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 $F : \mathcal{C} \to \textbf{Set}$.
• 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 $C$

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

\[
\begin{array}{ccc}
\text{Ob}(C) & \xleftarrow{p} & \text{Arr}(C) \\
\downarrow{t} & & \uparrow{s} \\
\end{array}
\]

- Composition data: paths are arrows.

\[
A \xrightarrow{f} B \xrightarrow{g} C.
\]

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

David I. Spivak  
Databases are categories
Examples of categories

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

  ![Diagram](#)

  \[
  A \leq B_1, B_2 \leq C
  \]

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

![Diagram](#)
Monoids

• A monoid is a category with one object but possibly many arrows.

• If $M$ is a monoid, $\text{Arr}(M)$ is a set with a unit and multiplication law.

• Example: $(\mathbb{N}, \cdot) = \begin{array}{c}
\text{n} \\
\circ \\
\text{1}
\end{array}$

  - Composition law is given by $n \cdot m = nm$, e.g. $2 \cdot 3 = 6$.
  - $\text{Arr}(\mathbb{N}, \cdot) = \mathbb{N}$, 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 $\text{Ob}(F) : \text{Ob}(\mathcal{C}) \to \text{Ob}(\mathcal{D})$ and
  - a function $\text{Arr}(F) : \text{Arr}(\mathcal{C}) \to \text{Arr}(\mathcal{D})$,

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 $C$, the identity functor $\text{id}_C : C \to C$.
• $\{\ast\} : C \to \text{Set}$. Everything in $C$ is sent to the one-point set.
• $\text{Inst} : \text{Hask} \to \text{Set}$. “Instances” for each data type.

• $\begin{array}{ccc} A & \xrightarrow{f} & B \end{array}$

• $\text{Cat}$ is the category whose objects are categories and morphisms are functors.
  • $\text{Set} \to \text{Cat}$.
  • $\text{Poset} \to \text{Cat}$
  • $\text{Graph} \to \text{Cat}$.
• If $M$ is a monoid, a functor $F: M \to \textbf{Set}$ is called an $M$-set.
  • The image of the unique object of $M$ is a set, say $S$,
  • And each arrow $m \in \text{Arr}(M)$ gives a function $S \to S$.
  • We call this function the \textit{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 $M \to \textbf{Set}$, \textquote{M as an $M$-set”
  • The set $S = \text{Arr}(M)$,
  • And the action is given by left multiplication.
Turning a functor into a category

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

  $C := \begin{array}{c} A \\ \downarrow g \\ C \end{array} \xrightarrow{f} \begin{array}{c} B \\ \downarrow \end{array}$; \hspace{1cm} F = \begin{array}{c} a_1 \cdot a_2 \cdot a_3 \\ \downarrow \end{array} \rightarrow \begin{array}{c} b_1 \cdot b_2 \\ \downarrow \end{array}$

- $\text{Gr}(F)$ 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:**

<table>
<thead>
<tr>
<th>Emp_Id</th>
<th>First</th>
<th>Last</th>
<th>Mgr</th>
<th>Dpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>David</td>
<td>Hilbert</td>
<td>103</td>
<td>q10</td>
</tr>
<tr>
<td>102</td>
<td>Bertrand</td>
<td>Russell</td>
<td>102</td>
<td>x02</td>
</tr>
<tr>
<td>103</td>
<td>Alan</td>
<td>Turing</td>
<td>103</td>
<td>q10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dept_Id</th>
<th>Name</th>
<th>Secr’y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q10</td>
<td>Sales</td>
<td>101</td>
</tr>
<tr>
<td>x02</td>
<td>Production</td>
<td>102</td>
</tr>
</tbody>
</table>

- 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:

<table>
<thead>
<tr>
<th>Char(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaaa</td>
</tr>
<tr>
<td>aaab</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Alan</td>
</tr>
<tr>
<td>Alao</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Emp_Id</th>
<th>First</th>
<th>Last</th>
<th>Mgr</th>
<th>Dpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>David</td>
<td>Hilbert</td>
<td>103</td>
<td>q10</td>
</tr>
<tr>
<td>102</td>
<td>Bertrand</td>
<td>Russell</td>
<td>102</td>
<td>x02</td>
</tr>
<tr>
<td>103</td>
<td>Alan</td>
<td>Turing</td>
<td>103</td>
<td>q10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dept_Id</th>
<th>Name</th>
<th>Secr’y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q10</td>
<td>Sales</td>
<td>101</td>
</tr>
<tr>
<td>x02</td>
<td>Production</td>
<td>102</td>
</tr>
</tbody>
</table>
Database schema as a category

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

\[ C = \]

- 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

\[ 
\begin{align*}
\mathcal{C} &= \text{Employee} \\
&\xrightarrow{\text{First}} \text{String} \\
&\xrightarrow{\text{Last}} \text{String} \\
&\xrightarrow{\text{Name}} \text{String} \\
&\xrightarrow{\text{Secr'y}} \text{Dept} \\
&\xrightarrow{\text{Dpt}} \text{Dept} \\
&\xrightarrow{\text{Mgr}} \text{Manager} \\
\end{align*}
\]

\[ \text{Mgr}.\text{Dpt} = \text{Dpt}; \]

\[ \text{Secr'y}.\text{Dpt} = \text{id}_\text{Dept} \]

- A functor \( F : \mathcal{C} \rightarrow \textbf{Set} \) 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.
A category $C$ is a schema. An object $x \in \text{Ob}(C)$ is a table.

A functor $F : C \rightarrow \text{Set}$ 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, \( G : \mathcal{D} \rightarrow \mathcal{C} \).
- We can pull back data along \( G \) by way of a functor
  \[ G^* : \text{Data}_\mathcal{C} \rightarrow \text{Data}_\mathcal{D}. \]
- For example, if my schema has no “department” table, I can load your data with \( G^* : \)

\[
\begin{tikzcd}
E \ar[rr, shift left=1] \ar[rr, shift right=1] \ar[rr, bend left=5] \ar[rr, bend right=5] \ar[rr, bend left=10] \ar[rr, bend right=10] & & D \\
S \ar[uu, shift left=1] \ar[uu, shift right=1] \ar[uu, bend left=5] \ar[uu, bend right=5] \ar[uu, bend left=10] \ar[uu, bend right=10]
\end{tikzcd}
\]

\[
m \cdot d = d; \]
\[
s \cdot d = \text{id}_D
\]

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

- Given a schema \( C \) and data \( F : C \to \text{Set} \), 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.
Example RDF

Under the Grothendieck construction, the database

<table>
<thead>
<tr>
<th>Employee</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp_Id</td>
<td>First</td>
</tr>
<tr>
<td>101</td>
<td>David</td>
</tr>
<tr>
<td>102</td>
<td>Bertrand</td>
</tr>
<tr>
<td>103</td>
<td>Alan</td>
</tr>
</tbody>
</table>

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 $F: M \to \textbf{Set}$; a set of records.
- Example: $M = (\mathbb{N}, \ast)$, Consider $M$ as an $M$-set.
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: $\text{Hask} \to \text{Set}$ 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.