Under consideration for publication in Math. Struct. in Comp. Science

Database queries and constraints via lifting problems

David I. Spivak[†]

Massachusetts Institute of Technology
Department of Mathematics
Cambridge, MA 02139
dspivak@mit.edu

Received 12 July 2012; Revised 18 June 2013

Previous work has demonstrated that categories are useful and expressive models for databases. In the present paper we build on that model, showing that certain queries and constraints correspond to lifting problems, as found in modern approaches to algebraic topology. In our formulation, each SPARQL graph pattern query corresponds to a category-theoretic lifting problem, whereby the set of solutions to the query is precisely the set of lifts. We interpret constraints within the same formalism and then investigate some basic properties of queries and constraints. In particular, to any database π we can associate a certain derived database $\mathbf{Qry}(\pi)$ of queries on π. As an application, we explain how giving users access to certain parts of $\mathbf{Qry}(\pi)$, rather than direct access to π, improves ones ability to manage the impact of schema evolution.

Contents

1. Introduction

In (Diskin and Kadish 1994), (Johnson 2001), (Johnson et al. 2002), and many others, a tight connection between database schemas and the category-theoretic notion of sketches was presented and investigated. This connection was carried further in (Spivak 2012)

[†] This project was supported by ONR grant N000141010841.

where the existence of three data migration functors was shown to follow as a simple consequence of using categories rather than sketches to model schemas. In this paper we shall show that a modern approach to the study of algebraic topology, the *lifting problem* approach (see (Quillen 1967)), provides an excellent model for typical queries and constraints (see (Prud'hommeaux et al. 2008)).

A database consists of a schema (a layout of tables in which *foreign key* columns connect one table to another) and an instance (the rows of actual data conforming to the chosen layout). One can picture the analogy between databases and topological spaces as follows. Imagine that a collection of data I and a schema S are each an abstract space, and suppose we have a projection from I to S. That is, we have some kind of continuous map $\pi\colon I \to S$ from a "data bundle" I to a "base space" S. Points in S represent tables, and paths in S represent foreign key columns (or iterates thereof), which point from one table to another. Over every point $s \in S$ in the base space, we can look at the corresponding fiber $\pi^{-1}(s) \subseteq I$ of the data bundle; this will correspond to the set of rows in table s. The map π, associating data with schema, is called a *database instance*.

A *query* on a database instance $\pi\colon I \to S$ is like a system of equations: it includes an organized collection of knowns and unknowns. In our model a query takes the form of a functor $m\colon W \to R$, such that W (standing for WHERE-clause) corresponds to the set of knowns, each of which maps to a specific value in the data bundle I, and such that the relationship between knowns and unknowns is captured in a schema R. More precisely, a query on the database instance $\pi\colon I \to S$ is presented as a commutative diagram to the left, which would be roughly translated into the pseudo-SQL to the right,[†] in (1):

$$\begin{array}{ccc} W & \xrightarrow{p} & I \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle \pi} \\ R & \xrightarrow[n]{} & S. \end{array} \qquad \begin{array}{ll} \text{SELECT} & * \\ \text{FROM} & R \xrightarrow{n} S \\ \text{WHERE} & R \xleftarrow{m} W \xrightarrow{p} I \end{array} \tag{1}$$

A *result* to the query is any mapping $\ell\colon R \to I$ making both triangles commute ($\ell \circ m = p, \pi \circ \ell = n$) in the diagram

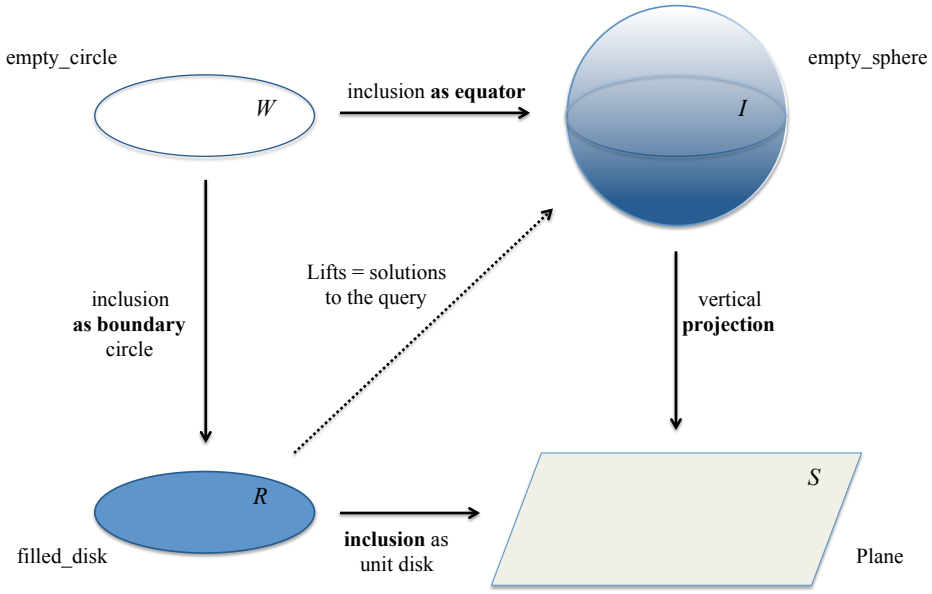
$$\begin{array}{ccc} W & \xrightarrow{p} & I \\ {\scriptstyle m}\downarrow & \overset{\ell}{\nearrow} & \downarrow{\scriptstyle \pi} \\ R & \xrightarrow[n]{} & S. \end{array} \tag{2}$$

The map ℓ is called a *lift* of Diagram 1, hence the term *lifting problem*. The idea is that a lift is a way to fill the result schema R with conforming data from the instance π.

We will now give a simple example from algebraic topology to strengthen the above image. By connecting databases and topology, we not only can visualize queries in a new way, but it is conceivable that algebraic topologists could use database interfaces to have computers work on lifting problems that arise in their research. Regardless, after the topological example, we will ground the discussion with an example database query.

[†] A more general SQL query, with a specific SELECT statement will be discussed in Example 4.3.2.

Fig. 3. A topological lifting problem



Consider an empty sphere, defined by the equation $x^2 + y^2 + z^2 = 1$; call it I. We project it down onto the (x, y)-coordinate plane ($z = 0$); call that plane $S \cong \mathbb{R}^2$. The sphere I serves as the database instance and the plane S serves as the schema. A query consists of some result schema mapping to the plane S, say a solid disk R (given by $z = 0, x^2 + y^2 \leq 1$), together with some constraints, say on the boundary circle W (given by $z = 0, x^2 + y^2 = 1$) of the disk. Graphically we have Figure 3.

The results of the lifting query from Figure 3 are the mappings $R \to I$ making the diagram commute. Under the guidance of (1) the query would look something like this:

```
SELECT ∗
FROM   filled_disk inclusion
WHERE  empty_circle as boundary = empty_circle as equator
```

Topologically one checks that there are exactly two lifts—the top hemisphere and the bottom hemisphere—so our pseudo-SQL query above would return exactly two results.

1.1. *Main example of a lifting query*

We now provide an example of a situation in which one may wish to query a database, and we show that this query naturally takes the structure of a lifting problem. We break a single example into three parts for clarity.

Example 1.1.1 (Main Example 1: Situation, SPARQL, and schema).

Suppose you have just come home from a party. There, you met and really hit it off with a married couple; the husband's name is Bob and the wife's name is Sue; they live in Cambridge. From your conversation, you know that Bob works at MIT and Sue works in the financial sector. You'd like to see them again, but you somehow forgot to ask for their contact information; in particular you'd like to know their last names.

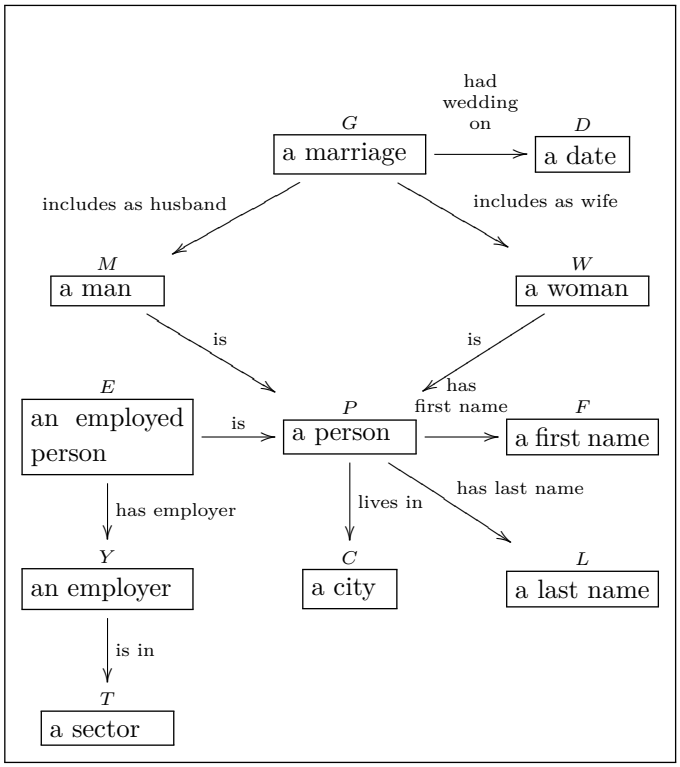
This is a typical database query problem. It can be phrased as the following SPARQL graph pattern query (which we arrange in two columns for space and readability reasons):

(?marriage includesAsHusband ?b)	(?marriage includesAsWife ?s)
(?b hasFirstName Bob)	(?s hasFirstName Sue)
(?b livesIn Cambridge)	(?s livesIn Cambridge)
(?employedb is ?b)	(?employeds is ?s)
(?employedb hasEmployer MIT)	(?employeds hasEmployer ?sueEmp)
	(?sueEmp isIn financial)
(?b hasLastName ?bobLast)	(?s hasLastName ?sueLast)

(4)

The query in (4) might be asked on the following database schema:[‡]

$S :=$



(5)

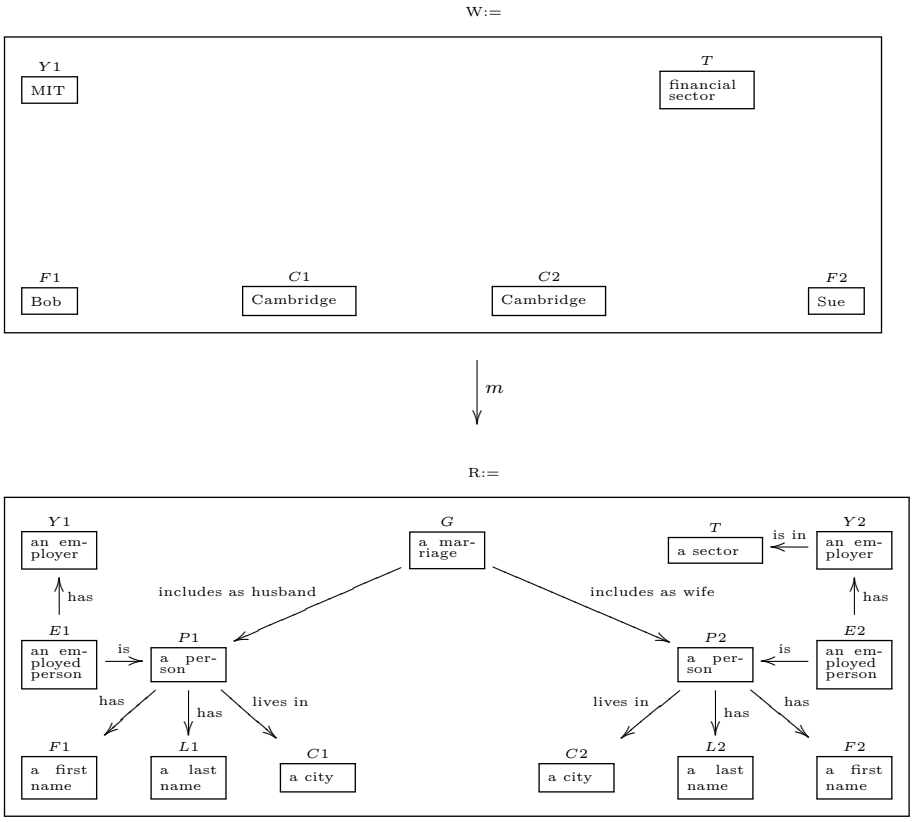
Given that S is instantiated with data $\pi\colon I \to S$, one can hope to find Bob and Sue,

[‡] The schema S in (5) deliberately includes a box D and an arrow $G \to D$ that are not part of our query (4).

and then determine their last name. In the following two examples (Examples 1.1.2 and 1.1.3) we will show that this query corresponds to a lifting problem for π.

Example 1.1.2 (Main Example 2: WHERE-clause and Result schema).
Recall the SPARQL query presented as (4) in Example 1.1.1, in which we wanted to find information about our new friends Bob and Sue. We will use a lifting problem to state this query; to do so we need to come up with a result schema R, a constraint schema (a set of knowns) W, and a mapping $m\colon W \to R$ embedding the known objects into the result schema. In this example we will present m, W, and R. In Example 1.1.3 we will explain the lifting diagram for the query and show the results.

In order to find our friends Bob and Sue, we will use the following mapping:



The functor $m\colon W \to R$ is indicated by sending each object in W to the object with the same label in R; e.g. ⌜MIT⌝ in $\mathrm{Ob}(W)$ is sent to ⌜an employer⌝ in $\mathrm{Ob}(R)$ because they are both labeled $Y1$.

To orient oneself, we suggest the following. Count the number of constants in the SPARQL query (4)—there are 6 (such as Bob, Cambridge, etc.); this is precisely the number of objects in W. Count the combined number of constants and variables in the SPARQL query—there are 14 (there are 8 variables, such as ?marriage, ?empoyedb, etc.); this is precisely the number of objects in R. Finally, count the number of triples in the

SPARQL query – there are 13; this is precisely the number of arrows in R. These facts are not coincidences.

Example 1.1.3 (Main Example 3: Lifting diagram and result set).

In Example 1.1.2 we showed a functor $m\colon W \to R$ corresponding to the SPARQL query stated in (4). In this example we will explain how this query can be formulated as a lifting problem of the form

$$\begin{array}{ccc} W & \xrightarrow{p} & I \\ {\scriptstyle m}\downarrow & \overset{\ell}{\nearrow} & \downarrow{\scriptstyle \pi} \\ R & \xrightarrow[n]{} & S \end{array} \tag{6}$$

which serves to pose our query to the database instance π. At this point we can ask for the set of solutions ℓ. So far, W, m, R, and S have been presented, I and π have been assumed, and the set of ℓ's is coming later, so it suffices to present p and n.

One should refer to our presentation of S in Example 1.1.1 (5). The functor $n\colon R \to S$ should be obvious from our labeling system (for example, the object E1=⌜an employed person⌝ in category R is mapped to the object E=⌜an employed person⌝ in category S). Note that, as applied to objects, n is neither injective nor surjective in this case: $n^{-1}(P) = \{P_1, P_2\}$ and $n^{-1}(D) = \emptyset$.

Suppose $\pi\colon I \to S$ is our data bundle, and assume that it contains enough data that the constants in the query have unique referents. [§] There is an obvious functor $p\colon W \to I$ that sends each object in category W to its referent in I. For example, we assume that there is an object in I labelled ⌜MIT⌝, which is mapped to by the object Y1=⌜MIT⌝ in W.

Thus our query from (4) is finally in the form of a lifting problem as in (6). We will show in Example 4.3.4, after we have built up the requisite theory, that the set of lifts can be collected into a single table, the most useful projection of which would look something like this:

Marriage								
ID	**Husband**				**Wife**			
	ID	**First**	**Last**	**City**	**ID**	**First**	**Last**	**City**
G3801	M881-36	Bob	Graf	Cambridge	W913-55	Sue	Graf	Cambridge

(7)

This concludes the tour of our main example: we have shown a typical query formulated as a lifting problem. The mathematical basis for the above ideas will be presented in Section 4.

1.2. *Relation to earlier work*

As mentioned above, there is a long history of applying category-theoretic formalism to database theory. For the purpose of our exposition here, we will divide these approaches

[§] This use of the term "query" is not standard. See Sections 1.5.1 and 4.2 for an explanation.

into two groupings. The first grouping, including (Tuijn and Gyssens 1992) and (Kato 1983), considers relational database tables as sets of attributes, using limits to discuss joins. This formulation is similar to that used in (Spivak 2009), in which simplicial sets were used as a geometric model for "sheaves of attributes". The second grouping, including (Diskin and Kadish 1994), (Johnson 2001), and (Johnson et al. 2002), uses *sketches* in the sense of (Ehresmann 1968). The latter approach is closer to that of (Spivak 2012) and of the present article. We now discuss the similarities and differences between the sketch approach and the lifting problems approach.

As we will go over in Section 2.1, we model database schemas as finite category presentations, whereas the above "second grouping" models them as sketches. A sketch is a category together with specified limit cones and colimit cones (see (Barr and Wells 2005)). Sketches are more expressive than categories; for example in the database context, using sketches allows a schema to convey when the set of rows in table T is the product of the sets of rows in tables U and V. This expressivity comes with some cost: whereas the categorical model in (Spivak 2012) has three built in data migration functors for moving data back and forth between schemas, the most general sketch model has only one, "pullback". If the modeler confines herself to the less-expressive limit sketches, a left adjoint to this pullback becomes available. The point is that with the capacity to express more in ones model, it seems her ability to transmit data to other models is reduced.

Still, it may be useful to find something in between sketches and bare categories, because using categories as models does not allow one to express constraints beyond foreign keys and commutative diagrams. For example it does not allow for injectivity, or "is a", constraints. It is here that the present paper fits in. Modern mathematical research, especially algebra and topology, has found surprisingly little use for sketches and sketch morphisms, given their naturality and simplicity. It is not clear why that is, especially given the success of the sketch model in applications (see (Barr and Wells 2005)); the future of category theory in mathematics may indeed make more use of sketches.

What we can say is that modern mathematical research has become deeply invested in categories and functors. Further, algebraic topology, which was the trailblazer for category theory, has for more than half a century found lifting problems to be a key tool for investigating abstract spaces. In this paper we make the connection between lifting problems and database theory. As mentioned above, we show that there are many constraints that are well-phrased as lifting problems, and that queries also fit nicely into this framework.

Sketches are often divided into three types: limit sketches, colimit sketches, and mixed sketches (i.e granting the architect the ability to impose limits, colimits, or both). Lifting constraints fully cover the expressivity of limit sketches and a bit more; see Section 3.5. In particular, lifting problems can enforce injectivity constraints, as shown in Example 3.3.4. However, colimit sketches can express things that lifting constraints cannot. For example, with colimit sketches one can express set-theoretic complements, and this cannot be done with lifting problems. The ability to enforce that one subset is the complement of another (also known as negation) comes with well-known problems such as domain dependence. However, there are certain real-world applications in which colimit sketches are simply unavoidable, e.g. something as simple as the data model for 3-packs of toothbrushes.

The takeaway from the present section is that lifting problems can express a different class of constraints than that expressible by sketches. We contend here that it is a valuable class of constraints, and that it corresponds quite well with SPARQL graph pattern queries. Our argument for the usefulness of lifting constraints is summarized in Section 3.3.

1.3. *Purpose of the paper*

The purpose of this paper is to:

— provide an efficient mathematical formulation of common database queries (modeling both SQL and SPARQL styles),
— attach a geometric image to database queries that can be useful in conceptualization, and
— explore theory and applications of the derived database schema $\mathbf{Qry}(\pi)$ of queries on a database instance π, and the derived instance of results.

We include several mathematical results that are well-known to experts, for the purpose of aiding those interested in using this paper to bridge the gap between database theory and category theory.

1.4. *Plan of the paper*

We begin in Section 2 with a review of the categorical approach to databases (see (Spivak 2012) for more details). Roughly this correspondence goes by the following slogan: "schemas are categories, instances are set-valued functors". In Section 2.3 we also discuss the Grothendieck construction, which will be crucial for our approach: a database instance can be converted into a *discrete opfibration*, which we will later use extensively to make the parallel with algebraic topology and lifting problems in particular.

In Section 3 we define constraints on a database in terms of lifting conditions and discuss some constraint implications. We give several examples to show how various common existence and uniqueness constraints (such as the constraint that a given foreign key column is surjective) can be framed in the language of lifting conditions. In Section 4 we discuss queries as lifting problems, and review the paper's main example. In Section 5, we show that the information in a given database instance can be collected into a new, derived database. This derived database of queries and their results can be queried, giving rise to nested queries. We explain how this formulation can be useful for managing the impact of schema evolution. Finally in Section 6 we briefly discuss some possible directions for future work, including tying in to Homotopy Type Theory (in the sense of (Awodey 2009) and (Voevodsky 2006)) and other projects.

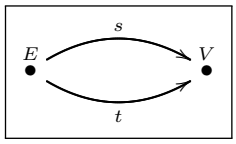
1.5. *Notation and terminology*

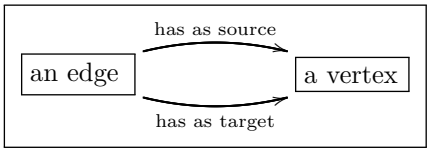
For any natural number $n \in \mathbb{N}$, let $\underline{n}$ denote the set $\{1, 2, \ldots, n\}$. We sometimes regard sets as discrete categories without mentioning it. Note that $\underline{0} = \emptyset$. Let $[n]$ denote the linear order $0 \leq 1 \leq \ldots \leq n$. We sometimes regard orders as categories without mentioning

that either. In particular $\underline{1}$ is the terminal category; it has one object and one morphism (the identity).

Given any category $\mathcal{C}$, we denote the category of all functors $\mathcal{C} \to \mathbf{Set}$ by $\mathcal{C}$–**Set**. The terminal object in $\mathcal{C}$–**Set** sends each object in $\mathcal{C}$ to $\underline{1}$; we denote it by $\underline{1}^{\mathcal{C}} \colon \mathcal{C} \to \mathbf{Set}$. For any category $\mathcal{C}$, there is a one-to-one correspondence between the objects in $\mathcal{C}$ and the functors $\underline{1} \to \mathcal{C}$. Thus we may denote an object $c \in \mathrm{Ob}(\mathcal{C})$ by a functor $\underline{1} \xrightarrow{c} \mathcal{C}$. In particular, we elide the difference between a set and a functor $\underline{1} \to \mathbf{Set}$.

We draw schemas in one of two ways. When trying to save space, we draw our objects as concisely-labeled nodes and our morphisms as concisely-labeled arrows; when trying to be more expressive, we draw our objects as text boxes and put as much text in them (and on each arrow) as is necessary to be clear (see (Spivak and Kent 2012)). For example, we might draw the indexing category for directed graphs in either of the following two ways:

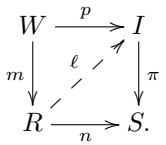




When in the typographical context of inline text we are discussing an object that has been elsewhere displayed as a textbox (such as an edge), we may represent it with corner symbols (e.g. as ⌜an edge⌝) to avoid various spacing issues that can arise.

Given two categories, there are generally many functors from one to the other; however, if the objects and arrows are labeled coherently, there are many fewer functors that roughly respect the labelings. We will usually be explicit when defining functors, but we will also take care that our functors respect labeling to the extent possible.

1.5.1. *"Queries on a database"* In wide-spread terminology for database queries, a query cannot depend on the current instance π of the database, but instead only on the schema S. This is perfectly reasonable for theoretical and practical reasons. Often in applications, however, one uses what is known as a *cursor*, which is basically a pre-defined query consisting of a join-graph and a set of variables to be bound at run-time. With respect to the diagram



the join-graph is R, the set of variables waiting to be bound is W, and the binding itself is $p \colon W \to I$. The mathematics will be covered more extensively in Section 4.2; in the remaining paragraphs of Section 1.5.1, we hope to get across how one might connect our use of the term "query" in the present paper to common ideas in database systems.

In applications, a query wizard may run the cursor in a 2-step query process: first it will query the database to offer the user a drop-down menu of choices in the active domain of each variable. The user will choose a row to which the variable will be bound (once for each variable). At this point the program will apply the actual query declared by the

cursor. This two step process corresponds to searching for possible functors $p\colon W \to I$ and then searching for lifts ℓ.

Throughout this article, when we speak of queries on a database, we mean queries for which the constant variables have been bound to elements in the active domain of a given instance. However, as we will see in Section 4.2, one can also use the same machinery in cursor-like fashion to pose queries in which variable values have been chosen without regard for whether or not they are in the active domain. In other words, we will see that what can be accomplished by queries in the sense of traditional relational database theory fits easily into our framework. Because it works either way, we thought that the unusual terminology "queries on a database" would be best because it neither lulls the reader into thinking that these gadgets are completely instance-independent, nor frightens the reader into thinking that the instance must be known in advance for the ideas here to work.

1.6. *Acknowledgments*

I would like to express my appreciation to Peter Gates, as well as to Henrik Forssell, Rich Haney, Eric Prud'hommeaux, and Emily Riehl for many useful discussions.

2. Elementary theory of categorical databases

2.1. *Review of the categorical description of databases*

The basic mantra is that a database schema is a small category S and an instance is a functor $\delta\colon S \to \mathbf{Set}$, where **Set** is the category of sets.[¶] To recall these ideas, we take liberally from (Spivak 2012), though more details and clarification are given there. Readers who are familiar with the basic setup and data migration functors can skip to Section 2.3.

In (Spivak 2012) a category **Sch** of categorical schemas and translations is defined and an equivalence of categories

$$\mathbf{Sch} \simeq \mathbf{Cat} \tag{8}$$

is proved, where **Cat** is the category of small categories. The difference between **Sch** and **Cat** is that an object of the former is a *chosen presentation* of a category, by generators and relations, as described below. Given the equivalence (8), we can and do elide the difference between schemas and small categories.

Roughly, a schema S consists of a graph G together with an equivalence relation on the set of paths of G. Each object $s \in \mathrm{Ob}(S)$ represents a table (or more precisely the ID column of a table), and each arrow $s \to t$ emanating from s represents a column of table s, taking values in the ID column of table t. An example should clarify the ideas.

[¶] If one prefers, **Set** can be replaced by the category of finite sets or by the category Types for some λ-calculus.

Example 2.1.1.

As a typical database example, consider the bookkeeping necessary to run a department store. We keep track of a set of employees and a set of departments. For each employee e, we keep track of

E.1 the **first** name of e, which is a `FirstNameString`,
E.2 the **last** name of e, which is a `LastNameString`,
E.3 the **manager** of e, which is an `Employee`, and
E.4 the department that e **works in**, which is a `Department`.

For each department d, we keep track of

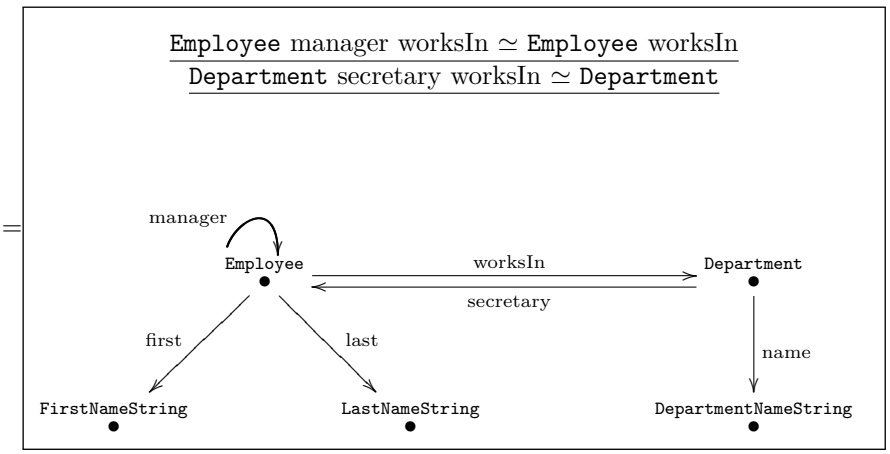
D.1 the **name** of d, which is a `DepartmentNameString`, and
D.2 the **secretary** of d, which is an `Employee`.

Suppose further that we make the following two rules.

Rule 1 For every employee e, the **manager** of e **works in** the same department that e **works in**.
Rule 2 For every department d, the **secretary** of d **works in** department d.

This is all captured neatly, with nothing left out and nothing else added, by the category presented below:

$S :=$



(9)

The underlined statements at the top indicate pairs of commutative (i.e. equivalent) paths; each path is indicated by its source object followed by the sequence of arrows that composes it. The objects, arrows, and equivalences in S correspond to the tables, columns, and rules laid out at the beginning of this example.

The collection of data on a schema is typically presented in table form. Display (10) shows how a database with schema S might look at a particular moment in time.

Employee				
ID	**first**	**last**	**manager**	**worksIn**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
ID	**name**	**secretary**
q10	Sales	101
x02	Production	102

(10)

FirstNameString
ID
Alan
Alice
Bertrand
Carl
David
⋮

LastNameString
ID
Arden
Hilbert
Jones
Russell
Turing
⋮

DepartmentNameString
ID
Marketing
Production
Sales
⋮

Every table has an ID column, and in every table each cell references a cell in the ID column of some table. For example, cells in the secretary column of the `Department` table refer to cells in the ID column of the `Employee` table. Finally, one checks that Rule 1 and Rule 2 hold. For example, let e be Employee 101. He works in Department q10 and his manager is Employee 103. Employee 103 works in Department q10 as well, as required. The point is that the data in (10) conform precisely to the schema S from Diagram (9).

A set of tables that conforms to a schema is called an *instance* of that schema. Let us denote the set of tables from (10) by δ; we noted above that δ conforms with, thus is an instance of, schema S. Mathematically, δ can be modeled as a functor

$$\delta\colon S \to \mathbf{Set}.$$

To each object $s \in S$ the instance δ assigns a set of row-IDs, and to each arrow $f\colon s \to t$ in S it assigns a function, as specified by the cells in the f-column of s.

2.2. *Review of data migration functors*

Once we realize that a database schema can be captured simply as a category S and each instance on S as a set-valued functor $\delta\colon S \to \mathbf{Set}$, classical category theory gives ready-made tools for migrating data between different schemas. The first definition we need is that of schema mapping.

Definition 2.2.1.
Let S and T be schemas (i.e. small categories). A *schema mapping* is a functor $F\colon S \to T$.

Thus a schema mapping assigns to each table in S a table in T, to each column in S a column in the corresponding table of T, and all this in such a way that the path equivalence relation is preserved.

Definition 2.2.2.
A schema mapping $F\colon S \to T$ induces three functors on instance categories, which we call *the data migration functors associated to* F and which we denote by Σ_F, Δ_F, and

Π_F, displayed here:

$$S\text{–}\mathbf{Set} \underset{\Pi_F}{\overset{\Sigma_F}{\leftarrow\!\!\Delta_F\!\!-}} T\text{–}\mathbf{Set}.$$

The functor $\Delta_F\colon T\text{–}\mathbf{Set} \to S\text{–}\mathbf{Set}$ sends an instance $\delta\colon T \to \mathbf{Set}$ to the instance $\delta \circ F\colon S \to \mathbf{Set}$. The functor Σ_F is the left adjoint to Δ_F, and the functor Π_F is the right adjoint to Δ_F. We call Δ_F the *pullback along* F , we call Σ_F the *left pushforward along* F, and we call Π_F the *right pushforward along* F.

The functors Δ_F, Σ_F, and Π_F are well-known in category theory literature, where the latter two are often referred to as the *left Kan extension along* F and the *right Kan extension along* F (see (Mac Lane 1988, Chapter X)). In databases, the left pushforward Σ_F will generally correspond to unions and quotients, and the right pushforward Π_F will generally correspond to products and joins. We explore these ideas a bit further in Section 5; see (Spivak 2012) for further explanation.

2.3. *RDF via the Grothendieck construction*

There is a well-known construction that associates to a functor $\delta\colon S \to \mathbf{Set}$, a pair $(\int(\delta), \pi_\delta)$ where $\int(\delta) \in \mathbf{Cat}$ is a new category, called *the category of elements of* δ, and $\pi_\delta\colon \int(\delta) \to S$ is a functor. The pair $(\int(\delta), \pi_\delta)$ is often called the *Grothendieck construction* of δ. The objects and morphisms of $\int(\delta)$ are given as follows

$$\mathrm{Ob}(\textstyle\int(\delta)) := \left\{(s, x) \mid s \in \mathrm{Ob}(S), x \in \delta(s)\right\}$$
$$\mathrm{Hom}_{\int(\delta)}((s, x), (s', x')) := \left\{f\colon s \to s' \mid \delta(f)(x) = x'\right\}$$

The functor $\pi_\delta\colon \int(\delta) \to S$ is straightforward: it sends the object (s, x) to s and sends the morphism $f\colon (s, x) \to (s', x')$ to $f\colon s \to s'$.

We call the pair $(\int(\delta), \pi_\delta)$ the *discrete opfibration associated to* δ. We will see in the next section (Definition 3.2.1) that π_δ is indeed a kind of fibration of categories. This construction, and in particular the category $\int(\delta)$, is also nicely connected with the *resource descriptive framework* (see (Prud'hommeaux et al. 2008)), in which data is captured in *RDF triples*. Indeed, the arrows $\bullet \overset{s}{\longrightarrow} \overset{p}{\bullet} \overset{b}{\longrightarrow} \bullet$ of $\int(\delta)$ correspond one-for-one with these RDF triples (**s**ubject, **p**redicate, o**b**ject). Thus we have shown a readymade conversion from relational databases to RDF triple stores via the Grothendieck construction. An example should clarify this discussion.

Example 2.3.1.

Recall the database instance $\delta\colon S \to \mathbf{Set}$ given by the tables in Diagram (10), whose schema S was presented as Diagram (9). Applying the Grothendieck construction to $\delta\colon S \to \mathbf{Set}$, we get a category $I := \int(\delta)$ and a functor $\pi := \pi_\delta$ as in Figure 11.

In the Introduction (Section 1), we discussed database instances in terms of mappings

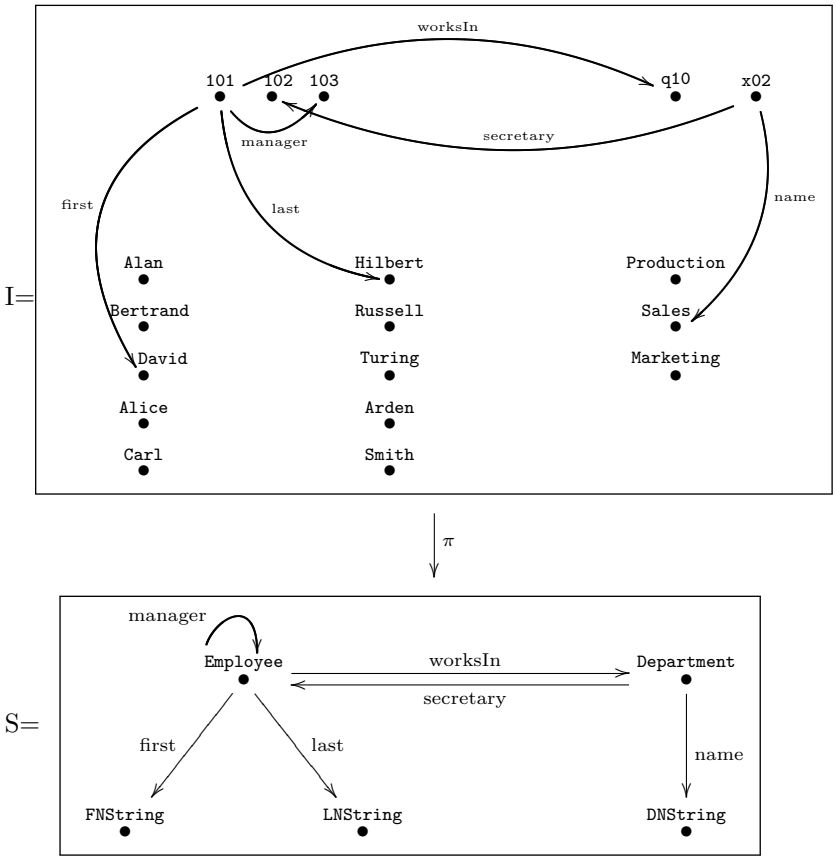


Fig. 11. An example of the Grothendieck construction, or category of elements, of a functor $\delta\colon S \to \mathbf{Set}$.. The functor $\pi\colon I \to S$ sends objects 101,102,103 in I to the object `Employee`, in S; it similarly sends the arrow labeled worksIn in $\int(\delta)$ to the arrow labeled worksIn in S, etc. In the tables in (10), which represents our instance δ, there are 16 non-ID cells, whereas in $I = \int(\delta)$, there are only six arrows drawn. The other ten arrows have been left out of the picture of I (e.g. the arrow $\overset{\texttt{102}}{\bullet} \xrightarrow{\text{Last}} \overset{\texttt{Russell}}{\bullet}$ is not drawn) for readability reasons. The point is that the RDF triple store associated to instance δ is nicely represented using the standard Grothendieck construction. For example, the arrow $\overset{\texttt{101}}{\bullet} \xrightarrow{\text{first}} \overset{\texttt{David}}{\bullet}$ represents the RDF triple (101 :first David).

π, each from a data bundle I to a base space S. We were referring to exactly the discrete opfibration picture in Figure 11.

In Section 3.2 we will give a definition of discrete opfibrations in terms of lifting constraints (Definition 3.2.1). First, however, we attempt to understand a discrete opfibration $\pi\colon I \to S$ by considering its various fibers and their relationships. More precisely, given an object $s \in \mathrm{Ob}(S)$, we consider the fiber $\pi^{-1}(s)$, and given a morphism $f\colon s \to s'$ in S we consider how the fibers $\pi^{-1}(s)$ and $\pi^{-1}(s')$ relate.

If $\pi\colon I \to S$ were not assumed to be a discrete opfibration but instead just a general functor, then all we would know about these various fibers would be that they are categories. But the first distinctive feature of a discrete opfibration is that the fiber $\pi^{-1}(s)$ is a *discrete category*, i.e. a set, for each object $s \in S$; that is, there are no morphisms between different objects in a chosen fiber (see Proposition 3.2.2). The pre-image $\pi^{-1}(f)$ of $f\colon s \to s'$ is a set of morphisms from objects in $\pi^{-1}(s)$ to objects in $\pi^{-1}(s')$. When π is a discrete opfibration, there exists a unique morphism in $\pi^{-1}(f)$ emanating from each object in $\pi^{-1}(s)$, so the subcategory $\pi^{-1}(f) \subseteq I$ can be cast as a single function $\pi^{-1}(f)\colon \pi^{-1}(s) \to \pi^{-1}(s')$.

To recap, the discrete opfibration $\pi_\delta\colon \int(\delta) \to S$ of a set-valued functor $\delta\colon S \to \mathbf{Set}$ contains the same information as δ does, but a different perspective. We have

$$\pi_\delta^{-1}(s) \cong \delta(s) \qquad \text{and} \qquad \pi_\delta^{-1}(f) \cong \delta(f),$$

for any $s, s' \in \mathrm{Ob}(S)$ and $f\colon s \to s'$.

2.3.2. *Basic behavior of the Grothendieck construction* Below are some simple results about the Grothendieck construction, all of which are well-known.

Proposition 2.3.3.
Let $\delta\colon S \to \mathbf{Set}$ be a functor. Then the Grothendieck construction $\int(\delta) \xrightarrow{\pi_\delta} S$ of δ can be described as a pullback in the diagram of categories

$$\begin{array}{ccc} \int(\delta) & \longrightarrow & \mathbf{Set}_* \\ {\scriptstyle \pi_\delta}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi} \\ S & \xrightarrow[\delta]{} & \mathbf{Set}, \end{array}$$

where $\mathbf{Set}_*$ is the category of pointed sets and π is the functor that sends a pointed set $(X, x \in X)$ to its underlying set X.

Proof.
This follows directly from definitions. □

Lemma 2.3.4.
Let S be a category. The functor $\int\colon S\text{–}\mathbf{Set} \to \mathbf{Cat}_{/S}$ is fully faithful. That is, given two instances, $\delta, \epsilon\colon S \to \mathbf{Set}$, there is a natural bijection,

$$\mathrm{Hom}_{S\text{–}\mathbf{Set}}(\delta, \epsilon) \xrightarrow{\cong} \mathrm{Hom}_{\mathbf{Cat}_{/S}}(\textstyle\int(\delta), \int(\epsilon)).$$

Proof.
This follows directly from definitions. □

Proposition 2.3.5.
Let $F\colon S \to T$ be a functor. Let $\delta\colon S \to \mathbf{Set}$ and $\epsilon\colon T \to \mathbf{Set}$ be instances, and suppose

we have a commutative diagram

$$\begin{array}{ccc} \int(\delta) & \longrightarrow & \int(\epsilon) \\ {\scriptstyle \pi_\delta}\downarrow & & \downarrow{\scriptstyle \pi_\epsilon} \\ S & \xrightarrow[F]{} & T. \end{array} \tag{12}$$

Then diagram (12) is a pullback, i.e. $\int(\delta) \cong S \times_T \int(\epsilon)$, if and only if $\delta \cong \Delta_F \epsilon$.

Proof.
This is checked easily by comparing the set of objects and the set of morphisms in $\int(\delta)$ with the respective sets in $S \times_T \int(\epsilon)$.

□

2.3.6. *Examples from algebraic topology* In algebraic topology (see (May 1999)), one associates to every topological space X a fundamental groupoid $Gpd(X)$. It is a category whose objects are the points of X and whose set of morphisms between two objects is the set of (equivalence classes of) continuous paths in X from one point to the other. Two paths in X are considered equivalent if one can be deformed to the other (without any part of it leaving X). Composition of morphisms is given by concatenation of paths.

One can reduce some of the study of a space X to the study of this algebraic object $G = Gpd(X)$, and the latter is well-suited for translation to the language of this paper.

Example 2.3.7.
Suppose that G is a groupoid. Then a covering of groupoids in the sense of (May 1999, Section 4.3) is precisely the same as a surjective discrete opfibration with schema G.

Let $G = Gpd(S^1)$ denote the fundamental groupoid of the circle with circumference 1. Explicitly we have $\mathrm{Ob}(G) = \{\theta \in \mathbb{R}\}/\!\sim$, where $\theta \sim \theta'$ if $\theta - \theta' \in \mathbb{Z}$; and we have

$$\mathrm{Hom}_G(\theta, \theta') = \{x \in \mathbb{R} \mid x + \theta \sim \theta'\}.$$

Think of G as the category whose objects are positions of a clock hand and whose morphisms are arbitrary durations of time (rotating the hands from one clock position around and around to another). Consider the functor $T\colon G \to \mathbf{Set}$ such that $T(\theta) = \{t \in \mathbb{R} \mid t - \theta \in \mathbb{Z}\}$ and such that for $x \in \mathrm{Hom}_G(\theta, \theta')$ we put $T(x)(t) = x + t$. So, for a clock position θ, the functor T returns all points in time at which the clock is in position θ.

Applying the Grothendieck construction to T, we get a covering $\pi\colon \int(T) \to G$, which corresponds to the universal cover of the circle S^1. One can think of it as a helix (modeling the time line) mapping down to the circle (modeling the clock).

A much more sophisticated example relating databases to classical questions in algebraic topology may be found in (Morava 2012).

3. Constraints via lifting conditions

In this section we introduce the lifting problem approach to database constraints. Roughly the same model will apply in the next section to database queries, the idea being that a lifting constraint is a lifting query that is guaranteed to have a result.

3.1. *Basic definitions*

Definition 3.1.1.

Let $S \in \mathbf{Cat}$ be a database schema. A *(lifting) constraint on* S is a pair (m, n) of functors

$$W \xrightarrow{m} R \xrightarrow{n} S.$$

A functor $\pi\colon I \to S$ is said to *satisfy the constraint* (m, n) if, for all solid arrow commutative diagrams of the form

$$\begin{array}{ccc} W & \longrightarrow & I \\ {\scriptstyle m}\downarrow & \nearrow & \downarrow{\scriptstyle \pi} \\ R & \xrightarrow[n]{} & S, \end{array} \tag{13}$$

there exists a dotted arrow lift making the diagram commute.

A *(lifting) constraint set* is a set $\xi := \{W_\alpha \xrightarrow{m_\alpha} R_\alpha \xrightarrow{n_\alpha} S \mid \alpha \in A\}$, for some set A. A functor $\pi\colon I \to S$ is said to *satisfy the constraint set* ξ if it satisfies each constraint (m_α, n_α) in ξ.

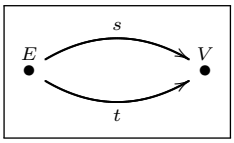
Given a constraint set ξ on S, we say that a constraint $W \xrightarrow{m} R \xrightarrow{n} S$ is *implied by* ξ if, whenever a functor $\pi\colon I \to S$ satisfies ξ it also satisfies (m, n).

Remark 3.1.2.

While not all constraints on databases are lifting constraints (for example, declaring a table to be the union of two others is not expressible by a lifting constraint), lifting constraints are the only type of constraint we will be considering in this paper. For that reason, we often leave off the word "lifting," as suggested by the parentheses in Definition 3.1.1.

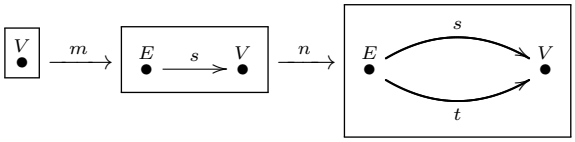
Example 3.1.3.

Consider the schema

$\mathcal{G} =$ 

The category $\mathcal{G}$–**Set** is precisely the category of (directed) graphs. Given a graph $X\colon \mathcal{G} \to \mathbf{Set}$, we have a function $X(s)\colon X(E) \to X(V)$ assigning to every edge its source vertex. Suppose we want to declare this function to be surjective, meaning that every vertex in

X is the source of some edge. We can do that with the following lifting constraint



where m and n respect labeling. A graph $\delta\colon \mathcal{G} \to \mathbf{Set}$ has the desired property, that every vertex is a source, iff $\int(\delta)$ satisfies the lifting constraint (m, n).

Definition 3.1.4.
Let $S \in \mathbf{Cat}$ be a schema. Given a functor $m\colon W \to R$, define a set $\langle m \rangle$ of lifting constraints as follows:

$$\langle m \rangle = \left\{ W \xrightarrow{m} R \xrightarrow{n} S \mid n \in \mathrm{Hom}_{\mathbf{Cat}}(R, S) \right\}.$$

(Note that there is a bijection $\langle m \rangle \cong \mathrm{Hom}_{\mathbf{Cat}}(R, S)$, but the form of the set $\langle m \rangle$ allows us to apply Definition 3.1.1.) Given a set of functors $M = \{m_j\colon W_j \to R_j \mid j \in J\}$, the union

$$\langle M \rangle := \bigcup_{j \in J} \langle m_j \rangle$$

is a constraint set, which we call the *universal constraint set generated by* M. A functor $\pi\colon I \to S$ satisfying the constraint set $\langle M \rangle$ is called an M-fibration. We say that elements of M are *generating constraints* for M-fibrations.

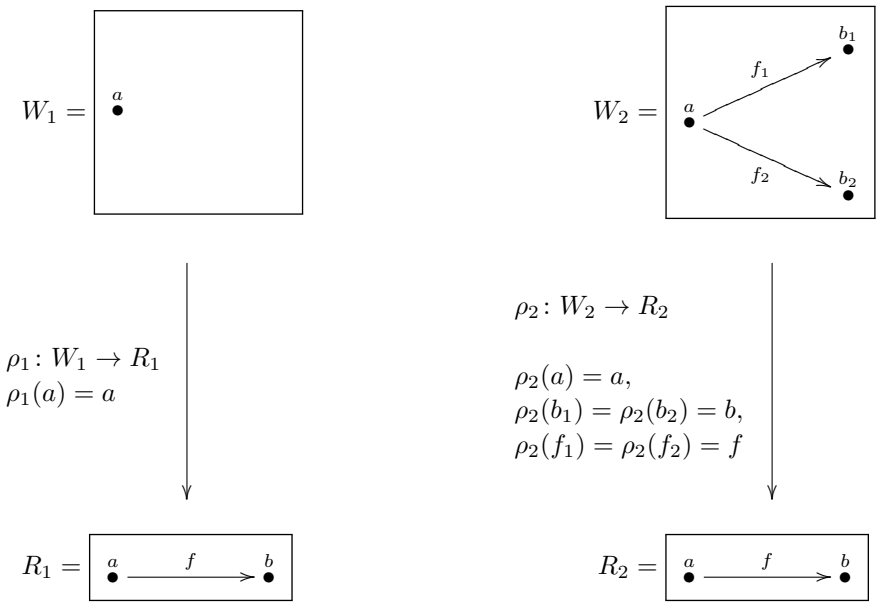
Remark 3.1.5.
Universal constraint sets seem to be more important in traditional mathematical contexts than in "informational" or database contexts. For example, in the world of simplicial sets, the Kan fibrations are M-fibrations for some universal constraint set $\langle M \rangle$, called *the set of generating acyclic cofibrations* (see (Hirschhorn 2003)).

3.2. *Discrete opfibrations via lifting constraints*

Our goal now is to express the notion of discrete opfibrations in terms of lifting constraints. In other words, we will exhibit a finite set of functors $\{m_\alpha\colon W_\alpha \to R_\alpha\}_{\alpha \in A}$ that serve to "check" whether an arbitrary functor $\pi\colon I \to S$ is a discrete opfibration. In fact, Definition 3.2.1 will define π to be a discrete opfibration if and only if it is a $\{\rho_1, \rho_2\}$-fibration, where $\rho_1\colon W_1 \to R_1$ and $\rho_2\colon W_2 \to R_2$ are functors displayed in Figure 14.

Definition 3.2.1.
Let I and S be categories and let $\pi\colon I \to S$ be a functor. Then I is a *discrete opfibration* if it satisfies the lifting constraints ρ_1 and ρ_2 from Figure 14. That is, for any pair of horizontal maps $W_1 \to I$ and $R_1 \to S$ (respectively for any pair of horizontal maps

Fig. 14. The generating constraints, ρ_1 and ρ_2, for discrete opfibrations



$W_2 \to I$ and $R_2 \to S$)

there exists a dotted arrow functor, as shown, such that the full diagram commutes.

Let $\pi : I \to S$ be a $\{\rho_1, \rho_2\}$-fibration. Then for any functor $R_1 = R_2 \to S$, i.e. for any arrow $f : s \to s'$ in S, we have two lifting conditions. A good way to understand the conditions of Definition 3.2.1 is that for any object $x \in \pi^{-1}(s)$ in the fiber over s,

1 there exists at least one arrow in I, emanating from x, whose image under π is f, and
2 there exists at most one arrow in I, emanating from x, whose image under π is f.

In the remainder of this section we give some consequences of Definition 3.2.1.

Proposition 3.2.2.
Let $\pi : I \to S$ be a discrete opfibration. Then for each object $s \in \mathrm{Ob}(S)$ the fiber $\pi^{-1}(s)$ is a discrete category.

Proof.
Let $s \in \mathrm{Ob}(S)$ be an object, and let $g : x \to y$ be a morphism in the fiber $\pi^{-1}(s) \subseteq I$; we will show that $x = y$ and that $g = \mathrm{id}_x$ is the identity morphism. Consider the map $\rho_2 : W_2 \to R_2$ from Figure 14. Let $n : R_2 \to S$ be the functor sending f to id_s. Let

$p\colon W_2 \to I$ send f_1 to id_x and send f_2 to g. We have a lifting diagram as in Definition 3.2.1, so a lift is guaranteed. This lift equates id_x and g.

□

Proposition 3.2.3.

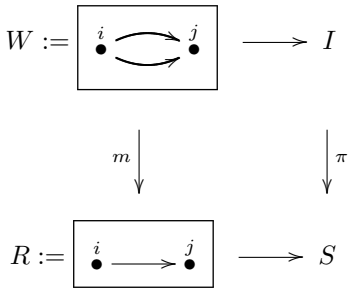
Let $\pi\colon I \to S$ be a discrete opfibration. Then π is faithful. In other words, for any two objects $i, j \in \mathrm{Ob}(I)$ the function

$$\pi\colon \mathrm{Hom}_I(i,j) \to \mathrm{Hom}_S(\pi(i), \pi(j))$$

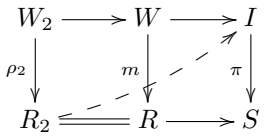
is injective.

Proof.

To prove that π is faithful, we need only find a solution to each lifting diagram of the form:



We can extend this diagram on the left with either surjective map from the relational constraint functor ρ_2 (see Figure 14) to m, as indicated in the diagram:



The result follows by noticing that the left-hand square is a pushout.

□

Let S be a category. We define a functor $\partial : \mathbf{Cat}_{/S} \to S\text{–}\mathbf{Set}$ as follows. For any $F\colon X \to S$, let $\underline{1}^X\colon X \to \mathbf{Set}$ denote the terminal object of X–**Set** (see Notation 1.5), and note that $\int(\underline{1}^X) \cong X$ in $\mathbf{Cat}_{/X}$. Define $\partial(F)\colon S \to \mathbf{Set}$ as

$$\partial(F) := \Sigma_F(\underline{1}^X).$$

We have the following proposition, which is well-known.

Proposition 3.2.4.

(i) The functor ∂ is left adjoint to $\int$:

$$\mathbf{Cat}_{/S} \underset{\int}{\overset{\partial}{\rightleftarrows}} S\text{–}\mathbf{Set}.$$

(ii) For any $\gamma\colon S \to \mathbf{Set}$ the counit map is an isomorphism

$$\partial \circ \textstyle\int(\gamma) \xrightarrow{\cong} \gamma.$$

(iii) An object $X \xrightarrow{F} S$ in $\mathbf{Cat}_{/S}$ is a discrete opfibration if and only if $F \cong \int \partial\,(F)$ in $\mathbf{Cat}_{/S}$.

Proof.

Let $F\colon X \to S$ be an object of $\mathbf{Cat}_{/S}$ and let $\gamma\colon S \to \mathbf{Set}$ be an object of S–$\mathbf{Set}$. By Proposition 2.3.5 we have a pullback diagram:

$$\begin{array}{ccc} \textstyle\int(\Delta_F\gamma) & \longrightarrow & \textstyle\int(\gamma) \\ \downarrow & \lrcorner & \downarrow \\ X & \xrightarrow[F]{} & S \end{array}$$

which implies the first isomorphism in the following chain:

$$\begin{aligned} \mathrm{Hom}_{\mathbf{Cat}_{/S}}(F, \textstyle\int(\gamma)) &\cong \mathrm{Hom}_{\mathbf{Cat}_{/X}}(\mathrm{id}_X, \textstyle\int(\Delta_F\gamma)) \\ &\cong \mathrm{Hom}_{X\text{–}\mathbf{Set}}(\underline{1}^X, \Delta_F\gamma) \\ &\cong \mathrm{Hom}_{S\text{–}\mathbf{Set}}(\Sigma_F(\underline{1}^X), \gamma) = \mathrm{Hom}_{S\text{–}\mathbf{Set}}(\partial\, F, \gamma). \end{aligned}$$

The second isomorphism follows from Lemma 2.3.4 and the third is adjointness; this proves Statement (i). Statement (ii) follows from the same lemma.

By construction, $\pi\colon \int(\delta) \to S$ is a discrete opfibration for any $\delta\colon S \to \mathbf{Set}$, so if $X \xrightarrow{F} S$ is not a discrete opfibration then $X \ncong \int \partial\,(F)$. Thus, it remains to show that if F is a discrete opfibration then $X \cong \int \partial\,(F)$. To see this, notice that for each $s \in \mathrm{Ob}(S)$ the set $F^{-1}(s)$ is final in $(F \downarrow s)$, so

$$\partial\,(F)(s) = \Sigma_F(\underline{1}^X)(s) = \operatorname*{colim}_{(F\downarrow s)} \underline{1}^X \cong F^{-1}(s).$$

This shows that the object structure in F is the same as that in $\int \partial\,(F)$. Similar analyses can be carried out for arrows and path equivalences.

□

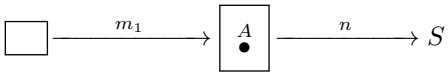
3.3. *Examples*

In this section we will show how to use lifting constraints (see Definition 3.1.1) to declare a number of different properties for tables and foreign keys in a database. Our examples include

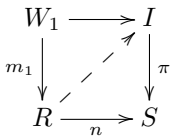
— declaring a table to be non-empty,
— declaring a table to have exactly one row,
— declaring a foreign key to be injective,
— declaring a foreign key to be surjective,
— declaring a binary relation to be reflexive, symmetric, and/or transitive,
— declaring a table to be a product (or more generally a limit) of other tables, and
— declaring that there are no nontrivial cycles in the data on a self-referencing table.

We will discuss these in the above order.

Example 3.3.1 (Nonempty).
Let S be a schema, and let $T \in \mathrm{Ob}(S)$ be a table, which we want to declare non-empty. We use the constraint drawn as follows



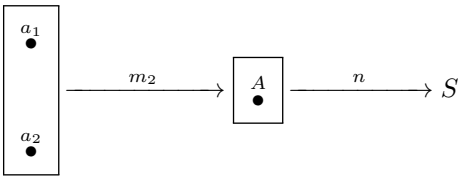
where $n(A) = T$. In other words, we set $W_1 = \emptyset$ to be the empty category, and we set $R = \{A\}$ to be the discrete category with one object, A. To say that the lifting problem



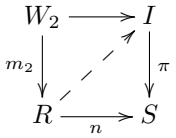
has a solution is to say that there exists an object in the instance category I whose image under π is T. In other words, there exists a row in table T. Here, the commutativity of the upper-left triangle does nothing, and the commutativity of the lower-right triangle does all the work.

Example 3.3.2 (Cardinality=1).
Let S be a schema, and $T \in \mathrm{Ob}(S)$ a table, which we want to declare to have exactly one row. We know a constraint guaranteeing the existence of a row in T from Example 3.3.1; in Section 3.4 we will give a general method for transforming existence constraints into uniqueness constraints, but here we will just give the result of that method.

To declare T to have at most one row, we use the constraint drawn as follows:



where $m_2(a_1) = m_2(a_2) = A$ and where $n(A) = T$. In other words, we set $W_2 = \{a_1, a_2\}$ to be a discrete category with two objects, and we set $R = \{A\}$ to be a discrete category with one object. The lifting problem

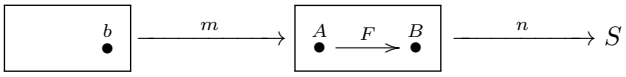


has a solution iff both triangles commute. We know already that the image of a and b in I consists of two rows in table T, because the square commutes. The commutativity of the upper-left triangle implies that a and b are the same, as desired. The commutativity of the lower-right triangle is implied by the surjectivity of m_2 and the commutativity of the square.

The set $\{(m_1, n), (m_2, n)\}$ is a constraint set on S that is satisfied by a discrete opfibration π if and only if the set $I(T)$ of rows in T has exactly one element.

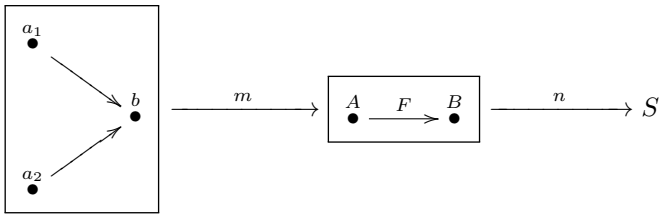
We will be more brief from here on out. The following constraint was used in Example 3.1.3.

Example 3.3.3 (Surjective foreign key).
The declaration that a foreign key $f\colon T \to T'$ be surjective is achieved by the constraint:



where $m(b) = B, n(A) = T, n(B) = T'$, and $n(F) = f$.

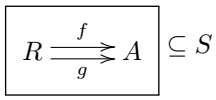
Example 3.3.4 (Injective foreign key).
The declaration that a foreign key $f\colon T \to T'$ be injective is achieved by the constraint:



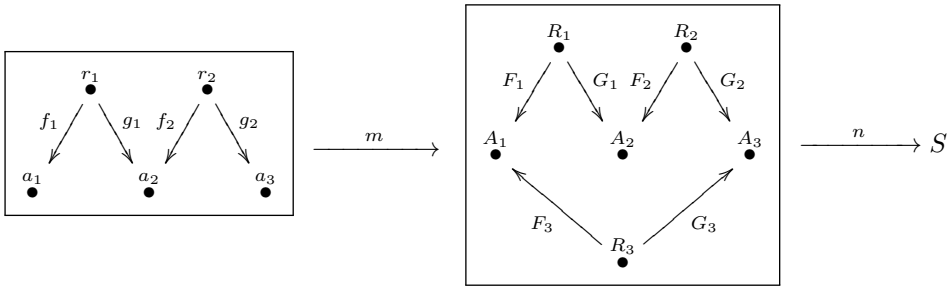
where $m(a_1) = m(a_2) = A$ and $m(b) = B$, and where $n(F) = f$.

There exist constraints that ensure a binary relation $R \subseteq A \times A$ is transitive, which we give in Example 3.3.5. There is another constraint to ensure it is symmetric, and another to ensure it is reflexive; we leave these as exercises.

Example 3.3.5 (Transitive binary relation).
The declaration that a relation



be transitive is achieved by the constraint

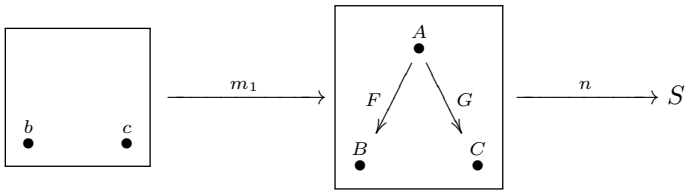


where the functors m and n should be clear by our labeling (e.g. $n(R_1) = n(R_2) = n(R_3) = R$).

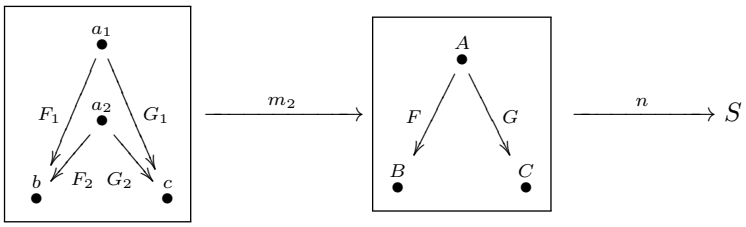
We now give the example of lifting constraints for products. This is part of a much larger story: In Section 3.5 we will show that any limit constraint can be modeled by lifting constraints.

Example 3.3.6 (Product).

Suppose we have a table T and two of its columns are $f\colon T \to U$ and $g\colon T \to V$. The declaration that (the set of rows in) table T is the product of (the sets of rows in) tables U and V is achieved by two constraints, an existence constraint and a uniqueness constraint. The existence constraint is



where $m_1(b) = B, m_1(c) = C$, and $n(F) = f, n(G) = g$. The uniqueness constraint is



where $m_2(F_1) = m_2(F_2) = F, m_2(G_1) = m_2(G_2) = G$, and $n(F) = f, n(G) = g$.

Thus the constraint set for (T, f, g) to be a product is $\{(m_1, n), (m_2, n)\}$.

Example 3.3.7 (Forests).

Let S be the free category generated by the graph with one object and one arrow, pictured here:

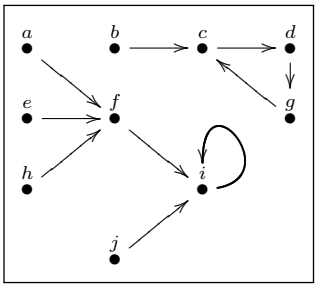
$$S := \quad \nu \circlearrowright p \tag{15}$$

This is just a self-referencing table. In mathematics, an instance $\delta\colon S \to \mathbf{Set}$ of such a self-referencing table is called a *discrete dynamical system* or *DDS*. The set $\delta(\nu)$ will be called the *set of nodes* of δ and given a node $x \in \delta(\nu)$, the node $\delta(p)(x)$ is called the *parent of* x. Here is a picture of a such an instance δ and its Grothendieck construction

$I = \int(\delta)$.

$\delta :=$

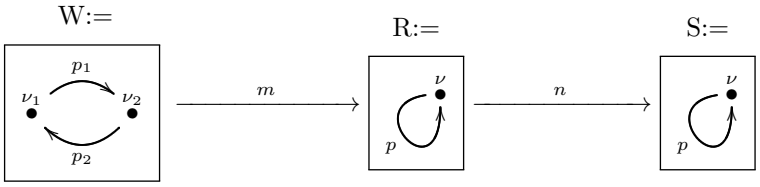
ν	
ID	**p**
a	f
b	c
c	d
d	g
e	f
f	i
g	c
h	f
i	i
j	i

$I :=$ 

(16)

Notice that a DDS looks like a forest (collection of trees) except that it may have cycles. These cycles can only occur at the root of a tree, and indeed each tree in the forest has a root cycle. In (16) we see that the tree containing a has a root cycle of length 1, and the tree containing b has a root cycle of length 3. Forests are a useful notion in computer science; we consider a DDS a forest if and only if each root cycle has length 1. This can be achieved by the following lifting constraint.

Let $R = S$ be the schema in (16), and let $n = \mathrm{id}\colon R \to S$. Let W be the free category on the graph below, and let $m\colon W \to R$ denote the functor sending p_1 and p_2 to p.



3.4. *Encoding uniqueness constraints*

Suppose given a constraint $W \xrightarrow{m} R \xrightarrow{n} S$. According to Definition 3.1.1 a functor $\pi\colon I \to S$ satisfies (m, n) if for every solid arrow diagram

$$\begin{array}{ccc} W & \longrightarrow & I \\ {\scriptstyle m}\downarrow & \nearrow & \downarrow{\scriptstyle \pi} \\ R & \xrightarrow[n]{} & S, \end{array} \tag{17}$$

there exists a dotted arrow lift making it commute. Thus it may appear that all lifting constraints are existence declarations. However, by employing a technique found in (Makkai 1997), we can always turn such an existence declaration into a uniqueness declaration using a related lifting diagram. In fact this was done a couple times (see Examples 3.3.2, 3.3.6) above. The uniqueness constraint corresponding to (m, n) is

$$R \amalg_W R \xrightarrow{(\mathrm{id}_R \amalg \mathrm{id}_R)} R \xrightarrow{n} S. \tag{18}$$

In other words, π satisfies constraint (18) if and only if there exists *at most one* dotted arrow lift making diagram (17) commute.

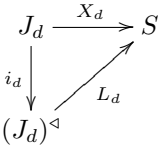
3.5. *Lifting constraints are more expressive than limit sketches*

In this section we show that lifting constraints are more expressive that limit sketches when it comes to set-models. We define limit sketches in Definition 3.5.1, prove that lifting constraints are at least as expressive as limit sketches in Proposition 3.5.2, and prove that lifting constraints are strictly more expressive than limit sketches in Proposition 3.5.3.

For any category $\mathcal{C}$, we let $\mathcal{C}^\triangleleft$ denote the category obtained by adjoining an initial object to $\mathcal{C}$.

Definition 3.5.1.

A *limit sketch* consists of a category S, and a set D of commutative diagrams in **Cat** of the form



one for each $d \in D$. Each X_d is called a *specified limit pre-cone* in S and each L_d is called a *specified limit cone* in S. We call S the *underlying category* of the sketch (S, D).

If (S, D) is a limit sketch, then an (S, D)-*model* is a functor $\delta\colon S \to \mathbf{Set}$ such that for each $d \in D$ the map i_d induces an isomorphism

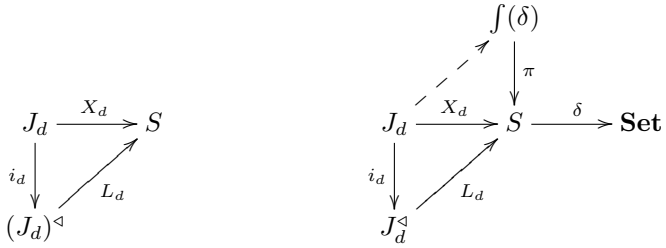
$$\lim_{(J_d)^\triangleleft}(\delta \circ L_d) \cong \lim_{J_d}(\delta \circ X_d). \tag{19}$$

Proposition 3.5.2.

Let (S, D) be a limit sketch. Then we can construct a set of lifting constraints ξ such that the functor $\int\colon S\text{–}\mathbf{Set} \to \mathbf{Cat}_{/S}$ induces a bijection between the set of functors $\delta\colon S \to \mathbf{Set}$ modeling (S, D) and the set of instances $\pi\colon I \to S$ satisfying ξ.

Proof.

It suffices to show that for each diagram $d = (J, X, L)$ as shown to the left

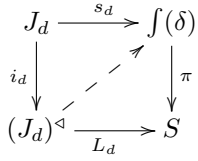


(20)

there exists a set K_d of lifting constraints $\{(m_k, n_k)\}_{k \in K_d}$ with the property that $\delta\colon S \to \mathbf{Set}$ satisfies (19) if and only if $\int(\delta) \to S$ satisfies the constraints in K_d.

The limit $\lim_J \delta \circ X_d$ is in bijection with the set of dotted lifts $s_d\colon J_d \to \int(\delta)$ (such that $\pi \circ s_d = X_d$) in the right-hand diagram of (20), and similarly the limit $\lim_{(}(J_d)^\triangleleft)\delta \circ L_d$ is in bijection with the set of lifts $(J_d)^\triangleleft \to \int(\delta)$. Thus to say that δ models d is to say

that for every commutative diagram of the form



there exists a unique dotted lift. We thus take K_d to be the set $\{(i_d, L_d), (i'_d, L_d)\}$, where $i'_d \colon (J_d)^\triangleleft \amalg_{J_d} (J_d)^\triangleleft \to (J_d)^\triangleleft$. In other words (i_d, L_d) encodes the existence of the dotted lift and (i'_d, L_d) encodes its uniqueness, as in Section 3.4. This completes the proof.

□

Proposition 3.5.3.

There exists a schema S and a set of lifting constraints ξ on it whose satisfaction is not modeled by any limit sketch with underlying category S.

Proof.

Let $S = \underline{1}$ be the terminal category, so that a functor $\delta \colon S \to \mathbf{Set}$ can be considered as just a set $\delta \in \mathrm{Ob}(\mathbf{Set})$ and we have $I := \int(\delta) \cong \delta$. Consider the unique lifting constraint of the form $\underline{2} \xrightarrow{m} \underline{1} \xrightarrow{n} S$. Up to isomorphism there exists precisely two instances δ satisfying $\xi = \{(m, n)\}$, namely either $\delta \cong \underline{0}$ or $\delta \cong \underline{1}$. We will show that there is no limit sketch with underlying category S having only two models up to isomorphism.

For a limit sketch on S each (J, X, L) either has $J = \emptyset$ or $J \xrightarrow{X} S$ is an epimorphism. In the first case, $\lim_{(J_d)^\triangleleft}(\delta \circ L_d) \cong \underline{1}$ and $\lim_{J_d}(\delta \circ X_d) \cong \delta$, so a model of (J, X, L) must have $\delta \cong \underline{1}$. In the second case $\lim_{(J_d)^\triangleleft}(\delta \circ L_d) \cong \delta \cong \lim_{J_d}(\delta \circ X_d)$, so every set δ models this constraint. Thus there is no set D such that the set of sketch models of (S, D) has precisely two elements up to isomorphism.

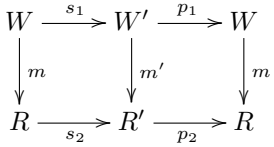
□

3.6. *Constraint implications*

Propositions 3.6.2 and 3.6.3 below are constraint implication results. That is, they show that instances satisfying one lifting constraint automatically satisfy another. These two constraint implications are not exhaustive, they merely give the idea.

Definition 3.6.1.

Suppose that one has a diagram of the form

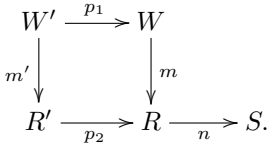


such that the top and bottom compositions are identity,

$$p_1 \circ s_1 = \mathrm{id}_W \qquad \text{and} \qquad p_2 \circ s_2 = \mathrm{id}_R.$$

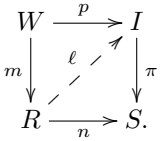
In this case we say that m is a *retract* of m'.

Proposition 3.6.2.
Suppose that (m, n) is a constraint for a schema S and that m is a retract of some m', part of which is shown to the left in the diagram



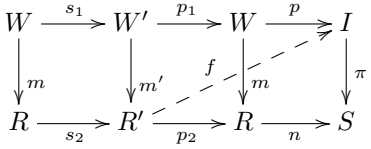
Then any discrete opfibration $\pi\colon I \to S$ satisfying $(m', n \circ p_2)$ also satisfies (m, n).

Proof.
The proof is straightforward but we include it for pedagogical reasons. Suppose given a lifting problem



(21)

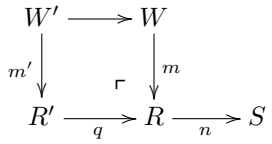
We assume by hypothesis that the dotted arrow lift f exists making the solid arrow diagram



commute. But then one checks that $\ell = f \circ s_2\colon R \to I$ is a lift as in (21).

□

Proposition 3.6.3.
Suppose that the square to the left in the diagram



is a pushout (as indicated by the corner symbol ⌜). If $\pi\colon I \to S$ satisfies the constraint $(m', n \circ q)$ then it satisfies (m, n).

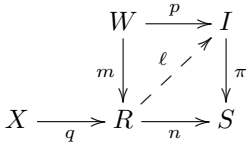
Proof.
Obvious.

□

4. Queries as lifting problems

In this section we will show a correspondence between queries and lifting problems, under which the set of results for a query corresponds to the set of solutions (i.e. lifts) for the associated lifting problem. The main example of this was discussed in Example 1.1.1. There we were interested in learning more about a married couple, given certain known information about them. After building up the necessary theory in Sections 4.1 and 4.3 we will apply it to the case of the married couple in Example 4.3.4.

In the Introduction, more specifically in (1), we alluded to a dictionary between certain SQL statements and lifting problems. In this section we will extend this a bit to include more specificity in the SELECT clause. Namely, we have this correspondence

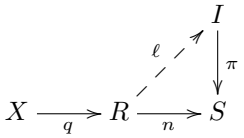


$$\begin{array}{lll} \text{SELECT} & X \xrightarrow{q} R \\ \text{FROM} & R \xrightarrow{n} S \\ \text{WHERE} & R \xleftarrow{m} W \xrightarrow{p} I \end{array} \tag{22}$$

The map q can be composed with any lift $\ell\colon R \to I$ to restrict our attention (i.e. project) to a certain segment of the result. We explain these ideas in Example 4.3.2. However, before getting to this general kind of query, we will discuss queries that do not include the WHERE-clause, i.e. the collection $W \to I$ of knowns.

4.1. *WHERE-less queries*

In this section we study queries as in Diagram (22) in which the where-clause W is empty, $W = \emptyset$. Such queries are often called *views*. In this case the two maps $R \xleftarrow{m} W \xrightarrow{p} I$ contain no information, so Diagram (22) reduces to the following:

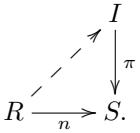


$$\begin{array}{lll} \text{SELECT} & X \xrightarrow{q} R \\ \text{FROM} & R \xrightarrow{n} S \end{array}$$

We call these *WHERE-less queries*.

Definition 4.1.1.

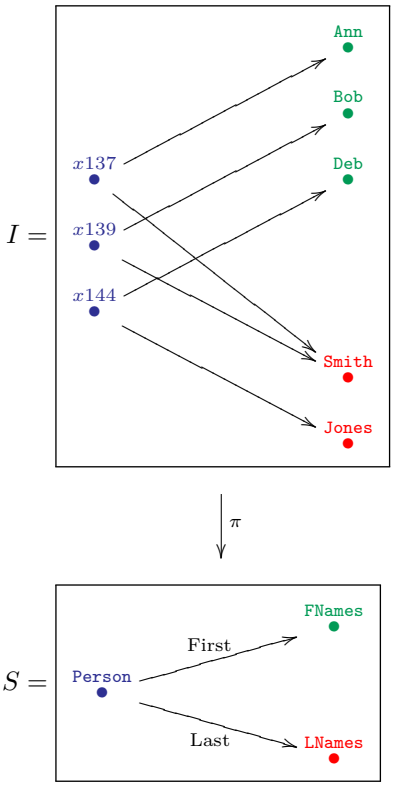
Let S be a schema. A *probe on* S is a functor $n\colon R \to S$; the category R is called *the result schema for the probe*. Given a discrete opfibration $\pi\colon I \to S$ the probe n is said to *set up the lifting problem*



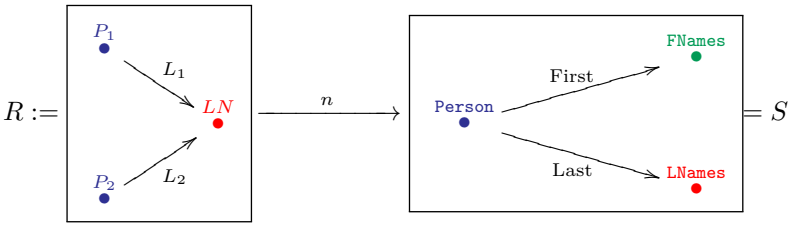
In the presence of a discrete opfibration π, we may refer to the probe n as *a where-less query*. We define the *set of solutions* to the query, denoted $\Gamma(n, \pi)$ as

$$\Gamma(n, \pi) := \{\ell\colon R \to I \mid \pi \circ \ell = n\}.$$

Example 4.1.2.
Consider the discrete opfibration $\pi\colon I \to S$ given here:



To find two people with the same last name, we find lifts of the where-less query



where both $n(L_1) = n(L_2) = (\overset{\texttt{Person}}{\bullet} \xrightarrow{\texttt{Last}} \overset{\texttt{LNames}}{\bullet})$. There are two people (Ann Smith, Bob Smith) with the same last name, so we may hope to get as our result set $\{(x137, \text{Smith}, x139)\}$.

Here is how to compute the result set for our query. We are looking for functors $\ell\colon R \to I$ that make the diagram

$$\begin{array}{ccc} & & I \\ & \overset{\ell}{\nearrow} & \downarrow{\scriptstyle \pi} \\ R & \xrightarrow[n]{} & S \end{array} \tag{23}$$

commute. Since L_1 and L_2 in R are sent to Last in S, we need to choose two "downward

sloping" arrows in I with the same target. Doing so, we indeed find all pairs of persons in I that have the same last name. Unfortunately, this query would return five results, which we can abbreviate as

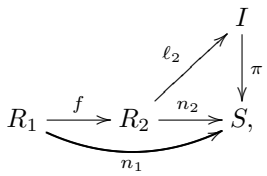
$$\begin{array}{lll} (x137, \mathrm{Smith}, x139), & (x139, \mathrm{Smith}, x137), & \\ (x137, \mathrm{Smith}, x137), & (x139, \mathrm{Smith}, x139), & (x144, \mathrm{Jones}, x144). \end{array} \tag{24}$$

The first two are what we are looking for, but they are redundant; the last three are degenerate (e.g. Deb Jones has the same last name as Deb Jones). We will deal with these issues in Example 4.1.5, after we discuss morphisms of queries.

Definition 4.1.3.

Let S be a schema. Given two probes $n_1 \colon R_1 \to S$ and $n_2 \colon R_2 \to S$, we define a *strict morphism* from n_1 to n_2, denoted $f \colon n_1 \to n_2$, to be a functor $f \colon R_1 \to R_2$ such that $n_2 \circ f = n_1$. Let $\widetilde{\mathbf{Prb}}(S) = \mathbf{Cat}_{/S}$ denote the category whose objects are probes and whose morphisms are strict morphisms. In the presence of a discrete opfibration $\pi \colon I \to S$, we may refer to f as a *strict morphism of where-less queries* (as in Definition 4.1.1).

Given a strict morphism $f \colon n_1 \to n_2$, one obtains a function $\Gamma(f, \pi) \colon \Gamma(n_2, \pi) \to \Gamma(n_1, \pi)$, because any lift ℓ_2 in the diagram



(25)

i.e. with $n_2 = \pi \circ \ell_2$, induces a lift $\ell_1 := \ell_2 \circ f \colon R_1 \to I$ with $n_1 = \pi \circ \ell_1$. We thus have produced a functor $\Gamma(-, \pi) \colon \widetilde{\mathbf{Prb}}(S)^{\mathrm{op}} \to \mathbf{Set}$. It is just the representable functor at π,

$$\Gamma(-, \pi) = \mathrm{Hom}_{\widetilde{\mathbf{Prb}}(S)}(-, \pi).$$

Remark 4.1.4.

We use the term *strict* morphism of probes in Definition 4.1.3 because a more lax version of morphism will be defined later, in Definition 5.1.2. Whereas above we consider commutative triangles of categories (e.g. $n_2 \circ f = n_1$ in (25)) and call the resulting category $\widetilde{\mathbf{Prb}}(S)$, the lax version will allow for natural transformations (e.g. $n_2 \circ f \Rightarrow n_1$) and will be denoted $\mathbf{Prb}(S)$. The functor $\Gamma(-, \pi) \colon \widetilde{\mathbf{Prb}}(S) \to \mathbf{Set}$ defined in Definition 4.1.3 can be extended to a functor (which we give the same name), $\Gamma(-, \pi) \colon \mathbf{Prb}(S) \to \mathbf{Set}$. This will all be discussed in Section 5.1.

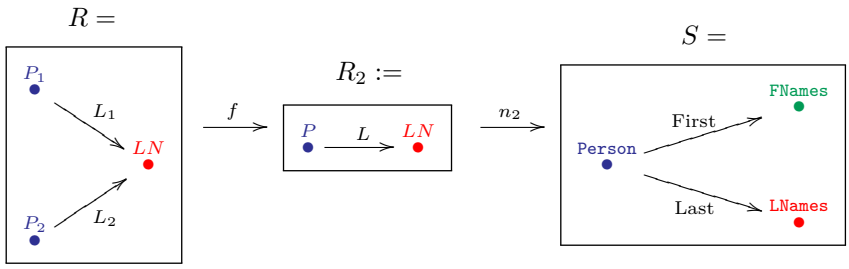
Example 4.1.5.

We again consider the situation from Example 4.1.2, where we were using the query $n \colon R \to S$ to look for pairs of people who had the same last name. The solution set in (24) had two problems:

— we were getting degenerate answers because every person has the same last name as him- or her-self, and

— we were getting order-redundancy because, given two people with the same last name, we can reverse the order and get another such pair.

In order to deal with the first issue, consider the strict morphism f of queries



where $f(L_1) = f(L_2) = L$, and note that indeed $n = n_2 \circ f$. By Definition 4.1.3 this induces a function between the solution sets; i.e. we get a function

$$\Gamma(f, \pi) \colon \Gamma(n_2, \pi) \to \Gamma(n, \pi).$$

In our example (24), the image of this function is precisely the set of duplicates. In other words, if we delete the elements in the image of $\Gamma(f, \pi)$ we get

$$\Gamma(n, \pi) - \Gamma(n_2, \pi) = \{(x137, \text{Smith}, x139), (x139, \text{Smith}, x137)\}.$$

In order to deal with the remaining order-redundancy issue, consider the swap map $s \colon R \to R$ given by $s(L_1) = L_2$ and $s(L_2) = L_1$. Note that $n \circ s = n$. Thus we have a strict morphism of probes $s \colon n \to n$, which induces a function $\Gamma(s, \pi) \colon \Gamma(n, \pi) \to \Gamma(n, \pi)$. By taking the orbits of this function, we effectively quotient out by order-swapping. In fact our swap map acts not just on (R, n) but on (R_2, n_2) as well, and so we can combine this method with the one above to obtain the desired answer, the one element set consisting of $(x137, \text{Smith}, x139)$, in unspecified order.

Proposition 4.1.6.

Let $\delta \colon S \to \mathbf{Set}$ be an instance and $\pi_\delta \colon I \to S$ the induced discrete opfibration. Given any probe $n \colon R \to S$, there is an isomorphism

$$\Gamma(n, \pi_\delta) \xrightarrow{\cong} \lim_R(\delta \circ n).$$

Proof.

Consider the diagram

$$\begin{array}{ccccc} I & \longrightarrow & \mathbf{Set}_* & & \\ \downarrow{\scriptstyle \pi_\delta} & \lrcorner & \downarrow{\scriptstyle \pi} & & \\ R & \xrightarrow[n]{} & S & \xrightarrow[\delta]{} & \mathbf{Set} \end{array}$$

where the right-hand square is a pullback, as shown in Proposition 2.3.3. We have a bijection

$$\text{Hom}_{\mathbf{Cat}/S}(n, \pi_\delta) \cong \text{Hom}_{\mathbf{Cat}/\mathbf{Set}}(\delta \circ n, \pi).$$

The left-hand side is $\Gamma(n, \pi_\delta)$ and the right-hand side is a standard formula for the limit of a set-valued functor, in this case for $\lim_R(\delta \circ n)$.

□

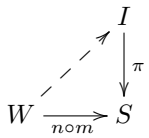
4.2. *Binding variables*

In Section 3.1.1 we defined lifting constraints on a schema S to be a pair of composable functors $W \xrightarrow{m} R \xrightarrow{n} S$. The idea is to think of R as a set of equations (or a *join graph* and of W as a set of variables to be bound at run-time. Our lifting approach below for queries will assume that the variables (in W) have already been bound to something in the active domain of π. As mentioned in the introduction, it is not standard to allow queries to depend on instances. In this short section we explain how to use where-less queries to determine the active domains. In this way, we will explain how instance-independent queries can be posed using the same lifting-problems approach.

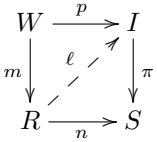
The idea is reminiscent of what is known in modern database practice as a *cursor*. Once the active domains for the variables in W are found, one can either run the cursor (i.e. the query) parameterized over all values in these active domains, or prompt the user to choose bindings for these variables.

We assume for this section that W is a discrete category; this will be most common in practice, but regardless all the ideas we will now discuss generalize to the non-discrete case.

Suppose given a cursor $W \xrightarrow{m} R \xrightarrow{n} S$. To determine the active domains of each variable in W, we simply apply the where-less query given by the diagram



The set of lifts $\Gamma(n \circ m, \pi)$ is the set of possible variable bindings. Once a lift $p\colon W \to I$ is chosen, we have a commutative square



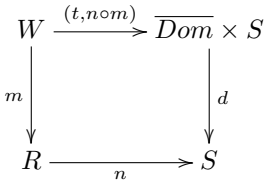
and as we will see in Section 4.3 below, the dotted arrow lifts ℓ will correspond to the results of the now-fully-defined query.

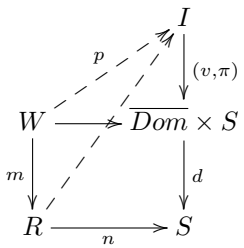
There is one more case we should discuss. Suppose one wants to pose a query such that it is not known in advance whether the chosen constants will or will not be available in the active domain—if they are not, the query must certainly return an empty set of results, and this is the intended behavior. In fact, this is the type of situation that is most often called a query in database literature. In the remainder of Section 4.2, we explain how this is handled by lifting queries.

Let Dom denote the set of all possible domain values, let $\overline{Dom}$ denote the indiscrete

category on Dom, and let $d\colon \overline{Dom} \times S \to S$ denote the projection. [|] Recall that for any category $\mathcal{C}$, the set of functions $\mathrm{Ob}(\mathcal{C}) \to Dom$ is in natural bijection with the set of functors $I \to \overline{Dom}$.

We are given the shape of the query $W \xrightarrow{m} R \xrightarrow{n} S$, and we are also given, for each $w \in W$ a value $t(w) \in Dom$. In other words, our query is represented by the commutative square to the left





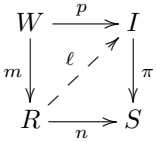
We also have are given a map $v\colon I \to \overline{Dom}$ that sends each datum $i \in \mathrm{Ob}(I)$ to its value in Dom. Form the solid-arrow diagram as to the right. As above, we perform the query in two steps. First we find all lifts $p\colon W \to I$ such that $(v, \pi) \circ p = (t, n \circ m)$. If this set is empty then the query will return an empty result set. However, if there do exist lifts p, then by choosing one, we bind our W-variables to their values found in the active domain. Finally, for each one we find all lifts $R \to I$ making the diagram to the right above commute. The set of all ways to do this is the set of results for our query.

4.3. *General lifting queries*

In this section we tackle the more general lifting query. These closely resemble graph pattern queries, as used in SPARQL (see (Prud'hommeaux et al. 2008)). We will show how to perform queries like (and including) the one suggested in Example 1.1.1, where we hoped to find the last names of our new acquaintances, Bob and Sue. We begin with the definition.

Definition 4.3.1.

Let S be a schema and $\pi\colon I \to S$ a discrete opfibration. A *query on* π is a solid-arrow commutative diagram of the form



(26)

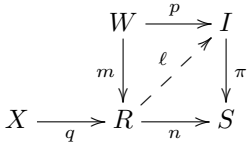
The categories W and R are called the *where-category* and the *result schema*, respectively. We define the *set of solutions* to the query, denoted $\Gamma^{m,p}(n, \pi)$, to be the set of lifts ℓ

[|] If one wants each table $s \in \mathrm{Ob}(S)$ to have its own data type, replace d with the appropriate category over S.

making the diagram commute. Precisely,

$$\Gamma^{m,p}(n,\pi) := \{\ell\colon R \to I \mid \pi \circ \ell = n \text{ and } \ell \circ m = p\}.$$

Example 4.3.2.
By this point, we have developed the theory necessary to make sense of the following dictionary.

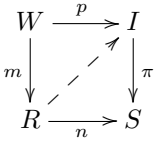


$$\begin{array}{ll} \text{SELECT} & X \xrightarrow{q} R \\ \text{FROM} & R \xrightarrow{n} S \\ \text{WHERE} & R \xleftarrow{m} W \xrightarrow{p} I \end{array}$$

Each lift ℓ in the commutative square is a solution to the SELECT $*$ statement, and composing ℓ with q projects to schema X.

The following proposition says that for any query on a dataset δ, there is a canonical embedding of the query result back into δ.

Proposition 4.3.3.
Let $\delta\colon S \to \mathbf{Set}$ be a instance on a schema and $\pi\colon I \to S$ the associated discrete opfibration. Suppose given a query (lifting problem)



with solution set $\Gamma^{m,p}(n,\pi) \in \mathbf{Set}$. Considering this set as a constant functor $\Gamma\colon R \to \mathbf{Set}$ (given by $\Gamma(r) = \Gamma^{m,p}(n,\pi)$ for all $r \in \mathrm{Ob}(R)$), there is an induced map of R-sets,

$$\mathrm{Res}\colon \Gamma \to \Delta_n \delta.$$

Proof.
Let $\Gamma(n,\pi) = \{\ell\colon R \to I \mid \pi \circ \ell = n\}$ denote the set of solutions to the where-less query $n\colon R \to S$. Clearly, we have an inclusion $\Gamma^{m,p}(n,\pi) \hookrightarrow \Gamma(n,\pi)$. By Proposition 4.1.6, there is an isomorphism $\Gamma(n,\pi) \cong \lim_R(\delta \circ n)$.

Let $t\colon R \to \underline{1}$ denote the terminal functor. It follows from definitions that for any functor $G\colon R \to \mathbf{Set}$, there is an isomorphism of [0]-Sets, $\lim_R(G) \cong \Pi_t(G)$, so in particular we have an inclusion $\Gamma^{m,p}(n,\pi) \to \Pi_t(\delta \circ n)$. By the (Δ_t, Π_t)-adjunction, there is an induced map

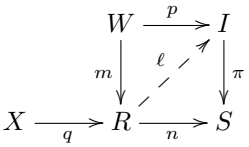
$$\Delta_t(\Gamma^{m,p}(n,\pi)) \to (\delta \circ n)$$

of R-sets. The result follows, since $\Delta_t(\Gamma^{m,p}(n,\pi)) = \Gamma$ and $\delta \circ n = \Delta_n \delta$.

□

Example 4.3.4 (Bob and Sue, revisited).
The motivating example for this paper was presented in Section 1.1. In particular,

we provided a SPARQL query to find all instances of married couples with the requisite characteristics (e.g. the husband's and wife's first names being Bob and Sue respectively). We showed that this SPARQL query could be straightforwardly transformed into a lifting problem of the form

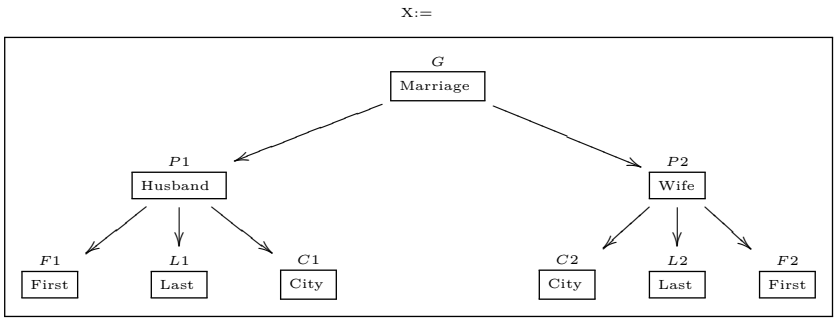


as in (6), and we specified the two functors $W \xrightarrow{m} R \xrightarrow{n} S$. We did not specify the discrete opfibration $I \xrightarrow{\pi} S$ or the inclusion of the known data $p\colon W \to I$, because writing out a convincing possibility for I would necessitate too much space to be worthwhile in this document.

The lifting diagram (6) was presumed to have only one solution, because it was presumed that we knew enough about Bob and Sue that no one else fit the description. In the language of Definition 4.3.1, the set $\Gamma^{m,p}(n,q)$ has one element. By Proposition 4.3.3, this element can be written as a database state on R. We output the result as a two-level table with one row in (7), repeated here,

Marriage								
ID	**Husband**				**Wife**			
	ID	**First**	**Last**	**City**	**ID**	**First**	**Last**	**City**
G3801	M881-36	Bob	Graf	Cambridge	W913-55	Sue	Graf	Cambridge

which in fact was a state on a schema $X \xrightarrow{q} R$, where X is the schema



(27)

While we have not discussed two-level tables before, we hope the idea is straightforward.

4.4. *SPARQL queries involving predicate variables*

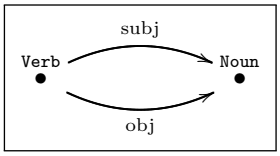
In Example 1.1.1 our SPARQL query (4) only has variables in subject and object positions (the nodes of the schema). It seems that most SPARQL queries used in practice also only have variables in the subject and object positions (see, e.g. (Deus et al. 2010)); still, general SPARQL queries can involve variables in any position including in predicate

positions, which correspond to the arrows of the schema. For example, we may use

$$\text{(John ?x Mary)} \tag{28}$$

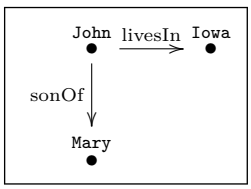
to find all known relationships between John and Mary. To deal with this type of query, one may proceed as follows.

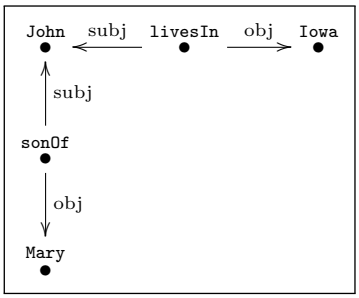
If $S = (V, E, s, t)$ is a graph (thought of as a schema with trivial path equivalences, which is in keeping with RDF schemas), then S itself can be viewed as a database instance $S\colon \mathcal{G} \to \mathbf{Set}$ on the schema

$\mathcal{G} =$ 

similar to Example 3.1.3. We will be working with the Grothendieck construction $\int(S) \to \mathcal{G}$. The category $\int(S)$ will be generated by a bipartite graph. The set of vertices in $\int(S)$ is the union $\texttt{Noun} \amalg \texttt{Verb}$, we might call them verb vertices and noun vertices. There is a unique edge in $\int(S)$ from every verb vertex to its subject noun vertex and another to its object noun vertex.

Example 4.4.1.
Let $X = (V, E, s, t)$ be the graph to the left below:

$X :=$ 

$\int(X) :=$ 

If X is conceived as an instance $X\colon \mathcal{G} \to \mathbf{Set}$, then $\int(X)$ is the category to the right above.

An instance $\pi\colon I \to S$ can be considered simply as a map of graphs, i.e. a map of instances on $\mathcal{G}$. Taking its Grothendieck construction yields a functor $\int(I) \to \int(S)$, whereby each arrow from S (representing a foreign key column) and each arrow from I (representing a cell in a foreign key column) have become a vertex in $\int(S)$ and $\int(I)$ respectively, as in Example 4.4.1. We can perform the original SPARQL query (28) to this derived form of the database because our original predicate can now be accessed as a subject or an object. For example our statement (John ?x Mary) would become the pair of statements (?x subj John) (?x obj Mary).

5. The category of queries on a database

In this section we will discuss some formal properties of the machinery developed in earlier sections. For example we will show that the queries on a given database can be arranged into a database of their own and subsequently queried. This process is commonly known as nesting queries. To this end, we define a category of queries and prove that the process of finding solutions is functorial. We do this in Sections 5.1 and 5.2. In Section 5.3 we extend some results from Section 3.6, giving more detail on the interaction between data migration functors, on the one hand, and query containment and constraint implication on the other.

This section is technical, but it may have fruitful applications. Given any database π, the category $\mathbf{Qry}(\pi)$ organizes the queries (or views) on π into a schema of their own. There is a canonical instance on $\mathbf{Qry}(\pi)$ populating each table (corresponding to a query) with its set of results. In typical applications, users of a database π are often better served by interacting with $\mathbf{Qry}(\pi)$ rather than with π. It is important to understand how schema evolution affects different parts of $\mathbf{Qry}(\pi)$; this is briefly discussed in Section 5.3.

5.1. *New discrete opfibrations from old*

The following theorem is not new, but its formulation in terms of databases is. Furthermore, the proof may be instructive.

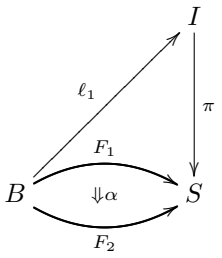
Theorem 5.1.1.

Let $\pi\colon I \to S$ be a discrete opfibration and let B be a category. Then the induced functor $\pi^B\colon I^B \to S^B$ is a discrete opfibration. If $\delta^B = \partial(\pi^B)\colon S^B \to \mathbf{Set}$ is the associated instance, then for any $F\colon B \to S$ in $\mathrm{Ob}(S^B)$, there is a bijection

$$\delta^B(F) \cong \Gamma(F, \pi).$$

Proof.

We begin our proof of the first claim by drawing a figure for reference:



(29)

To see that π^B is a discrete opfibration, suppose that $F_1, F_2\colon B \to S$ are functors and $\alpha\colon F_1 \to F_2$ is a natural transformation. Given a functor $\ell_1\colon B \to I$ with $\pi \circ \ell_1 = F_1$, we must show that there exists a unique functor $\ell_2\colon B \to I$ and natural transformation $\beta\colon \ell_1 \to \ell_2$ such that $\pi \circ \ell_2 = F_2$ and $\pi \circ \beta = \alpha$. For any object $b \in \mathrm{Ob}(B)$, the map $\alpha_b\colon F_1(b) \to F_2(b)$ in S together with the object $\ell_1(b) \in I$, such that $\pi(\ell_1(b)) = F_1(b)$, induces a unique arrow $\beta_b\colon \ell_1(b) \to i_b$ in I for some $i_b \in \mathrm{Ob}(I)$, because π is a discrete opfibration. Define $\ell_2(b) = i_b$. This defines $\ell_2\colon B \to I$ on objects.

Now suppose that $f\colon b \to b'$ is any morphism in B. Applying what we have so far, we get a functor $X \to Y$, where X is the solid-arrow portion of the category to the left and Y is the commutative square category to the right,

$$X := \begin{array}{ccc} \ell_1(b) & \xrightarrow{\beta_b} & \ell_2(b) \\ {\scriptstyle \ell_1(f)}\downarrow & & \vdots\, {\scriptstyle ?} \\ \ell_1(b') & \xrightarrow[\beta_{b'}]{} & \ell_2(b') \end{array} \longrightarrow \begin{array}{ccc} F_1(b) & \xrightarrow{\alpha_b} & F_2(b) \\ {\scriptstyle F_1(f)}\downarrow & & \downarrow{\scriptstyle F_2(f)} \\ F_1(b') & \xrightarrow[\alpha_{b'}]{} & F_2(b') \end{array} =: Y \tag{30}$$

and we get a commutative diagram

$$\begin{array}{ccc} X & \longrightarrow & I \\ \downarrow & & \downarrow{\scriptstyle \pi} \\ Y & \longrightarrow & S \end{array}$$

In order to complete our definition of ℓ_2, our goal is to fill in the missing side (the dotted arrow labeled "?") in square X.

The map $F_2(f)\colon F_2(b) \to F_2(b')$ in S together with the object $\ell_2(b) \in \mathrm{Ob}(I)$ with $\pi(\ell_2(b)) = F_2(b)$ induces a unique arrow $h_{b'}\colon \ell_2(b) \to j_{b'}$ for some $j_{b'} \in \mathrm{Ob}(I)$ with $\pi(j_b) = b'$. But now we have two maps in I over the composite $F_1(b) \to F_2(b')$ both with source $\ell_1(b) \in \mathrm{Ob}(I)$, namely $\beta_{b'} \circ \ell_1(f)\colon \ell_1(b) \to \ell_2(b')$ and $h_{b'} \circ \beta_b\colon \ell_1(b) \to j_{b'}$. Since π is a discrete opfibration, their codomains must be equal, so we have a map $\ell_2(f) := h_{b'}\colon \ell_2(b) \to \ell_2(b') = j_{b'}$, and we have completed the commutative square X in Diagram (30). We have now defined our functor $\ell_2\colon B \to I$ and natural transformation $\beta\colon \ell_1 \to \ell_2$ over α, and they are unique: we made no choices in their constructions. We have shown that $\pi^B\colon I^B \to S^B$ is a discrete opfibration.

Let $\delta^B := \partial(\pi^B)\colon S^B \to \mathbf{Set}$ be the instance associated to π^B and let $F \in \mathrm{Ob}(S^B)$ be an object. We can consider F as a map $\underline{1} \xrightarrow{F} S^B$, and $\delta^B(F)$ is isomorphic to the set of lifts in the left-hand diagram

$$\begin{array}{ccc} & & I^B \\ & \nearrow & \downarrow{\scriptstyle \pi^B} \\ \underline{1} & \xrightarrow[F]{} & S^B \end{array} \qquad\qquad \begin{array}{ccc} & & I \\ & \nearrow & \downarrow{\scriptstyle \pi} \\ B & \xrightarrow[F]{} & S \end{array}$$

which by adjointness is in bijection with the set of lifts $\Gamma(F, \pi)$ in the right-hand diagram. Therefore we have $\delta^B(F) \cong \Gamma(F, \pi)$, completing the proof.

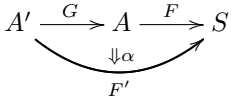
□

The following definition of $\mathbf{Prb}(S)$ extends the notion of $\widetilde{\mathbf{Prb}}(S)$ from Definition 4.1.3: $\widetilde{\mathbf{Prb}}(S) \subseteq \mathbf{Prb}(S)$ is a subcategory with the same set of objects. The category $\mathbf{Prb}(S)$ is a 2-category-theoretic version of slice categories, and is not new (see e.g. (Kelly 1974)).

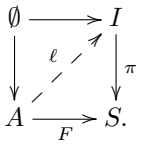
Definition 5.1.2.

Let S be a category. We define the *category of probes on* S, denoted $\mathbf{Prb}(S)$, as follows.

$$\begin{aligned} \mathrm{Ob}(\mathbf{Prb}(S)) &= \{(A, F) \mid A \in \mathrm{Ob}(\mathbf{Cat}), F\colon A \to S \text{ a functor}\} \\ \mathrm{Hom}_{\mathbf{Prb}(S)}((A, F), (A', F')) &= \{G, \alpha) \mid G\colon A' \to A, \quad \alpha\colon F \circ G \to F'\} \end{aligned}$$



Remark 5.1.3.
In the presence of a discrete opfibration $\pi\colon I \to S$, a probe $F\colon A \to S$ sets up a *where-less query* on π for which the results are the lifts $\ell \in \Gamma(F, \pi)$ for the diagram



We call these where-less queries to emphasize that the where-category (upper left of the diagram) is empty.

For any category B, there is an obvious functor $S^B \to \mathbf{Prb}(S)$. The following corollary extends Theorem 5.1.1 in the obvious sense. One way to understand its content is that we can query over where-less queries. In other words, this is a formalization of nested queries. For example, we can create a join graph of where-less queries and look for a set of coherent results. Corollary 5.1.4 (which is not new) implies that given a morphism between two where-less queries on S and given a result for the first query, there is an induced result for the second query. We will deal with the general case of nested queries (those having non-trivial where-categories) in Proposition 5.2.2.

Corollary 5.1.4.
Let $\pi\colon I \to S$ be a discrete opfibration. Then the induced functor

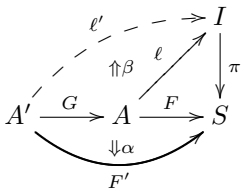
$$\overline{\pi} = \mathbf{Prb}(\pi))\colon \mathbf{Prb}(I) \to \mathbf{Prb}(S)$$

is a discrete opfibration. The instance associated to π is

$$\Gamma(-, \pi) = \partial\,(\overline{\pi})\colon \mathbf{Prb}(S) \to \mathbf{Set}.$$

Proof.
Proving this corollary is really just a matter of writing down the appropriate diagram. In order to show that $\overline{\pi}$ is a discrete opfibration, we choose an object $\ell\colon A \to I$ in $\mathbf{Prb}(I)$ with $\overline{\pi}(\ell) = F\colon A \to S$, we choose a morphism $(G, \alpha)\colon (A, F) \to (A', F')$ in $\mathbf{Prb}(S)$, and we show that there exists a unique morphism $(G, \beta)\colon (A, \ell) \to (A', \ell')$ in $\mathbf{Prb}(I)$, for some $\ell'\colon A' \to I$, such that $\pi \circ \beta = \alpha$. In diagrams, we begin with the solid-arrow

diagram



(31)

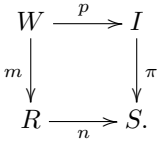
and hope to find such an $\ell' \colon A' \to I$ and $\beta \colon \ell \circ G \to \ell'$.

We have $\pi \circ (\ell \circ G) = F \circ G$. Applying Theorem 5.1.1, there is a unique induced functor $\ell' \colon A' \to I$ and natural transformation $\beta \colon \ell \to \ell'$ such that $\pi \circ \beta = \alpha$, having the required properties. This completes the proof.

□

Remark 5.1.5.

There is a way to express the set of solutions to a lifting problem using limits. Let $\pi \colon I \to S$ be a discrete opfibration, and consider the query



We can consider m as a strict morphism of probes on S, so it induces a function $\Gamma(m, \pi) \colon \Gamma(n, \pi) \to \Gamma(nm, \pi)$, and we can consider $p \in \Gamma(nm, \pi)$ as an element in the codomain. There is a bijection

$$\Gamma^{m,p}(n, \pi) \cong \Gamma(n, \pi) \times_{\Gamma(nm,\pi)} \{p\}, \tag{32}$$

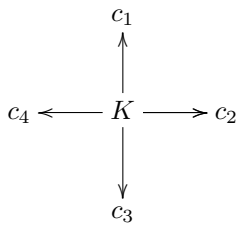
expressing the set $\Gamma^{m,p}(n, \pi)$ of solutions to the lifting problem as the fiber of $\Gamma(m, \pi)$ over p. This idea may be useful when one has disjunctions in the WHERE-clause of a query, as one could replace $\{p\}$ with the set of disjuncts.

Next we present examples of two types of morphisms of where-less queries, namely projection and indirection. These types generate all morphisms of where-less queries.

Example 5.1.6 (Projection).

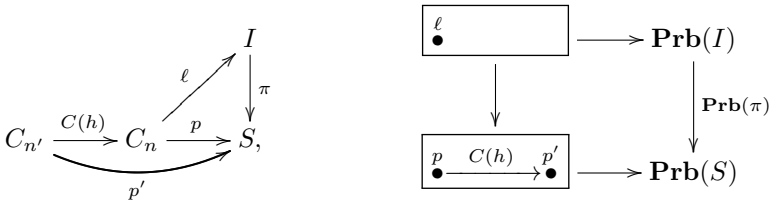
Let $\delta \colon S \to \mathbf{Set}$ be an instance and let $\pi \colon I \to S$ be the associated discrete opfibration. Let $n \in \mathbb{N}$ be a natural number. The *n-column table schema*, here denoted C_n, is the category with an initial object K, precisely n other objects, and precisely n non-identity

arrows; it follows that C_n looks like an asterisk (or "star schema"), e.g. C_4 is drawn:



A functor $p\colon C_n \to S$ is called an *n-column table schema in S*. For each object $x \in C_n$, we call $p(x) \in \mathrm{Ob}(S)$ a *column of p* and we call $p(K)$ the *primary key column of p*. In fact, p is a probe or where-less query. The result set $\Gamma(p, \pi)$ can be thought of as the set of records for instance δ in table p; indeed $\Gamma(p, \pi)$ is isomorphic to $\delta(p)(K)$ as sets.

For any injection $h\colon \{1, 2, \dots, n'\} \hookrightarrow \{1, 2, \dots, n\}$, there is an induced functor $C(h)\colon C_{n'} \to C_n$, which we can compose with p to get a new morphism $p' := p \circ C(h)\colon C_{n'} \to S$ and a strict morphism of probes $p \to p'$. A record in table p is given by a lift ℓ as shown to the left:



(33)

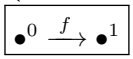
and composing ℓ with $C(h)$ gives its projection as a record in table p'. Thus h induces a function $\Gamma(p, \pi) \to \Gamma(p', \pi)$, and its image is the associated projection. The righthand diagram in (33) is another way of viewing the lefthand diagram.

Remark 5.1.7.

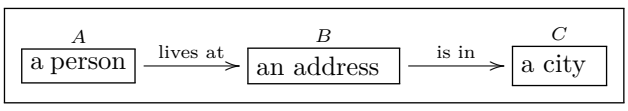
In Example 5.1.6, we did not really need to assume that the function h was injective. If h were not injective, then the morphism of queries $C(h)$ would result in some duplication of columns rather than a pure projection. In other words, the morphism of queries is simply given by substitution along the function $C(h)$.

In Example 5.1.6 we changed the shape of the result schema and used a strict morphism of probes (the natural transformation $p \circ C(h) \to p'$ was the identity). In Example 5.1.8 we will keep the result schema fixed but allow a non-strict morphism.

Example 5.1.8 (Indirection).

Let $R = [1] =$ 

and let S be the schema

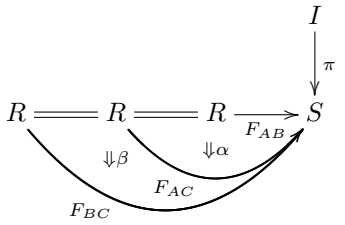
$S :=$ 

There are three non-constant functors $R \to S$, which we denote F_{AB}, F_{AC}, and F_{BC};

there is a natural transformation $\alpha\colon F_{AB} \to F_{AC}$ and a natural transformation $\beta\colon F_{AC} \to F_{BC}$. Thus we get two morphisms in $\mathbf{Prb}(S)$, namely

$$(\mathrm{id}_R, \alpha)\colon (R, F_{AB}) \to (R, F_{AC}) \qquad \text{and} \qquad (\mathrm{id}_R, \beta)\colon (R, F_{AC}) \to (R, F_{BC}).$$

Suppose $\pi\colon I \to S$ is an instance. We can draw the setup as



We can take global sections $\Gamma(-, \pi)$ for each of these three probes and obtain maps between the result sets by Theorem 5.1.1:

$$\Gamma(F_{AB}, \pi) \xrightarrow{\alpha} \Gamma(F_{AC}, \pi) \xrightarrow{\beta} \Gamma(F_{BC}, \pi).$$

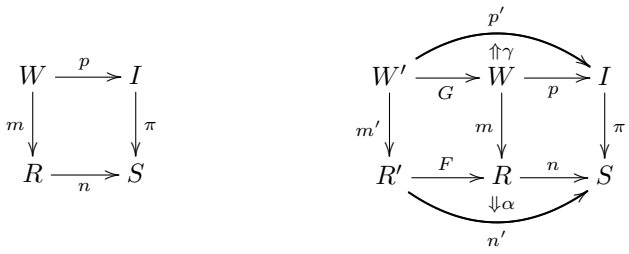
In other words, the morphism of queries induces a morphism of result sets. Simply, given some person and her address we can return a person and the city she lives in; given some person and his city we can return an address and the city it is in.

5.2. *The category of queries*

We are now ready to generalize the category $\mathbf{Prb}(S)$ of where-less queries on S to a category of all (lifting) queries on S.

Definition 5.2.1.

Let $\pi\colon I \to S$ denote a discrete opfibration. We define the *category of (lifting) queries on* π, denoted $\mathbf{Qry}(\pi)$ as follows. The objects of $\mathbf{Qry}(\pi)$ are commutative diagrams as to the left

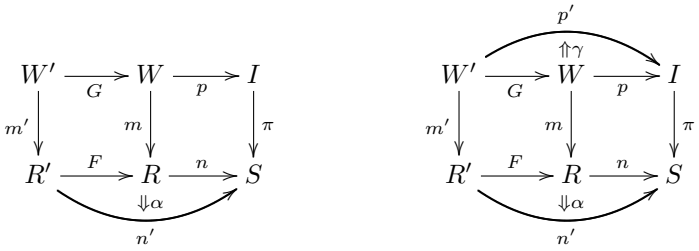


and the morphisms $(F, G, \alpha, \gamma)\colon (R, W, n, p) \to (R', W', n', p')$ are diagrams as to the right, where $m \circ G = F \circ m'$ and where $\pi \circ \gamma = \alpha \circ m'$.

Proposition 5.2.2.

Let $\pi\colon I \to S$ be a discrete opfibration, and suppose given the diagram to the left,

where the two squares commute:



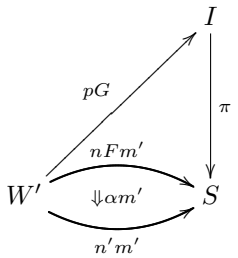
Then there exists a unique morphism of queries

$$(F, G, \alpha, \gamma) \colon (R, W, n, , p) \to (R', W', n', p')$$

as to the right.

Proof.

This is a direct application of Theorem 5.1.1. Indeed, in place of Diagram (29), we draw

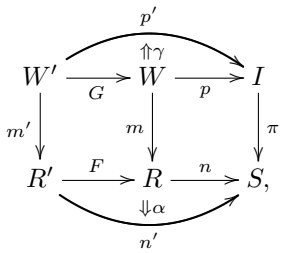


The unique functor and transformation labeled ℓ_2 and β given by the theorem serve as p' and γ here.

□

Theorem 5.2.3.

Let $\pi \colon I \to S$ be a discrete opfibration. Then $\Gamma^{-,-}(-, \pi) \colon \mathbf{Qry}(\pi) \to \mathbf{Set}$ is functorial. That is, given a morphism of queries



there is an induced function, natural in $\mathbf{Qry}(\pi)$,

$$\Gamma^{m,p}(n,\pi) \longrightarrow \Gamma^{m',p'}(n',\pi).$$

Sketch of proof

Suppose given a lift $\ell\colon R \to I$ in $\Gamma^{m,p}(n,\pi)$. By Corollary 5.1.4 we have a map $\ell'\colon R' \to I$, with $\pi \circ \ell' = n'$, and a natural transformation $\beta\colon \ell \circ F \to \ell'$, with $\pi \circ \beta = \alpha$. We need to show that $\ell' \circ m' = p'$ and $\beta \circ m' = \gamma$. But using the proof technique from Proposition 5.2.2, this follows from Theorem 5.1.1 and the definition of discrete opfibration.

□

Remark 5.2.4.

Given a discrete opfibration $\pi\colon I \to S$, we sometimes denote the functor $\Gamma^{-,-}(-,\pi)$ simply by

$$\Gamma(\pi)\colon \mathbf{Qry}(\pi) \to \mathbf{Set}.$$

5.3. *Data migration functors*

Recall (from Definition 2.2.2) that, given a functor $F\colon S \to T$, three data migration functors are induced between the categories S–**Set** and T–**Set**. The most straightforward is denoted $\Delta_F\colon T\text{–}\mathbf{Set} \to S\text{–}\mathbf{Set}$. It has both a left adjoint, denoted $\Sigma_F\colon S\text{–}\mathbf{Set} \to T\text{–}\mathbf{Set}$, and a right adjoint, denoted $\Pi_F\colon S\text{–}\mathbf{Set} \to T\text{–}\mathbf{Set}$.

In standard database contexts, schemas evolve over time. We model these schema evolutions as zigzags of functors from one schema to another, along which one can migrate data using a data migration functor. It is useful to know how this will affect queries. Typically, users of a database $\pi\colon I \to S$ are given access to a subset of $\mathbf{Qry}(\pi)$—they do not see the whole database, but instead some collection of queries. As the schema evolves it is important to understand how $\mathbf{Qry}(\pi)$ evolves. In this section we describe some results; for example under a pullback query results are unchanged.

Let us begin by giving a description of Π_F in terms of where-less queries (see Section 4.1). Recall that for any object $d \in \mathrm{Ob}(T)$ the "comma" category $(d \downarrow F)$ is defined as follows:

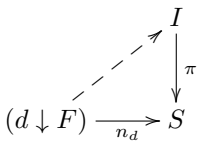
$$\begin{aligned} \mathrm{Ob}(d \downarrow F) &= \{(c,f) \mid c \in \mathrm{Ob}(S), f\colon d \to F(c)\} \\ \mathrm{Hom}_{(d\downarrow F)}((c,f),(c',f')) &= \{g\colon c \to c' \mid f' \circ F(g) = f\}. \end{aligned}$$

There is a natural functor $n_d\colon (d \downarrow F) \to S$, and given a morphism $h\colon d \to d'$ in T we have a morphism $(d' \downarrow F) \to (d \downarrow F)$, or more precisely $n_{d'} \to n_d$, in $\mathbf{Cat}_{/S}$.

Proposition 5.3.1.

Let $F\colon S \to T$ be a functor and $\gamma\colon S \to \mathbf{Set}$ an instance of S with associated discrete opfibration $\pi\colon I \to S$. Given any object $d \in \mathrm{Ob}(T)$, there is an associated where-less

query



and we have $\Pi_F(\gamma)(d) \cong \Gamma(n_d, \pi)$. Moreover, a morphism $d \to d'$ in T induces a strict morphism of where-less queries $n_{d'} \to n_d$; thus we have a functor $T \to \widetilde{\mathbf{Prb}}(\pi)^{\mathrm{op}}$. Then $\Pi_F(\gamma)\colon T \to \mathbf{Set}$ is the composition

$$T \xrightarrow{d \mapsto n_d} \widetilde{\mathbf{Prb}}(\pi)^{\mathrm{op}} \xrightarrow{\Gamma(-,\pi)} \mathbf{Set}.$$

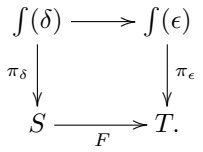
Proof.

Let F, γ, π, d, and $n_d\colon (d \downarrow F) \to S$ be as in the proposition statement. By Proposition 4.1.6, we have $\Gamma(n_d, \pi) \cong \lim_R(\gamma \circ n_d)$. This is exactly the formula for $\Pi_F(\gamma)(d)$ by (Mac Lane 1988, Theorem X.3.1), since Π_F is a right Kan extension. The statement for morphisms follows similarly.

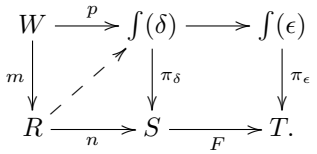
□

While Proposition 5.3.1 provides an interesting relationship between right pushforwards and queries, it does not allow us to relate queries on a database with queries on its right pushforward. In the following paragraphs, we will show briefly that graph pattern queries do transform nicely with respect to data migration functors Σ_F and Δ_F.

We begin by discussing the left pushforward functor. Given a functor $F\colon S \to T$, we have a migration functor $\Sigma_F\colon S\text{–}\mathbf{Set} \to T\text{–}\mathbf{Set}$. If $\delta \in S\text{–}\mathbf{Set}$ and $\epsilon \in T\text{–}\mathbf{Set}$ are instances, then there is a bijection between the set of natural transformations $\Sigma_F\delta \to \epsilon$ and the set of commutative diagrams



Given a query on π_δ, we clearly obtain an induced query on π_ϵ, and a solution to the former yields a solution to the latter:

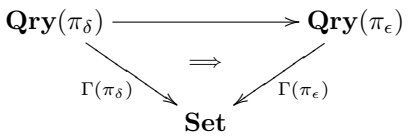


We state this formally in the following proposition.

Proposition 5.3.2.

Let $F\colon S \to T$ be a functor, $\delta \in S\text{–}\mathbf{Set}$ and $\epsilon \in T\text{–}\mathbf{Set}$ instances, and $\Sigma_F\delta \to \epsilon$ a map of T-sets. There exists an induced functor of query categories and a natural transformation

diagram:

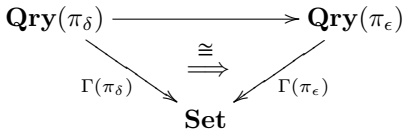


Proof.
The proof follows from the discussion above.

□

We now consider the case that $\delta \cong \Delta_F \epsilon$.

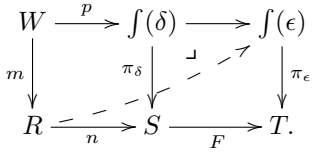
Proposition 5.3.3.
Let $F\colon S \to T$ be a functor, let $\epsilon\colon T \to \mathbf{Set}$ be a functor, let $\delta = \Delta_F \epsilon\colon S \to \mathbf{Set}$ be its pullback, and let π_δ and π_ϵ be as in Diagram (34) below. Then the results of any query on π_δ are the same as the results of the induced query on π_ϵ. That is, we have a natural isomorphism diagram



Proof.
Consider the diagram

$$\begin{array}{ccc} \int(\delta) & \longrightarrow & \int(\epsilon) \\ {\scriptstyle \pi_\delta}\downarrow & \lrcorner & \downarrow{\scriptstyle \pi_\epsilon} \\ S & \xrightarrow[F]{} & T, \end{array} \tag{34}$$

which is a pullback by Proposition 2.3.5. Given a query on π_δ, we obtain a query on π_ϵ as in Proposition 5.3.2. The function from solutions for π_δ to solutions for π_ϵ is a bijection by the universal property of pullbacks.



Indeed, given a lift $R \to \int(\epsilon)$ of π_ϵ, the fact that (34 is a pullback means that there is a unique lift of π_δ mapping to it.

□

6. Future work

This paper has set up an analogy between database queries and constraints on the one hand, and a now classical approach to algebraic topology—the lifting problem—on the

other. Data on a schema is analogous to a covering space or fibration: the local quality of this fibration is determined by constraints, and the locating of sections that satisfy a set of properties is the posing of a query.

There are a few interesting directions for future research. The first is to make a connection to the relatively new field of *homotopy type theory (HoTT)* (see (Awodey 2009),(Voevodsky 2006)). The idea is that instead of two paths through a database schema being *equal*, one could declare them merely *equivalent*; if paths are declared equivalent in more than one way, these equivalences may also be declared as equivalent (or not). In this context, two observations on data may not be definitionally equal, but provably equal, and we consider the proofs and the differences between proofs as part of the data. To make this connection, the schema of a database should be a quasi-category ((Joyal 2002),(Lurie 2009)) $\mathcal{X}$ rather than an ordinary category. Each higher simplex encodes a proof that different paths (or paths of paths, etc.) through the schema are equivalent. We might replace the instance data by a functor (map of quasi-categories) $\mathcal{X} \to \mathbf{Type}$, where **Type** is the quasi-category of homotopy types. In this context, classical homotopical questions, e.g. from the theory of model categories ((Hirschhorn 2003)) may be even more applicable.

Another direction for future research is to use topological tools to investigate or "mine" data. For example, given a functor $\delta\colon S \to \mathbf{Set}$, we can compose with the functor $i\colon \mathbf{Set} \to \mathbf{Top}$ which sends each set to the corresponding discrete topological space. The homotopy colimit of $i \circ \delta$ is a topological space, of possibly any dimension and homotopy type, that encodes the connection pattern of the data. This space is homotopy equivalent to the nerve of the data bundle,

$$\operatorname{hocolim}(i \circ \delta) \simeq N(\textstyle\int \delta)$$

(see (Dugger 2008)). Thus we could report homotopy invariants of the data δ, such as connected components, loops, etc. The question is whether these invariants would be meaningful and useful. For schemas of classical mathematical interest, such as the simplicial indexing category $S = \mathbf{\Delta}^{\mathrm{op}}$, the homotopy colimit of $i \circ \delta$ is exactly what we want; it is the geometric realization of δ. It remains to be seen whether such homotopy invariants may be useful in other contexts; e.g. there may be some connection to the analysis given by persistent homology (see (Ghrist 2008),(Carlsson et al. 2004)).

A third and fairly straightforward project would be to adapt Garner's small object argument (see (Garner 2009)) to our notion of constraints. Garner's argument works, and provides nice universal properties, in the case of what we have called "universal constraint sets" (see Section 3.2). The question is, if we apply his techniques to local constraints, such as those in Example 3.3.6 used to declare that one table is the product of two others, does his procedure still result in a discrete opfibration with all the nice universal properties enjoyed in the universal case? We conjecture that it will. One should also check whether the results obtained from that procedure agree with those from the *universal chase procedure* (see (Deutsch et al. 2008)). Indeed, they should provide equivalent results, since both claim to be universal in the same way.

References

Awodey, S., Warren, M. A. (2009) "Homotopy theoretic models of identity types." *Math. Proc. Cambridge Philos. Soc.* 146 (1), pp. 45-55.

Bancilhon, F; and Spyratos, N. (1981) "Update semantics of relational views." *ACM TODS* 6. pp. 557–575.

Barr, M., Wells, C. (2005) *Toposes, triples, and theories.* Corrected reprint of the 1985 original. Repr. Theory Appl. Categ. No. 12.

Borceux, F. (1994) *Handbook of categorical algebra 1., 2., 3.* Encyclopedia of Mathematics and its Applications 50, 51, 52. Cambridge University Press, Cambridge.

Carlsson, G., Zomorodian, A., Collins, A., and Guibas, L. (2004) "Persistence barcodes for shapes." *Proc. Sympos. Geom. Process.*, pp. 127Ð138.

Diskin, Z.; Kadish, B. (1994) "Algebraic graph-oriented=category-theory-based manifesto of categorizing data base theory", Tech. report, Frame Inform Systems.

Deus, H.F., Zhao, J., Sahoo, S., Samwald, M., Prud'hommeaux, E., Miller M., Marshall, M.S., Cheung, K.-H. (2010) "Prevanance of microarray experiments for a better understanding of experiment results."

Deutsch, A., Nash, A., and Remmel, J. "The Chase Revisited" *Proceedings of PODS* 2008.

Dugger, D. "A primer on homotopy colimits" (2008). ePrint available, http://math.uoregon.edu/∼ddugger/hocolim.pdf.

Ehresmann, C. (1968), "Esquisses et types des structures algèbriques", *Bul. Inst. Politehn. Iasi* (N.S.), 14 (18) (fasc. 1-2). pp. 1–14.

Gambino, N.; Kock, J. (2013) " Polynomial functors and polynomial monads". *Math. Proc. Cambridge Phil. Soc.* 154, pp. 153-192.

Garner, R. (2009) "Understanding the small object argument", *Appl. Categ. Structures* 17 (3), pp. 247–285.

Ghrist, R. (2008) "Barcodes: the persistent topology of data." Bull. Amer. Math. Soc. (N.S.) 45, no. 1, 61Ð75.

Hartshorne, R. *Algebraic Geometry.* Graduate Texts in Mathematics, No. 52. Springer-Verlag 1977.

Johnson, M. (2001)"On Category Theory as a (meta) Ontology for Information Systems Research" *Proceedings of the international conference on Formal Ontology in Information Systems.*

Johnstone, P. (2002) *Sketches of an elephant, Volume 1,2.* Oxford logic guides 43, 44. The Clarendon Press, Oxford University Press, Oxford.

Joyal, A., *Catlab*, available online: http://ncatlab.org/joyalscatlab/show/Factorisation+systems

Joyal, A., (2002) "Quasi-categories and Kan complexes." J. Pure Appl. Algebra 175, no. 1-3, 207Ð222.

Johnson, M.; Rosebrugh, R.; Wood, R.J. *Entity-relationship-attribute designs and sketches*, Theory Appl. Categ. **10** (2002), 94–112 (electronic).

Hirschhorn, P. (2003) *Model categories and their localizations.* Mathematical surveys and monographs, 99. AMS.

Kato, A. (1983) "An abstract relational model and natural join functors." *Bull. Inform. Cybernet.* 20, 95–106.

Lurie, J. *Higher topos theory.* Annals of Mathematical Studies, 170. Princeton University Press, 2009.

Kelly, G.M. (1974) "On clubs and doctrines", *Category Seminar (Lecture Notes in Mathematics)* Volume 420. Springer. pp 181-256

Mac Lane, S. (1988) *Categories for the working mathematician* 2nd edition. Graduate texts in mathematics 5, Springer Verlag, New York.

Makkai, M. (1997) *Generalized sketches as a framework for completeness theorems I.*. J. Pure Appl. Algebra 115, no. 1, 49–79.

May, J.P. (1999) *A concise course in Algebraic Topology*. Chicago Lectures in Mathematics. University of Chicago Press, Chicago, IL.

Morava, J. (2012) "Theories of anything", ePrint available: http://arxiv.org/abs/1202.0684v1

Mac Lane, S. and Moerdijk, I. (1994)*Sheaves in Geometry and Logic: a first introduction to topos theory*, Universitext. Springer-Verlag, New York.

Prud'hommeaux, E., Seaborne, A. (Editors). "SPARQL Query Language for RDF" *W3C Recommendation* 2008/01/15. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/ . Accessed 2012/07/03.

Quillen, D.G. (1967) *Homotopical Algebra*. Lecture notes in mathematics, No. 43. Springer-Verlag.

Spivak, D.I. "Simplicial databases". ePrint available: http://arxiv.org/abs/0904.2012

Spivak, D.I. "Functorial data migration". *Information and Computation* 2012, DOI: 10.1016/j.ic.2012.05.001. ePrint available: http://arxiv.org/abs/1009.1166

Spivak, D.I., Kent, R.E. (2012) "Ologs: A Categorical Framework for Knowledge Representation." *PLoS ONE* 7(1): e24274. doi:10.1371/journal.pone.0024274.

Tuijn, C.; Gyssens, M. (1992) "Views and decompositions from a categorical perspective." In *4th Int. Conf. on Database Theory, ICDT* (Vol. 92, pp. 99 – 112).

Voevodsky, V. (2006) "A very short note on the homotopy λ-calculus." Unpublished note.