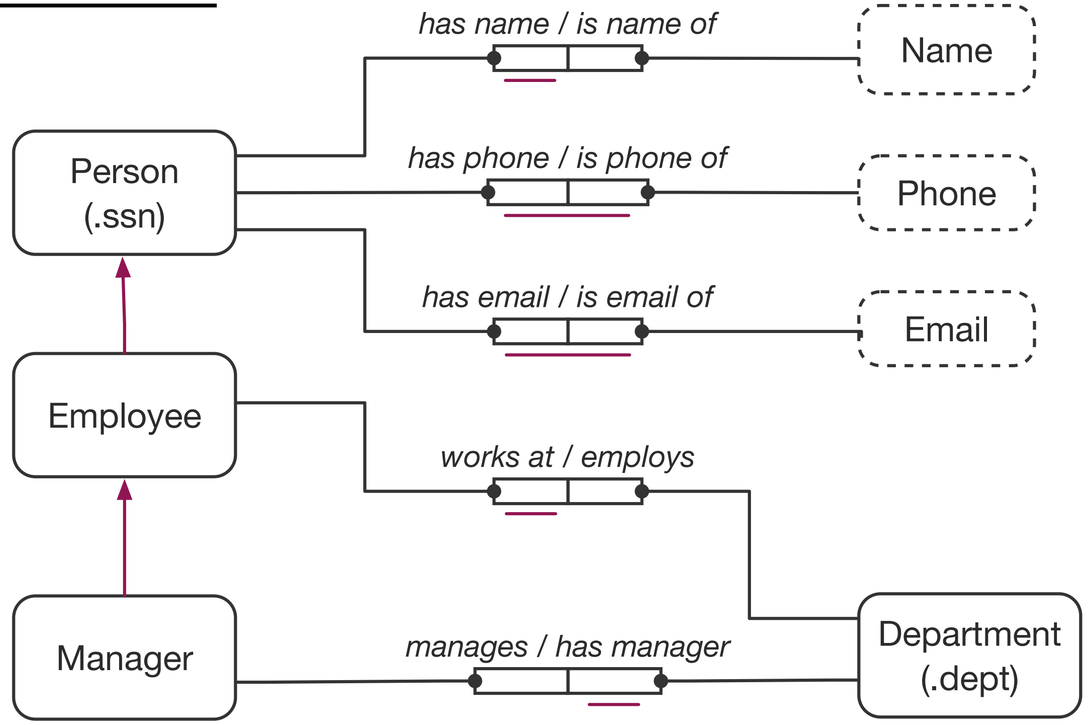
ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

$$\text{ssn, phone, email} \rightarrow \text{name, dept, manager}$$
$$\text{ssn} \rightarrow \text{name, dept}$$
$$\text{ssn} \twoheadrightarrow \text{phone}$$
$$\text{ssn} \twoheadrightarrow \text{email}$$
$$\text{dept} \rightarrow \text{manager}$$
$$\text{jointly\_nullable}(\text{dept, manager})$$
$$\sigma_{\text{not\_null}(\text{dept})}(\text{manager} \subseteq \text{ssn})$$

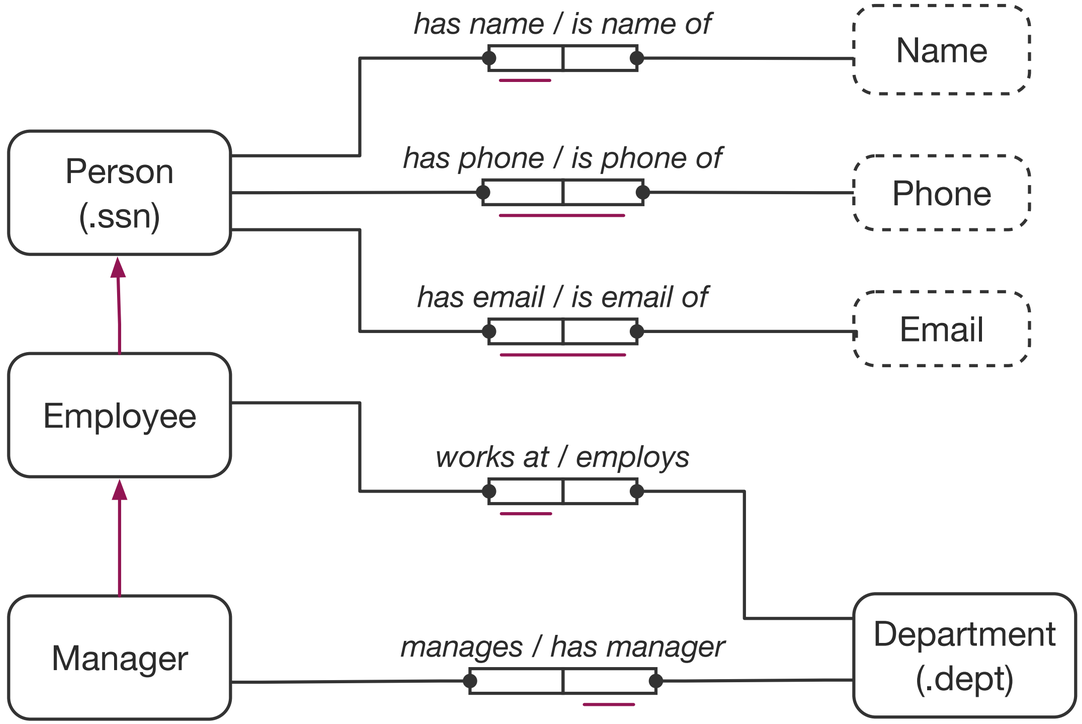
ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

$$\text{ssn, phone, email} \rightarrow \text{name, dept, manager}$$
$$\text{ssn} \rightarrow \text{name, dept}$$
$$\text{ssn} \twoheadrightarrow \text{phone}$$
$$\text{ssn} \twoheadrightarrow \text{email}$$
$$\text{dept} \rightarrow \text{manager}$$
$$\text{jointly\_nullable}(\text{dept, manager})$$
$$\sigma_{\text{not\_null}(\text{dept})}(\text{manager} \subseteq \text{ssn})$$

THE CORE CONCEPTUAL SCHEMA



THE CORE CONCEPTUAL SCHEMA



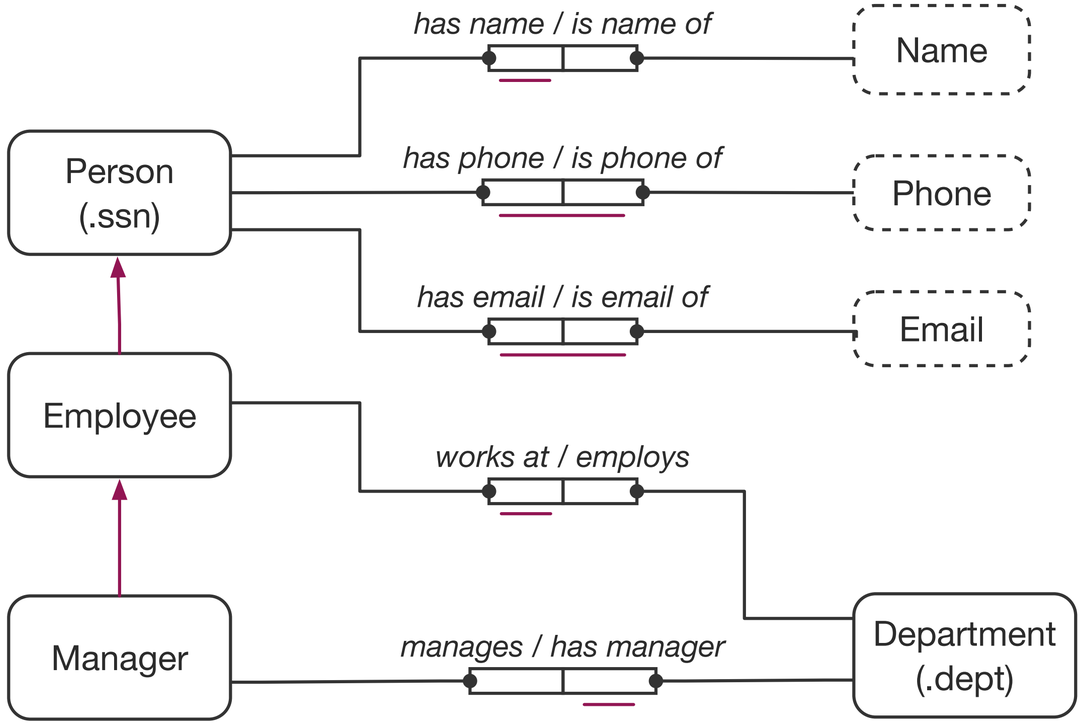
:Manager **rdfs:subClassOf** :Employee .
:Employee **rdfs:subClassOf** :Person .

:has-ssn **rdfs:domain** :Person .
:has-ssn **rdfs:range** xsd:string .
:has-dept **rdfs:domain** :Department .
:has-dept **rdfs:range** xsd:string .

:has-name **rdfs:domain** :Person .
:has-name **rdfs:range** xsd:string .
:has-phone **rdfs:domain** :Person .
:has-phone **rdfs:range** xsd:string .
:has-email **rdfs:domain** :Person .
:has-email **rdfs:range** xsd:string .

:employs **rdfs:domain** :Department .
:employs **rdfs:range** :Employee .
:manages **rdfs:domain** :Manager .
:manages **rdfs:range** :Department .

+ cardinalities: SHACL OWL



ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

:Person/ssn=ssn1 **rdf:type** :Person .
:Person/ssn=ssn1 **:has-ssn** “ssn1” .
:Person/ssn=ssn2 **rdf:type** :Employee .
:Employee/ssn=ssn2 **:has-ssn** “ssn2” .
:Employee/ssn=ssn2 **:has-name** “Mary” .
:Employee/ssn=ssn2 **:has-phone** 2345 .
:Employee/ssn=ssn2 **:has-phone** 3456 .
:Department/dept=D1 **rdf:type** :Department .
:Department/dept=D1 **:employs** :Employee/ssn=ssn2 .
:Employee/ssn=ssn3 **rdf:type** :Manager .
:Employee/ssn=ssn3 **:manages** :Department/dept=D1 .
. . .etc. . .etc. . .etc. . .etc. . .etc. . .etc. . .

+

:Manager **rdfs:subClassOf** :Employee .
:Employee **rdfs:subClassOf** :Person .

:has-ssn **rdfs:domain** :Person .
:has-ssn **rdfs:range** xsd:string .
:has-dept **rdfs:domain** :Department .
:has-dept **rdfs:range** xsd:string .

:has-name **rdfs:domain** :Person .
:has-name **rdfs:range** xsd:string .
:has-phone **rdfs:domain** :Person .
:has-phone **rdfs:range** xsd:string .
:has-email **rdfs:domain** :Person .
:has-email **rdfs:range** xsd:string .

:employs **rdfs:domain** :Department .
:employs **rdfs:range** :Employee .
:manages **rdfs:domain** :Manager .
:manages **rdfs:range** :Department .

+ cardinalities: 

ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

```
INSERT DATA
{ :Person/ssn=ssn1 :has-phone 6789 . }

INSERT DATA
{ :Person/ssn=ssn1 :has-name ‘John Doe’ . }

INSERT DATA
{
:Person/ssn=ssn5 :has-name ‘Linda’ .
:Person/ssn=ssn5 :has-ssn ‘ssn5’ .
:Person/ssn=ssn5 :has-phone 1234 .
:Person/ssn=ssn5 :has-email ‘linda@company.com’
:Department/dept=2 :employs :Person/ssn=ssn5 .
}

DELETE DATA
{ :Person/ssn=ssn2 :has-phone 2345 . }

DELETE DATA
{ :Person/ssn=ssn4 :has-phone 5678 . }
```

```
INSERT INTO table
VALUES (‘ssn1’, ‘John’, 1234, ‘john@company.com’, NULL, NULL);
```

ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

```
INSERT INTO table
VALUES (‘ssn5’, ‘Linda’, 1234, ‘linda@company.com’, ‘D2’, ‘ssn4’);
```

```
DELETE FROM table
WHERE ssn = ‘ssn4’ AND phone = 5678 ;
```

```
INSERT DATA
{ :Person/ssn=ssn1 :has-phone 6789 . }
```

```
INSERT DATA
{ :Person/ssn=ssn1 :has-name ‘John Doe’ . }
```

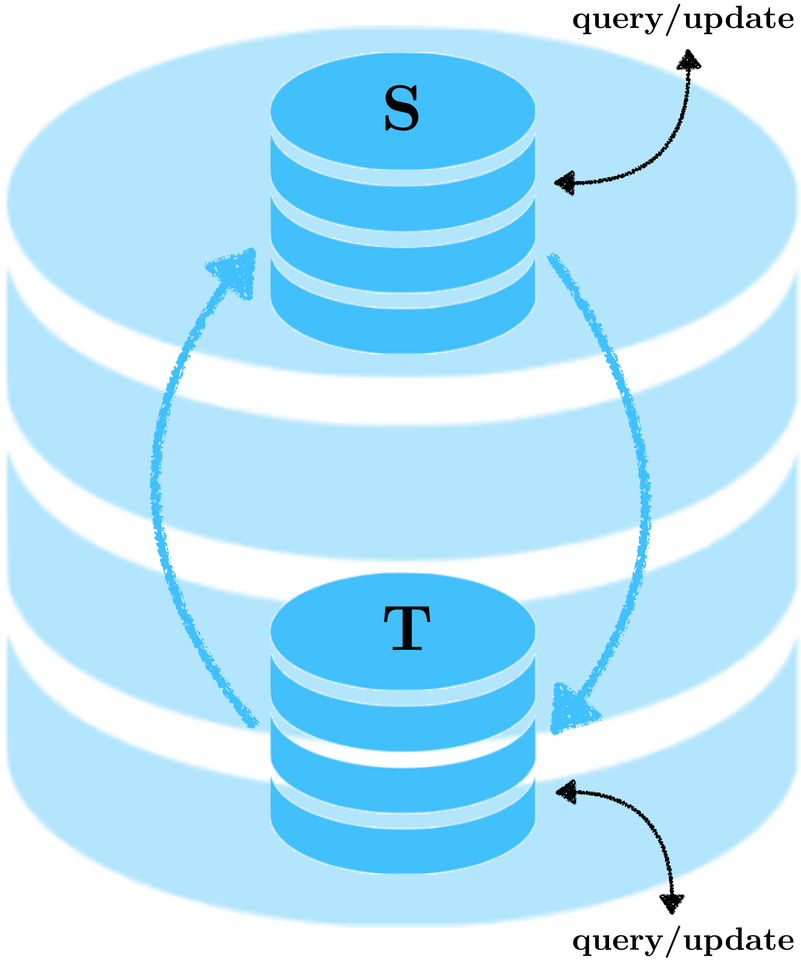
FAIL

```
INSERT DATA
{
:Person/ssn=ssn5 :has-name ‘Linda’ .
:Person/ssn=ssn5 :has-ssn ‘ssn5’ .
:Person/ssn=ssn5 :has-phone 1234 .
:Person/ssn=ssn5 :has-email ‘linda@company.com’
:Department/dept=2 :employs :Person/ssn=ssn5 .
}
```

```
DELETE DATA
{ :Person/ssn=ssn2 :has-phone 2345 . }
```

```
DELETE DATA
{ :Person/ssn=ssn4 :has-phone 5678 . }
```

FAIL



LOSSLESS DYNAMIC REPLICATION

ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

+ constraints

Person

PID
‘:Person/ssn=ssn1’
‘:Person/ssn=ssn2’
‘:Person/ssn=ssn3’
‘:Person/ssn=ssn4’

Employee

PID
‘:Person/ssn=ssn2’
‘:Person/ssn=ssn3’
‘:Person/ssn=ssn4’

Manager

PID
‘:Person/ssn=ssn3’
‘:Person/ssn=ssn4’

Department

DID
‘:Department/dept=D1’
‘:Department/dept=D2’

has-ssn

PID	ssn
‘:Person/ssn=ssn1’	‘ssn1’
‘:Person/ssn=ssn2’	‘ssn2’
‘:Person/ssn=ssn3’	‘ssn3’
‘:Person/ssn=ssn4’	‘ssn4’

has-email

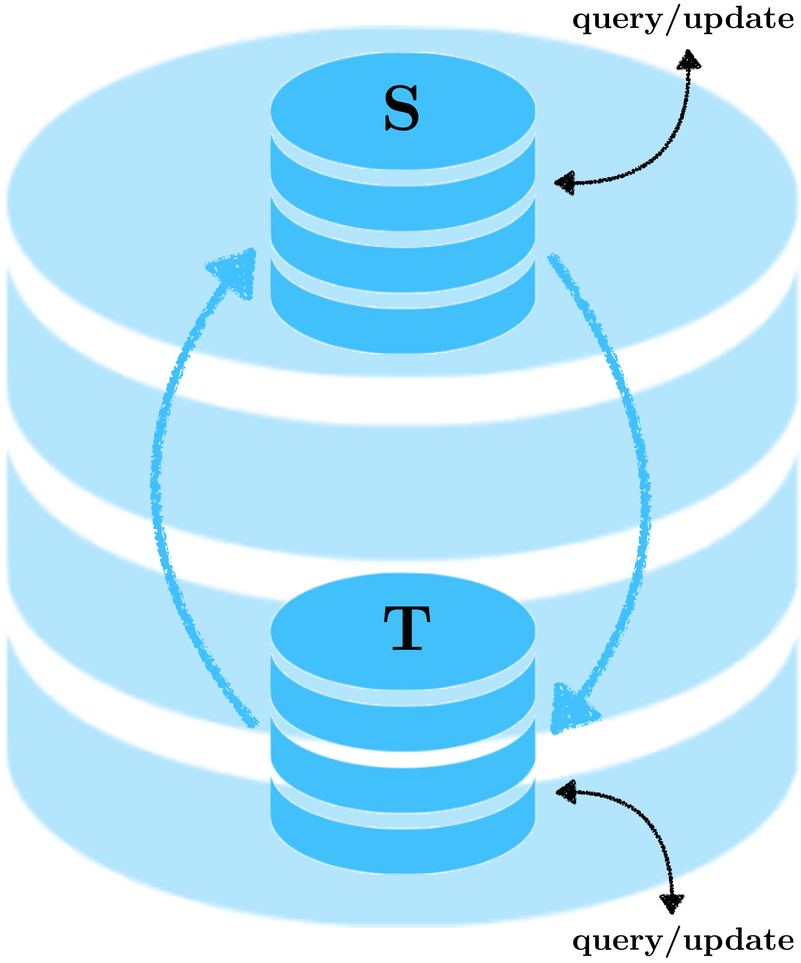
PID	email
‘:Person/ssn=ssn1’	‘john@company.com’
‘:Person/ssn=ssn2’	‘mary@company.com’
‘:Person/ssn=ssn2’	‘mary@gmail.com’
‘:Person/ssn=ssn3’	‘sue@company.com’
‘:Person/ssn=ssn4’	‘paul@company.com’

employs

DID	PID
‘:Department/dept=D1’	‘:Person/ssn=ssn3’
‘:Department/dept=D2’	‘:Person/ssn=ssn4’

. . .etc. . .etc. . .etc. . .etc. . .etc. . .etc. . .

+ constraints



provably correct

LOSSLESS DYNAMIC REPLICATION

ssn	name	phone	email	dept	manager
‘ssn1’	‘John’	1234	‘john@company.com’	NULL	NULL
‘ssn2’	‘Mary’	2345	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@company.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary’	2345	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn2’	‘Mary	3456	‘mary@gmail.com’	‘D1’	‘ssn3’
‘ssn3’	‘Sue’	4567	‘sue@company.com’	‘D1’	‘ssn3’
‘ssn4’	‘Paul’	5678	‘paul@company.com’	‘D2’	‘ssn4’

+ constraints

Person

PID
‘:Person/ssn=ssn1’
‘:Person/ssn=ssn2’
‘:Person/ssn=ssn3’
‘:Person/ssn=ssn4’

Employee

PID
‘:Person/ssn=ssn2’
‘:Person/ssn=ssn3’
‘:Person/ssn=ssn4’

Manager

PID
‘:Person/ssn=ssn3’
‘:Person/ssn=ssn4’

Department

DID
‘:Department/dept=D1’
‘:Department/dept=D2’

has-ssn

PID	ssn
‘:Person/ssn=ssn1’	‘ssn1’
‘:Person/ssn=ssn2’	‘ssn2’
‘:Person/ssn=ssn3’	‘ssn3’
‘:Person/ssn=ssn4’	‘ssn4’

has-email

PID	email
‘:Person/ssn=ssn1’	‘john@company.com’
‘:Person/ssn=ssn2’	‘mary@company.com’
‘:Person/ssn=ssn2’	‘mary@gmail.com’
‘:Person/ssn=ssn3’	‘sue@company.com’
‘:Person/ssn=ssn4’	‘paul@company.com’

employs

DID	PID
‘:Department/dept=D1’	‘:Person/ssn=ssn3’
‘:Department/dept=D2’	‘:Person/ssn=ssn4’

. . .etc. . .etc. . .etc. . .etc. . .etc. . .etc. . .

+ constraints

Advantages

- native **queries** possible at each end, with local response time
- native **update** possible at each end, with local response time,
  with **propagation** of the update at the other end,
  and **integrity** maintained at both ends
- supports native ACID **transactions**
- given a source database in a RDBMS, the transducer is just a bunch of SQL **materialised views with constraints** and SQL **triggers** added to the RDBMS, without changing the source database
- given a source database in a RDBMS, the transducer is generated by an **automatic** procedure, which applies lossless transformation patterns
- the designer **interacts** by providing class names and predicate names
- implemented in **PostgreSQL**

Advantages, Disadvantages

- native **queries** possible at each end, with local response time
- native **update** possible at each end, with local response time,
  with **propagation** of the update at the other end,
  and **integrity** maintained at both ends
- supports native ACID **transactions**
- given a source database in a RDBMS, the transducer is just a bunch of SQL **materialised views with constraints** and SQL **triggers** added to the RDBMS, without changing the source database
- given a source database in a RDBMS, the transducer is generated by an **automatic** procedure, which applies lossless transformation patterns
- the designer **interacts** by providing class names and predicate names
- implemented in **PostgreSQL**

Advantages, Disadvantages, Status

- native **queries** possible at each end, with local response time
- native **update** possible at each end, with local response time,
  with **propagation** of the update at the other end,
  and **integrity** maintained at both ends
- supports native ACID **transactions**
- given a source database in a RDBMS, the transducer is just a bunch of SQL **materialised views with constraints** and SQL **triggers** added to the RDBMS, without changing the source database
- given a source database in a RDBMS, the transducer is generated by an **automatic** procedure, which applies lossless transformation patterns
- the designer **interacts** by providing class names and predicate names
- implemented in **PostgreSQL**

And now?
(seeking collaborations)

- **Open-source serious implementation**
- Documentation of the methodology

- Also:
  - collaboration with RDBMS profiling companies
  - deployment in enterprise settings