Databases are categories

David I. Spivak

dspivak@uoregon.edu
Mathematics Department
University of Oregon

Presented on 2010/06/03
Galois Connections

My background

- Coming to mathematics.
- Coming to category theory.
- Coming to information science.

Need for coherence

- The world of information suffers from lack of coherence.
  - Databases are incompatible;
  - Vocabularies are mismatched;
  - Different systems do not work together.
- Need an overarching framework.
  - Standards can alleviate this type of problem.
  - Need a standard for information of all types.
  - A standard for information must be flexible yet rigorous.

Role of Mathematics

- Mathematics is a powerful language.
  - In science, strongest findings are mathematical.
  - In computer science, math gives firmest foundation. Examples
    - Functional programming;
    - Lambda calculus;
    - Trees, graphs;
    - Relational databases.
  - Mathematics offers "high assurance"!
- Can mathematics model information itself?

Category theory

- History
  - Invented in 1941 to relate geometry and algebra.
  - Considered at first to be too abstract,
  - Now dominating mathematical literature.
- Current role: modeling mathematics
  - Category theory gets to the heart of what is being modeled.
  - Different categories can be related by functors.
  - Functors are maps that preserve relationships.
- Branching out:
  - Now useful in computer science, physics, and linguistics.
- Similar in feeling to Haskell.
- "Correctness in complexity."

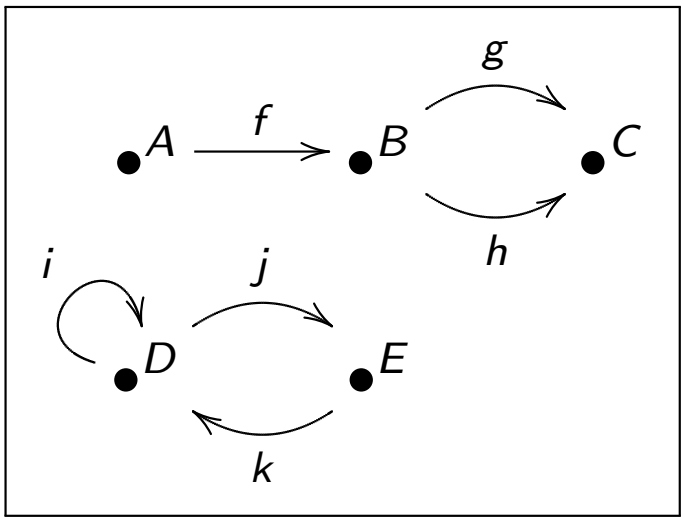
In this talk

Categories = Database schemas

- I will show that a category $\mathcal{C}$ is just a database schema.
- The data itself is given by a functor $F : \mathcal{C} \to \mathbf{Set}$.
- Thus categories are not so foreign to computer scientists and databases are not so foreign to mathematicians.

Categories

- Idea: A category models objects of a certain sort and the relationships between them.



- Think of it like a graph: the nodes are objects and the arrows are relationships.
- Some paths can be equated with others (example: $j.k = i^3$).

Definition of a category $\mathcal{C}$

- A set of objects, $\mathbf{Ob}(\mathcal{C})$, and a set of arrows $\mathbf{Arr}(\mathcal{C})$.
- Each arrow has a source and target object; every object has a primary arrow (also called an "identity arrow"):

$$\mathbf{Arr}(\mathcal{C}) \underset{t}{\overset{s}{\underset{\longleftarrow}{\overset{\longleftarrow}{p}}}} \mathbf{Ob}(\mathcal{C}).$$

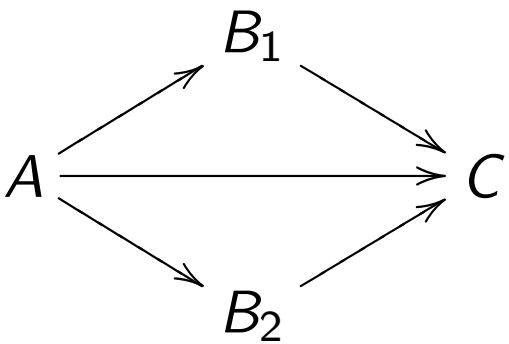
- Composition data: paths are arrows.

$$A \xrightarrow{f} B \xrightarrow{g} C. \qquad f.g : A \to C$$

- Associative law: $(f.g).h = f.(g.h)$
- Identity law for any $A \xrightarrow{f} B$, $p(A).f = f = f.p(B)$.

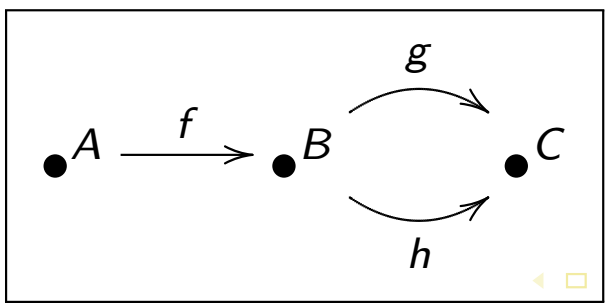
Examples of categories

- **Set**. Objects are sets, arrows are functions.
- **Hask**. Objects are Haskell data-types, arrows are functions.
- Partial orders, e.g.:



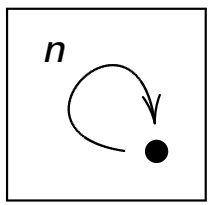
$$\text{“}A \leq B_1, B_2 \leq C\text{”}$$

- Graphs: Any graph can be turned into a category using a “free” composition law – make each path a new arrow.



Monoids

- A monoid is a category with one object but possibly many arrows.
- If M is a monoid, $\mathbf{Arr}(M)$ is a set with a unit and multiplication law.
- Example: $(\mathbb{N}, *) =$



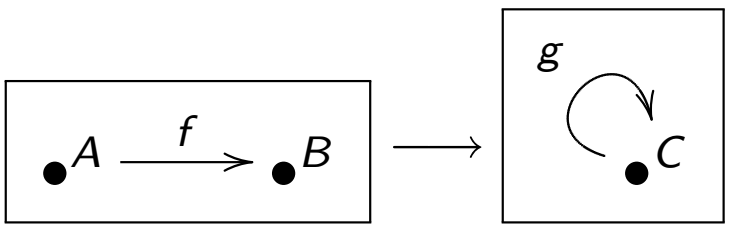
  - Composition law is given by $n * m = nm$, e.g. $2 * 3 = 6$.
  - $\mathbf{Arr}(\mathbb{N}, *) = \mathbb{N}$, the set of natural numbers.

Functors

- Idea: A functor is a graph morphism that is required to respect the composition law.
- Definition: A functor $F : \mathcal{C} \to \mathcal{D}$ consists of
  - A function $\mathbf{Ob}(F) : \mathbf{Ob}(\mathcal{C}) \to \mathbf{Ob}(\mathcal{D})$ and
  - a function $\mathbf{Arr}(F) : \mathbf{Arr}(\mathcal{C}) \to \mathbf{Arr}(\mathcal{D})$,

  that respect
  - the source and target of every arrow,
  - the primary arrow of every node, and
  - the composition law.

Examples of functors

- For any category $\mathcal{C}$, the identity functor $\mathrm{id}_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$.
- $\{*\} : \mathcal{C} \to \mathbf{Set}$. Everything in $\mathcal{C}$ is sent to the one-point set.
- Inst : **Hask** $\to$ **Set**. "Instances" for each data type.
- 

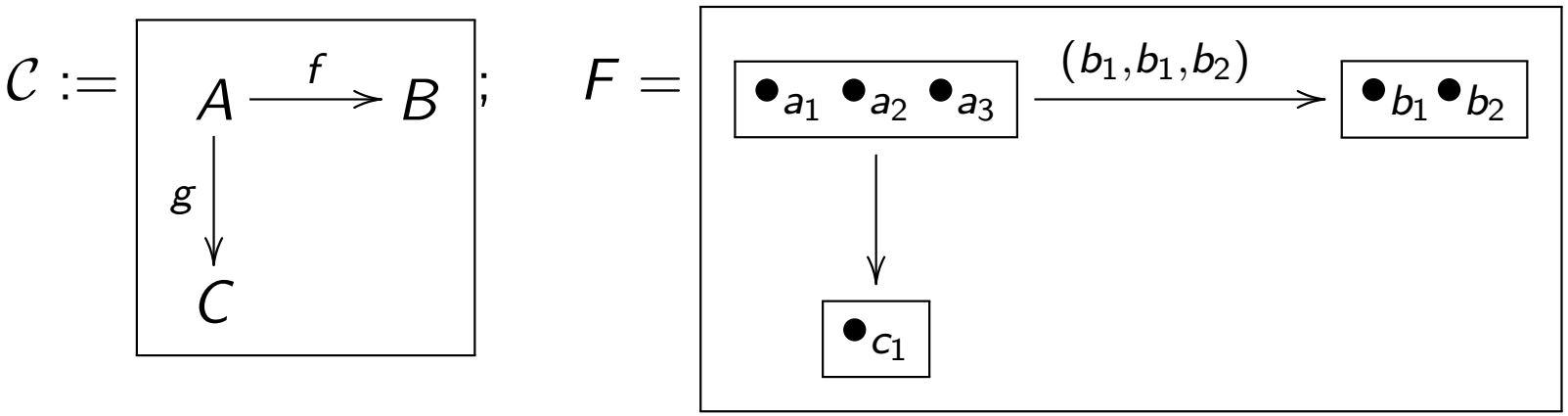
- **Cat** is the category whose objects are categories and morphisms are functors.
  - **Set** $\to$ **Cat**.
  - **Poset** $\to$ **Cat**
  - **Graph** $\to$ **Cat**.

M-sets

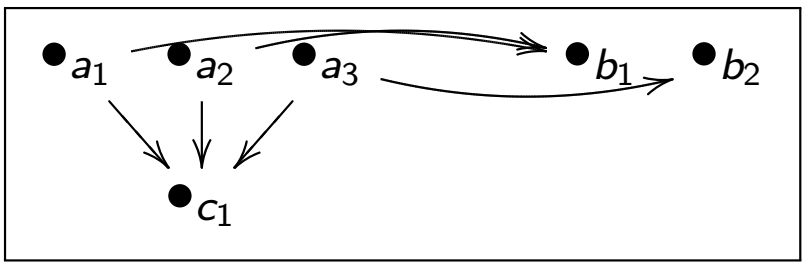
- If M is a monoid, a functor $F : M \to \mathbf{Set}$ is called an M-set.
  - The image of the unique object of M is a set, say S,
  - And each arrow $m \in \mathbf{Arr}(M)$ gives a function $S \to S$.
  - We call this function the *action* of m on S.
- Example: Finite state automaton..
  - S is the set of states,
  - M is the monoid of strings in the alphabet,
  - A curried version of F acts as the state transition function.
- There is a canonical functor $M \to \mathbf{Set}$, "M as an M-set"
  - The set $S = \mathbf{Arr}(M)$,
  - And the action is given by left multiplication.

Turning a functor into a category

- The Grothendieck construction.
- A functor $F : \mathcal{C} \to \mathbf{Set}$, "a model and its instances."
- Example:



- $Gr(F)$ is the category of instances:



What is a database?

- A database consists of a bunch of tables and relationships between them.
- The rows of a table are called "records," "tuples," or "instances."
- The columns are called "attributes."
- Columns may be "pure data" or may be "keys."
  - We take the convention that every table has a distinguished **Primary Key** column.
  - 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.

Foreign Keys

- Example:

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

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

- Note the primary key columns and foreign key columns.
- Perhaps we should enforce certain integrity constraints:
  - 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.
- Example:

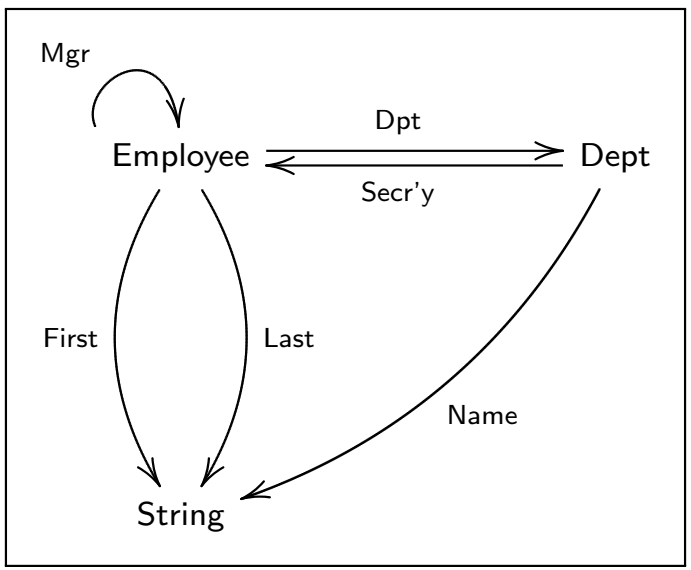
Char(4)
aaaa
aaab
⋮
Alan
Alao
⋮

- So any data column can be considered a foreign key to a 1-column table.
- Conclusion: each column in a table is a key – one primary, the rest foreign.

Example again

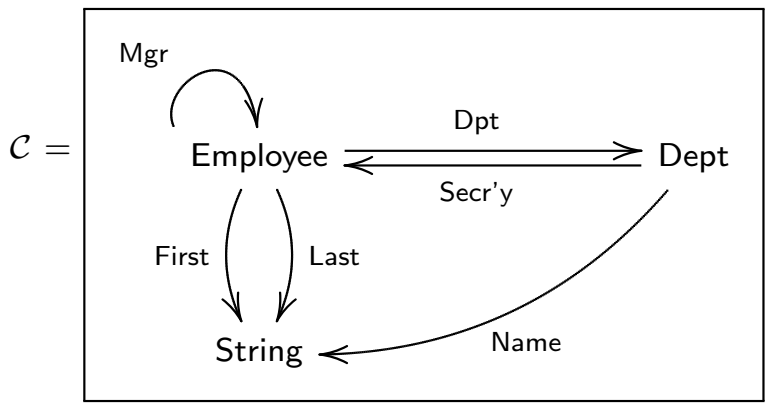
Employee				
Emp_Id	**First**	**Last**	**Mgr**	**Dpt**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

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



Database schema as a category

- A database schema is a system of tables linked by foreign keys.
- This is just a category!

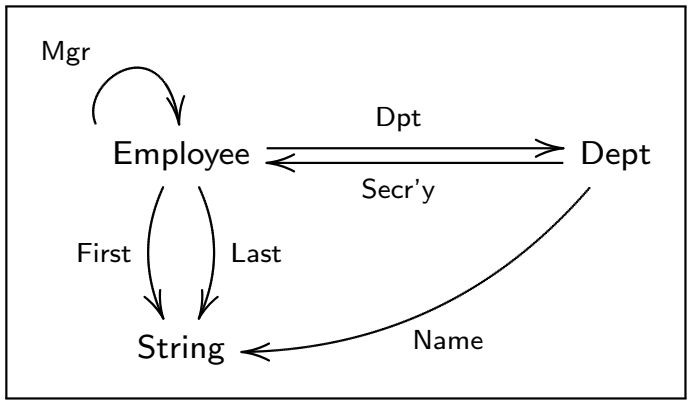


- Objects are tables, arrows are columns.
- Primary key column of a table is primary arrow of an object.
- Declaring integrity constraints (e.g. `Mgr.Dpt=Dpt`) is declaring composition law.

Schema=Category, Data=Functor

- Let

$\mathcal{C} =$



`Mgr.Dpt = Dpt;`

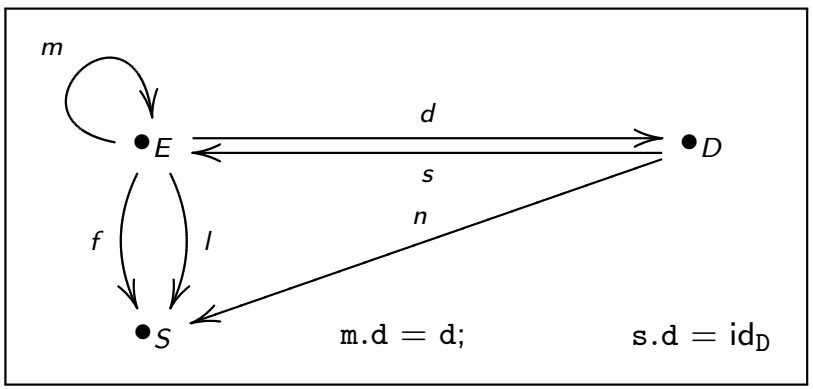
`Secr'y.Dpt = id_Dept`

.

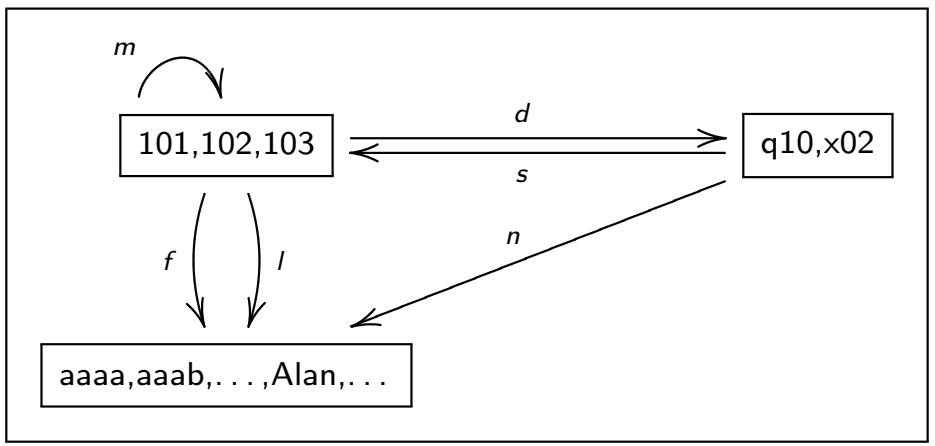
- A functor $F : \mathcal{C} \to \mathbf{Set}$ consists of
  - A set for each object of $\mathcal{C}$ and
  - a function for each arrow of $\mathcal{C}$, such that
  - the declared equations hold
- In other words, F fills the schema with data.

Data as a functor

$\mathcal{C} =$



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



.

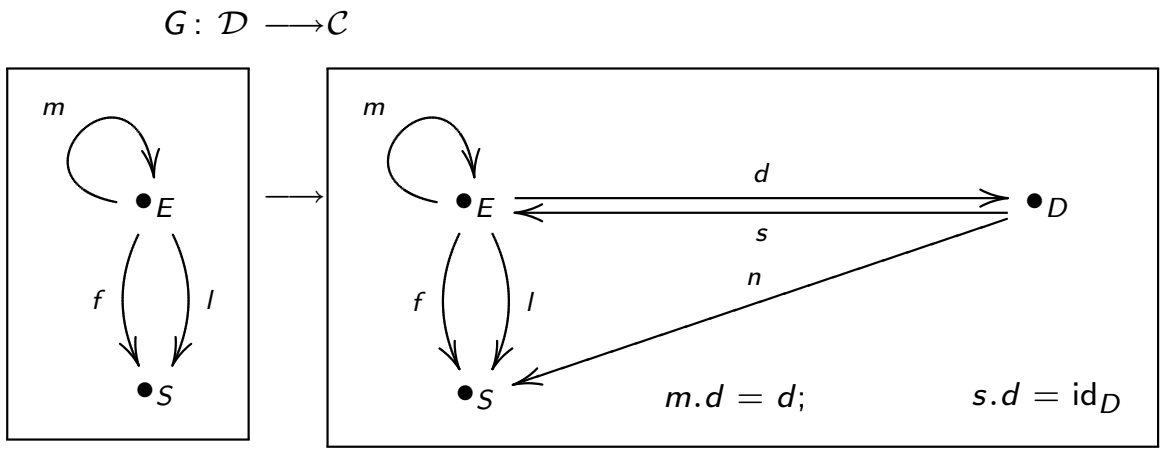
- A category $\mathcal{C}$ is a schema. An object $x \in \mathbf{Ob}(\mathcal{C})$ is a table.
- A functor $F : \mathcal{C} \longrightarrow \mathbf{Set}$ fills the tables with compatible data.
- For each table x, the set $F(x)$ is its set of rows.

Morphisms of schemas

- Morphisms of schemas are functors, $G : \mathcal{D} \to \mathcal{C}$.
- We can pull back data along G by way of a functor

$$G^* : \mathrm{Data}_{\mathcal{C}} \to \mathrm{Data}_{\mathcal{D}}.$$

- For example, if my schema has no "department" table, I can load your data with G^*:

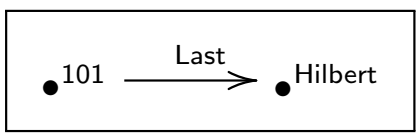


- I can also push data from $\mathcal{D}$ into data on $\mathcal{C}$ in canonical ways.

RDF

- Given a schema $\mathcal{C}$ and data $F : \mathcal{C} \to \mathbf{Set}$, we can apply the Grothendieck construction to F.
- Every row in the Employee, Department, and String tables becomes a vertex in a new "RDF graph."
- What are the arrows?
- Answer: each cell in a table becomes an arrow from the current row to the row in the foreign table.

	Last
101	Hilbert



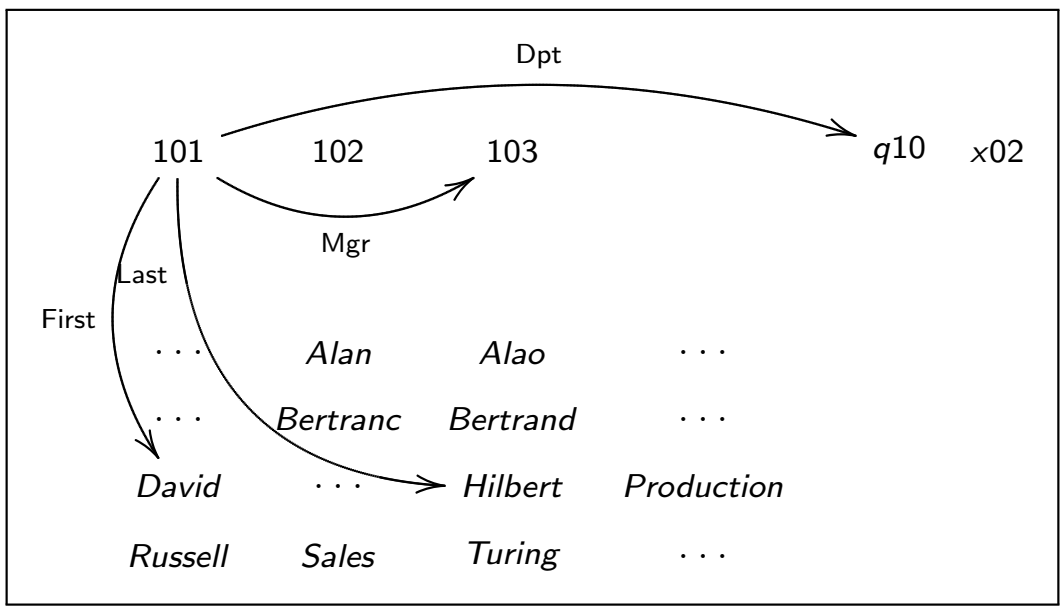
Example RDF

Under the Grothendieck construction, the database

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

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

becomes the RDF graph



(with 16 arrows left out for ease of reading).

Monoids

- Let M be a monoid. What is it as a database schema?
- It's a single table schema with many foreign keys referencing itself. One column for each arrow (element) of M.
- An M-set is a functor $F : M \to \mathbf{Set}$; a set of records.
- Example: $M = (\mathbb{N}, *)$, Consider M as an M-set.

$(\mathbb{N}, *)$					
1	**2**	**3**	**4**	**5**	$\cdots$
1	2	3	4	5	$\cdots$
2	4	6	8	10	$\cdots$
3	6	9	12	15	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Integrating disparate fields

- The purpose of a category is to distill the essence of a certain topic.
- This is also the goal of a database schema.
- In this talk we learned that categories and database schemas are the same thing.
- By integrating databases and category theory, we have linked two very different disciplines.
- These two disciplines can learn and benefit from one another.

A mathematical foundation for databases

- The usual logical foundation of databases is not sufficient for comparing different databases – too many levels upon levels.
- Category theory is designed for levels upon levels.
- A categorical foundation for databases could be useful in practice.
- Case in point: the immediate connection between relational databases and RDF.

Integrating data and programs

- **Hask** is a category, and Inst: **Hask** $\to$ **Set** is a functor.
- That means **Hask** can be considered a database.
  - Each type A can be considered a table:
  - a function $c: A \to B$ is a column of A (with values in B),
  - each instance of type A is a row r of table A.
  - The (r, c) cell of table A is the image $c(r) \in B$.
- Any other database can be considered as a category of "user-defined types."
- Defining Haskell functions on these types connects the user-defined category and the Haskell category.
- Thus databases and programs can be smoothly integrated.