Information and Computation 217 (2012) 31–51

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

# Functorial data migration ☆

David I. Spivak

*Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, United States*

ARTICLE INFO

*Article history:*
Received 14 January 2011
Revised 21 March 2012
Available online 11 May 2012

*Keywords:*
Category theory
Databases
Data migration
Adjoint functors
Queries

ABSTRACT

In this paper we present a simple database definition language: that of categories and functors. A database schema is a small category and an instance is a set-valued functor on it. We show that morphisms of schemas induce three "data migration functors", which translate instances from one schema to the other in canonical ways. These functors parameterize projections, unions, and joins over all tables simultaneously and can be used in place of conjunctive and disjunctive queries. We also show how to connect a database and a functional programming language by introducing a functorial connection between the schema and the category of types for that language. We begin the paper with a multitude of examples to motivate the definitions, and near the end we provide a dictionary whereby one can translate database concepts into category-theoretic concepts and vice versa.



## 1. Introduction

This paper has two main goals. The first goal is to present a straightforward category-theoretic model of databases under which every theorem about small categories becomes a theorem about databases. To do so, we will present a category **Sch** of database schemas, which has three important features:

- the category **Sch** is equivalent to **Cat**, the category of small categories,
- the category **Sch** is a faithful model for real-life database schemas, and
- the category **Sch** serves as a foundation upon which high-level database concepts rest easily and harmoniously.

The second goal is to apply this category-theoretic formulation to provide new data migration functors, so that for any translation of schemas $F : \mathcal{C} \to \mathcal{D}$, one can transport instances on the source schema $\mathcal{C}$ to instances on the target schema $\mathcal{D}$ and vice versa, with provable "round-trip" properties. For example, homomorphisms of instances are preserved under all migration functors. While these migration functors do not appear to have been discussed in database literature, their analogues are well-known in modern programming languages theory, e.g. the theory of dependent types [34], and polynomial data types [26]. This is part of a deeper connection between database schemas and *kinds* (structured collections of types, see [46,20]) in programming languages. See also Section 3.6.

An increasing number of researchers in an increasing variety of disciplines are finding that categories and functors offer high-quality models, which simplify and unify their respective fields.[1] The quality of a model should be judged by its

☆ This project was supported by ONR grant N000141010841.
*E-mail address:* dspivak@math.mit.edu.

[1] Aside from mathematics, in which category-theoretic language and theorems are indispensable in modern algebra, geometry, and topology, category theory has been successful in: programming language theory [35,37]; physics [3,12]; materials science [45,18]; and biology [40,16].

0890-5401/$ – see front matter 
http://dx.doi.org/10.1016/j.ic.2012.05.001

efficiency as a proxy or interface—that is, by the ease with which an expert can work with only an understanding of the model, and in so doing successfully operate the thing itself. Our goal in this paper is to provide a high-quality model of databases.

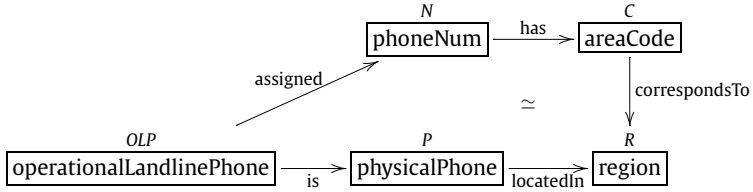
Other category-theoretic models of databases have been presented in the past [23,7,25,39,21,38,14,47]. Almost all of them used the more expressive notion of sketch where we have used categories. The additional expressivity came at a cost that can be cast in terms of our two goals for this paper. First, the previous models were more complex and this may have created a barrier to wide-spread understanding and adoption. Second, morphisms of sketches do not generally induce the sorts of data migration functors that morphisms of categories do.

It is our hope that the present model is simple enough that anyone who has an elementary understanding of categories (i.e. who knows the definition of category, functor, and natural transformation) will, without too much difficulty, be able to understand the basic idea of our formulation: database schemas as categories, database instances as functors. Moreover, we will provide a dictionary (see Section 3.7, Table 1) whereby the main results and definitions in this paper will simply correspond to results or definitions of standard category theory; this way, the reader can rely on tried and true sources to explain the more technical ideas presented here. Moreover, one may hope to leverage existing mathematical theory to their own database issues through this connection.

Before outlining the plan of the paper in Section 1.2 we will give a short introduction to the fundamental idea on which the paper rests, and provide a corresponding "categorical normal form" for databases, in Section 1.1.

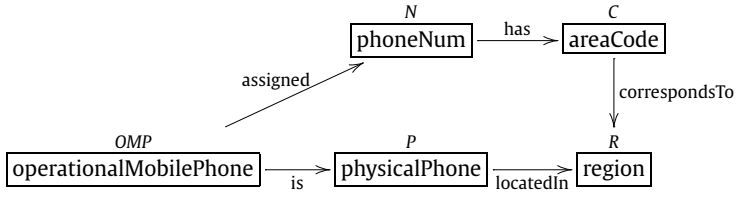
### 1.1. Categorical normal form

A database schema may contain hundreds of tables and foreign keys. Each foreign key links one table to another, and each sequence of foreign keys $T_1 \to T_2 \to \cdots \to T_n$ results in a function $f$ from the set of records in $T_1$ to the set of records in $T_n$. It is common that two different foreign key paths, both connecting table $T_1$ to table $T_n$, may exist; and they may or may not define the same mapping on the level of records. For example, an operational "landline" phone is assigned a phone number whose area code corresponds to the region in which the physical phone is located. Thus we have two paths $OLP \to R$:




(1)

and the data architect for this schema knows whether or not these two paths should always produce the same mapping. In (1), the two paths $OLP \to R$ do produce the same mapping, and the $\simeq$ sign is intended to record that fact. For contrast, if we replace operationalLandlinePhone with operationalMobilePhone (OMP), the two paths $OMP \to R$ would not produce the same mapping, because a cellphone need not be currently located in the region indicated by its area code. Thus we would get a similar but different diagram,




(2)

We are emphasizing here that the notion of *path equivalence* is an important and naturally-arising integrity constraint, which provides a crucial clue into the intended semantics of the schema. Its enforcement is often left to the application layer, but it should actually be included as part of the schema [22]. Including path equivalence information in the database schema has three main advantages:

- it permits the inclusion of "hot" query columns without redundancy,
- it provides an important check for creating schema mappings, and
- it promotes healthy schema evolution.

A *category* (in the mathematical sense) is roughly a graph with one additional bit of expressive power: the ability to declare two paths equivalent. We now have the desired connection between database schemas and categories: Tables in a schema are specified by vertices (or as we have drawn them in diagram (1), by boxes); columns are specified by arrows; and