Categorical ontologies and databases

David I. Spivak

dspivak@math.mit.edu
Mathematics Department
Massachusetts Institute of Technology

Presented on 2012/07/26
at the Stanford Center for Biomedical Research

Research goal

- My goal is to develop a rigorous understanding of information:
  - what it is,
  - how it is used,
  - how it evolves,
  - how it can be faithfully transmitted.
- A good mathematical abstraction will be characterized by
  - clarity and obviousness;
  - coincidence: a matching of well-known ideas in math with well-known ideas in information science;
  - grace: coherent and well-coordinated extensions to differing perspectives on data.

Purpose of this talk:

To communicate a modern mathematical approach to databases and ontologies that is

- simple: intended that non-mathematicians can follow most of it without struggle;
- expressive: the model includes both databases and ontologies;
- efficient: well-established notation lets us speak easily about big ideas;
- effective: mathematical theorems can be applied to improve productivity.

Mantra: database schemas are categories

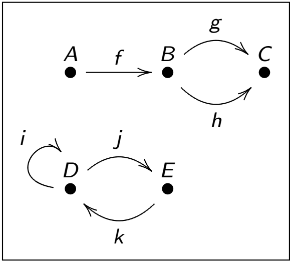
- A high-level view of data is captured in a database schema.
- A high-level view of mathematical objects is captured in a category.
- There is a deep similarity between database schemas and categories.
- We will examine this similarity and see where it takes us.

Plan of the talk

- Lay out the basic idea of categories and that of databases, and show the tight connection between them.
- Discuss schema evolution, data migration, and querying.
- Show how the RDF representation flows naturally from this model.
- Discuss SPARQL graph pattern queries in this language.

What is a category?

- A category is like a directed graph.

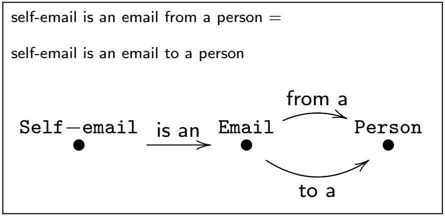
$\mathcal{C} :=$ 

- Categories come with one extra piece of expressivity: the ability to declare different paths to be equivalent.
  - Example: declare that $j; k \simeq i; i; i$ and $f; g \simeq f; h$.

Example

- How could one interpret this kind of abstraction?

$\mathcal{C} :=$



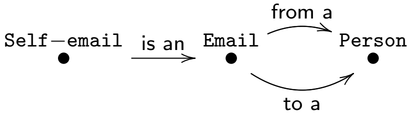
- Such "business rules" can be encoded into the category.

Example continued

Category:

self-email is an email from a person =

self-email is an email to a person



Self-email	
ID	**is an Email**
SEm1207	Em1207
SEm1210	Em1210
SEm1211	Em1211

Email		
ID	**from a Person**	**to a Person**
Em1206	Bob	Sue
Em1207	Carl	Carl
Em1208	Sue	Martha
Em1209	Chris	Bob
Em1210	Chris	Chris
Em1211	Julia	Julia
Em1212	Martha	Chris

Person
ID
Bob
Carl
Chris
Julia
Martha
Sue

A little history

- Since its invention in the early 1940s, category theory has revolutionized math.
  - Most modern papers in topology, algebra, or geometry could not even be written without category theory.
- It's like set theory and logic, except less floppy, more principles-based.
- It was invented to build bridges between disparate branches of math by distilling the essence of mathematical structure.
  - In a similar way, it can build bridges between different schemas....

Branching out

- Category theory naturally fosters connections between disparate fields.
- It has branched out of math and into physics, linguistics, and materials science.
- It has had much success in the theory of programming languages.
  - The pure category-theoretic concept of *monads* has vastly extended the reach of functional programming.
- I propose category theory as the natural language of informatics.

What is the essence of structure?

- If mathematics is the art of getting organized, what organizes math?
- After thousands of years, people realized that there was a common abstraction by which to structure much of mathematics.
- It consists of: objects, arrows, paths, and path equivalence.
- Or: things, tasks, processes, and "sameness of outcome".
- Or: primary keys, foreign keys, paths of FKs, and path equations.
- Let's give the definition.

Definition of a category I: Constituents

A *category* $\mathcal{C}$ consists of the following constituents:

1. A set $\mathbf{Ob}(\mathcal{C})$, called *the set of objects of* $\mathcal{C}$.
   - I'll denote each object $x \in \mathbf{Ob}(\mathcal{C})$ by $\overset{x}{\bullet}$.
2. A set $\mathbf{Arr}(\mathcal{C})$, called *the set of arrows of* $\mathcal{C}$, and two functions

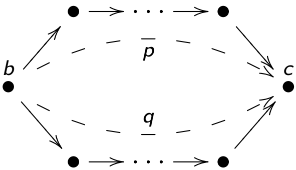
   $$src, tgt\colon \mathbf{Arr}(\mathcal{C}) \to \mathbf{Ob}(\mathcal{C}),$$

   assigning to each arrow its *source* and its *target* object, respectively.
   - An arrow $f \in \mathbf{Arr}(\mathcal{C})$ is often written $\overset{x}{\bullet}\xrightarrow{f}\overset{y}{\bullet}$, where $x = src(f), y = tgt(f)$.
   - We define a *path in* $\mathcal{C}$ to be a finite "head-to-tail" sequence of arrows in $\mathcal{C}$, e.g. $\overset{x}{\bullet}\xrightarrow{f}\overset{y}{\bullet}\xrightarrow{g}\overset{z}{\bullet}$.
   - Paths can have length n for any $n \in \mathbb{N}$, including $n = 0$ and $n = 1$.
3. An notion of equivalence for paths, denoted $\simeq$.

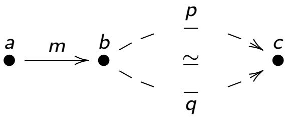
Definition of a category II: Rules

These constituents must satisfy the following requirements:

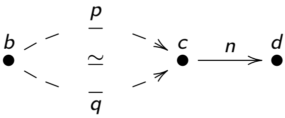
1. If $p \simeq q$ are equivalent paths then the sources agree: $src(p) = src(q)$.
2. If $p \simeq q$ are equivalent paths then the targets agree: $tgt(p) = tgt(q)$.
3. Suppose we have two paths (of any lengths) $b \to c$:



If $p \simeq q$ then for any extensions



or

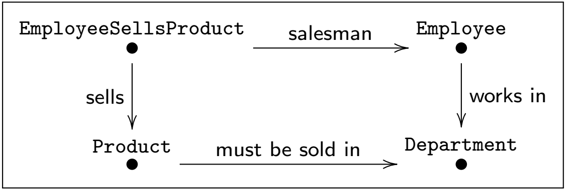


$m; p \simeq m; q$ and $p; n \simeq q; n$.

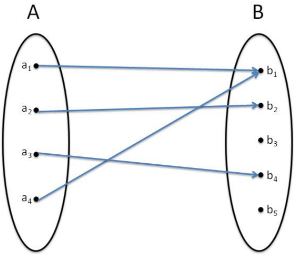
What does equivalence of paths mean?

- Nodes represent tables; arrows represent foreign keys.
- A path $p\colon \overset{a}{\bullet}\to\overset{b}{\bullet}$ represents "following foreign keys" from table a to table b.
- Following a path p, we can take any record in table a and return a record in table b.
- We declare two paths $p, q\colon \overset{a}{\bullet}\to\overset{b}{\bullet}$ equivalent if they should return the same record in b for any record in a.

Category:



The category of Sets



- Above we see two sets and a function between them. We would denote this categorically by $\overset{A}{\bullet}\xrightarrow{f}\overset{B}{\bullet}$
  - The objects of **Set** represent sets.
  - The arrows in **Set** represent functions.
  - A path represents a sequence of composable functions.
  - Two paths are equivalent if their compositions are the same.
- Note that b_3 and b_5 have been inserted, and a_1 and a_4 have been merged.

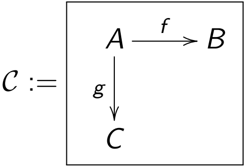
Functors: mappings between categories

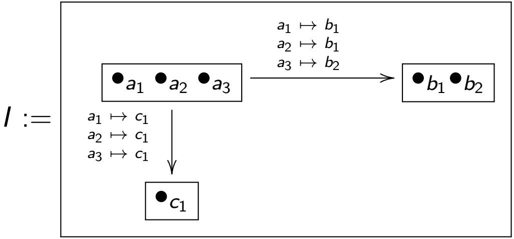
- One way to think of a category is as a directed graph, where certain paths have been declared equivalent.
- A functor is a graph mapping that is required to respect equivalence of paths.
- **Definition**: A functor $F\colon \mathcal{C} \to \mathcal{D}$ consists of
  - a function $\mathbf{Ob}(\mathcal{C}) \to \mathbf{Ob}(\mathcal{D})$ and
  - a function $\mathbf{Arr}(\mathcal{C}) \to \mathbf{Path}(\mathcal{D})$,

  such that F
  - respects sources and targets,
  - respects equivalences of paths.

Functors to **Set**

- A category $\mathcal{C}$ is a system of objects and arrows, and an equivalence relation on its paths.
- A functor $\mathcal{C} \to \mathcal{D}$ is a mapping that preserves these structures.
- **Set** is the category whose objects are sets, whose arrows are functions, and where paths are equivalent if they compose to the same function.
- If $\mathcal{C}$ is the category on the left below, then a functor $I \colon \mathcal{C} \to$ **Set** might look like this:





What is a database?

- A database consists of a bunch of tables and relationships between them.
- The rows of a table are called "records" or "tuples."
- The columns are called "attributes."
- An attribute may be "pure data" or may be a "key."
  - A table may have "foreign key columns" that link it to other tables.
  - A foreign key of table A links into the primary key of table B.
- A schema may have "business rules."

Foreign Keys and business rules

- Example:

Employee				
ID	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
ID	**Name**	**Secr**
q10	Sales	101
x02	Production	102

- Note the ID (primary key) columns and the foreign key columns.
- Perhaps we should enforce certain integrity constraints (business rules):
  - The manager of an employee E must be in the same department as E,
  - The secretary of a department D must be in D.

Data columns as foreign keys

- Theoretically we can consider a data-type as a 1-column table.
- Examples:

String
ID
a
b
⋮
z
aa
ab
⋮

Integer
ID
0
1
⋮
9
10
11
⋮

- So even data columns can be considered as foreign keys (to respective 1-column tables).
- Conclusion: each column in a table is a key – one primary, the rest foreign.

Example again

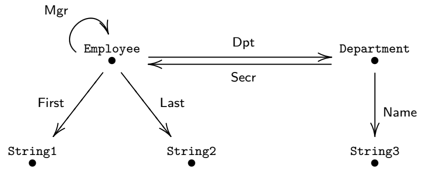
Employee				
ID	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
ID	**Name**	**Secr**
q10	Sales	101
x02	Production	102

String
ID
a
b
⋮
z
aa
ab
⋮

$\mathcal{C} :=$

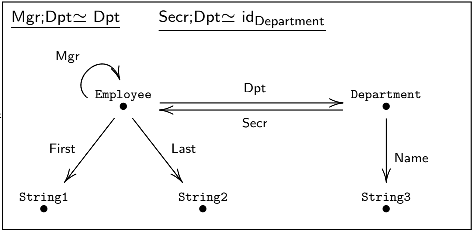
Mgr;Dpt $\simeq$ Dpt Secr;Dpt $\simeq$ $\text{id}_{\text{Department}}$



Database schema as a category

- A database schema is a system of tables linked by foreign keys.
- In case you missed it, this is the same thing as a category.

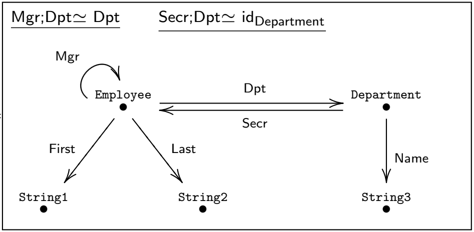
$\mathcal{C} :=$



  - Each object x in $\mathcal{C}$ is a table (Employee, Departments, String);
  - each arrow $x \to y$ is a column of table x.
- ID column of a table corresponds to the trivial path on that object.
- Declaring business rules (e.g. Mgr;Dpt $\simeq$ Dpt) is declaring the path equivalence.

Schema=Category, Instance=Set-valued functor

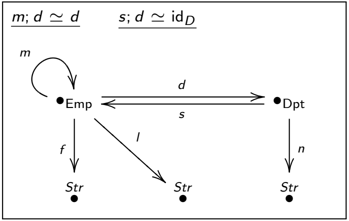
- Let $\mathcal{C}$ be the following category

$\mathcal{C} :=$ 

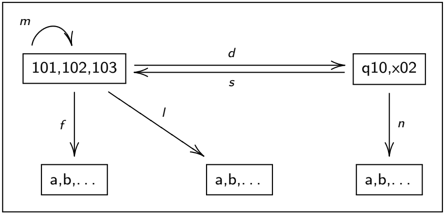
- A functor $I : \mathcal{C} \to$ **Set** consists of
  - A set for each object of $\mathcal{C}$ and
  - a function for each arrow of $\mathcal{C}$, such that
  - the declared equations hold.
- In other words, I fills the schema with compatible data.
- Categorical databases could also be called *functional databases*.

Data as a set-valued functor

$\mathcal{C} :=$



$I\colon \mathcal{C} \to \mathbf{Set}$



- A category $\mathcal{C}$ is a schema. An object $c \in \mathbf{Ob}(\mathcal{C})$ is a table.
- A functor $I\colon \mathcal{C} \to \mathbf{Set}$ fills the tables with compatible data.
- For each table c, the set $I(c)$ is its set of rows.
- The path equivalences in $\mathcal{C}$ are enforced by I as business rules.

Summary of basic setup

- The connection between categories and databases is simple.
- A schema is a custom category.
  - Note a key departure from Codd's model: categories are holistic.
  - The schema is captured as a whole.
  - Normal form results naturally from the language of path equivalence.
- A functors $I\colon \mathcal{C} \to \mathbf{Set}$ is an instance of $\mathcal{C}$.
- What about functors $F\colon \mathcal{C} \to \mathcal{D}$ between schemas?

Changes

- We've discussed the situation as though static: a single schema and a single instance.
- Next we'll discuss changes.
- Changing the schema (schema mappings).
  - Without holistic model, the language isn't really there to discuss schema evolution, data migration, etc.
- Changing the data (updates).

Changes in schema

- Suppose in our modeling of a given situation, we evolve from schema $\mathcal{C}$ to schema $\mathcal{D}$.
- We should find a functorial connection between them.
- Over time we may have something like

$$\mathcal{C} = \mathcal{C}_0 \xrightarrow{F_0} \mathcal{C}_1 \xrightarrow{F_1} \cdots \xrightarrow{F_{n-1}} \mathcal{C}_n = \mathcal{D}$$

- We want to be able to migrate data from $\mathcal{C}$ to $\mathcal{D}$ and vice versa.
- We want to be able to migrate queries against $\mathcal{C}$ to queries against $\mathcal{D}$ and vice versa.
- And we want this all to work as it "should".

Composing functors

- Suppose $F\colon \mathcal{C} \to \mathcal{D}$ and $G\colon \mathcal{D} \to \mathcal{E}$ are functors.
- What is their composition $\mathcal{C} \to \mathcal{E}$?
  - We have a way to take objects in $\mathcal{C}$ to objects in $\mathcal{E}$,
  - Arrows in $\mathcal{C}$ turn into paths in $\mathcal{D}$ and arrows in $\mathcal{D}$ turn into paths in $\mathcal{E}$.
  - We can concatenate these, thus taking arrows in $\mathcal{C}$ to paths in $\mathcal{E}$.
  - Our rules ensure that the equivalences in $\mathcal{C}$ will be preserved in $\mathcal{E}$.
- Composing functors is going to make migrating data more straightforward.

Changes in data

- Let $\mathcal{C}$ be a schema and let $I, J \colon \mathcal{C} \to \mathbf{Set}$ be two instances.
- A *natural transformation* $u \colon I \to J$ consists of the following:
  - For each object (table) $T \in \mathbf{Ob}(\mathcal{C})$ we get a map of record sets
    $$u_T \colon I(T) \to J(T).$$
  - For each arrow (foreign key) $f \colon T \to T'$, we get data consistency; formally,
    $$J(f) \circ u_T = u_{T'} \circ I(f).$$
- These correspond to updates like insert, merge, delete, and split.

The category of instances

- Given a schema $\mathcal{C}$, the *category of instances* on $\mathcal{C}$ is denoted $\mathcal{C}$–**Inst**.
    - The objects of $\mathcal{C}$–**Inst** are instances, i.e. functors $I : \mathcal{C} \to \mathbf{Set}$.
    - The arrows are natural transformations (updates).
    - The paths are sequences of updates.
    - Two paths are equivalent if they result in the same update.
- The category $\mathcal{C}$–**Inst** is a topos; it has an internal language and logic supporting the *typed lambda calculus*.
- That means, it works well with the theory of programming languages.

Data migration

- Let $\mathcal{C}$ and $\mathcal{D}$ be different schemas.
- We call a functor between them, $F\colon \mathcal{C} \to \mathcal{D}$, a *schema mapping*.
- Given such a mapping, we want to be able to canonically transfer instances on $\mathcal{C}$ to instances on $\mathcal{D}$ and vice versa.
- That means, given $F\colon \mathcal{C} \to \mathcal{D}$ we want functors

$$\mathcal{C}\text{–}\mathbf{Inst} \to \mathcal{D}\text{–}\mathbf{Inst}$$

and

$$\mathcal{D}\text{–}\mathbf{Inst} \to \mathcal{C}\text{–}\mathbf{Inst}.$$

What a functor $\mathcal{C}$–**Inst** $\to$ $\mathcal{D}$–**Inst** means.

A functor $\mathcal{C}$–**Inst** $\to$ $\mathcal{D}$–**Inst** means:

- **Objects:** To every instance on $\mathcal{C}$ we associate an instance on $\mathcal{D}$.
- **Arrows:** For every update on a $\mathcal{C}$-instance there is a corresponding update on the associated $\mathcal{D}$-instance.
- **Path equivalences:** If two different sequences of updates on $\mathcal{C}$-instances result in the same mapping, then the same will hold of the corresponding sequences on $\mathcal{D}$-instances.

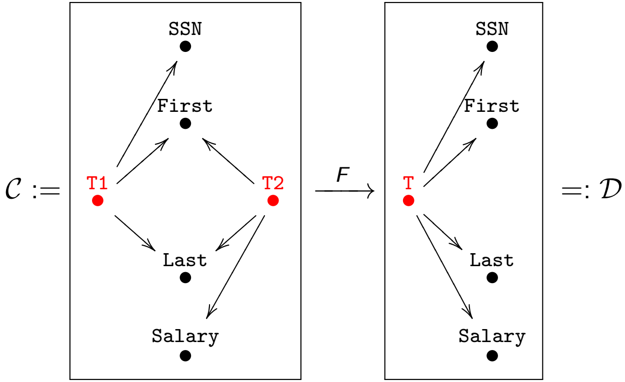
Functorial data migration for CT experts

- For any schema (category) $\mathcal{C}$, we have the category $\mathcal{C}$–**Inst** of set-valued functors $I: \mathcal{C} \to \textbf{Set}$ and natural transformations. These are the instances of the database.
- A functor $F: \mathcal{C} \to \mathcal{D}$ serves as a translation between schemas.
- Composition with F induces a functor $\Delta_F: \mathcal{D}\text{–}\textbf{Inst} \to \mathcal{C}\text{–}\textbf{Inst}$,

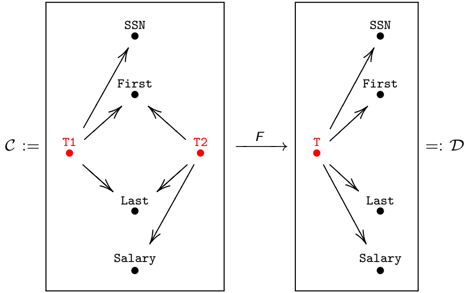
$$\mathcal{C} \xrightarrow{F} \mathcal{D} \xrightarrow{I} \textbf{Set}.$$

- The functor Δ_F migrates data from $\mathcal{D}$ back to $\mathcal{C}$.
- It has two adjoints $\Sigma_F: \mathcal{C}\text{–}\textbf{Inst} \to \mathcal{D}\text{–}\textbf{Inst}$ and $\Pi_F: \mathcal{C}\text{–}\textbf{Inst} \to \mathcal{D}\text{–}\textbf{Inst}$.

Uses of functorial data migration 1: Translation F



Uses of functorial data migration 2: Projection via Δ_F



$J\colon \mathcal{D} \to$ **Set**:

T				
ID	**SSN**	**First**	**Last**	**Salary**
XF667	115-234	Bob	Smith	$250
XF891	122-988	Sue	Smith	$300
XF221	198-877	Alice	Jones	$100

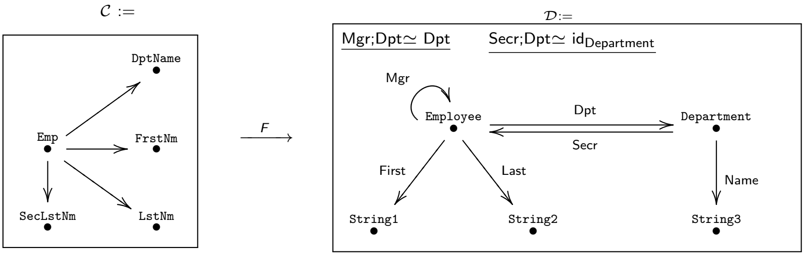
$\Delta_F(J)\colon \mathcal{C} \to$ **Set**:

T1			
ID	**SSN**	**First**	**Last**
XF667T1	115-234	Bob	Smith
XF891T1	122-988	Sue	Smith
XF221T1	198-877	Alice	Jones

T2			
ID	**First**	**Last**	**Salary**
XF667T2	Bob	Smith	$250
XF891T2	Sue	Smith	$300
XF221T2	Alice	Jones	$100

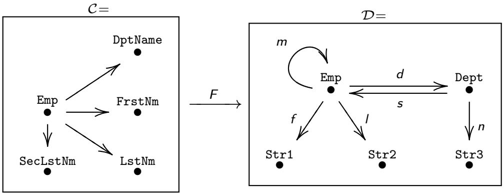
Another example of Δ_F

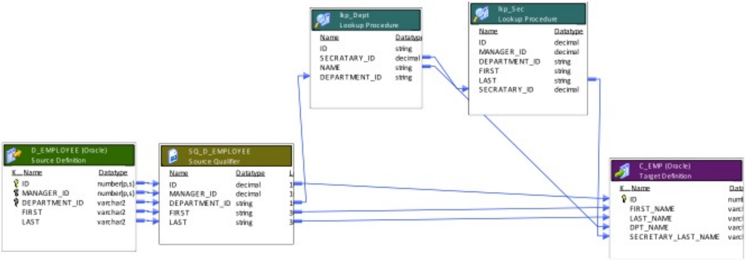
- Consider the schema mapping



- We get $\Delta_F \colon \mathcal{D}\text{–}\mathbf{Inst} \to \mathcal{C}\text{–}\mathbf{Inst}$
- Given an instance on $\mathcal{D}$ we get one on $\mathcal{C}$.
- Given an update on $\mathcal{D}$ we get one on $\mathcal{C}$.

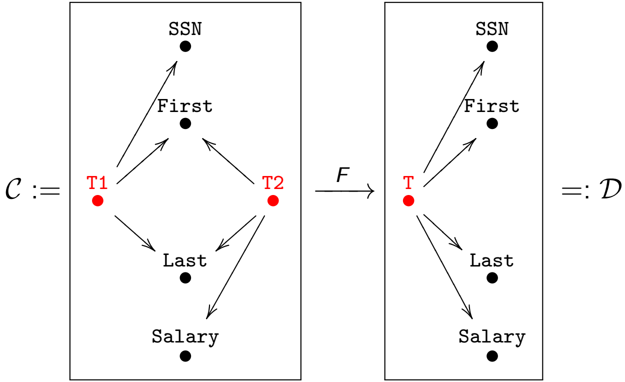
Compare the Informatica picture



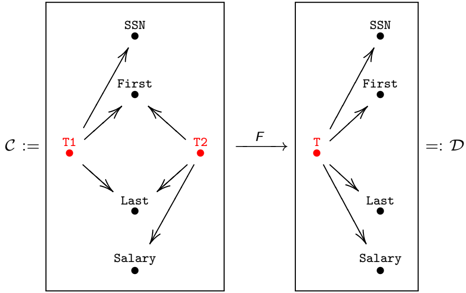


How could you move data forward along F?

Question: How could you take data on $\mathcal{C}$ and get data on $\mathcal{D}$?



Uses of functorial data migration 3: Joins via Π_F



$I\colon \mathcal{C} \to$ **Set**:

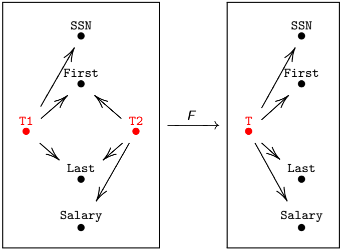
T1			
ID	**SSN**	**First**	**Last**
T1-001	115-234	Bob	Smith
T1-002	122-988	Sue	Smith
T1-003	198-877	Alice	Jones

T2			
ID	**First**	**Last**	**Salary**
T2-A101	Alice	Jones	\$100
T2-A102	Sam	Miller	\$150
T2-A104	Sue	Smith	\$300
T2-A110	Carl	Pratt	\$200

$\Pi_F(I)\colon \mathcal{D} \to$ **Set**:

T				
ID	**SSN**	**First**	**Last**	**Salary**
T1-002T2-A104	122-988	Sue	Smith	\$300
T1-003T2-A101	198-877	Alice	Jones	\$100

Uses of functorial data migration 4: Unions via Σ_F

$\mathcal{C} :=$ 

$=: \mathcal{D}$

$I : \mathcal{C} \to \mathbf{Set}$:

T1

ID	SSN	First	Last
T1-001	115-234	Bob	Smith
T1-002	122-988	Sue	Smith
T1-003	198-877	Alice	Jones

T2

ID	First	Last	Salary
T2-A101	Alice	Jones	$100
T2-A102	Sam	Miller	$150
T2-A104	Sue	Smith	$300
T2-A110	Carl	Pratt	$200

$\Sigma_F(I) : \mathcal{D} \to \mathbf{Set}$:

T

ID	SSN	First	Last	Salary
T1-001	115-234	Bob	Smith	T1-001.Salary
T1-002	122-988	Sue	Smith	T1-002.Salary
T1-003	198-877	Alice	Jones	T1-003.Salary
T2-A101	T2-A101.SSN	Alice	Jones	$100
T2-A102	T2-A102.SSN	Sam	Miller	$150
T2-A104	T2-A104.SSN	Sue	Smith	$300
T2-A110	T2-A110.SSN	Carl	Pratt	$200

Project, join, union

- Every category theorist knows about the functors Δ, Π, and Σ
  - They occur throughout mathematics.
  - They have been well-studied and there are theorems about their behavior.
- It is a great coincidence that they correspond so well to Project, Union, and Join.
- We can leverage established theorems to predict the results of queries and data migration.

Summary of data migration and views

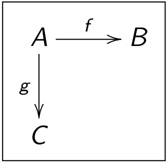
- The language of categories and functors puts everything on an equal footing.
  - Each schema is a category; each schema mapping is a functor.
  - **Set** is a category; an instance is a functor from a schema to **Set**.
  - Instances on a schema form a category; different instance categories can be compared with functors.
  - Everything is kept organized, but it's all interoperable.
- Schema mappings are totally graphical.
  - As a UI, just drag and drop bullets from one schema to another.
  - Creating or evolving schemas should be much easier.
  - In CTDB, instead of having a graphical interface on top, the graph is part of the formal structure.
  - Should increase productivity over current clumsy ETL software.

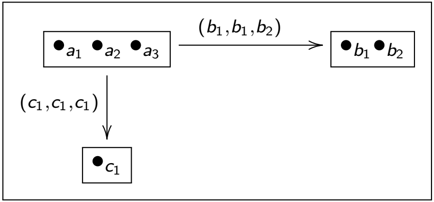
Semi-structured data

- The relational model demands adherence to a pre-defined schema.
- RDF allows for semi-structured data.
- The category-theoretic model extends easily, and it translates between these two viewpoints.
  - I will show how to convert a database to a triple store.
  - I will formulate graph pattern queries mathematically.
  - Everything below is classical mathematics, viewed from a data perspective.

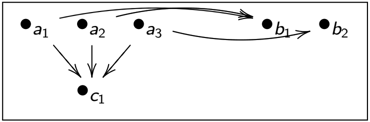
The Grothendieck construction

- Let $\mathcal{C}$ be a category and let $I\colon \mathcal{C} \to \mathbf{Set}$ be a functor.
- We can convert I into a category $Gr(I)$ in a canonical way:
  - Example:

$\mathcal{C} :=$ 

$I :=$ 

  - $Gr(I)$ is also known as *the category of elements of I*:

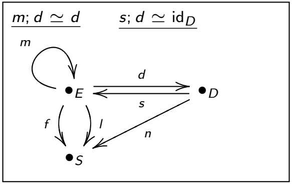
$Gr(I) =$ 

Grothendieck construction applied to database instances

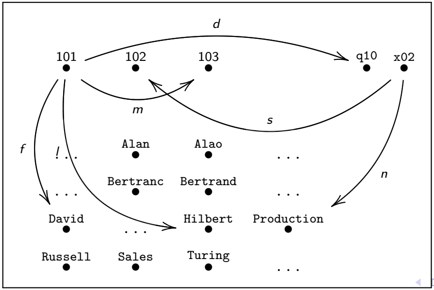
- Suppose given the following instance, considered as $I\colon \mathcal{C} \to \mathbf{Set}$

Employee				
ID	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
ID	**Name**	**Secr'y**
q10	Sales	101
x02	Production	102

$\mathcal{C} =$ 

Here is $Gr(I)$, the category of elements of I:

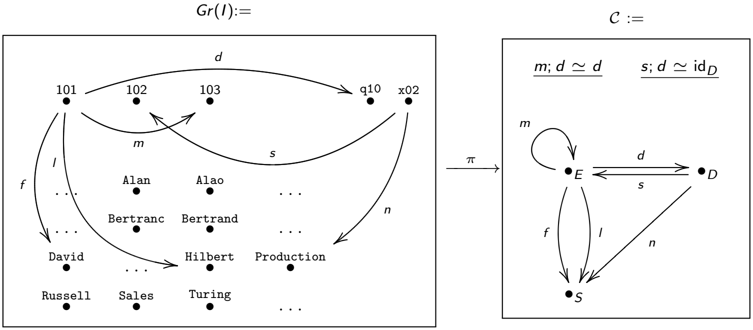
$Gr(I) =$ 

10 arrows left out.

A different perspective on data

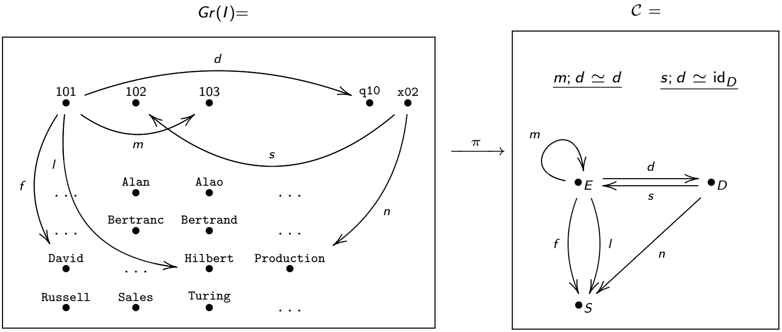
In fact, the Grothendieck construction of $I\colon \mathcal{C} \to \mathbf{Set}$ always yields not only a category $Gr(I)$ but a functor

$$\pi\colon Gr(I) \to \mathcal{C}.$$



For each $X \in \mathcal{C}$, its inverse image is $\pi^{-1}(X) = I(X)$, the set of rows in table X.

RDF schema and stores



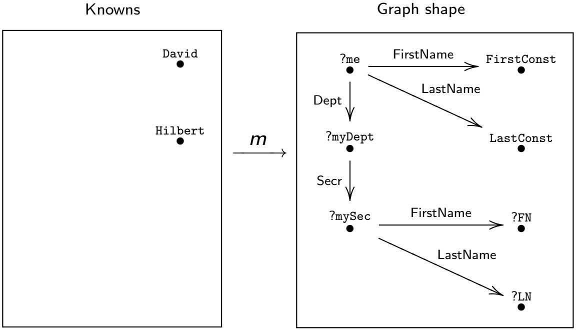
- The relation to RDF triples is clear: each arrow $f : x \to y$ in $Gr(I)$ is a triple with subject x, predicate f, and object y.
- For example (101 department q10), (x02 name Production), etc..
- $\mathcal{C}$ is the RDF schema and $Gr(I)$ is the triple store.

Graph pattern queries: Hilbert's problem

- Suppose David Hilbert wants to know the name of his department's secretary.
- The relevant SPARQL query would look something like

```
SELECT  ?FN ?LN
WHERE   {   ?me FirstName David
            ?me LastName Hilbert
            ?me Dept ?myDept
            ?myDept Secr ?mySec
            ?mySec FirstName ?FN
            ?mySec Lastname ?LN     }
```

Encoding a graph pattern query categorically 1

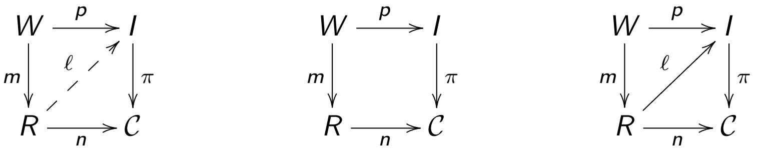



```
SELECT   ?FN ?LN
WHERE    {    ?me FirstName David
              ?me LastName Hilbert
              ?me Dept ?myDept
              ?myDept Secr ?mySec
              ?mySec FirstName ?FN
              ?mySec Lastname ?LN       }
```

Encoding a graph pattern query categorically 2

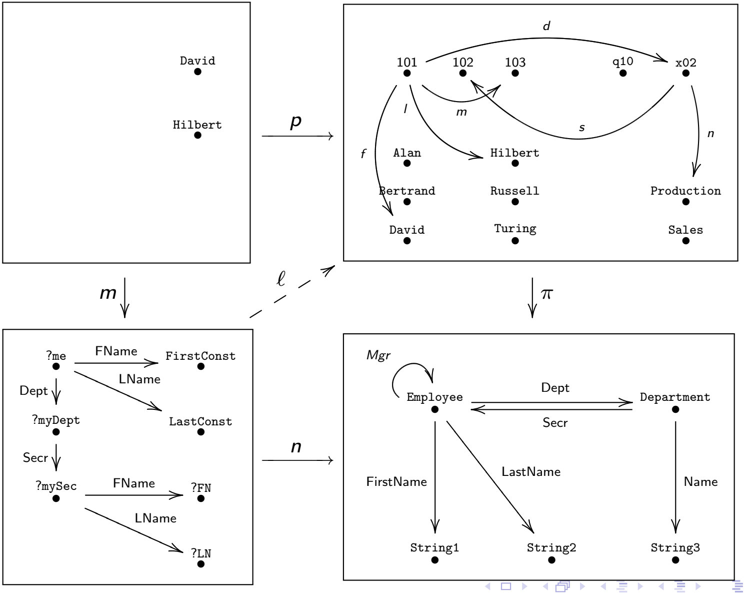
The "lifting problem approach."

Abbreviation: Query: Solutions:



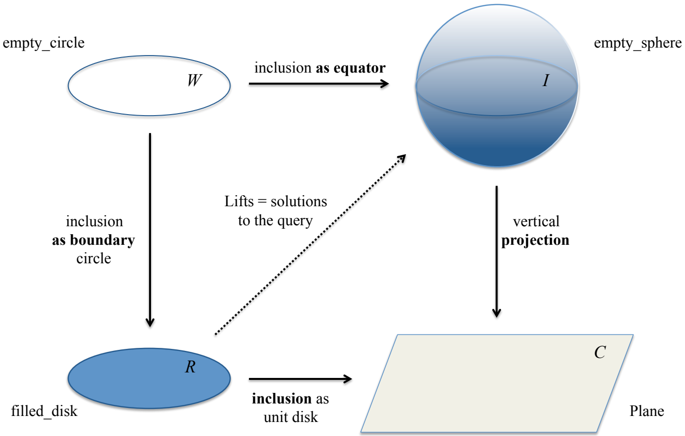
In the next slide you'll see such an arrangement for Hilbert's problem.

Encoding a graph pattern query categorically 3



The lifting problem approach in topology

This kind of picture pops up all over algebraic topology.



It's all in the same language

$$\begin{array}{ccc} W & \xrightarrow{p} & I \\ {\scriptstyle m}\downarrow & \overset{\ell}{\nearrow} & \downarrow{\scriptstyle \pi} \\ R & \xrightarrow[n]{} & \mathcal{C} \end{array}$$

- Everything in the query above is a category or a functor.
- Queries and constraints can both be phrased in terms of lifting problems.
- For example, one can use lifting problems to declare that
  - a foreign key must be injective
  - a foreign key must be surjective
  - a relation must be reflexive, symmetric, or transitive,
  - a table must be non-empty, etc.

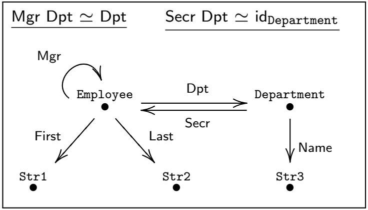
Conclusion

- I hope the connection between databases and categories is clear.

Employee				
ID	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
ID	**Name**	**Secr**
q10	Sales	101
x02	Production	102

$\mathcal{C}=$



- I discussed how one can use this connection to facilitate:
  - schema mapping and data migration;
  - formalizing views;
  - merging relational and RDF outlooks.
- Category theory is well-suited for modeling informatics.

Thanks for the invitation to speak!