The Functorial Data Model

Ryan Wisnesky
Department of Mathematics
Massachusetts Institute of Technology
wisnesky@math.mit.edu

PlanBig December 2014

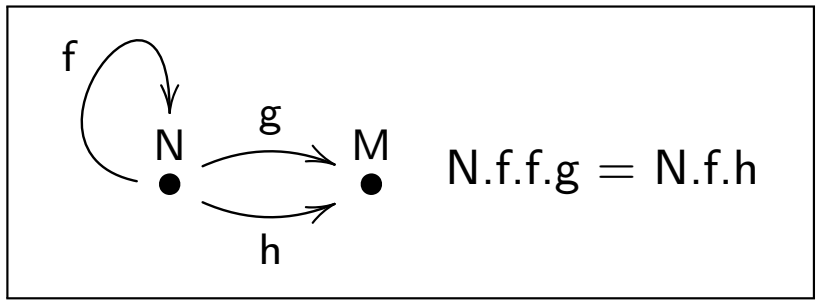
Introduction

- Goal: describe new work on the mathematical foundations of information management: the **functorial data model**.
  - It is conceptually similar to the Entity-Relationship (ER) model, but is formalized using the modern language of **category theory**.
  - Its clean mathematical foundations enable many useful applications.
  - Project webpage: **categoricaldata.net/fql.html**.

- Sponsored by:
  - ONR grant N000141310260
  - AFOSR grant FA9550-14-1-0031

Categorical Data Models

- A **category** is a *reflexive*, *directed*, *labelled*, *multi-graph* and a set of *path equations*:



- Category theory was instrumental in the development of two extensions to the relational model, both of which inform work on language-integrated query (LINQ):
  - **The nested relational model** generalizes sets to nested collections and is inspired by *monads*.
  - **Algebraic datatypes** implement nested collections using recursion and are inspired by *algebras*.
- **The functorial data** model generalizes relational schemas to categories and is inspired by *adjunctions*.
  - Discovered by Rosebrugh et al in the early 2000s, Spivak, myself, and others have lately proposed it for information integration.

The **Relational Model**

- Schemas are first-order theories, and instances are models:

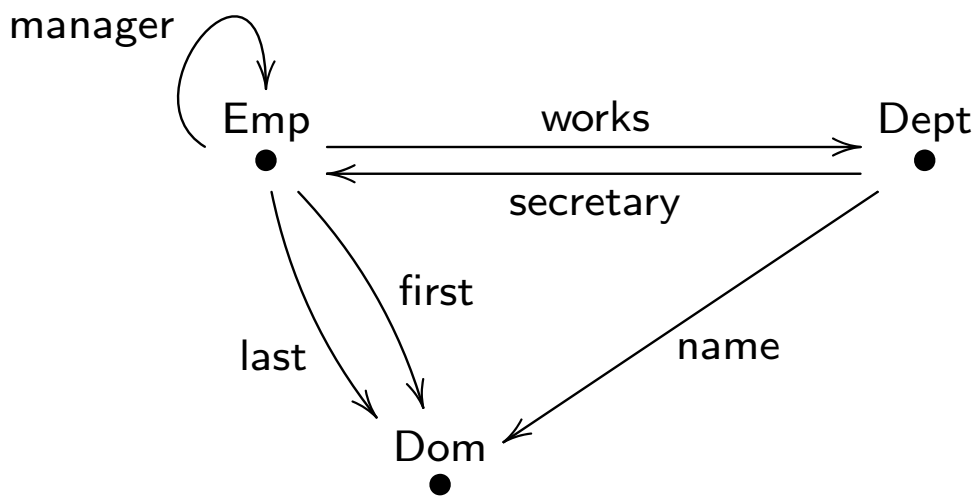
- Emp, Dept (1); manager, secretary, first, last, name, works (2)
- All relations functional, e.g., $\mathsf{name}(d, n) \wedge \mathsf{name}(d, n') \rightarrow n = n'$
- All relations total, e.g., $\mathsf{Dept}(d) \rightarrow \exists e.\mathsf{Emp}(e) \wedge \mathsf{secretary}(d, e)$
- Secretaries work in their depts: $\mathsf{secretary}(d, e) \rightarrow \mathsf{works}(e, d)$
- Managers work with emps: $\mathsf{mgr}(e, m) \wedge \mathsf{works}(e, d) \rightarrow \mathsf{works}(m, d)$

Emp				
ID	**mgr**	**works**	**first**	**last**
101	103	q10	Al	Akin
102	102	x02	Bob	Bo
103	103	q10	Carl	Cork

Dept		
ID	**sec**	**name**
q10	101	CS
x02	102	Math

The **Functorial Data Model**

- Schemas are categories, and instances are set-valued functors:



$$\text{Emp.manager.works} = \text{Emp.works}$$

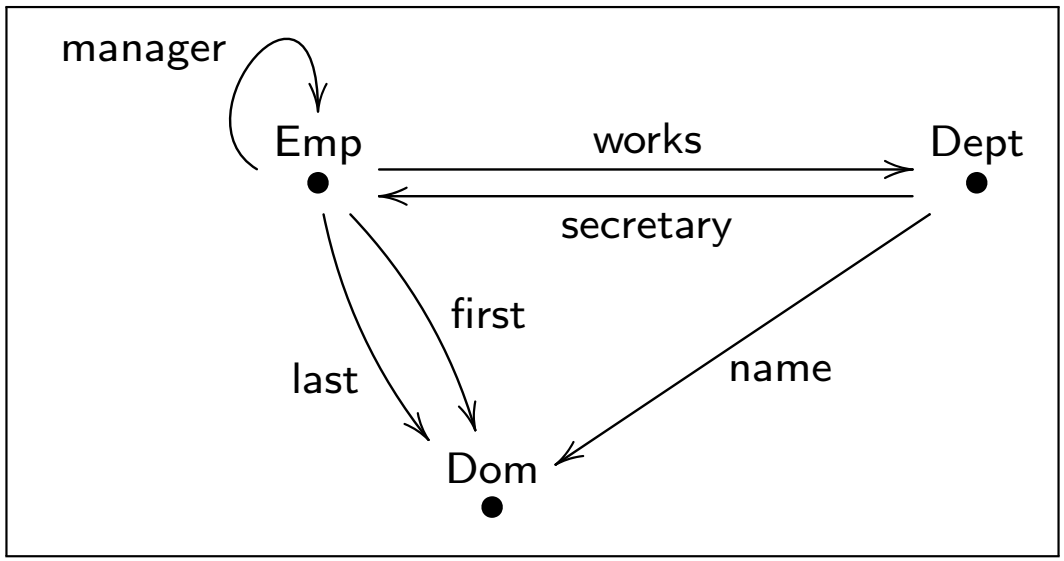
$$\text{Dept.secretary.works} = \text{Dept}$$

Emp				
ID	**mgr**	**works**	**first**	**last**
101	103	q10	Al	Akin
102	102	x02	Bob	Bo
103	103	q10	Carl	Cork

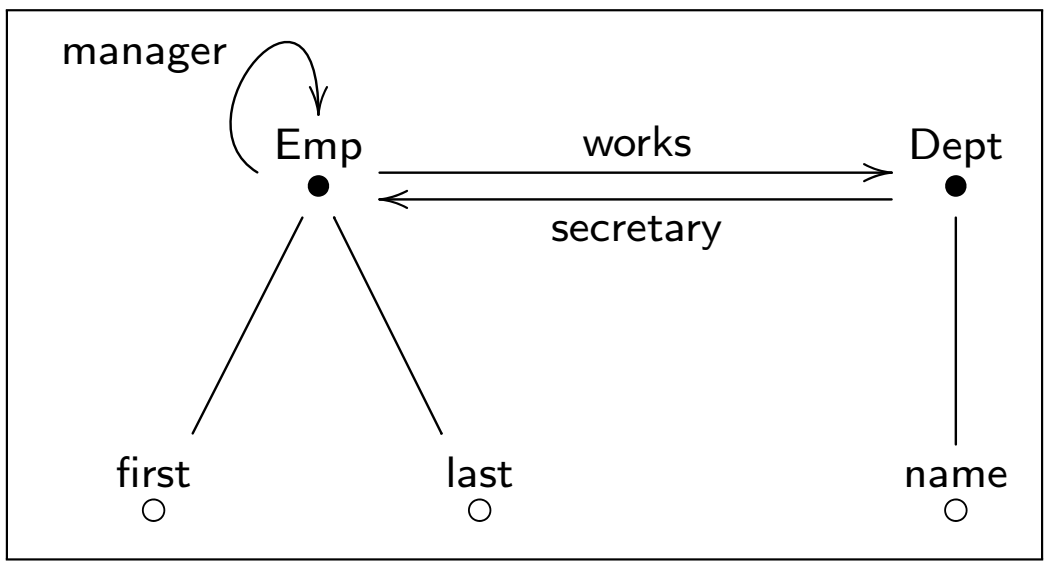
Dept		
ID	**sec**	**name**
q10	101	CS
x02	102	Math

Categories as Entity-Relationship (ER) Diagrams

- Draw edges $\bullet \to_f \bullet_{Dom}$ as "attributes" $\bullet - \circ_f$:



=



FQL: A Functorial Query Language

- A **schema mapping** $F : S \to T$ is a path-equality respecting function:

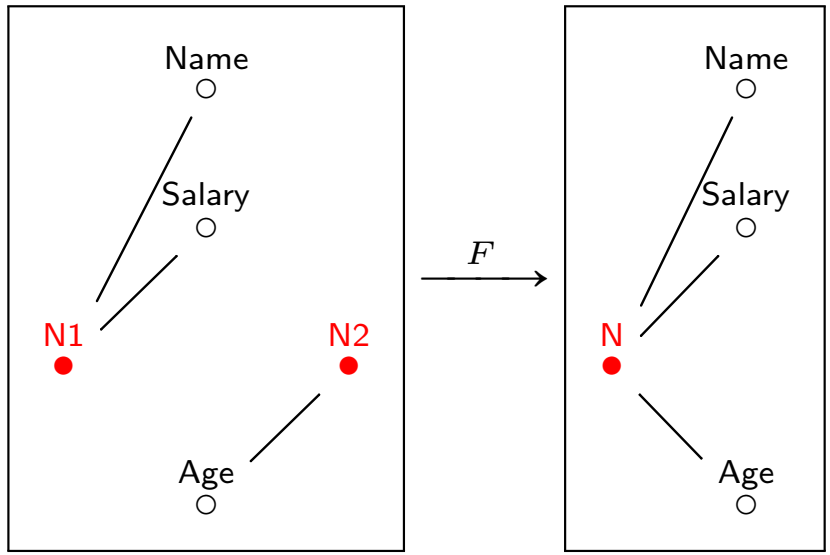
$$nodes(S) \to nodes(T) \qquad edges(S) \to paths(T)$$

  and it induces three adjoint **data migration** functors:

  - Δ_F : T-inst $\to$ S-inst (like projection)
  - Σ_F : S-inst $\to$ T-inst (like union) (also like the chase)
  - Π_F : S-inst $\to$ T-inst (like join)

- A **FQL query** has the form $\Sigma_F \circ \Pi_G \circ \Delta_H$, where:
  - G is a surjection on attributes (implies "domain independence")
  - F is a discrete op-fibration (implies "union compatibility")
- Theorem: FQL queries are closed under composition.

Δ (Project)



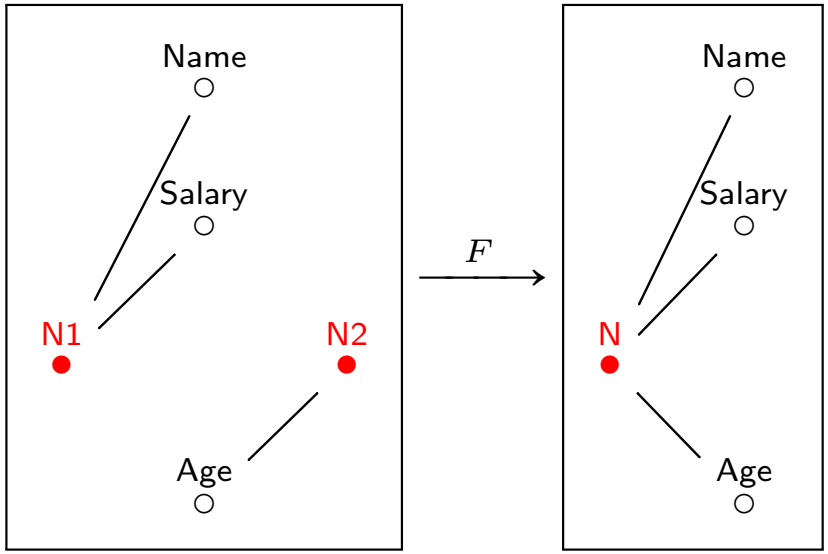
N1		
ID	**Name**	**Salary**
1	Bob	$250
2	Sue	$300
3	Alice	$100

N2	
ID	**Age**
1	20
2	20
3	30

$\xleftarrow{\Delta_F}$

N			
ID	**Name**	**Age**	**Salary**
1	Bob	20	$250
2	Sue	20	$300
3	Alice	30	$100

Π (Join)



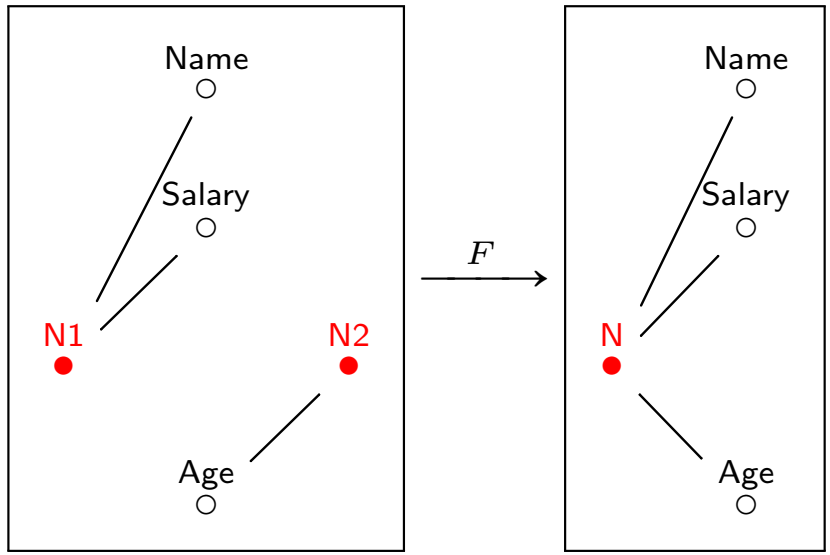
N1		
ID	**Name**	**Salary**
1	Bob	$250
2	Sue	$300
3	Alice	$100

N2	
ID	**Age**
1	20
2	20
3	30

$\xrightarrow{\Pi_F}$

N			
ID	**Name**	**Age**	**Salary**
1	Alice	20	$100
2	Alice	20	$100
3	Alice	30	$100
4	Bob	20	$250
5	Bob	20	$250
6	Bob	30	$250
7	Sue	20	$300
8	Sue	20	$300
9	Sue	30	$300

Σ (Union)



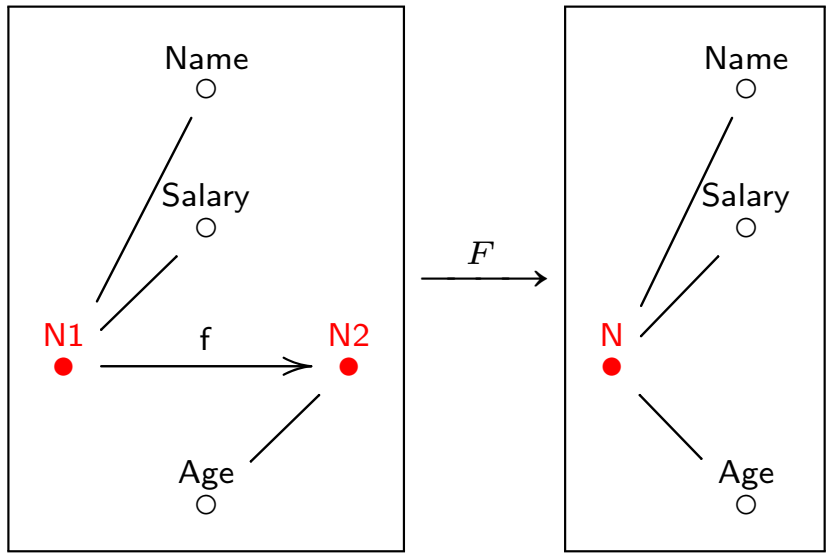
N1		
ID	**Name**	**Salary**
1	Bob	$250
2	Sue	$300
3	Alice	$100

N2	
ID	**Age**
1	20
2	20
3	30

$\xrightarrow{\Sigma_F}$

N			
ID	**Name**	**Age**	**Salary**
1	Alice	null_1	$100
2	Bob	null_2	$250
3	Sue	null_3	$300
4	null_4	20	null_7
5	null_5	20	null_8
6	null_6	30	null_9

Foreign keys



N1			
ID	**Name**	**Salary**	**f**
1	Bob	$250	1
2	Sue	$300	2
3	Alice	$100	3

N2	
ID	**Age**
1	20
2	20
3	30

Δ_F ← → Π_F, Σ_F

N			
ID	**Name**	**Age**	**Salary**
1	Alice	20	$100
2	Bob	20	$250
3	Sue	30	$300

Results

- SPCU + keygen (sets) can implement FQL queries.
- FQL queries can implement SPCU + keygen (bags).
- FQL queries + post-processing can implement SPCU + keygen (sets).
- The instances on each schema form a topos, and hence can interpret higher-order logic.
- There is a deep relationship between "non-union-compatible" Σ and "the chase".
- There is a relationship between FQL queries and polynomial functors.

The FQL Integrated Development Environment

- The **FQL IDE** is an open-source java GUI:
  - It translates from SQL to FQL, and FQL to SQL (when possible).
  - It emits RDF encodings of instances.
  - It comes with many built-in examples.
  - It can be used as a command-line compiler.
  - Download at **categoricaldata.net/fql.html**.

A textual employee instance

FQL IDE

Compile | New | Open | Save | Help | Options | Load Example: Typed ...

Untitled 1 × | Typed employees ×

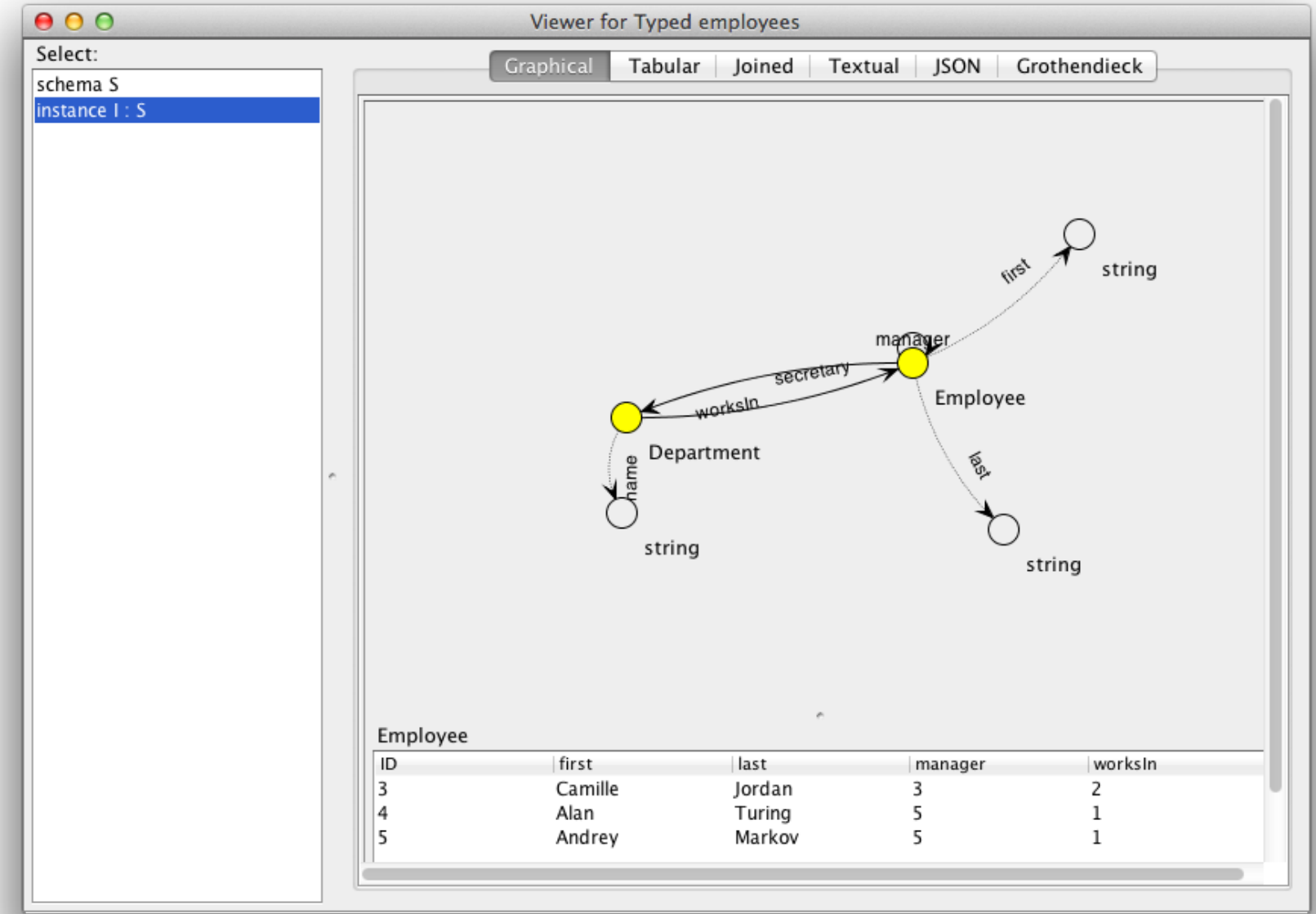
```
1  schema S = {
2   nodes
3       Employee, Department;
4   attributes
5       name  : Department -> string,
6       first : Employee -> string,
7       last  : Employee -> string;
8   arrows
9       manager   : Employee -> Employee,
10      worksIn   : Employee -> Department,
11      secretary : Department -> Employee;
12  equations
13      Employee.manager.worksIn = Employee.worksIn,
14      Department.secretary.worksIn = Department,
15      Employee.manager.manager = Employee.manager;
16  }
17
18 instance I : S = {
19  nodes
20      Employee -> { 101, 102, 103 },
21      Department -> { q10, x02 };
22  attributes
23      first -> { (101, Alan), (102, Camille), (103, Andrey) },
24      last  -> { (101, Turing), (102, Jordan), (103, Markov) },
25      name  -> { (q10, AppliedMath), (x02, PureMath) };
26  arrows
27      manager -> { (101, 103), (102, 102), (103, 103) },
28      worksIn -> { (101, q10), (102, x02), (103, q10) },
29      secretary -> { (q10, 101), (x02, 102) };
30  }
```

Compiler response

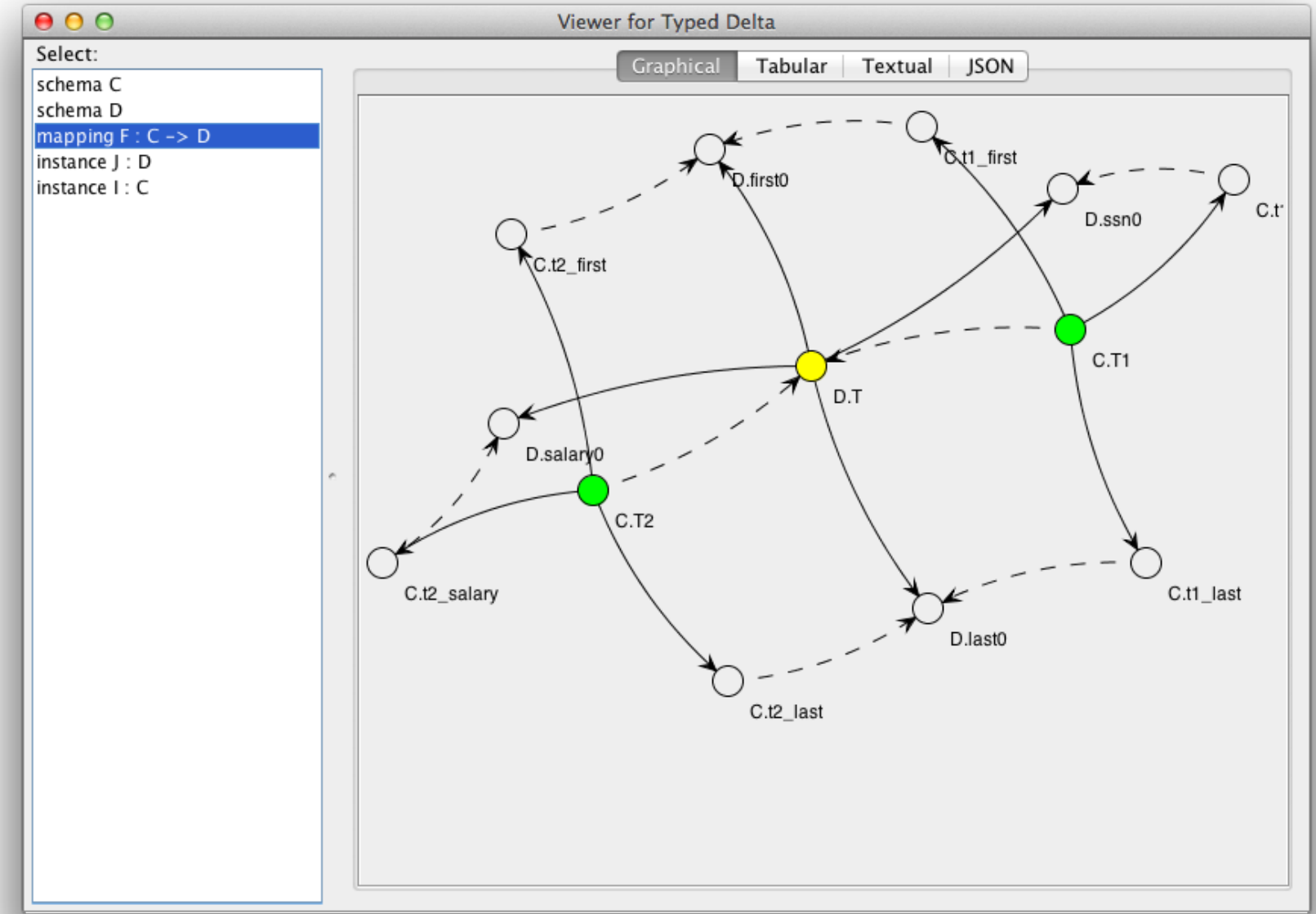
```
DROP DATABASE FQL; CREATE DATABASE FQL; USE FQL; SET @guid := 0;

CREATE TABLE I_Department(c1 VARCHAR(128), c0 VARCHAR(128));
```

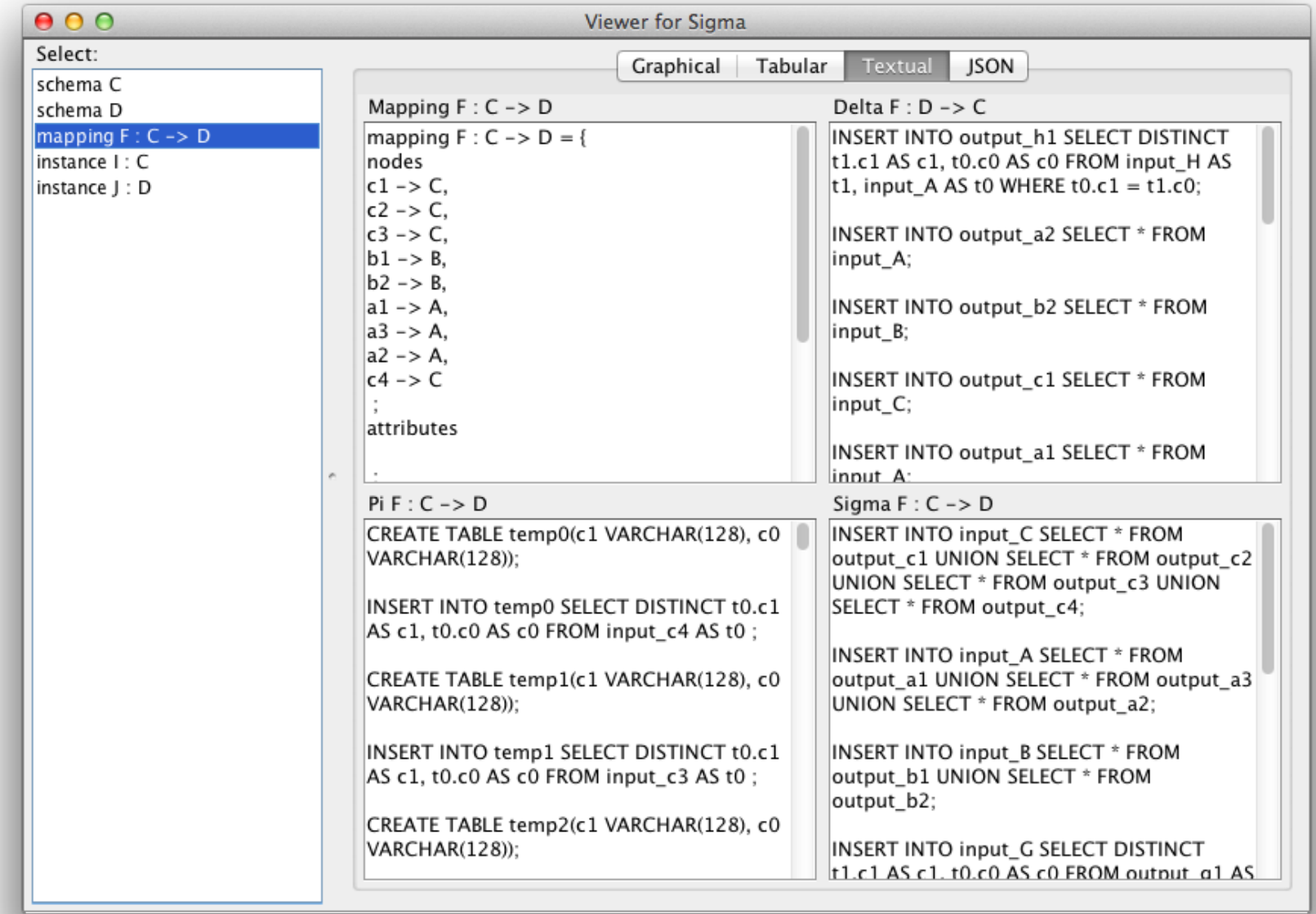
A graphical employee instance



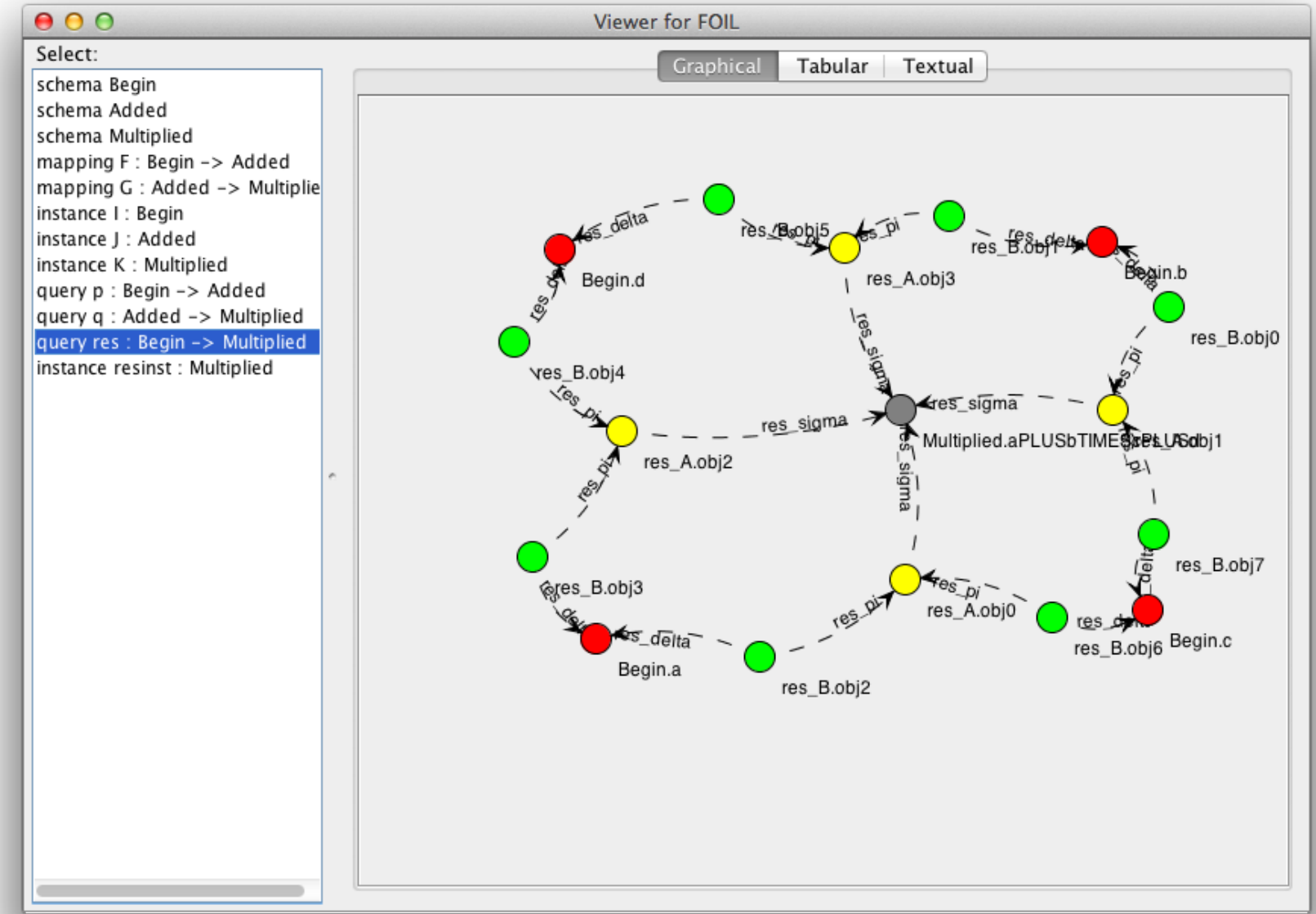
A graphical schema mapping



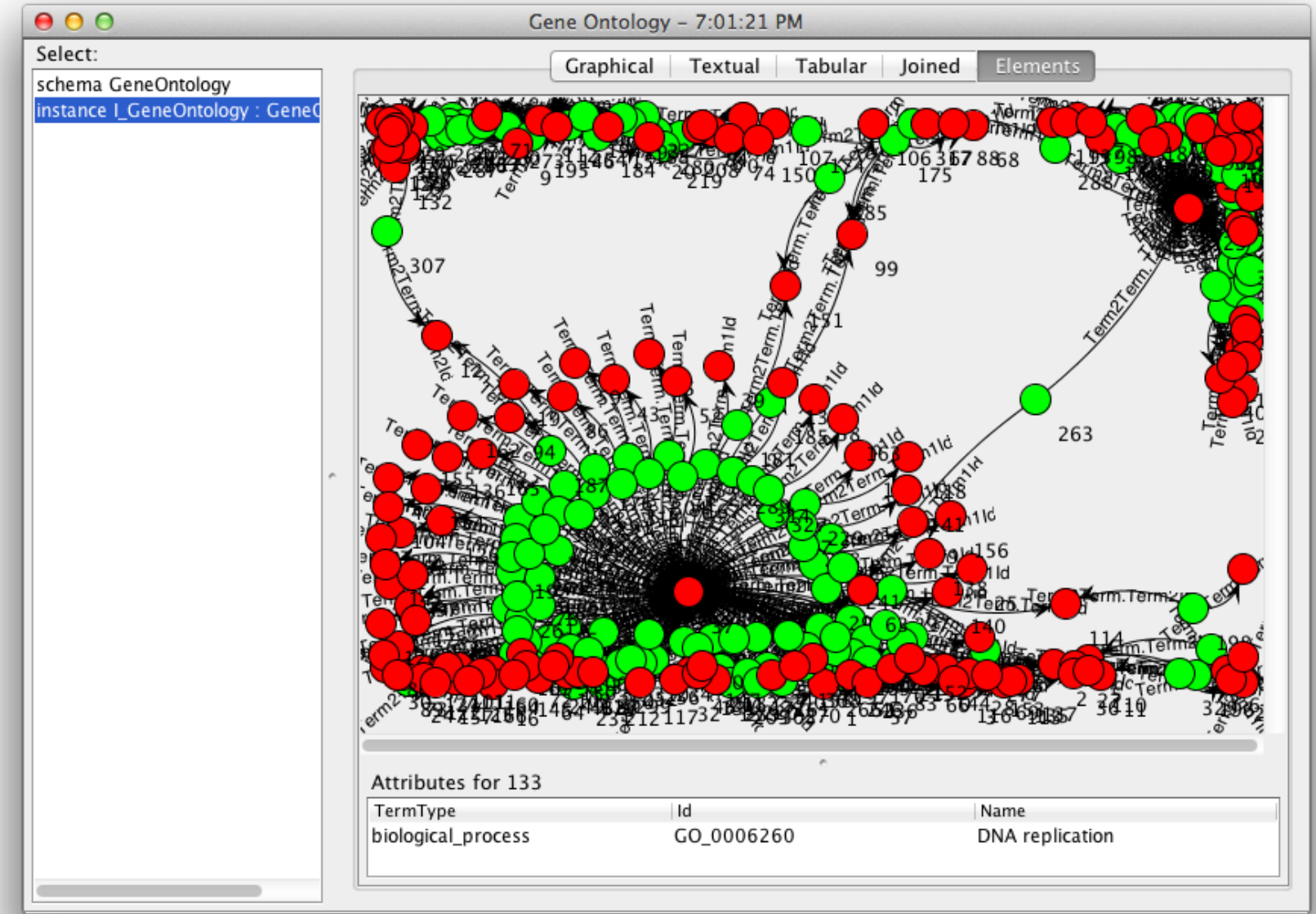
SQL generation



A graphical query



A genomics instance



Conclusion

- The **functorial data model** extends the relational model by generalizing schemas to categories.
  - It is conceptually similar to the Entity-Relationship (ER) model, but formalized using category theory.

- The **FQL IDE** is a graphical schema-mapping tool for developing functorial data migrations.
  - Project webpage: **categoricaldata.net/fql.html**.

The **Nested Relational Model**

- Schemas are higher-order theories, and instances are models:

  - Emp: Set (ID:Dom,emps:Set Dom,works:Dom,first:Dom,last:Dom)
    Dept: Set (ID:Dom, sec:Dom, name:Dom)
  - Query languages include the Nested Relational Calculus:

    $$\texttt{for}\ (e \in \mathsf{Emp})(e' \in e)\ \texttt{where}\ e.\mathsf{ID} = e'\ \texttt{return}\ (\mathsf{name} : e.\mathsf{first})$$

Emp				
ID	**emps**	**works**	**first**	**last**
101	{ }	q10	Al	Akin
102	{ 102 }	x02	Bob	Bo
103	{ 101,103 }	q10	Carl	Cork

Dept		
ID	**sec**	**name**
q10	101	CS
x02	102	Math

Algebraic Datatypes

- Implement collections as algebraic datatypes:

  - List a = nil | cons t (List a)

    Emp:List(ID:Dom,emps:List Dom,works:Dom,first:Dom,last:Dom)
    Dept:List(ID:Dom, sec:Dom, name:Dom)

  - Folds used to process collections:

    fold : (a → b → b) → b → List a → b
    fold f x nil = x
    fold f x (cons h t) = f h (fold f x t)

    sum = fold + 0

Emp				
ID	**emps**	**works**	**first**	**last**
101	nil	q10	Al	Akin
102	cons 102 nil	x02	Bob	Bo
103	cons 101 (cons 103 nil)	q10	Carl	Cork

Dept		
ID	**sec**	**name**
q10	101	CS
x02	102	Math