Optimal Knowledge Graph Merge

using Category Theory

Ryan Wisnesky
Conexus AI

April 6, 2021

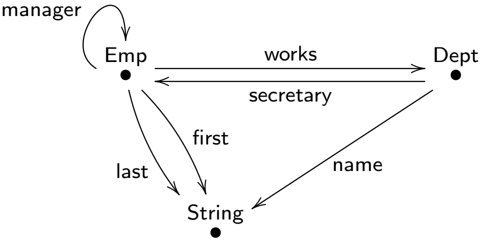
$$\Sigma \dashv \Delta \dashv \Pi$$

Introduction

- There is a branch of math called *category theory* that allows you to convert relational data to graph data and vice-versa in a way that is guaranteed to respect data integrity/business rules.
- This branch of mathematics also describes an optimal way to merge knowledge graphs, which we at Conexus have so far been using to merge ontologies (and merge relational databases, and merge ontologies with relational databases, and more).
- Work has culminated in an open-source language, CQL, available at categoricaldata.net, being commercialized by Conexus, conexus.com.

- Categories are graphs with extra structure, and so category theory has deep connections to "algebraic property graphs" (joint work with Joshua Shinavier).

Example Categorical Schema and Database



$$\text{Emp} \xrightarrow{\text{manager}} \text{Emp} \xrightarrow{\text{works}} \text{Dept} \;=\; \text{Emp} \xrightarrow{\text{works}} \text{Dept}$$

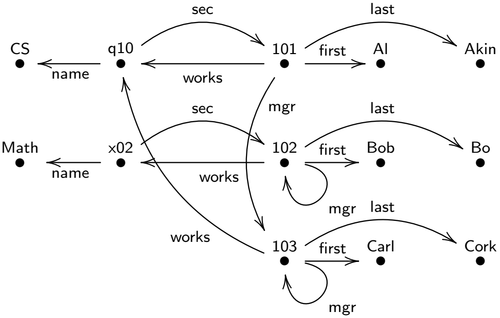
$$\text{Dept} \xrightarrow{\text{secretary}} \text{Emp} \xrightarrow{\text{works}} \text{Dept} \;=\; \text{Dept}$$

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

String
ID
Al
Bob
. . .

Categorical Databases to Triples (Graphs) and Back

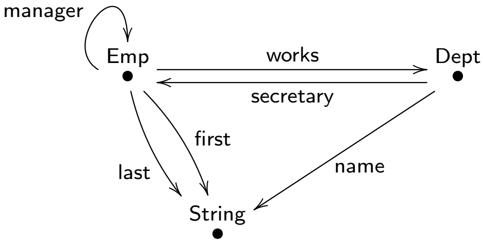


Emp				
ID	**mgr**	**works**	**first**	**last**
101	103	q10	Al	Akin
102	102	x02	Bob	Bo
103	103	q10	Carl	Cork

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

String
ID
Al
Bob
...

Categorical Select-From-Where/For-Where-Return Syntax



Find the name of every manager's department:

```
CQL
select e.manager.works.name
from Emp as e
```

```
SQL
select d.name
from Emp as e1, Emp as e2, Dept as d
where e1.manager = e2.ID and
      e2.works = d.ID
```

How to optimally merge knowledge graph schemas

1. Describe each input knowledge graph schema as a directed labelled multi-graph with equational constraints (i.e., a category).
2. Union together the knowledge graph schemas, define a set of equational constraints that describe their semantic overlap, and merge their nodes, and if possible, edges, according to these constraints. (i.e. compute a "co-limit" in the category of schemas).
   - The result of the schema merge is unique up to unique isomorphism, but in practice, people invariably want to choose a particular way of merging (e.g., prefer Person to People).
   - The merged schema posses a unique mapping to any other way of merging the schemas.

How to optimally merge knowledge graphs

1. Transform each input knowledge graph into tables over its schema.
2. "OUTER UNION" the input tables onto the merged schemas, then recursively "OUTER MERGE" their rows them according to the equations in the merged schema. (i.e. compute a "co-limit" in the category set-valued functors out of the merged schema).
   - "OUTER UNION" and "OUTER MERGE" must create *labelled nulls*, not SQL-style nulls, so this process can diverge, and it is undecidable predicting if it does.
   - The result of the data merge is unique up to unique isomorphism, and posses a unique mapping to any other way of merging the data.
3. (Optional). Arbitrary "row linking" algorithms may be added to the above data merge step by materializing their output row links as data on merged knowledge graph schema and adding corresponding equations to the merged schema.
   - For example, chemical links can be imported as tuples of the form (Hydrogen, H), (Helium, He), etc.

Schema Integration in CQL

```
schema S1 = literal : sql {
    entities
        Observation Person Gender ObsType
    foreign_keys
        f: Observation -> Person    g: Observation -> ObsType    h: Person -> Gender
    attributes
         att: Person -> String    att: Gender -> String      att: ObsType -> String }

schema S2 = literal : sql {
    entities
        Observation Patient Method Type
    foreign_keys
        f : Observation -> Patient      g1: Observation -> Method g2: Method -> Type
    attributes
         att: Patient -> String      att: Type -> String  }


schema_colimit ColimAuto = quotient S1 + S2 : sql {
    entity_equations
        S1.Observation = S2.Observation         S1.Person = S2.Patient         S1.ObsType = S2.Type
    path_equations
        Observation.S1_Observation_f = Observation.S2_Observation_f
        Observation.g = Observation.g1.g2      }
```

Data Integration (Tabular) - Overlap Given as Data

Observation

ID	f	g

Person

ID
p

Type

ID
BP
Wt

→

Method

ID	g2
m_1	BP
m_2	BP
m_3	Wt
m_4	Wt

Type

ID
BP
Wt

Observation

ID	f	g1
o_1	Pete	m_1
o_2	Pete	m_2
o_3	Jane	m_3
o_4	Jane	m_1

Patient

ID
Jane
Pete

↓

Gender

ID
F
M

Type

ID
BP
Wt
HR

Observation

ID	f	g
o_5	Peter	BP
o_6	Paul	HR
o_7	Peter	Wt

Person

ID	h
Paul	M
Peter	M

→

Method

ID	g2
$null_1$	BP
$null_2$	Wt
$null_3$	HR
m_1	BP
m_2	BP
m_3	Wt
m_4	Wt

Observation

ID	f	g1
o_1	Peter	m_1
o_2	Peter	m_2
o_3	Jane	m_3
o_4	Jane	m_1
o_5	Peter	$null_1$
o_6	Paul	$null_2$
o_7	Peter	$null_3$

Gender

ID
F
M
$null_4$

ObsType

ID
BP
Wt
HR

Person

ID	h
Jane	$null_4$
Paul	M
Peter	M

Data Integration in CQL

Gender (2)

Row	att
0	M
1	F

ObsType (3)

Row	att
2	BloodPressure
3	BodyWeight
4	HeartRate

Observation (3)

Row	f	g
5	8	2
6	8	3
7	9	4

Person (2)

Row	att	h
8	Peter	0
9	Paul	0

Method (4)

Row	g2
0	10
1	10
2	11
3	11

Observation (4)

Row	f	g1
4	8	0
5	8	1
6	9	2
7	9	0

Patient (2)

Row	att
8	Pete
9	Jane

Type (2)

Row	att
10	BloodPressure
11	BodyWeight

G (3)

Row	att
0	M
1	?0
2	F

M (7)

Row	g2
3	20
4	20
5	21
6	21
7	20
8	21
9	22

O (7)

Row	f	g1
10	17	3
11	17	4
12	18	5
13	18	3
14	17	7
15	17	8
16	19	9

P (3)

Row	P_att1	P_att2	h
17	Peter	Pete	0
18	?1	Jane	1
19	Paul	?2	0

T (3)

Row	T_att1	T_att2
20	BloodPressure	BloodPressure
21	BodyWeight	BodyWeight
22	HeartRate	?3

Conclusion

- There is a branch of math called *category theory* that allows you to convert relational data to graph data and vice-versa in a way that is guaranteed to respect data integrity/business rules.
- This branch of mathematics also describes an optimal way to merge knowledge graphs, which we at Conexus have so far been using to merge ontologies (and merge relational databases, and merge ontologies with relational databases, and more). See categoricaldata.net and conexus.com.
- We're looking for partners to merge knowledge graphs in practice! CQL works with Tinkerpop graphs and RDF/OWL out of the box.

Tinkerpop Import

```
schema tp = tinkerpop

constraints tpc = tinkerpop

command c1 = spawn_bitsy

command c2 = exec_tinkerpop {
"g.V().drop()"
"g.addV('root').property('data',9).as('root').
  addV('node').property('data',5).as('b').
  addV('node').property('data',2).as('c').
  addV('node').property('data',11).as('d').
  addV('node').property('data',15).as('e').
  addV('node').property('data',10).as('f').
  addV('node').property('data',1).as('g').
  addV('node').property('data',8).as('h').
  addV('node').property('data',22).as('i').
  addV('node').property('data',16).as('j').
  addE('left').property('data',16).from('root').to('b').
  addE('left').from('b').to('c').
  addE('right').from('root').to('d').
  addE('right').from('d').to('e').
  addE('right').from('e').to('i').
  addE('left').from('i').to('j').
  addE('left').from('d').to('f').
  addE('right').from('b').to('h').
  addE('left').from('c').to('g')"
}

instance g = import_tinkerpop_all
```

Edge (9)

Row	id	label	in	out
0	6e0edabb-5bb4-4770-9...	right	24	23
1	407b8f04-92a8-41a5-8a...	right	28	20
2	c9355b58-74f7-450d-91...	left	21	29
3	ccc54499-b09d-41bc-9b...	right	22	24
4	d79980ba-e2af-40b6-81...	left	27	25
5	af6a003b-03ce-4d2c-85...	right	25	22
6	cb2b2c08-d4fa-45ed-a4...	left	20	23
7	2ccf43ae-f665-452b-83...	left	29	20
8	d008b756-31fc-4de3-a8...	left	26	24

HasEdgeProperty (1)

Row	key	value	edge
9	data	16	6

HasVertexProperty (10)

Row	key	value	vertex
10	data	5	20
11	data	1	21
12	data	15	22
13	data	9	23
14	data	11	24
15	data	22	25
16	data	10	26
17	data	16	27
18	data	8	28
19	data	2	29

Vertex (10)

Row	id	label
20	493481f1-9994-4601-9642-dbf0234cd8ef	node
21	77b14e20-868d-4b0f-a211-019dc39d0f16	node
22	a28fa796-bd87-412f-b685-fe17c6d1da55	node
23	b8f832ba-bbac-4aa1-89cf-4a9bf258deba	root
24	c16cc69e-fc73-40cc-bcbd-052e282a258d	node
25	49b8e52f-3e9e-45b9-8627-5fd1ce4ebc02	node
26	2143d4c5-a2df-4359-8fd3-34ae16da9c5c	node
27	1d73752c-f8ef-4e63-a5f1-e8ac869c3d29	node
28	c30480e7-2b3a-4590-8e2b-05442b6d91a6	node
29	d343b0de-554d-47b2-861d-81a5d8fcfbf9	node

Rdf Import

R (45)

Row	object	predicate	subject
0	Optional[Math]	cql://attribute/name	?0
1	?1	cql://foreign_key/secretary	?0
2	cql://entity/Department	http://www.w3.org/1999/02/22-r...	?0
3	cql://entity/Department	http://www.w3.org/2000/01/rdf-s...	cql://foreign_key/secretary
4	cql://entity/Employee	http://www.w3.org/2000/01/rdf-s...	cql://foreign_key/secretary
5	http://www.w3.org/1999/02/22-r...	http://www.w3.org/1999/02/22-r...	cql://foreign_key/secretary
6	Optional[Bob]	cql://attribute/first	?1
7	Optional[Bo]	cql://attribute/last	?1
8	Optional[1]	cql://attribute/age	?1
9	?1	cql://foreign_key/manager	?1
10	?0	cql://foreign_key/worksIn	?1
11	cql://entity/Employee	http://www.w3.org/1999/02/22-r...	?1
12	cql://entity/Department	http://www.w3.org/2000/01/rdf-s...	cql://attribute/name
13	http://www.w3.org/2001/XMLSch...	http://www.w3.org/2000/01/rdf-s...	cql://attribute/name
14	http://www.w3.org/1999/02/22-r...	http://www.w3.org/1999/02/22-r...	cql://attribute/name
15	Optional[Al]	cql://attribute/first	?2
16	Optional[2]	cql://attribute/age	?2
17	?1	cql://foreign_key/manager	?2
18	?0	cql://foreign_key/worksIn	?2
19	cql://entity/Employee	http://www.w3.org/1999/02/22-r...	?2
20	http://www.w3.org/2000/01/rdf-s...	http://www.w3.org/1999/02/22-r...	cql://entity/Department
21	cql://entity/Employee	http://www.w3.org/2000/01/rdf-s...	cql://attribute/first
22	http://www.w3.org/2001/XMLSch...	http://www.w3.org/2000/01/rdf-s...	cql://attribute/first
23	http://www.w3.org/1999/02/22-r...	http://www.w3.org/1999/02/22-r...	cql://attribute/first
24	cql://entity/Employee	http://www.w3.org/2000/01/rdf-s...	cql://foreign_key/worksIn

Backup Slides

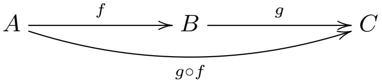
Category Theory

A category $\mathcal{C}$ consists of

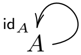
- *objects* A, B, C ... and *arrows* (also called *morphisms*) f, g, h ... such that:
- For every arrow f there is an object $\text{src}(f)$ called the *source* of f and an object $\text{tgt}(f)$ called the *target* of f. When $S = \text{src}(f)$ and $T = \text{tgt}(f)$, we may write $f : S \to T$. Visually:

$$S \xrightarrow{f} T$$

- For every arrow $f : A \to B$ and arrow $g : B \to C$ there is an arrow $g \circ f : A \to C$ called the *composite* of f and g:



- Composition is *associative*, i.e. $h \circ (g \circ f) = (h \circ g) \circ f$ for arbitrary f, g, and h.
- For every object A there is an *identity arrow* $\text{id}_A : A \to A$:



- Furthermore, for any arrow $f : A \to B$, $f \circ \text{id}_A = f = \text{id}_B \circ f$.

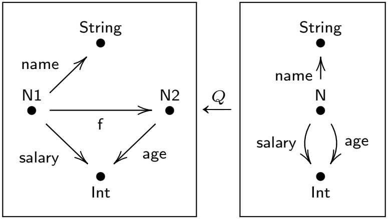
A *functor* $F : \mathcal{C} \to \mathcal{D}$ is a function from $\mathcal{C}$'s objects to $\mathcal{D}$'s objects and $\mathcal{C}$'s arrows to $\mathcal{D}$'s arrows that preserves composition and identity:

$$F(\text{id}_c) = \text{id}_{F(c)} \qquad F(f \circ g) = F(f) \circ F(g).$$

Categorical Schemas and Databases

- A *schema* S is a directed multi-graph and a set of paths through the graph called "equivalent".
- A schema S denotes a category $[\![S]\!]$:
  - The objects of $[\![S]\!]$ are the nodes of S.
  - The arrows of $[\![S]\!]$ are the paths through S, modulo the path equivalences in S.
- An S-instance (database on schema S) is a collection of sets, one per node in S, and a collection of (unary) functions, one per edge in S, satisfying the path equivalences in S.
- For example, these sets and functions may be represented as a collection of SQL tables, one per node in S, each with columns for edges out of that node.
- An S-instance denotes a functor $[\![S]\!] \to \mathbf{Set}$, where $\mathbf{Set}$, the category of sets, has for objects all sets and for arrows all (unary) functions.

Query Evaluation and Co-evaluation



N1			
ID	**name**	**salary**	**f**
1	Alice	$100	4
2	Bob	$250	5
3	Sue	$300	6

N2	
ID	**age**
4	20
5	20
6	30

$\xleftarrow{eval_Q}$

$\xrightarrow{coeval_Q}$

N			
ID	**name**	**salary**	**age**
a	Alice	$100	20
b	Bob	$250	20
c	Sue	$300	30

Example Round Trip

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

N2	
ID	**Age**
4	20
5	20
6	30

$\xrightarrow{coeval_Q}$

N			
ID	**Name**	**Salary**	**Age**
a	Alice	$100	$null_1$
b	Bob	$250	$null_2$
c	Sue	$300	$null_3$
d	$null_4$	$null_5$	20
e	$null_6$	$null_7$	20
f	$null_8$	$null_9$	30

$\downarrow \eta$ $\swarrow eval_Q$

N1		
ID	**Name**	**Salary**
a	Alice	$100
b	Bob	$250
c	Sue	$300
d	$null_4$	$null_5$
e	$null_6$	$null_7$
f	$null_8$	$null_9$

N2	
ID	**Age**
a	$null_1$
b	$null_2$
c	$null_3$
d	20
e	20
f	30