SIMPLICIAL DATABASES

DAVID I. SPIVAK

Abstract. In this paper, we define a category **DB**, called the category of simplicial databases, whose objects are databases and whose morphisms are data-preserving maps. Along the way we give a precise formulation of the category of relational databases, and prove that it is a full subcategory of **DB**. We also prove that limits and colimits always exist in **DB** and that they correspond to queries such as select, join, union, etc.

One feature of our construction is that the schema of a simplicial database has a natural geometric structure: an underlying simplicial set. The geometry of a schema is a way of keeping track of relationships between distinct tables, and can be thought of as a system of foreign keys. The shape of a schema is generally intuitive (e.g. the schema for round-trip flights is a circle consisting of an edge from A to B and an edge from B to A), and as such, may be useful for analyzing data.

We give several applications of our approach, as well as possible advantages it has over the relational model. We also indicate some directions for further research.

Contents

1. Introduction 1
2. The category of Tables 3
3. Constructions and formal properties of Tables 8
4. Schemas and databases 11
5. Constructions and formal properties of Simplicial Databases 21
6. Applications, advantages, and further research 29
References 34

1. Introduction

The theory of relational databases is generally formulated within mathematical logic. We provide a more modern and more flexible approach using methods from category theory and algebraic topology. Category theory is useful both as a language and as a tool, and has been successfully applied to many areas of computer science. Using an inefficient language can hamper one's ability to implement, work with, and reason about a subject. This can be seen as one reason that SQL implements tables, rather than relational databases in their pure form: perhaps mathematical logic is not a sufficiently flexible language for discussing databases as they are used in practice.

This project was supported in part by a grant from the Office of Naval Research: N000140910466.

One reason that relational databases have been so successful is that their definition can be phrased within a precise mathematical language. The definition we provide in this paper is just as precise, if not more so (see the discussion at the beginning of Section 4). However, we go beyond simply defining the *objects* of study (databases), but instead continue on to define *morphisms* between databases. With these definitions, we have a category of databases.

There are many categories whose objects are databases (the difference being in their morphisms); what makes one definition better than another? First, a good definition should make sense – the morphisms should somehow preserve the structure of the databases. Second, applying common categorical constructions (colimits, limits, etc.) to the category of databases should result in common database constructions, such as unions, joins, etc. Third, the categorical approach should make reasoning about databases, such as that needed for maintaining and restructuring databases, easier.

Our formulation accomplishes these three goals (see Remark 4.3.8, and Sections 5 and 6, respectively). As an added bonus, the schemas for our databases have geometric structure (more precisely, the structure of a simplicial set). In other words, the schema is given as a geometric object which one should think of as a kind of Entity-Relationship diagram for the schema. This approach may lead to improvements in query optimization because one can adjust the "shape" of the schema to fit with the purposes of the queries to be taken. The ability to visualize data should also prove useful, because these visualizations seem to "make sense" in practice. Examples of this phenomenon are given in 6.1.1 and 6.1.2, where we respectively discuss round trip flights and a sociological experiment involving 4-cycles in high school partnerships.

The data on a given schema is given by a sheaf of sets on that schema. Sheaves are widely used in modern mathematics because they generalize sets and functions and because they have good formal properties. Classical operations on sheaves (such as direct images) allow one to transport data from one schema to another in a functorial way. One of the main purposes of this paper is to provide a good language for discussing databases mathematically, and the consideration of data as a sheaf on a given schema helps to accomplish that goal.

Other researchers have formulated databases in terms of category theory (for example, see [RW92],[JRW02],[PS95],[Ber01],[DK94],[Dis96],[GB92]). Of note is work by Cadish and Diskin, and work by Rosebrugh and Wood. There are many differences between previous viewpoints and our own. Most notably, our work uses simplicial methods to give a geometric structure to the schemas of databases and uses sheaves over these spaces to model the data itself. Both of these approaches appear to be new.

We assume throughout this paper that the reader has a basic knowledge of category theory which includes knowing the definition of category, functor, limit, and colimit, as well as basic facts such as Yoneda's lemma. Good references for this material include [ML98],[BW90], and [Bor94a]. We do not assume that the reader has a prior knowledge of sheaves or of simplicial sets.

We begin by defining the category of tables, in Section 2. In Section 3, we prove that the category of tables is closed under limits and certain colimits, and that these constructions correspond to joins and unions. We also prove that projections and deletions are easily defined under our formulation. In Section 4, we first give a brief description of simplicial sets. We then proceed to define the category of simplicial databases. In Section 5, we prove that the category of simplicial databases is closed under all limits and colimits and prove that they again correspond to joins and unions. Finally in Section 6, we discuss some applications of our model and directions for future research.

1.1. **Acknowledgments.** I would like to thank Paea LePendu for explaining relational databases to me, for suggesting that databases should be categorified, and for his advice and encouragement throughout the process. I would also like to thank Chris Wilson for several useful conversations.

2. The category of Tables

It is no accident that SQL uses tables instead of relations: Tables are inherently more useful, yet just as easy to implement. They are disliked by the purists of relational database theory ([Dat05]) not because they are bad, but because they do not fit in with that theory. In this section we provide a categorical structure to the set of tables, thus firmly grounding it in rigorous mathematics.

2.1. **Data types.**

In order to define schemas, records, and tables of a given type, we need to define what we mean by "type."

Definition 2.1.1. A *type specification* is simply a function between sets $\pi\colon U \to \mathbf{DT}$. The set $\mathbf{DT}$ is called the set of *data types* for π, and the set U is called the *domain bundle* for π. Given any element $T \in \mathbf{DT}$, the preimage $\pi^{-1}(T) \subset U$ is called the *domain of* T, and an element $x \in \pi^{-1}(T)$ is called an *object of type* T.

Example 2.1.2. Let U denote the disjoint union $U := (\mathbb{Z} \amalg \mathbb{R} \amalg \mathbf{Strings})$ and let $\mathbf{DT}$ denote the three element set $\{`\mathbb{Z}', `\mathbb{R}', `\mathbf{Strings}'\}$. Let $\pi\colon U \to \mathbf{DT}$ denote the obvious function, which send all of $\mathbb{Z}$ to the element '$\mathbb{Z}$', all of $\mathbb{R}$ to '$\mathbb{R}$', and all of **Strings** to '**Strings**'. The preimage $\pi^{-1}(`\mathbf{Strings}') \subset U$, which we have called the domain of the type '**Strings**', is indeed the set of strings.

As another example, the mod 2 function $\pi\colon \mathbb{Z} \to \{`\text{even}', `\text{odd}'\}$ is a type specification in which the objects of type 'even' are the even integers.

2.2. **Schemas.**

We quickly recall the definition of fiber product (for sets).

Definition 2.2.1. Let A, B, and C be sets, and suppose $f\colon A \to B$ and $g\colon C \to B$ are functions with the same codomain. The *fiber product of* A *and* C *over* B, denoted $A \times_B C$, is the set

$$A \times_B C := \{(a,c) \in A \times C | f(a) = g(c) \in B\}.$$

The fiber product moreover comes equipped with obvious projection maps making the diagram

$$\begin{tikzcd}
A \times_B C \arrow[r, "f'"] \arrow[d, "g'"'] \arrow[dr, phantom, "\lrcorner", very near start] & C \arrow[d, "g"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

commute. The corner symbol $\lrcorner$ serves to remind the reader that the object in the upper left is a fiber product. We sometimes call $g'\colon A \times_B C \to A$ the *pullback of* g *along* f; similarly f' is the pullback of f along g.

Remark 2.2.2. The fiber product of the diagram $A \xrightarrow{f} B \xleftarrow{g} C$ above should probably be denoted $f \times_B g$ instead of $A \times_B C$, since it depends on the maps f and g, not just their domains. However, this is not often done, and in this paper the maps will be clear from context.

Definition 2.2.3. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification. A *simple schema of type* π consists of a pair (C, σ), where C is a finite (totally) ordered set and $\sigma\colon C \to \mathbf{DT}$ is a function. We sometimes denote the simple schema (C, σ) by σ. We refer to C as the *column list* or *list of attributes* for σ and π as the *type specification* for σ.

Let $U_\sigma := \sigma^{-1}(U)$ denote the fiber product $U \times_{\mathbf{DT}} C$. We call the pullback $\pi_\sigma\colon U_\sigma \to C$, i.e. the left hand map in the diagram

$$\begin{tikzcd}
U_\sigma \arrow[r] \arrow[d, "\pi_\sigma"'] \arrow[dr, phantom, "\lrcorner", very near start] & U \arrow[d, "\pi"] \\
C \arrow[r, "\sigma"'] & \mathbf{DT},
\end{tikzcd}$$

the *domain bundle on C* induced by σ.

Remark 2.2.4. We do not worry much about the ordering on C, as evidenced by the fact that we do not record it in the notation (C, σ) for the simple schema. In fact the ordering requirement can be dropped from the definition if one so chooses.

The reason we include it is first because the columns of a displayed table naturally come with an order (left to right), and second because it results in a more commonly used mathematical object in Section 4. See Remark 4.1.1.

Example 2.2.5. Let $\pi\colon U \to \mathbf{DT}$ denote the type specification of Example 2.1.2. Let $C =$ (‘First Name’, ‘Last Name’,‘BYear’), and define $\sigma\colon C \to \mathbf{DT}$ by

$$\sigma(\text{‘First Name’}) = \text{‘}\mathbf{Strings}\text{’}$$
$$\sigma(\text{‘Last Name’}) = \text{‘}\mathbf{Strings}\text{’}$$
$$\sigma(\text{‘BYear’}) = \text{‘}\mathbb{Z}'$$

We see that C is a list of attributes for the simple schema σ. We call C the column list because, once we arrange data in terms of tables, the columns of these tables will each be headed by an element of C.

One can check that the domain bundle $U_\sigma \to C$ induced by σ is the obvious function

$$(\mathbf{Strings} \amalg \mathbf{Strings} \amalg \mathbb{Z}) \longrightarrow C.$$

Thus an object of type ‘First Name’ is a string in this example.

Definition 2.2.6. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification. A *morphism of simple schemas (of type π)*, written $f\colon (C, \sigma) \to (C', \sigma')$, is an order-preserving function $f\colon C \to C'$ such that the triangle

$$\begin{tikzcd}
C \arrow[rr, "f"] \arrow[dr, "\sigma"'] & & C' \arrow[dl, "\sigma'"] \\
& \mathbf{DT} &
\end{tikzcd}$$

commutes.

The *category of simple schemas on π*, denoted $\mathcal{S}^\pi$ is the category whose objects are simple schemas and whose morphisms are morphisms thereof.

Remark 2.2.7. Let $\mathbf{\Delta}$ denote the category of finite ordered sets. Let $(\mathbf{\Delta} \downarrow \mathbf{DT})$ denote the category for which an object is a finite ordered set with a map to $\mathbf{DT}$ and for which a morphism is an order-preserving function, over $\mathbf{DT}$. One can easily see that the category $\mathcal{S}^{\pi}$ is isomorphic to $(\mathbf{\Delta} \downarrow \mathbf{DT})$, regardless of π. However, we should think of π as part of the data for a simple schema.

Note that the symbol $\mathbf{\Delta}$ typically refers to the category of *non-empty* finite ordered sets; one typically denotes the category of all finite ordered sets as $\mathbf{\Delta}_{+}$. For typographical reasons, we do not follow the standard convention in this paper.

2.3. **Records and Tables.**

Definition 2.3.1. Let (C, σ) be a simple schema. A *record on* (C, σ) is a function $r\colon C \to U_{\sigma}$ such that $\pi_{\sigma} \circ r = \mathrm{id}_C$, i.e. a section of the domain bundle for σ. We denote the set of records on σ by $\Gamma^{\pi}(\sigma)$, or simply by $\Gamma(\sigma)$ if π is understood.

In other words, a record must produce, for each attribute $c \in C$, an object of type $\sigma(c) \in \mathbf{DT}$.

Example 2.3.2. Let π and (C, σ) be as in Example 2.2.5. A record on that simple schema is a section r as depicted in the diagram

$$\begin{tikzcd}
\mathbf{Strings} \amalg \mathbf{Strings} \amalg \mathbb{Z} \arrow[d, "\pi_{\sigma}"] \\
\{\text{‘First Name’, ‘Last Name’,‘BYear’}\}. \arrow[u, bend left, "r"]
\end{tikzcd}$$

That is, a record is a way to designate a first name and a last name (in **Strings**) and a birth year (in $\mathbb{Z}$). For example (Barack; Obama; 1961) denotes a record on this simple schema; that is, it defines a section of π_{σ}.

The set $\Gamma(\sigma)$ of records on (C, σ) is simply the set of all possible such sections. In this example $\Gamma(\sigma) = \mathbf{Strings} \times \mathbf{Strings} \times \mathbb{Z}$.

Definition 2.3.3. Let $\pi\colon U \to \mathbf{DT}$ be a type specification. A *table of type* π consists of a sequence (K, C, σ, τ), where K is a set, (C, σ) is a simple schema of type π, and $\tau\colon K \to \Gamma(\sigma)$ is a function. We sometimes denote the table (K, C, σ, τ) simply by τ. The set K is called the *set of keys of* τ, and (C, σ) is called the *simple schema of* τ.

Remark 2.3.4. Given a table (K, C, σ, τ), those familiar with SQL should think of the set K of keys as the set of row identifiers for a table. These row ids are always unique identifiers and serve as an internal key system for the table; they are generally not considered as part of the data.

Remark 2.3.5. We do not require our tables to have finitely many rows. One could easily enforce such a restriction if desired, and follow the rest of the paper with that restriction in mind. The resulting category would be a full subcategory of the one we present in Definition 2.4.1, it would still be closed under finite limits (etc.), and queries would be taken in precisely the same way as they are here.

Example 2.3.6. Given a simple schema (C, σ), a table on it is simply a collection of records indexed by a set K. The records need not be distinct because the set K keeps track of the distinctions. Continuing with π and (C, σ) as in Example 2.3.2, we could have $K = \{1, 2, `foo'\}$ and let $\tau\colon K \to \Gamma(\sigma)$ be the assignment

$$\begin{aligned} 1 &\mapsto (\text{Barack; Obama; 1961}) \\ 2 &\mapsto (\text{Michelle; Obama; 1964}) \\ `foo' &\mapsto (\text{Barack; Obama; 1961}) \end{aligned}$$

This table can be written in more standard form as:

K	‘First Name’	‘Last Name’	‘BYear’
1	Barack	Obama	1961
2	Michelle	Obama	1964
‘foo’	Barack	Obama	1961

We indicate with the double vertical line the fact that this table corresponds to a function whose domain is K.

Lemma 2.3.7. *Let $\pi\colon U \to \mathbf{DT}$ denote a type specification, let (C_1, σ_1) and (C_2, σ_2) denote simple schemas on π, and let $f\colon (C_2, \sigma_2) \to (C_1, \sigma_1)$ denote a morphism of simple schemas. There is an induced map on record sets $f^*\colon \Gamma(\sigma_1) \to \Gamma(\sigma_2)$.*

Proof. Consider the diagram

$$\begin{tikzcd}
U_{\sigma_2} \arrow[r] \arrow[d, "\pi_2"'] \arrow[dr, phantom, "\lrcorner", very near start] & U_{\sigma_1} \arrow[r] \arrow[d, "\pi_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & U \arrow[d, "\pi"'] \\
C_2 \arrow[r, "f"] \arrow[rr, bend right, "\sigma_2"'] & C_1 \arrow[r, "\sigma_1"] & \mathbf{DT}.
\end{tikzcd}$$

Note that the left hand square is a fiber product square. This follows by applying basic category theory (specifically the “pasting lemma” for fiber products; see [ML98]) to the fact that the right hand square and the big rectangle are fiber product squares. We must show that a section $r_1\colon C_1 \to U_{\sigma_1}$ of π_1 induces a section $r_2\colon C_2 \to U_{\sigma_2}$ of π_2, because this assignment will constitute $f^*\colon \Gamma(\sigma_1) \to \Gamma(\sigma_2)$.

Suppose given r_1 with $\pi_1 \circ r_1 = \mathrm{id}_{C_1}$. We have a map $r_1 \circ f\colon C_2 \to U_{\sigma_1}$ and a map $\mathrm{id}_{C_2}\colon C_2 \to C_2$ such that $f \circ \mathrm{id}_{C_2} = f = \pi_1 \circ (r_1 \circ f)$. By the universal property, these two maps define a map $r_2\colon C_2 \to U_{\sigma_2}$ such that, in particular $\pi_2 \circ r_2 = \mathrm{id}_{C_2}$. This is the desired section of π_2.

□

Given a morphism $f\colon \sigma_2 \to \sigma_1$ of simple schemas, the function $f^*\colon \Gamma(\sigma_1) \to \Gamma(\sigma_2)$ defined in the above lemma is said to be *induced* by f.

Definition 2.3.8. Let $\pi\colon U \to \mathbf{DT}$ be a type specification, and let $(K_1, C_1, \sigma_1, \tau_1)$ and $(K_2, C_2, \sigma_2, \tau_2)$ denote tables. A *morphism of tables* $\varphi\colon \tau_1 \to \tau_2$ consists of a pair (g, f), where $g\colon K_1 \to K_2$ is a function and $f\colon (C_2, \sigma_2) \to (C_1, \sigma_1)$ is a morphism of simple schemas such that the diagram of sets

(1)
$$\begin{tikzcd}
K_1 \arrow[r, "\tau_1"] \arrow[d, "g"'] & \Gamma(\sigma_1) \arrow[d, "f^*"] \\
K_2 \arrow[r, "\tau_2"'] & \Gamma(\sigma_2)
\end{tikzcd}$$

commutes, where $f^*\colon \Gamma(\sigma_1) \to \Gamma(\sigma_2)$ is the function induced by f.

Example 2.3.9. Let us continue with Example 2.3.6, except for a slight renaming of objects: $C_1 := C, \sigma_1 := \sigma, K_1 := K$, and $\tau_1 := \tau$. Let $C_2 = \{$'First', 'Last'$\}$ and let σ_2 send both elements to the data type **Strings** $\in$ **DT**; thus $\Gamma(\sigma_2) =$ **Strings** $\times$ **Strings**.

Let $K_2 = \{5, 6, \text{'}bar'\}$ and τ_2 be the assignment

$$\begin{aligned} 5 &\mapsto \text{(Barack; Obama)} \\ 6 &\mapsto \text{(Michelle; Obama)} \\ \text{'}bar' &\mapsto \text{(George; Bush).} \end{aligned}$$

A morphism of tables $\varphi\colon \tau_1 \to \tau_2$ should consist of a map $g\colon K_1 \to K_2$ and a map $f^*\colon \Gamma(C_1) \to \Gamma(C_2)$. We have an obvious map of simple schemas $f\colon C_2 \to C_1$, namely 'First' ↦ 'First name' and 'Last' ↦ 'Last name'. Then $f^*\colon \Gamma(\sigma_1) \to \Gamma(\sigma_2)$ is just the projection **Strings** $\times$ **Strings** $\times\, \mathbb{Z} \to$ **Strings** $\times$ **Strings**.

Now, to define a morphism of tables $\varphi\colon \tau_1 \to \tau_2$, our choice of g must send both of the records (Barack; Obama; 1961) in τ_1 to the record (Barack; Obama) and send the record (Michelle; Obama; 1964) to the record (Michelle; Obama). There is a unique such morphism ϕ in this case.

For a variety of reasons, there *does not* exist a morphism of tables $\tau_2 \to \tau_1$.

Remark 2.3.10. The morphism of tables in Example 2.3.9 has a common form. As in the example, a morphism of tables often is composed of a projection (in the columns) together with an inclusion (in the rows). The requirement that the square (1) in Definition 2.3.8 commutes is simply the requirement that morphisms preserve the integrity of the data.

2.4. **The category of tables.** We have now defined tables and morphisms between tables. Given morphisms depicted

$$\begin{tikzcd} K_1 \arrow[r, "\tau_1"] \arrow[d] & \Gamma(\sigma_1) \arrow[d] \\ K_2 \arrow[r, "\tau_2"] \arrow[d] & \Gamma(\sigma_2) \arrow[d] \\ K_3 \arrow[r, "\tau_3"] & \Gamma(\sigma_3) \end{tikzcd}$$

it is easy to see how composition is defined. It is also easy to understand the identity morphism on a table $\tau\colon K \to \Gamma(C)$. Thus we have a category.

Definition 2.4.1. Let $\pi\colon U \to$ **DT** denote a type specification. The category whose objects are tables $K \to \Gamma(\sigma)$ and whose morphisms are commutative squares as in Definition 2.3.8 is called *the category of tables on* π and is denoted $\mathbf{Tables}^{\pi}$, or simply **Tables**, if π is understood.

Example 2.4.2. Suppose $\pi\colon U \to$ **DT** is as in Example 2.2.5. Suppose that $C = \{c_1, c_2\}$ and $C' = \{c_1'\}$, and that $\sigma\colon C \to$ **DT** and $\sigma'\colon C' \to$ **DT** are the unique maps such that $\Gamma(\sigma) = \mathbb{Z} \times \mathbb{Z}$ and $\Gamma(\sigma') = \mathbb{Z}$. Let K and K' be any two sets and $\tau\colon K \to \Gamma(\sigma)$ and $\tau'\colon K' \to \Gamma(\sigma')$ be any two tables.

For a morphism $\tau_1 \to \tau_2$ in the category of tables, we are allowed any kind of function between key sets $K \to K'$, but the only permitted maps $\mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}$

are the two projections, because they are the only maps which are induced by morphisms of simple schemas.

Definition 2.4.3. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification and let $\sigma\colon C \to \mathbf{DT}$ denote a simple schema. The *category of tables on* σ *of type* π, denoted $\mathbf{Tables}_\sigma^\pi$ is the category whose objects are tables $\tau\colon K \to \Gamma(\sigma)$ and whose morphisms are triangles

$$\begin{tikzcd} K_1 \arrow[d, "g"'] \arrow[dr, "\tau_1"] & \\ K_2 \arrow[r, "\tau_2"'] & \Gamma(\sigma) \end{tikzcd}$$

denoted by $g\colon \tau_1 \to \tau_2$.

2.5. **Relational tables.** The most common formulation of databases used today is the relational model, invented by E.F. Codd (see [Cod70]). It is based on the theory of mathematical logic, and more specifically on relations. One can find a modern treatment of the subject in [Dat05]. We define a relation in Definition 2.5.1 as a type of table, where the map $\tau\colon K \to \Gamma(\sigma)$ is required to be an injection.

Definition 2.5.1. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification, and let $\sigma\colon C \to \mathbf{DT}$ denote a simple schema on π. A *relation on* σ is a table $\tau\colon K \to \Gamma(\sigma)$ for which τ is an injective function.

A morphism of relations is a morphism of tables, for which the source and target tables are relations. That is, the *category of relations*, denoted $\mathbf{Rel}^\pi$ is the full subcategory of $\mathbf{Tables}^\pi$ spanned by the relations. Similarly, given a simple schema σ, the *category of relations on* σ is the full subcategory of $\mathbf{Tables}_\sigma^\pi$ spanned by the relations. As usual the superscript π can be dropped if it is understood.

There is a functor $\mathbf{Rel} \to \mathbf{Tables}$ and a functor $\mathbf{Rel}_\sigma \to \mathbf{Tables}_\sigma$, both of which are simply inclusions of full subcategories.

3. Constructions and formal properties of Tables

Our definition for the category of tables (Definition 2.4.1) is sensible because objects are tables and morphisms are data-preserving maps. In this section we show that category-theoretic operations on tables correspond to operations on databases, such as joins and other queries. Fix a type specification $\pi\colon U \to \mathbf{DT}$ for the remainder of the section. We will drop π as a superscript in this section; for example the category $\mathcal{S}^\pi$ of simple schemas on π will be denoted simply by $\mathcal{S}$.

We sometimes refer to the underlying keys or underlying simple schema of a table, so we record these trivial constructions in a remark.

Remark 3.1.2. There is a forgetful functor $\mathbf{Tables} \to \mathbf{Sets}$ given by sending a table $\tau\colon K \to \Gamma(\sigma)$ to the key set K and a morphism of tables to the underlying map of keys. There is another forgetful functor $\mathbf{Tables} \to \mathcal{S}^{\mathrm{op}}$ which sends the table τ to its simple schema σ and a morphism $\varphi = (g, f)$ of tables to the underlying morphism of simple schemas f.

Lemma 3.1.3. *There exists a final object and an initial object in* $\mathbf{Tables}$.

Proof. One checks immediately that if we take K to be a terminal object in $\mathbf{Sets}$ (i.e. any set K with cardinality 1) and σ to be the inital object $\emptyset \to \mathbf{DT}$ in $\mathcal{S}$, then there is exactly one table with these as its underlying keys and simple schema, and this table is the terminal object in **Tables**.

One also checks immediately that if we take $K = \emptyset$ to be the initial object in **Sets** and $\sigma = \mathrm{id}_{\mathbf{DT}} \colon \mathbf{DT} \to \mathbf{DT}$ to be the final object in $\mathcal{S}$, then there is exactly one table with these as its underlying keys and simple schema, and this table is the initial object in **Tables**.

$\square$

Certain colimits exist in **Tables**; namely colimits of diagrams that are constant in the underlying simple schema.

Construction 3.1.4. Let $\tau_1 \colon K_1 \to \Gamma(\sigma)$ and $\tau_2 \colon K_2 \to \Gamma(\sigma)$ be two tables with the same simple schema. By taking the disjoint union of K_1 and K_2 we get a new table $\tau \colon K_1 \amalg K_2 \to \Gamma(\sigma)$. This query is called UNION ALL in SQL.

We can also take the (non-disjoint) union of these two tables, if we know how they overlap. That is, if there is some set K with maps $g_1 \colon K \to K_1$ and $g_2 \colon K \to K_2$ in such a way that $\tau_1 \circ g_1 = \tau_2 \circ g_2$, then we can obtain a new table $\tau \colon K_1 \amalg_K K_2 \to \Gamma(\sigma)$. This query is called UNION in SQL.

We will see that limits in the category of tables correspond to generalized joins.

Proposition 3.1.5. *All finite limits exist in* **Tables**.

Proof. It suffices (see, for example, [MLM94, p. 30]) to show that **Tables** has a terminal object and is closed under taking fiber products; the first of these facts was shown in Lemma 3.1.3. For the second, suppose we have a diagram

$$\begin{tikzcd}
K_1 \arrow[r, "\tau_1"] \arrow[d] & \Gamma(\sigma_1) \arrow[d, "f_1^*"] \\
K \arrow[r, "\tau"] & \Gamma(\sigma) \\
K_2 \arrow[r, "\tau_2"] \arrow[u] & \Gamma(\sigma_2) \arrow[u, "f_2^*"']
\end{tikzcd} \tag{2}$$

in **Tables**, where $\sigma \colon C \to \mathbf{DT}$ and $\sigma_i \colon C_i \to \mathbf{DT}$ for $i = 1, 2$ are simple schemas. As indicated, the maps $\Gamma(\sigma_i) \to \Gamma(\sigma)$ are induced by morphisms of simple schemas $f_i \colon \sigma \to \sigma_i$, for $i = 1, 2$.

Consider the simple schema

$$(\sigma_1 \amalg_\sigma \sigma_2) \colon C_1 \amalg_C C_2 \longrightarrow \mathbf{DT}$$

induced by taking the colimit of the column lists. We would like to show that the natural function

$$\Gamma(\sigma_1 \amalg_\sigma \sigma_2) \longrightarrow \Gamma(\sigma_1) \times_{\Gamma(\sigma)} \Gamma(\sigma_2) \tag{3}$$

is a bijection.

Let us first calculate the set $\Gamma(\sigma_1 \amalg_\sigma \sigma_2)$. It is the set of all sections r of the map π' in the diagram

$$\begin{tikzcd}
(\sigma_1 \amalg_\sigma \sigma_2)^{-1}(U) \arrow[r] \arrow[d, "\pi'"] \arrow[dr, phantom, very near start, "\lrcorner"] & U \arrow[d, "\pi"] \\
C_1 \amalg_C C_2 \arrow[r, "\sigma_1 \amalg_\sigma \sigma_2"'] \arrow[u, dotted, bend left, "r"] & \mathbf{DT}.
\end{tikzcd}$$

To give such a section is to give, for each $c_1 \in C_1$ an element of $\pi^{-1}(\sigma_1(c_1))$, and for each $c_2 \in C_2$ an element of $\pi^{-1}(\sigma_2(c_2))$, in such a way that for all $c \in C$, the induced elements in $\pi^{-1}(\sigma_i(f_i(c)))$ are the same for $i = 1, 2$. This is precisely the data needed for a unique element of the set $\Gamma(\sigma_1) \times_{\Gamma(\sigma)} \Gamma(\sigma_2)$; this proves the claim that the map in (3) is a bijection.

It now follows that the fiber product of Diagram (2) is the table

$$\tau_1 \times_\tau \tau_2 \colon K_1 \times_K K_2 \longrightarrow \Gamma(\sigma_1 \amalg_\sigma \sigma_2)$$

obtained by taking the fiber product of sources and targets in (2), and the induced map between them.

□

Proposition 3.1.5 gives the formula for the join of two tables over a third. As one sees from the construction, the columns of the join are the union of the columns of the given tables, and the key set is the fiber product of the key sets of the given tables.

Lemma 3.1.6. *Let $\sigma \colon C \to \mathbf{DT}$ denote a simple schema. The category $\mathbf{Tables}_\sigma$ of tables on σ is closed under small limits and colimits.*

Proof. The category of sets is closed under small limits and colimits. To take the limit or colimit of a diagram $X \colon I \to \mathbf{Tables}_\sigma$, simply take the limit or colimit (respectively) of the underlying diagram of key sets – see Definition 3.1.2. This set comes with a natural map to $\Gamma(\sigma)$, and one shows easily that it is the limit or colimit (respectively) of X.

□

Example 3.1.7. Let $\sigma \colon C \to \mathbf{DT}$ denote a simple schema. The initial and final objects in $\mathbf{Tables}_\sigma$ are $\emptyset \to \Gamma(\sigma)$ and $\mathrm{id}_{\Gamma(\sigma)} \colon \Gamma(\sigma) \to \Gamma(\sigma)$, respectively.

Construction 3.1.8. Let $\tau \colon K \to \Gamma(\sigma)$ be a table with simple schema $\sigma \colon C \to \mathbf{DT}$, and let $C' \subset C$ be a sublist of its column list. There is an induced table

$$\tau|_{C'} \colon K \to \Gamma(\sigma|_{C'}).$$

In SQL this construction is called the *projection* of τ onto the sublist $C' \subset C$ of columns.

Using the projection query, one can realize a SELECT query as a limit of databases.

Construction 3.1.9. Let us construct the SELECT query. One begins with a table $\tau \colon K \to \Gamma(\sigma)$ with simple schema $\sigma \colon C \to \mathbf{DT}$, from which to select. Let $f \colon C' \subset C$ be a sublist of its columns, and let $\sigma' = \sigma|_{C'} \colon C' \to \mathbf{DT}$ be the restricted simple schema. One may select from τ all records whose restriction to C' is a member of some list. We encode this list as a table $\tau' \colon K' \to \Gamma(\sigma')$ on σ'.

In order to select from τ all records whose restriction to C' is in the table τ', take the limit of the diagram

$$\begin{tikzcd}
K \arrow[r, "\tau"] \arrow[d, "f^*\circ\tau"'] & \Gamma(\sigma) \arrow[d, "f^*"] \\
\Gamma(\sigma') \arrow[r, "\mathrm{id}"] & \Gamma(\sigma') \\
K' \arrow[u, "\tau'"] \arrow[r, "\tau'"'] & \Gamma(\sigma'). \arrow[u, "\mathrm{id}"']
\end{tikzcd}$$

This limit is the desired SELECT query.

Example 3.1.10. Let $\tau\colon K \to \Gamma(\sigma)$ be the table from Example 2.3.6. To select all instances for which the first name is Barack, let $C' = \{\text{'First Name'}\}$. Let τ' denote the one-row table

K'	'First Name'
k'	Barack

Both τ and τ' have a canonical map to the terminal table on C', the table with one column ('First Name') and with a row for each element of **Strings**. Of course, this terminal table is too big to write down, but we do not need it. The fiber product is easily computed to be the table

K	'First Name'	'Last Name'	'BYear'
1	Barack	Obama	1961
'foo'	Barack	Obama	1961

We conclude this section by a quick remark on the category-theoretic properties of the relational tables.

Remark 3.1.11. Relations behave much like ordinary tables. Limits exist in **Rel** and $\mathbf{Rel}_\sigma$. The functor **Rel** → **Tables** preserves limits, and the functor $\mathbf{Rel}_\sigma \to \mathbf{Tables}_\sigma$ preserves limits but *does not* preserve colimits.

We take the viewpoint that the "correct" way to take a colimit of a diagram $X\colon I \to \mathbf{Rel}_\sigma$ is to pass to the diagram $I \to \mathbf{Tables}_\sigma$ and take its colimit instead. This claim, in particular, says that sometimes UNION ALL is more appropriate than UNION is. Since UNION ALL is not legal in the strict relational database theory (or it would be the same as UNION), our viewpoint could be seen as controversial to purists of the relational model.

4. Schemas and databases

A relational database is a set of relations, together with a system of keys and foreign keys which link the relations together. The definition of relations themselves is, of course, quite mathematically precise. However, the precise way in which these relations are allowed to be linked together is rarely written down as a mathematical structure in its own right, either in research papers or textbooks (we could not find it in [Dat05] or [EN07], for example). For example, ER diagrams are exemplified or even defined, but not as a mathematical object (like relations are). There are exceptions, such as [RW92, 2.1], but as far as we know, these definitions are not actually the ones used, either by practitioners or by theorists.

In this section we will define simplicial databases in a rigorous way (see Definition 4.3.3). Although examples will be plentiful, they will never stand in for precise definitions. We will also define morphisms of databases, thus making explicit the idea of "data-preserving maps." Providing a new and precise definition of the category of databases may be useful to database theorists, as well as to people interested in studying mathematical informatics.

4.1. **Schemas.** Roughly, a simplicial set is a picture that can be drawn with vertices, edges, solid triangles, solid tetrahedra, and solid "higher-dimensional tetrahedra." For any integer $n \geq 0$, an n-dimensional solid tetrahedron, or *n-simplex*, is the "diagonal triangle" shape in $\mathbb{R}^{n+1}$ given by the algebraic equation $x_1 + x_2 + \cdots + x_{n+1} = 1$ and the inequalities $x_i \geq 0$ for $1 \leq i \leq n+1$. To draw with these shapes is to connect various tetrahedra together along their faces (or subfaces). For example, one could connect four triangles together along various faces to obtain an empty tetrahedron, the boundary of the 3-simplex.

Simplicial sets are a fundamental tool in algebraic topology, and are important in many other fields within mathematics, such as combinatorial commutative algebra. See [Fri08] or [GJ99] for details.

A database is a system of tables which are connected together via foreign keys. This information is part of the schema for the database. In our formulation, we keep track of this information using (something akin to) simplicial sets as our schemas. Tables are connected together when the corresponding simplices are connected.

We use a slight variant of simplicial sets, which we will define in Definition 4.1.2. Namely, since columns can only take entries in a given data type, we must keep track of this information. For this reason, the simplicial sets we use as schemas have labeled vertices, where each label is an element of $\mathbf{DT}$. We do not define schemas exactly this way, however, because a more generalizable way to phrase it may be useful for future generalizations.

Remark 4.1.1. As mentioned in Remark 2.2.4, some prefer the columns of each table in a database to be unordered, whereas we have chosen to consider them as an ordered set. Simply using symmetric simplicial sets, a variant of simplicial sets in which vertices are unordered, will solve any such issue. See [Gra01] for details on symmetric simplicial sets.

Definition 4.1.2. Let $\mathbf{\Delta}$ denote the category of finite totally ordered sets, let $\pi\colon U \to \mathbf{DT}$ be a type specification, and let

$$\mathcal{S} \cong (\mathbf{\Delta} \downarrow \mathbf{DT})$$

denote the category of simple schemas on π (see Definition 2.2.3 and Remark 2.2.7). We define *the category of schemas on* π, denoted $\mathbf{Sch}^{\pi}$ to be the category whose objects are functors $X\colon \mathcal{S}^{\mathrm{op}} \to \mathbf{Sets}$ and whose morphisms are natural transformations of functors.

Let $X \in \mathbf{Sch}^{\pi}$ denote a schema. Given a simple schema $\sigma\colon C \to \mathbf{DT}$, the *$\sigma$-simplices* of X are the elements of the set $X(\sigma)$, and we write X_σ to denote $X(\sigma)$.

Remark 4.1.3. Given a category $\mathcal{C}$, the category whose objects are functors $\mathcal{C}^{\mathrm{op}} \to \mathbf{Sets}$ and whose morphisms are natural transformations of functors is called *the category of presheaves on* $\mathcal{C}$ and denoted $\mathbf{Pre}(\mathcal{C})$. It is a common mathematical construction which "formally adds all colimits to $\mathcal{C}$." That is, $\mathbf{Pre}(\mathcal{C})$ is closed under taking colimits, and for any functor $\mathcal{C} \to \mathcal{D}$ to a category $\mathcal{D}$ which is closed under taking colimits, there is a unique colimit-preserving functor $\mathbf{Pre}(\mathcal{C}) \to \mathcal{D}$ over $\mathcal{C}$. See, for example, [MLM94, I.5.4].

Thus, we have $\mathbf{Sch}^\pi = \mathbf{Pre}(\mathcal{S}^\pi)$. Since $\mathcal{S}^\pi$ signifies the category of ways to set up columns of a tables, $\mathbf{Pre}(\mathcal{S}^\pi)$ is the category of ways to glue such things together.

Remark 4.1.4. The category of (augmented) simplicial sets is the category $\mathbf{Pre}(\mathbf{\Delta})$. The only difference between it and $\mathbf{Pre}(\mathcal{S}^\pi) \cong \mathbf{Pre}(\mathbf{\Delta} \downarrow \mathbf{DT})$ is that each simplex in $\mathbf{Sch}^\pi$ has labeled vertices, whereas simplices in $\mathbf{Pre}(\mathbf{\Delta})$ do not. In the introduction to this section we described simplicial sets in terms of tetrahedra. After making the necessary modifications, we see that a schema is constructed by gluing together labeled tetrahedra along their faces, where we only allow these tetrahedra to be glued if their labels match.

If X is a schema, we sometimes refer to the simplices of its underlying simplicial set as simplices of X. Thus, the n-simplices of X is the union of all σ-simplices of X, where $\sigma\colon C \to \mathbf{DT}$ is a simple schema with cardinality $\mathrm{card}(C) = n+1$. That is, we write

$$X_n = \coprod_{\{\sigma\colon C \to \mathbf{DT} | \mathrm{card}(C) = n+1\}} X_\sigma.$$

There is a classifying map $s\colon X_0 = \amalg_{a \in \mathbf{DT}}(X_a) \to \mathbf{DT}$ which sends all of X_a to a, for each $a \in \mathbf{DT}$.

One of the best features of the schemas we are presenting here is their geometric nature, as described in the first paragraph of this section. Unfortunately, Definition 4.1.2 does not make the geometry explicit at all. Hopefully the next few examples will help make it more clear.

Example 4.1.5. Let $\sigma\colon C \to \mathbf{DT}$ denote a simple schema. It naturally defines a schema $X = \Delta^\sigma$ as the functor which sends a simple schema $\sigma'\colon C' \to \mathbf{DT}$ to the set $X_{\sigma'} = \mathrm{Hom}_{\mathcal{S}}(\sigma', \sigma)$. If C has $n+1$ elements, one visualizes Δ^σ as an n-dimensional tetrahedron whose vertices are labeled by elements in the image of σ.

This is not just a heuristic: there is a *geometric realization* functor $Re : \mathbf{Sch} \to \mathbf{Top}$ which realizes every schema as a topological space in a natural way, and behaves as we have described for simplices Δ^σ.

As an example, suppose C has two elements and their images under σ are $a, b \in \mathbf{DT}$. We imagine Δ^σ as a line segment, whose vertices are labeled a and b. If C' has three elements and σ' sends two of them to a and one of them to b, we imagine $\Delta^{\sigma'}$ as a filled-in triangle, whose vertices are labeled a, a, and b. The figures we have imagined are the images of σ and σ' under Re.

Definition 4.1.6. Let $\sigma \in \mathcal{S}$ denote a simple schema. The schema $\Delta^\sigma \in \mathbf{Sch}$ defined in Example 4.1.5 is called *the σ-simplex* and, as a functor $\mathcal{S}^{\mathrm{op}} \to \mathbf{Sets}$, is said to be *represented by σ*.

Example 4.1.7. We have mentioned that every object in $\mathbf{Sch}^\pi$ can be obtained by gluing together simplices. This is proven in [Bor94a, 2.15.6]. Let us explain how we would construct the union X of two edges along a common vertex. Suppose that the common vertex is labeled b and the other vertices are labeled a and c. The schema X is obtained as the colimit of the diagram

$$\Delta^{(a,b)} \leftarrow \Delta^{(b)} \rightarrow \Delta^{(b,c)}$$

taken in $\mathbf{Sch}^{\pi}$.

We will now write down this schema explicitly as a presheaf on $\mathcal{S}^{\pi}$, i.e. as a functor $X\colon (\mathbf{\Delta} \downarrow \mathbf{DT})^{\mathrm{op}} \rightarrow \mathbf{Sets}$. Given $\sigma\colon C \rightarrow \mathbf{DT}$, we let X_{σ} be a single element if the image of σ is contained in $\{a, b\}$ or contained in $\{b, c\}$. Otherwise we take X_{σ} to be the empty set.

Example 4.1.8. A basic example of a schema is that of a set of labeled vertices with no edges or higher simplices connecting them. This is obtained as a coproduct of 0-simplices (see Remark 4.1.3), and it is called a *discrete schema.*

4.2. **Sheaves on a schema.**

Definition 4.2.1. Let $X \in \mathbf{Sch}^{\pi}$ denote a schema. A *subschema of* X consists of a schema $X' \in \mathbf{Sch}^{\pi}$ such that for every $\sigma \in \mathcal{S}^{\pi}$ we have $X'_{\sigma} \subset X_{\sigma}$. The subschemas of X form a category $\mathbf{Sub}(X)$, in which there is a morphism $X'' \rightarrow X'$ in $\mathbf{Sub}(X)$ if and only if X'' is a subschema of X'.

We will soon be discussing colimits in the category $\mathbf{Sub}(X)$. One should note that $\mathbf{Sub}(X)$ is particularly nice, in that the colimit of any diagram $D\colon I \rightarrow \mathbf{Sub}(X)$ is the smallest subschema $X' \subset X$ which contains $D(i)$ for all $i \in I$. In the language of lattices or locales, one writes $\mathrm{colim}(D) = \bigvee_{i \in I} D(i)$. See [Bor94b, 1.3].

Definition 4.2.2. We define a *sheaf on* X to be a functor $\mathcal{K}\colon \mathbf{Sub}(X)^{\mathrm{op}} \rightarrow \mathbf{Sets}$ such that, for every diagram $D\colon I \rightarrow \mathbf{Sub}(X)$, the natural map

$$\mathcal{K}(\mathrm{colim}(D)) \longrightarrow \lim(\mathcal{K}(D))$$

is an isomorphism. That is, $\mathcal{K}$ must send colimits of subschemas to corresponding limits of sets.

A *morphism of sheaves on* X is a natural transformation of functors $\mathbf{Sub}(X)^{\mathrm{op}} \rightarrow \mathbf{Sets}$. Let $\mathbf{Shv}(X)$ denote the category of sheaves on X.

Remark 4.2.3. Category theory experts will recognize $\mathbf{Shv}(X)$ as the category of sheaves on a certain Grothendieck site (the locale of subobjects of X). It is well known that $\mathbf{Shv}(X)$ is therefore closed under small limits and colimits. Moreover, there is an adjunction

$$\begin{tikzcd}\mathbf{Pre}(X) \arrow[r, shift left, "Sh"] & \mathbf{Shv}(X) \arrow[l, shift left]\end{tikzcd}$$

for which the right adjoint is the forgetful functor and the left adjoint is called *sheafification.* Roughly, one sheafifies a presheaf on a schema by replacing its value on each union of simplices by the fiber product of its values on those simplices. See [MLM94] for details.

Example 4.2.4. For any schema X, there is an object $\emptyset \in \mathbf{Sub}(X)$, which is the colimit of the empty diagram on $\mathbf{Sub}(X)$. Hence if $\mathcal{K}$ is to be a sheaf on X, one must have $\mathcal{K}(\emptyset) \cong \{*\}$.

If X is a discrete schema (see Example 4.1.8), then X is the coproduct its 0-simplices. Thus, if $\mathcal{K}$ is to be a sheaf on X, we must have

$$\mathcal{K}(X) = \prod_{x \in X_0} \mathcal{K}(x).$$

Example 4.2.5. Suppose that $X \in \mathbf{Sch}^{\pi}$ is the schema $\Delta^{(\text{`}\mathbf{Str}\text{'},\text{`}\mathbb{Z}\text{'})}$, which looks like this:

$$^{\text{`}\mathbf{Str}\text{'}}\bullet\!\!-\!\!\!-\!\!\!-\!\!\!-\!\!\bullet^{\text{`}\mathbb{Z}\text{'}}\,.$$

The category $\mathbf{Sub}(X)$ is a partially ordered set with five objects: $\emptyset$, $\bullet^{\text{`}\mathbf{Str}\text{'}}$,$\bullet^{\text{`}\mathbb{Z}\text{'}}$, $(\bullet^{\text{`}\mathbf{Str}\text{'}}, \bullet^{\text{`}\mathbb{Z}\text{'}})$, and X itself; $\mathbf{Sub}(X)$ has inclusions as morphisms.

A sheaf $\mathcal{K} \in \mathbf{Shv}(X)$ assigns a set to each of these five objects, and functions to each inclusion. However, by Example 4.2.4, it must assign to $\emptyset$ the terminal set, $\mathcal{K}(\emptyset) = \{*\}$, and it must assign to $(\bullet^{\text{`}\mathbf{Str}\text{'}}, \bullet^{\text{`}\mathbb{Z}\text{'}})$ the product $\mathcal{K}(\bullet^{\text{`}\mathbf{Str}\text{'}}) \times \mathcal{K}(\bullet^{\text{`}\mathbb{Z}\text{'}})$. Thus, to specify a sheaf, we need only specify two values, and one morphism, namely $\mathcal{K}(X) \to \mathcal{K}(\bullet^{\text{`}\mathbf{Str}\text{'}}) \times \mathcal{K}(\bullet^{\text{`}\mathbb{Z}\text{'}})$.

For example we may choose on objects the assignments $\mathcal{K}(X) = \{4, cc, 10\}$, $\mathcal{K}(\bullet^{\text{`}\mathbf{Str}\text{'}}) = \{1, 2\}$, and $\mathcal{K}(\bullet^{\text{`}\mathbb{Z}\text{'}}) = \{x, y, z\}$; this implies $\mathcal{K}((\bullet^{\text{`}\mathbf{Str}\text{'}}, \bullet^{\text{`}\mathbb{Z}\text{'}}))$ is isomorphic to $\{1x, 1y, 1z, 2x, 2y, 2z\}$. Any function from $\{4, cc, 10\}$ to this six element set, say $4 \mapsto 1x, cc \mapsto 2z, 10 \mapsto 2z$, defines the restriction maps in our sheaf $\mathcal{K}$. These restriction maps can be thought of as "foreign keys."

Definition 4.2.6. Given a schema $X \in \mathbf{Sch}^{\pi}$, we have been working with the category $\mathbf{Sub}(X)$ of subschemas of X. There is a related category, called *the category of nonempty non-degenerate simple schemas over* X and denoted $\mathbf{ND}(X)$, whose objects are monomorphisms $\Delta^{\sigma} \hookrightarrow X$ in $\mathbf{Sch}^{\pi}$, where $\sigma\colon C \to \mathbf{DT}$ is a schema with $C \neq \emptyset$ (see Example 4.1.5), and whose morphisms are commutative triangles.

Every simplex in a schema has a unique underlying non-degenerate simplex (of which it is the degeneracy), so one can define a functor $\mathbf{ND}\colon \mathbf{Sch}^{\pi} \to \mathbf{Cat}$.

Since every injection $\Delta^{\sigma} \hookrightarrow X$ is in particular a subschema, there is an obvious functor

$$\mathbf{ND}(X) \to \mathbf{Sub}(X).$$

This induces an adjunction $\mathbf{Pre}(\mathbf{ND}(X)) \rightleftarrows \mathbf{Pre}(\mathbf{Sub}(X))$. No nontrivial unions exist in $\mathbf{ND}(X)$, so this adjunction becomes

$$\begin{tikzcd}\mathbf{Pre}(\mathbf{ND}(X)) \arrow[r, shift left, "L"] & \mathbf{Shv}(\mathbf{Sub}(X)), \arrow[l, shift left, "R"]\end{tikzcd}$$

where $\mathbf{Pre}(\mathbf{ND}(X))$ is the category of presheaves $\mathbf{ND}(X)^{\mathrm{op}} \to \mathbf{Sets}$. See [Joh02, C.1.4.3] for more details on this type of construction.

Proposition 4.2.7. *Let $X \in \mathbf{Sch}^{\pi}$ be a schema, and let $\mathbf{ND}(X)$ denote the category of non-degenerate nonempty simple schemas over X. The adjunction*

$$\begin{tikzcd}\mathbf{Pre}(\mathbf{ND}(X)) \arrow[r, shift left, "L"] & \mathbf{Shv}(\mathbf{Sub}(X)), \arrow[l, shift left, "R"]\end{tikzcd}$$

is an equivalence of categories.

Proof. It is an easy exercise to show that the composition $L \circ R$ is equal to the identity on $\mathbf{Pre}(\mathbf{ND}(X))$ and that $K \circ L$ is canonically isomorphic to the identity on $\mathbf{Shv}(\mathbf{Sub}(X))$.

□

Proposition 4.2.7 says that one does not have to worry about sheaves: the category $\mathbf{Shv}(X)$ is equivalent to a category of functors (without "sheaf" requirements).

Lemma 4.2.8. *Let $\pi\colon U \to \mathbf{DT}$ denote a type specification and let $f\colon X \to Y$ denote a morphism of schemas on π. There is an adjunction*

$$\begin{tikzcd}\mathbf{Shv}(\mathbf{Sub}(Y)) \arrow[r, shift left, "f^*"] & \mathbf{Shv}(\mathbf{Sub}(X)) \arrow[l, shift left, "f_*"]\end{tikzcd}$$

defined as follows for sheaves $\mathcal{K}_X \in \mathbf{Shv}(\mathbf{Sub}(X))$ and $\mathcal{K}_Y \in \mathbf{Shv}(\mathbf{Sub}(Y))$. For any $V_X \in \mathbf{Sub}(X)$ we take

$$f^*\mathcal{K}_Y(V_X) \coloneqq \mathcal{K}_Y(f(V_X)),$$

where $f(V_X) \in \mathbf{Sub}(Y)$ is the image of V_X in Y. For any $V_Y \in \mathbf{Sub}(Y)$ we take

$$f_*\mathcal{K}_X(V_Y) \coloneqq \mathcal{K}_X(f^{-1}(V_Y)),$$

where $f^{-1}(V_Y)$ is the preimage of V_Y in X.

Proof. Colimits of presheaves are computed objectwise, and it follows from Proposition 4.2.7 that the functor f^*, defined above, preserves colimits. Hence, it suffices to show that for any representable sheaf $rY' = \mathrm{Hom}_{\mathbf{Sub}(Y)}(-, Y') \in \mathbf{Shv}(\mathbf{Sub}(Y))$ and sheaf $T \in \mathbf{Shv}(\mathbf{Sub}(X))$, one has an isomorphism

$$\mathrm{Hom}(f^*(rY'), T) \cong^? \mathrm{Hom}(rY', f_*T).$$

To begin, note that for any $U \in \mathbf{Sub}(X)$ one has a chain of natural isomorphisms

$$\begin{aligned} f^*(rY')(U) \coloneqq (rY')(f(U)) &\cong \mathrm{Hom}_{\mathbf{Sub}(Y)}(f(U), Y') \\ &\cong \mathrm{Hom}_{\mathbf{Sub}(X)}(U, f^{-1}(Y')) \cong r(f^{-1}(Y'))(U). \end{aligned}$$

That is, $f^*(rY') \cong r(f^{-1}(Y'))$. By another chain of natural isomorphisms, we have

$$\begin{aligned} \mathrm{Hom}(f^*(rY'), T) &\cong \mathrm{Hom}(r(f^{-1}(Y')), T) \\ &\cong T(f^{-1}(Y')) \\ &=: f_*T(Y') = \mathrm{Hom}(rY', f_*T). \end{aligned}$$

This proves the lemma.

□

4.3. **Simplicial databases.** We think of a schema as a way of organizing the data in a database. Before we say what a database is, let us give one more example of a schema. In some sense it will be the fundamental example of a schema; however, it should not really be thought of as a way to organize the data, but as the meaning of the data itself.

Example 4.3.1. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification, and let $\mathcal{S} = \mathcal{S}^\pi$ denote the category of simple schemas on π. Let $\Gamma^\pi\colon \mathcal{S}^{\mathrm{op}} \to \mathbf{Sets}$ denote the functor which assigns to a schema $\sigma\colon C \to \mathbf{DT}$ the set $\Gamma^\pi(\sigma)$ of records on σ (see Definition 2.3.1).

By Lemma 2.3.7, a map $\sigma \to \sigma'$ induces a function $\Gamma^\pi(\sigma') \to \Gamma^\pi(\sigma)$, so Γ^π is indeed a contravariant functor. By definition we can consider Γ^π as a schema on π and write $\Gamma^\pi \in \mathbf{Sch}^\pi$.

We call Γ^π *the universal record on π,* for reasons which will be clear soon. If the type specification $\pi\colon U \to \mathbf{DT}$ is obvious from context, we may denote Γ^π simply by Γ.

Definition 4.3.2. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification, let Γ^π denote the universal record on π, and let $X \in \mathbf{Sch}^\pi$ denote a schema on π. The *universal sheaf on X of type* π is the sheaf $\mathcal{U}^\pi$ whose value on a subschema $X' \subset X$ is the set

$$\mathcal{U}^\pi(X') = \mathrm{Hom}_{\mathbf{Sch}^\pi}(X', \Gamma^\pi).$$

Each element of $\mathcal{U}^\pi(X')$ is called a *record on X' of type* π. If π is clear from context, we may write $\mathcal{U}$ to denote $\mathcal{U}^\pi$.

Now let $X, Y \in \mathbf{Sch}^\pi$ be schema and let $\mathcal{U}_X$ and $\mathcal{U}_Y$ denote the universal sheaf of type π on X and Y, respectively. A map of schemas $f\colon Y \to X$ induces a morphism $\mathcal{U}_f\colon f^*\mathcal{U}_X \to \mathcal{U}_Y$ as follows. Let $Y' \subset Y$ denote an object in $\mathbf{Sub}(Y)$; then composing with f induces a natural map

$$f^*\mathcal{U}_X(Y') = \mathrm{Hom}_{\mathbf{Sch}^\pi}(f(Y'), \Gamma^\pi) \longrightarrow \mathrm{Hom}_{\mathbf{Sch}^\pi}(Y', \Gamma^\pi) = \mathcal{U}_Y(Y'),$$

which we denote $\mathcal{U}_f$; it is similarly defined on morphisms.

Definition 4.3.3. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification. A *simplicial database (* or simply *database) of type* π is a triple $(X, \mathcal{K}, \tau)$ where $X \in \mathbf{Sch}^\pi$ is a schema of type π, $\mathcal{K} \in \mathbf{Shv}(X)$ is a sheaf of sets on $\mathbf{Sub}(X)$, and $\tau\colon \mathcal{K} \to \mathcal{U}_X$ is a morphism of sheaves on X (see Definition 4.3.2). We refer to X as *the schema*, $\mathcal{K}$ as *the sheaf of keys*, and τ as *the data* of the database $(X, \mathcal{K}, \tau)$.

Remark 4.3.4. Given a set of ways to measure objects, it often happens that we have several objects with the same measurements. For example, we may have three green apples, or two 1999 Toyota Corollas. In relational databases, if two objects have the same attributes, then they are taken to be the same instance. To keep them distinct, one introduces a unique identifier, an artificial key, which becomes part of the data. This causes problems with database integration, because the arbitrarily-chosen artificial keys in one database will generally not match with those in another.

In our definition, the keys for the data are kept separate, as the sheaf of sets $\mathcal{K}$. Different names for the keys in no way affect the data itself and therefore do not interfere with database integration. We say more about this in Section 5.3.

Example 4.3.5. In Example 4.2.5, we wrote down a sheaf $\mathcal{K} \in \mathbf{Shv}(X)$ on the schema

$$X = {}^{\text{‘}\mathbf{Str}\text{’}}\bullet\!\!\longrightarrow\!\!\bullet^{\text{‘}\mathbb{Z}\text{’}},$$

and we will continue to use it in this example. To specify a database on X of type π, we must give a morphism $\tau\colon \mathcal{K} \to \mathcal{U}^\pi$ of sheaves on X.

We defined the universal sheaf $\mathcal{U}_X$ of type π on X in Definition 4.3.2. We have

$$\begin{aligned}
\mathcal{U}_X(X) = \mathcal{U}_X((\bullet^{\text{‘}\mathbf{Str}\text{’}}, \bullet^{\text{‘}\mathbb{Z}\text{’}})) &= \mathbf{Str} \times \mathbb{Z} \\
\mathcal{U}_X(\bullet^{\text{‘}\mathbf{Str}\text{’}}) &= \mathbf{Str} \\
\mathcal{U}_X(\bullet^{\text{‘}\mathbb{Z}\text{’}}) &= \mathbb{Z} \\
\mathcal{U}_X(\emptyset) &= \{*\}.
\end{aligned}$$

To define a map $\tau\colon \mathcal{K} \to \mathcal{U}_X$, we must give maps

$$\tau(\bullet^{\text{‘}\mathbf{Str}\text{’}})\colon \mathcal{K}(\bullet^{\text{‘}\mathbf{Str}\text{’}}) \to \mathcal{U}_X(\bullet^{\text{‘}\mathbf{Str}\text{’}}), \qquad \tau(\bullet^{\text{‘}\mathbb{Z}\text{’}})\colon \mathcal{K}(\bullet^{\text{‘}\mathbb{Z}\text{’}}) \to \mathcal{U}_X(\bullet^{\text{‘}\mathbb{Z}\text{’}})$$

and

$$\tau(X)\colon \mathcal{K}(X) \to \mathcal{U}_X(X)$$

that compose correctly with the restriction maps. We arbitrarily assign

$$\begin{array}{rclrcl} \tau(1) & = & \text{Barack} & \tau(x) & = & 1961 \\ \tau(2) & = & \text{Michelle} & \tau(y) & = & 1946 \\ & & & \tau(z) & = & 1964. \end{array}$$

Now $\mathcal{K}(X) = \{4, cc, 10\}$, and the restriction map sends $4 \mapsto 1x$, $cc \mapsto 2z$, and $10 \mapsto 2z$. This forces $\tau(4) = (\text{Barack}; 1961)$ and $\tau(cc) = \tau(10) = (\text{Michelle}; 1964)$. The other values and restriction maps for $\mathcal{K}$ are now also forced.

Example 4.3.6. In Example 4.3.5, we followed the definitions very closely, perhaps to the detriment of the big ideas. In this example, we write down how the sheaf "looks" as a collection of tables.

Let us first change the schema X very slightly, by instead using the schema $\sigma\colon \{\text{First, BYear}\} \to \mathbf{DT}$, where $\sigma(\text{First}) = \text{'Str'}$ and $\sigma(\text{BYear}) = \text{'}\mathbb{Z}\text{'}$, and now taking $X = \Delta^\sigma$. The only difference is that we have labeled our columns by more specific attribute names. We write $\tau(X)\colon \mathcal{K}(X) \to \mathcal{U}_X(X)$ as the table

$\tau(X) =$

$\mathcal{K}(X)$	First	BYear
4	Barack	1961
cc	Michelle	1964
10	Michelle	1964

We write $\tau(\bullet^{\text{First}})$ and $\tau(\bullet^{\text{BYear}})$ as the tables

$\tau(\bullet^{\text{First}}) =$

$\mathcal{K}(\bullet^{\text{First}})$	First
1	Barack
2	Michelle

$\tau(\bullet^{\text{BYear}}) =$

$\mathcal{K}(\bullet^{\text{BYear}})$	BYear
x	1961
y	1946
z	1964

We can consider the restriction maps $\mathcal{K}(X) \to \mathcal{K}(\bullet^{\text{First}})$ and $\mathcal{K}(X) \to \mathcal{K}(\bullet^{\text{BYear}})$ as foreign keys attached to the $\tau(X)$ table. The way things are set up, this foreign key information is kept in the restriction maps of the sheaf $\mathcal{K}$. See Example 4.2.5.

Definition 4.3.7. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification, let $\mathcal{X} = (X, \mathcal{K}_X, \tau_X)$ and $\mathcal{Y} = (Y, \mathcal{K}_Y, \tau_Y)$ denote databases of type π, and let $\mathcal{U}_X$ and $\mathcal{U}_Y$ denote the universal sheaf on X and Y (see Definition 4.3.2). A *morphism of databases*, denoted

$$(f, f^\sharp)\colon \mathcal{X} \to \mathcal{Y},$$

consists of a map $f\colon Y \to X$ of schemas (see Definition 4.1.2) and a morphism of sheaves $f^\sharp\colon f^*\mathcal{K}_X \to \mathcal{K}_Y$ on Y such that the diagram of sheaves

(4)
$$\begin{tikzcd} f^*\mathcal{K}_X \arrow[r, "f^*(\tau_X)"] \arrow[d, "f^\sharp"'] & f^*\mathcal{U}_X \arrow[d, "\mathcal{U}_f"] \\ \mathcal{K}_Y \arrow[r, "\tau_Y"'] & \mathcal{U}_Y \end{tikzcd}$$

commutes.

The *category of simplicial databases on* π, whose objects are simplicial databases as defined in Definition 4.3.3 and whose morphisms have just been defined, is denoted $\mathbf{DB}^\pi$, or simply $\mathbf{DB}$ if π is understood. Fixing a schema X, the *category of databases on* X, denoted $\mathbf{DB}_X$, is the category whose objects are databases with schema X and whose morphisms are identity on X.

Remark 4.3.8. A database is roughly a bunch of tables glued together by foreign key mappings. A morphism of databases is a way to coherently assign to each table in one database, a table in another database, and a morphism between the two tables. Recall that a morphism of tables is a "data-preserving map" (see Definition 2.3.8, Example 2.3.9, and Remark 2.3.10). Thus, a morphism of databases should be thought of as a coherent system of data-preserving maps.

We might make the following definition. A *morphism without integrity* is a pair $(f, f^\sharp)\colon \mathcal{X} \to \mathcal{Y}$ as above, but *without* the requirement that diagram (4) commute.

Remark 4.3.9. Let Y be a schema and let $\mathcal{U}_Y$ denote the universal database on Y. One can identify $\mathbf{DB}_Y$ with the category $\mathbf{Shv}(Y)_{/\mathcal{U}_Y}$ of sheaves over $\mathcal{U}_Y$. Explicitly, this is the category whose objects are arrows $\mathcal{K} \to \mathcal{U}_Y$ and whose morphisms are commutative triangles.

4.4. **Relational simplicial databases.** In this subsection, we present a category of relational databases as a full subcategory of the category $\mathbf{DB}$ of simplicial databases. We also give an adjunction which allows one to convert a database in our sense to a relational database in a functorial way.

Definition 4.4.1. Let π denote a type specification. A simplicial database $\mathcal{X} = (X, \mathcal{K}, \tau)$ on π is called *relational* if $\tau\colon \mathcal{K} \to \mathcal{U}_X$ is a monomorphism of sheaves. The *category of relational simplicial databases*, denoted $\mathcal{R}\mathbf{el}^\pi$ is the full subcategory of $\mathbf{DB}^\pi$ spanned by the relational simplicial databases.

Note the precise similarity of this definition with Definition 2.5.1: the schema X is a gluing together of simple schemas σ, the sheaf $\mathcal{U}_X$ evaluated on a simplex $\Delta^\sigma \subset X$ is $\Gamma(\sigma)$, and a monomorphism of sheaves is a morphism which restricts to an injective function on each simplex.

Every function $f\colon A \to B$ between sets has an image $\mathrm{im}(f) \subset B$ and an injection $f^m\colon \mathrm{im}(f) \to B$; similarly, given a schema X, every morphism $f\colon \mathcal{A} \to \mathcal{B}$ of sheaves of sets on X has an image sheaf denoted $\mathrm{im}(f) \subset \mathcal{B}$ and a monomorphism of sheaves $f^m\colon \mathrm{im}(f) \to \mathcal{B}$. If $\mathcal{X} = (X, \mathcal{K}, \tau)$ is a database, we can take the image sheaf $\mathrm{im}(\tau)$ of $\tau\colon \mathcal{K} \to \mathcal{U}_X$, and the database $(X, \mathrm{im}(\tau), \tau^m)$ will be a relational simplicial database.

Lemma 4.4.2. *Let π denote a type specification. There is an adjunction*

$$\begin{tikzcd}\mathbf{DB}^\pi \arrow[r, shift left] & \mathcal{R}\mathbf{el}^\pi \arrow[l, shift left]\end{tikzcd}$$

in which the left adjoint is given by $(X, \mathcal{K}, \tau) \mapsto (X, \mathrm{im}(\tau), \tau^m)$ and the right adjoint is the forgetful functor which realizes a relational simplicial database as a simplicial database.

Proof. This is a simple exercise that reduces to the fact that the image functor, which sends the category of sets and functions to the category of sets and injections, is a left adjoint to the forgetful functor.

□

Since the forgetful functor $\mathcal{R}\mathbf{el}^\pi \to \mathbf{DB}^\pi$ is fully faithful, the counit of the adjunction in Lemma 4.4.2 is the identity functor on $\mathcal{R}\mathbf{el}^\pi$. Another way to say this is that one does not lose information when considering a relational database as a simplicial database, but one often does lose information when converting a simplicial database to a relational database. Strictly "more information" can be contained in a simplicial database than in a relational database.

4.5. **Tables vs. simplicial databases.** In this last subsection we present the functor $F\colon \mathbf{Tables} \to \mathbf{DB}$ which realizes a table as a simplicial database. We will also present the "global table" construction, which roughly takes a database and joins everything together to make one big (unnormalized!) table.

Construction 4.5.1. Let $\pi\colon U \to \mathbf{DT}$ denote a type specification and (K, C, σ, τ) a table on π (see Definition 2.3.3). Let $X = \Delta^{\sigma} \in \mathbf{Sch}^{\pi}$ be the associated schema, let $\mathcal{U}_X$ denote the universal database on X, and let $\mathcal{K}_X$ denote the constant sheaf on $\mathbf{Sub}(X)$ which takes each subschema to the set K. Define $\tau_X : \mathcal{K}_X \to \mathcal{U}_X$ in the unique way such that $\tau_X(X)\colon \mathcal{K}_X(X) \to \mathcal{U}_X(X)$ is the function $\tau\colon K \to \Gamma(\sigma)$. We are ready to assign

$$F((K, C, \sigma, \tau)) \coloneqq (X, \mathcal{K}_X, \tau_X).$$

Given a map of tables $\varphi\colon (K_1, C_1, \sigma_1, \tau_1) \to (K_2, C_2, \sigma_2, \tau_2)$, we will now show that there is a canonical map of simplicial databases $(X_1, \mathcal{K}_1, \tau_1) \to (X_2, \mathcal{K}_2, \tau_2)$. Recall from Definition 2.3.8 that $\varphi = (g, f)$ where $g\colon K_1 \to K_2$ is a function and $f\colon \sigma_2 \to \sigma_1$ is a morphism of simple schemas such that Diagram (1), rewritten for the readers convenience here:

$$\begin{tikzcd}
K_1 \arrow[r, "\tau_1"] \arrow[d, "g"'] & \Gamma(\sigma_1) \arrow[d, "f^*"] \\
K_2 \arrow[r, "\tau_2"'] & \Gamma(\sigma_2),
\end{tikzcd}$$

commutes.

The morphism $f\colon \sigma_2 \to \sigma_1$ of simple schemas induces a morphism $\Delta^{\sigma_2} \to \Delta^{\sigma_1}$ of schemas, i.e. a map $f\colon X_2 \to X_1$. The sheaf $f^*\mathcal{K}_1$ on X_2 is the constant sheaf with value K_1, so g gives a map $f^\sharp\colon f^*\mathcal{K}_1 \to \mathcal{K}_2$. We will skip some details, but one can easily show that the commutativity of the Diagram (4) is equivalent to the commutativity of Diagram (1), completing the construction.

We can also extract a single table from a simplicial database, by looking at its global sections. This requires a functor called f_+ defined in Section 5.1. We include the construction here, rather than later, in order to keep like topics together, and conclude nicely with Remark 4.5.3.

Construction 4.5.2. Let $\mathcal{X} = (X, \mathcal{K}, \tau)$ denote a simplicial database. Recall from Remark 4.1.4 that there is an induced classification map $s\colon X_0 \to \mathbf{DT}$. Assuming that X has finitely many vertices, we can construct a table whose simple schema is s.

To do so, we need only note that there is a unique map of schemas $f\colon X \to \Delta^s$. Indeed, given any simplex in X, its set of vertices classifies a unique simplex in Δ^s; this defines f. If we write $K = \mathcal{K}(X) = f_+\mathcal{K}(\Delta^s)$ and $t = f_+\tau_X(\Delta^s)\colon K \to \Gamma(s)$, then we are ready to construct the table

$$(K, X_0, s, t) \in \mathbf{Tables}.$$

Its columns are given by the vertices X_0 of X; its rows are difficult to describe in general, but in specific cases are quite sensible.

Remark 4.5.3. It is not hard to show that the two above constructions establish an adjunction

$$\begin{tikzcd}
\mathbf{Tables} \arrow[r, shift left] & \mathbf{DB} \arrow[l, shift left]
\end{tikzcd}$$

Given a database $\mathcal{X}$, the table obtained by the right adjoint will be called the *global table on* $\mathcal{X}$.

5. Constructions and formal properties of Simplicial Databases

The point of the formalism in Section 4 is to find a language in which to describe databases such that the typical operations performed when working with databases are sensible in the language. In other words, queries of databases should make sense as categorical constructions, as they did in Section 3 for tables.

5.1. **Changing the schema.** Let us begin with some ways that one can import data from one schema into another. In Lemma 4.2.8 we discussed the adjunction

$$\begin{tikzcd}\mathbf{Shv}(\mathbf{Sub}(Y)) \arrow[r, shift left, "f^*"] & \mathbf{Shv}(\mathbf{Sub}(X)) \arrow[l, shift left, "f_*"]\end{tikzcd} \tag{5}$$

induced by a map of schemas $f\colon Y \to X$. Given a database $\mathcal{X} = (X, \mathcal{K}_X, \tau_X)$ on X there is an induced database $(Y, f^*\mathcal{K}_X, \mathcal{U}_f \circ (f^*\tau_X))$, denoted $f^*\mathcal{X}$; see Definition 4.3.2 and refer to the diagram

$$\begin{tikzcd}f^*\mathcal{K}_X \arrow[r, "f^*\tau_X"] \arrow[dr] & f^*\mathcal{U}_X \arrow[d, "\mathcal{U}_f"] \\ & \mathcal{U}_Y.\end{tikzcd}$$

A slightly more complicated construction creates a database on X from a database $\mathcal{Y} = (Y, \mathcal{K}_Y, \tau_Y)$ on Y and a map of schemas $f\colon Y \to X$. By the adjunction (5), we have the diagram

$$\begin{tikzcd} & \mathcal{U}_X \arrow[d] \\ f_*\mathcal{K}_Y \arrow[r, "f_*\tau_Y"'] & f_*\mathcal{U}_Y,\end{tikzcd} \tag{6}$$

but since there is no canonical map $f_*\mathcal{K}_Y \to \mathcal{U}_X$, we have not yet constructed a database on X.

To do so, let $f_+(\mathcal{K}_Y)$ denote the limit of Diagram (6). This sheaf comes with a canonical map to $\mathcal{U}_X$, which we denote $f_+\tau_Y\colon f_+\mathcal{K}_Y \to \mathcal{U}_X$. The triple

$$(X, f_+\mathcal{K}_Y, f_+\tau_Y)$$

is a database on X, which we denote $f_+\mathcal{Y}$.

Proposition 5.1.1. *Let π denote a type specification, and let $f\colon Y \to X$ be a morphism of schemas of type π. The functors f^* and f_+ define an adjunction which we denote by a slight abuse of notation*

$$\begin{tikzcd}\mathbf{DB}_X \arrow[r, shift left, "f^*"] & \mathbf{DB}_Y. \arrow[l, shift left, "f_+"]\end{tikzcd}$$

Proof. Let $\mathcal{X} = (X, \mathcal{K}_X, \tau_X)$ and $\mathcal{Y} = (Y, \mathcal{K}_Y, \tau_Y)$ be databases. Giving a morphism $f^*\mathcal{X} \to \mathcal{Y}$ of databases over Y amounts to a giving a map α of sheaves making the diagram

$$\begin{tikzcd}
f^*\mathcal{K}_X \arrow[r, "f^*\tau_X"] \arrow[d, dashed, "\alpha"'] & f^*\mathcal{U}_X \arrow[d, "\mathcal{U}_f"] \\
\mathcal{K}_Y \arrow[r, "\tau_Y"'] & \mathcal{U}_Y
\end{tikzcd}$$

commute. By the adjunction (5) this diagram is equivalent to the diagram

$$\begin{tikzcd}
\mathcal{K}_X \arrow[r, "\tau_X"] \arrow[d, dashed, "\alpha"'] & \mathcal{U}_X \arrow[d, "\mathcal{U}_f"] \\
f_*\mathcal{K}_Y \arrow[r, "f_*\tau_Y"'] & f_*\mathcal{U}_Y,
\end{tikzcd}$$

by Lemma 4.2.8. Supplying a morphism α making this diagram commute is equivalent to supplying a morphism $\mathcal{K}_X \to f_+\mathcal{K}_Y$ over $\mathcal{U}_X$, because $f_+\mathcal{K}_Y$ is the limit of Diagram 6. The proof now follows from Remark 4.3.9.

□

Definition 5.1.2. Let π denote a type specification, and let $f \colon Y \to X$ be a morphism of schemas of type π. The functor $f^* \colon \mathbf{DB}_X \to \mathbf{DB}_Y$, defined above, is called the *pullback functor*, and the functor $f_+ \colon \mathbf{DB}_Y \to \mathbf{DB}_X$, defined above, is called the *push-forward functor*.

Given a sheaf of sets $\mathcal{K}_X$ on X, we also refer to $f^*\mathcal{K}_X \in \mathbf{Shv}(Y)$ as *the pullback of* $\mathcal{K}_X$, and given a sheaf of sets $\mathcal{K}_Y$ on Y, we also refer to $f_+\mathcal{K}_Y \in \mathbf{Shv}(X)$ as *the push-forward of* $\mathcal{K}_Y$.

Example 5.1.3. Let X and Y be the schemas

$$X := \overset{\text{`}\mathbf{Str}\text{'}}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\overset{\text{`}\mathbb{Z}\text{'}}{\bullet}, \qquad Y := \overset{\text{`}\mathbf{Str}\text{'}}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\overset{\text{`}\mathbf{Str}\text{'}}{\bullet},$$

and let $f \colon Y \to X$ be the unique morphism of schemas between them.

By Remark 4.3.9, a database on X is given by a morphism of sheaves $\tau_X \colon \mathcal{K}_X \to \mathcal{U}_X$, for some sheaf of sets $\mathcal{K}_X$. We roughly think of it as a table of strings and integers, with some values not filled in. (In fact, τ_X has more information because, for example, two keys in $\mathcal{K}(X)$ might be sent to the same key in $\mathcal{K}(\bullet^{\text{`}\mathbf{Str}\text{'}})$).

The pullback database $f^*\tau_X \colon f^*\mathcal{K}_X \to \mathcal{U}_Y$ is degenerate in the sense that every row has the same string repeated in two columns. In some sense, this is to be expected.

Now suppose that $\tau_Y \colon \mathcal{K}_Y \to \mathcal{U}_Y$ is a database on Y. We roughly think of it as a table whose rows are pairs of strings. The push-forward $f_+\tau_Y$ consists of three tables: one has two columns (strings and integers) and the other two just have one column. The one column table of integers $f_+\tau_Y(\bullet^{\text{`}\mathbb{Z}\text{'}})$ is empty. The one column table of strings $f_+\tau_Y(\bullet^{\text{`}\mathbf{Str}\text{'}})$ consists of those strings S for which there is a row in $\tau_Y(Y)$ consisting of a repeated string (S, S). Finally, the two column table $f_+\tau_Y(X)$ consists of an element (S, n) for every row S in the one-column table of strings and every integer $n \in \mathbb{Z}$.

One sees that by this example that if $f \colon Y \to X$ is not surjective, then the pushforward functor f_+ results in huge tables. It is not meant to be implemented as a hash table but as a theoretical construct.

Given a map of schemas $f\colon Y \to X$, there is one more important way to send a database on X to a database on Y, but only if f is a monomorphism of schemas. A monomorphism of schemas corresponds to the relationship often known as "is a", in which every object of type x "is an" object of type y. In this situation, there is a functor which takes as input a database of y's, and produces as output a database of x's with all of the y-information filled in, but nothing else. The functor that accomplishes this task is denoted $f_!\colon \mathbf{DB}_Y \to \mathbf{DB}_X$ and is called "extension by $\emptyset$," meaning that on every simplex in X that is not in $f(Y)$, the value of the sheaf there is an empty table.

To define $f_!$ rigorously, we first notice that $f^*\colon \mathbf{Shv}(X) \to \mathbf{Shv}(Y)$ not only has a right adjoint (f_*), but a left adjoint as well, which we also denote $f_!\colon \mathbf{Shv}(Y) \to \mathbf{Shv}(X)$. If f is a monomorphism, then every subschema $Y' \subset Y$ is sent to a subschema $f(Y') \subset X$.

Let us define $f_!\mathcal{U}_Y$ and its canonical map to $\mathcal{U}_X$. Every subschema $X' \subset X$ is either of the form $X' = f(Y')$ or not. If so, we set $f_!\mathcal{U}_Y(X') = \mathcal{U}_Y(Y') = \mathcal{U}_X(X')$. If not, we set $f_!\mathcal{U}_Y(X') = \emptyset$. There is a canonical map $a_f\colon f_!\mathcal{U}_Y \to \mathcal{U}_X$ which is the identity map on $X' = f(Y')$ and which is $\emptyset \to \mathcal{U}_X(X')$ when $X' \notin \mathrm{im}(f)$.

Now that we have a canonical map $a_f\colon f_!\mathcal{U}_Y \to \mathcal{U}_X$ in the case that $f\colon Y \to X$ is an inclusion, we can define $f_!\colon \mathbf{DB}_Y \to \mathbf{DB}_X$ to be given by

$$f_!(Y, \mathcal{K}_Y, \tau_Y) \coloneqq (X, f_!\mathcal{K}_Y, a_f \circ \tau_Y).$$

The functor $f_!$ is left adjoint to the functor $f^*\colon \mathbf{DB}_X \to \mathbf{DB}_Y$ (but $f_!$ is defined only when $f\colon Y \to X$ is an injection.)

5.2. **Nulls.** Nulls do not conform with the mathematical logic that underlies the strict theoretical foundation of relational databases. They are easy enough to deal with, however, by use of foreign keys. That is, for each column $c \in C$ of a schema $\sigma\colon C \to \mathbf{DT}$ for which a table may contain a null, one creates a new schema σ' on columns $C' = C - \{c\}$. By an easy use of foreign keys, one considers objects classified by σ to be also classified by σ'. This is a way to get around the problem of nulls. Other approaches can be found in [JR03].

The same technique is done (automatically) in simplicial databases. Over a simplex Δ^σ, one puts objects for which the value on each column is known. If the value on some set of columns is unknown for a certain object, it is represented as a record on the subsimplex for which it is total.

If one so desired, he or she could implement simplicial databases so that local sections of the database (records over subschema) appeared as global sections of the database (records over the whole schema) by putting the value "Null" in appropriate places. From our perspective it is preferable just to leave local data as local data and not try to promote it to global data, at least for theoretical purposes.

5.3. **Duplicate records.** SQL allows for a table to have the same record in two different rows. Therefore, tables are not relations and SQL does not strictly implement relational databases. One could argue that SQL is "wrong" in not conforming to the theory (see [Dat05, p. 14]), but perhaps the pure relational theory is overly strict; this is the position we take.

Simplicial databases allow for duplicate entries. This should not be threatening because internal keys ensure the integrity of the data. If $\Gamma = A \times B \times C$, then relations on this simple schema are subsets $K \subset \Gamma$. In the theory of simplicial databases, we allow non-injective functions $\tau\colon K \to \Gamma$, called tables.

Philosophically, we see the relational model as "confusing the object with its attributes." A schema, or set of attributes, gives a set of ways to measure a collection of objects. It is entirely possible that two objects in that collection could have the same measurements according to the schema. In the relational model, these two objects would be *identified* in the sense that only one row of the table would be representing both. From now on, the database and its users will have no choice but to consider these objects to be the same.

The only alternative to this is to introduce arbitrary identifiers, in the form of "Primary Keys." These artificial keys are not part of the data being measured about the objects. In our view, it is best to keep these arbitrary identifiers "internal" to the database management system. Among several advantages, the most obvious is database integration, in which it is important to know what aspects of the data are "measured" and invariant, and what aspects are contrived. We will say more about this in Section 6.5.3.

5.4. **Limits and colimits of databases.** We will see shortly that limits and colimits taken in the category of simplicial databases have meaning in terms of the general theory of databases, such as joins and unions.

Theorem 5.4.1. *Let $\pi\colon U \to \mathbf{DT}$ denote a type specification. The category $\mathbf{DB}^\pi$ of databases of type π is closed under taking small colimits and small limits.*

Proof. Let I denote a small category and let $\mathcal{X}\colon I \to \mathbf{DB}$ denote an I-shaped diagram in $\mathbf{DB} = \mathbf{DB}^\pi$. There is a functor $\mathbf{DB} \to \mathbf{Sch}^{\mathrm{op}}$ taking a database $(A, \mathcal{K}_A, \tau_A)$ to its underlying schema A, and composing this functor with $\mathcal{X}$ gives a functor which we denote $X\colon I \to \mathbf{Sch}^{\mathrm{op}}$. For an object $i \in I$, we denote the database $\mathcal{X}(i)$ by $\mathcal{X}_i$ and write

$$\mathcal{X}_i = (X_i, \mathcal{K}_i, \tau_i).$$

To define the colimit (respectively limit) of the diagram $\mathcal{X}$, we must first specify its schema. Since $\mathbf{Sch} = \mathbf{Pre}(\mathcal{S})$, where $\mathcal{S}$ is the category of simple schemas (see Definition 2.2.6), it is closed under colimits and limits ([MLM94, p. 22]); hence so is $\mathbf{Sch}^{\mathrm{op}}$. Let $C = \mathrm{colim}(X)$ (resp. $L = \lim(X)$) denote the colimit (resp. limit) of the diagram $X\colon I \to \mathbf{Sch}^{\mathrm{op}}$. Let $\mathcal{U}_C$ and $\mathcal{U}_L$ denote the universal databases on C and L, respectively.

As a colimit in $\mathbf{Sch}^{\mathrm{op}}$, the schema C comes equipped with morphisms in $c_i\colon C \to X_i$ in $\mathbf{Sch}$, for each $i \in I$, making the appropriate diagrams commute. There is a pullback sheaf $c_i^*\tau\colon c_i^*\mathcal{K}_i \to \mathcal{U}_C$. If $f\colon i \to j$ is a morphism in I, then the map $X_j \to X_i$ in $\mathbf{Sch}$ induces a morphism

$$c_i^*\mathcal{K}_i \to c_j^*\mathcal{K}_j$$

of pullback sheaves over $\mathcal{U}_C$ on C. Let $c^*\colon I \to \mathbf{Shv}(C)_{/\mathcal{U}_C}$ denote the I-shaped diagram of these pullback sheaves over $\mathcal{U}_C$. Define $\tau_C\colon \mathcal{K}_C \to \mathcal{U}_C$ to be the colimit of this diagram. Then the database

$$\mathcal{C} = (C, \mathcal{K}_C, \tau_C)$$

is our candidate for the colimit of the diagram $\mathcal{X}$. It is a matter of tracing through the construction to show that $\mathcal{C}$ has the necessary universal property.

Defining the limit of $\mathcal{X}$ is similar. As a limit in $\mathbf{Sch}^{\mathrm{op}}$, the schema L comes equipped with morphisms $\ell_i\colon X_i \to L$ in $\mathbf{Sch}$, for each $i \in I$, making the appropriate diagrams commute. There is a push-forward sheaf $(\ell_i)_+\mathcal{K}_i$ on L, which comes equipped with a map $(\ell_i)_+\tau\colon (\ell_i)_+\mathcal{K}_i \to \mathcal{U}_L$. If $f\colon i \to j$ is a morphism in I, then the map $X_j \to X_i$ in **Sch** induces a morphism

$$(\ell_i)_+\mathcal{K}_i \to (\ell_j)_+\mathcal{K}_j$$

of push-forward sheaves over $\mathcal{U}_L$ on L. Let $(\ell_+)\colon I \to \mathbf{Shv}(L)_{/\mathcal{U}_L}$ denote the I-shaped diagram of these push-forward sheaves over $\mathcal{U}_L$. Define $\tau_L\colon \mathcal{K}_L \to \mathcal{U}_L$ to be the limit of this diagram. Then the database

$$\mathcal{L} = (L, \mathcal{K}_L, \tau_L)$$

is our candidate for the limit of the diagram $\mathcal{X}$. Again, it is a matter of tracing through the construction to show that $\mathcal{L}$ has the necessary universal property.

This completes the proof.

□

Remark 5.4.2. The final object in the category $\mathbf{DB}^\pi$ of databases on $\pi\colon U \to \mathbf{DT}$ is the empty database (with empty schema and trivial sheaf). The initial object $(X, \mathcal{K}, \tau)$ in $\mathbf{DB}^\pi$ has, as its schema X, a single n-simplex for every map $\sigma\colon \{0, 1, \ldots, n\} \to \mathbf{DT}$; the sheaf is $\mathcal{K} = \mathcal{U}_X$, and the map $\tau\colon \mathcal{U}_X \to \mathcal{U}_X$ is the identity.

If one knows the Čech nerve construction, one can realize the initial object in those terms, by applying the Čech nerve functor to $\pi\colon U \to \mathbf{DT}$. See [Spi08, 3.1] for details.

Corollary 5.4.3. *Let $X \in$ **Sch** be a schema and let $\mathbf{DB}_X$ denote the category of databases with schema X and with morphisms which restrict to the identity on X. Colimits and limits exist in $\mathbf{DB}_X$; in particular $\mathbf{DB}_X$ has an initial object and a final object.*

Proof. Given a non-empty diagram which restricts to the identity on a certain schema X, one sees by the construction of limits and colimits in the proof of Theorem 5.4.1 that the limit and the colimit of that diagram will also have schema X.

The limit (respectively the colimit) of the empty diagram in $\mathbf{DB}_X$, if it exists, is the final (resp. initial) object in $\mathbf{DB}_X$; we must show it does exist. One immediately sees that the final object is $(X, \mathcal{U}_X, \mathrm{id}_{\mathcal{U}_X})$, and the initial object is $(X, \emptyset, \emptyset \to \mathcal{U}_X)$, where $\emptyset$ here denotes the sheaf on X whose value is constantly the empty set, and where $\emptyset \to \mathcal{U}_X$ is the unique morphism of sheaves.

□

5.5. **Projections.** The PROJECT query is built into the theory of simplicial databases. Given a database $(X, \mathcal{K}, \tau)$ and a subschema $X' \subset X$, we have the database $(X', \mathcal{K}|_{X'}\tau|_{X'})$ given by restricting the sheaf $\mathcal{K}$ and the map of sheaves $\tau\colon \mathcal{K} \to \mathcal{U}$ to the subschema X'. One can view it as a table using Construction 4.5.2.

5.6. **Unions and insertions.** Given two databases with the same schema, one can apply the UNION query. To do so, one keeps the same columns but takes the union of the rows. An insertion is a special kind of union; namely it is a union of two databases on the same schema, where one of the databases consists only of a single row.

We have a few more options in simplicial databases than one does in relational databases; these differences are analogous to the difference between the UNION query and the UNION ALL query in SQL. That is, since we allow duplicate entries (see Section 5.3), the user can decide when an object in one database is the same as an object with the same attributes stored in another database and when it is different. Let us make all of this precise.

We can represent unions, insertions, and more by taking colimits of various diagrams of databases. Let $\mathcal{X} = (X, \mathcal{K}, \tau)$ denote a simplicial database, and let $\mathcal{X}' = (X, \mathcal{K}', \tau')$ be a database with the same schema, X. Both receive a map from the initial database on X, and the coproduct will be $(X, \mathcal{K} \amalg \mathcal{K}', \tau \amalg \tau')$ as desired. (See the proof of Theorem 5.4.1 for details on the colimit construction.)

The above construction gives a UNION ALL query: duplicated tuples will remain distinct. There are two ways of having that not be the case. The first is to simply eliminate the duplicates by converting the database to a relational database; see Lemma 4.4.2. However, this may result in information loss if there really were two entities with the same attributes, because these duplicates will be eliminated.

The other way can occur if the user has more information about which instances in the first database correspond to instances in the second database. This can be accomplished by having a third database $\mathcal{X}'' = (X, \mathcal{K}'', \tau'')$ and maps from it to $\mathcal{X}$ and $\mathcal{X}'$. The colimit of this diagram, $(X, \mathcal{K} \amalg_{\mathcal{K}''} \mathcal{K}', \tau \amalg_{\tau''} \tau')$, will be the union of the records in $\mathcal{X}$ with those in $\mathcal{X}'$, and will identify two records if they agree in $\mathcal{X}''$.

As mentioned above, inserting a row is a special case of taking the union of databases.

We can take much more general colimits than those mentioned above, all of which were constant in the schema. These constructions appear to be new; perhaps they can provide useful ways to analyze and assemble data.

5.7. **Join.** Two databases can be joined together by specifying a common subschema of each and "gluing together" along that subschema. If no common subschema is mentioned we take the initial schema, which is empty, and join along that; the result is called the natural join. The concept of gluing is rigorously formulated as taking limits of certain diagrams in $\mathbf{Sch}^{\mathrm{op}}$; thus the point we are making is that joining databases in the usual sense can be accomplished by taking limits in the category of simplicial databases. Let us make all of this precise.

Recall from Theorem 5.4.1 that the limit of the diagram of databases

$$(X_1, \mathcal{K}_1, \tau_1) \longrightarrow (X, \mathcal{K}, \tau) \longleftarrow (X_2, \mathcal{K}_2, \tau_2)$$

has schema $X' = X_1 \amalg_X X_2$. This induces a diagram

$$\begin{tikzcd}
X \arrow[r] \arrow[d] & X_1 \arrow[d] \\
X_2 \arrow[r] \arrow[ur, phantom, "\ulcorner"] & X'
\end{tikzcd}$$

in $\mathbf{Sch}$. We can thus push-forward $\mathcal{K}_1$, $\mathcal{K}$, and $\mathcal{K}_2$ to X' and get a diagram of push-forward sheaves there (see Definition 5.1.2), all naturally mapping to $\mathcal{U}_{X'}$. For typographical reasons, we leave out the fact that these are push-forwards and write the diagram $\mathcal{K}_1 \to \mathcal{K} \leftarrow \mathcal{K}_2$ over $\mathcal{U}_{X'}$. We are ready to write the limit database as

$$(X_1 \amalg_X X_2, \mathcal{K}_1 \times_{\mathcal{K}} \mathcal{K}_2, \tau'),$$

where $\tau' \colon \mathcal{K}_1 \times_{\mathcal{K}} \mathcal{K}_2 \to \mathcal{U}_{X'}$ is the structure map.

Example 5.7.1. Suppose we have the two schemas pictured here:

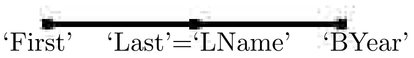
$$X_1 := {}^{\text{'First'}}\bullet\!\!-\!\!\!-\!\!\!-\!\!\bullet^{\text{'Last'}}, \qquad X_2 := {}^{\text{'L.Name'}}\bullet\!\!-\!\!\!-\!\!\!-\!\!\bullet^{\text{'BYear'}},$$

and wish to join them together by equating 'Last' with 'L.Name' (both of which have the same data type, namely **Str**). To do so, we use the schema $X = \bullet^{\textbf{'Str'}}$, which maps to each of X_1 and X_2 in an obvious way.

Now given any databases $\mathcal{X}_1 = (X_1, \mathcal{K}_1, \tau_1)$ and $\mathcal{X}_2 = (X_2, \mathcal{K}_2, \tau_2)$ on X_1 and X_2, we can join them by taking the limit of the solid arrow diagram

$$\begin{tikzcd}
\mathcal{X}_1 \times_{\mathcal{X}} \mathcal{X}_2 \arrow[r, dashed] \arrow[d, dashed] & \mathcal{X}_2 \arrow[d] \\
\mathcal{X}_1 \arrow[r] & \mathcal{X}
\end{tikzcd}$$

where $\mathcal{X} = (X, \mathcal{U}_X, \mathrm{id}_{\mathcal{U}_X})$ is the final database on X. The schema of the resulting database is



This *does not* represent a table with three columns, but two tables, each with two columns, and each projecting to a common 1-column table. However, its global table does have three columns (see Remark 4.5.3). Its records are those triples of the form (First,Last,BYear) for which there is a (First,Last) pair in $\mathcal{X}_1$ and a (Last,BYear) pair in $\mathcal{X}_2$ with matching values of Last. This is indeed their join.

Remark 5.7.2. The "join" we are working with here could be thought of as a combination of equi-join and outer join. Because databases are sheaves on a schema, they do not have just one table but a system of tables, and the idea of nulls is built into the theory (see Section 5.2).

More precisely, if $\mathcal{X}_1 \to \mathcal{X} \leftarrow \mathcal{X}_2$ is a diagram of databases, the limit $\mathcal{X}'$ represents the join of $\mathcal{X}_1$ and $\mathcal{X}_2$ along a shared set of columns (those of $\mathcal{X}$). Its schema is roughly the union of the schemas of $\mathcal{X}_1$ and $\mathcal{X}_2$. Its global table will be the equi-join of the global tables for $\mathcal{X}_1$ and $\mathcal{X}_2$.

The point of this remark, however, is that the new table $\mathcal{X}'$ does not only contain global information, but local information as well. Much of the data of $\mathcal{X}_1$ (respectively $\mathcal{X}_2$) is preserved upon passage to $\mathcal{X}'$, and that which cannot be extended to global data could still be viewed globally if one uses Null values. It is in this sense that colimits in **DB** are related to outer joins.

When joining databases together, one first chooses a set C of columns to equate. When two distinct objects have the same C-attributes, then the join is "lossy" in the sense that there will be false information in the join. To remedy this, one must be careful to distinguish between objects, even when considered only in terms of C. The following example will hopefully make this more clear.

Example 5.7.3. Suppose one wants to join the following two tables:

τ_1	Title	LastName
1	Dr.	Marx
2	Mr.	Marx

τ_2	FirstName	LastName
A	Karl	Marx
B	Groucho	Marx

The outcome will be the following table:

Title	FirstName	LastName
Dr.	Karl	Marx
Dr.	Groucho	Marx
Mr.	Karl	Marx
Mr.	Groucho	Marx

This table has four entries, two of which are "accurate," in that they describe real instances, and two of which are not. This occurs because the relational database cannot distinguish between the two instances of the last name Marx.

Achieving a lossless join is easy, when databases are allowed to have duplicate entries with the same attributes. Consider the table

τ	LastName
x	Marx
y	Marx

which accepts maps from both τ_1 and τ_2 by sending both 1 and A to x, and sending both 2 and B to y (see Definition 2.3.8). The limit of this diagram is the table

Title	FirstName	LastName
Dr.	Karl	Marx
Mr.	Groucho	Marx

as desired.

In the example above, the table τ has two instances of the same string. This is not superfluous because there are two people named Marx. They are differentiated by their internal keys, but not by their attributes. Keeping distinct objects distinct, even if they have the same attributes is very useful in practice. It not only allows for lossless joins, but it is well-suited for database integration as well.

5.8. **Select.** In Example 3.1.10, we selected from a table τ with columns $C = \{$'First Name', 'Last Name', 'BYear'$\}$ all instances for which the value of 'First Name' was "Barack." This was computed as follows. First, we made a table τ' whose column set C' consisted of a single element, labeled 'First Name', and filled in τ' with a single entry, 'Barack'. We might call this table the *selection table.* The SELECT operation was performed by taking the fiber product $\tau \to \mathrm{id}_{C'} \leftarrow \tau'$, where $\mathrm{id}_{C'}$ denotes the table of all possible values of 'First Name'.

Performing SELECT operations in a general simplicial database has the same flavor, in that it is always computed as a certain kind of fiber product. Denote the database from which we are selecting as $\mathcal{X} = (X, \mathcal{K}_X, \tau_X)$, let $S \subset X$ denote a subschema and $\mathcal{S} = (S, \mathcal{K}_S, \tau_S)$ a relational table on S, to serve as the selection table. That is, we will be selecting from X all instances that have the designated S-attributes. Finally, we let $1_{\mathcal{S}} = (S, \mathcal{U}_S, \mathrm{id}_{\mathcal{U}_S})$ denote the final database on the schema S. The fiber product $\mathcal{X}_{\mathcal{S}}$ in the diagram

$$\begin{tikzcd}
\mathcal{X}_{\mathcal{S}} \arrow[r] \arrow[d] \arrow[dr, phantom, very near start, "\lrcorner"] & \mathcal{S} \arrow[d] \\
\mathcal{X} \arrow[r] & 1_{\mathcal{S}}
\end{tikzcd}$$

is the desired result.

5.9. **Deletions.** Deletion can be subtle. If one deletes entries over a subschema, the action must "cascade" up the hierarchy, deleting entries in larger schemas when they refer or point to the deleted entries. To that end, we define the following construction.

Definition 5.9.1. Suppose given a schema X and a subsheaf $\mathcal{K}_1 \subset \mathcal{K}$ on X. Let $\overline{\mathcal{K}_1} \subset \mathcal{K}$ denote the presheaf on X with

$$\overline{\mathcal{K}_1}(X') \coloneqq \{r \in \mathcal{K}(X') | \exists X'' \subset X', X'' \neq \emptyset, r_{X''} \in \mathcal{K}_1(X'')\}$$

for subschema $X' \in \mathbf{Sub}(X)$. Here $r_{X''}$ denotes the image of r under the restriction map $\mathcal{K}(X') \to \mathcal{K}(X'')$. We call $\overline{\mathcal{K}_1}$ the *closure of* $\mathcal{K}_1$ *in* $\mathcal{K}$.

Suppose now we want to delete all entries of a given type from a database. More concretely, suppose $\mathcal{X} = (X, \mathcal{K}_X, \tau_X)$ is a database with schema X, that $i\colon S \subset X$ is a subschema, and that $\mathcal{S} = (S, \mathcal{K}_S, \tau_S)$ is a relational database of objects of this subtype, all of which we would like to delete from X. As explained in Section 5.8, we can select the rows of $\mathcal{X}$ of the type specified by $\mathcal{S}$ by defining $\mathcal{X}_S$ to be the limit as in the diagram

$$\begin{tikzcd}
\mathcal{X}_S \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathcal{S} \arrow[d] \\
\mathcal{X} \arrow[r] & (S, \mathcal{U}_S, \mathrm{id}_{\mathcal{U}_S}).
\end{tikzcd}$$

We know that $\mathcal{X}_\mathcal{S}$ has schema $X = X \amalg_S S$ and we momentarily invent notation and write $\mathcal{X}_\mathcal{S} = (X, \mathcal{K}_{\mathcal{S}\subset\mathcal{X}}, \tau_{\mathcal{S}\subset\mathcal{X}})$.

The map $\mathcal{X}_\mathcal{S} \to \mathcal{X}$ defines an inclusion of sheaves $\mathcal{K}_{\mathcal{S}\subset\mathcal{X}} \subset \mathcal{K}_X$ on X, and we take its closure $\overline{\mathcal{K}_{\mathcal{S}\subset\mathcal{X}}} \subset \mathcal{K}_X$. By construction we can now delete this subsheaf objectwise on $\mathbf{Sub}(X)$. That is, we define for $X' \subset X$

$$\mathcal{K}_{\mathcal{X}\backslash\mathcal{S}}(X') = \mathcal{K}_X(X') \backslash \mathcal{K}_{\mathcal{S}\subset\mathcal{X}}(X'),$$

where $A \backslash B$ denotes the maximal subset of A which contains no elements in B.

The database

$$\mathcal{X}' \coloneqq (X, \mathcal{K}_{\mathcal{X}\backslash\mathcal{S}}, \tau),$$

where τ is shorthand for $\tau|_{\mathcal{K}_{\mathcal{X}\backslash\mathcal{S}}}\colon \mathcal{K}_{\mathcal{X}\backslash\mathcal{S}} \to \mathcal{U}_X$, is the deletion of $\mathcal{S}$ from $\mathcal{X}$. There is a canonical map $\mathcal{X}' \to \mathcal{X}$ in $\mathbf{DB}$, and one can show that $\mathcal{X}'$ is the final object under $\mathcal{X}$ whose join with $\mathcal{S}$ is empty.

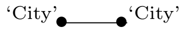
6. Applications, advantages, and further research

In this section, we discuss the applications of the category of simplicial databases. First, simplicial databases can be used wherever relational databases are used; though simplicial databases are more general, they are still closed under applying the usual queries. On the other hand, there are many advantages to using simplicial databases as opposed to relational ones.

In Section 6.1, we discuss how the geometry of a schema can provide an intuitive picture for the content and layout of a database. As an example of using category theory to reason about databases, we show in Section 6.2 that query equivalences are trivially verified when one phrases them in categorical language. In Section 6.3 we discuss how diagrams of databases can give various users different privileges in terms of accessing and modifying data. In Section 6.4 we address the issue of comparing our categorification of databases to others' versions. Finally, in Section 6.5, we discuss further research on the subject and open questions.

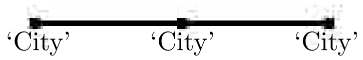
6.1. **Geometric intuition.** In Section 4.1, we defined the category $\mathbf{Sch}^{\pi}$ of schemas for a given type specification π. They are based on geometric objects called simplicial sets. In this section, we show that the geometry of these objects is intuitive and therefore useful in practice.

Example 6.1.1. In this example, we consider a simplified situation in which one keeps track of the cities from which airplane flights take off and those at which they land. So suppose we have only one type, $\mathbf{DT} = \{\text{`City'}\}$ and U is the set of cities in the world that have airports. Let X be the schema



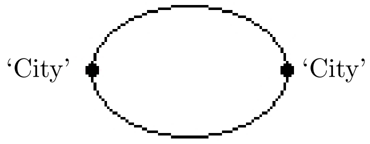
For our sheaf of keys $\mathcal{K}$, we take $\mathcal{K}(\text{`City'}) = U$. Over the 1-simplex X take $\mathcal{K}(X)$ to be the set of pairs (c_1, c_2) for which c_1 is the city of departure and c_2 is the city of arrival for some flight. Let $\mathcal{X}$ denote this database of flights.

Now, joining this database with itself yields a database with schema whose global



sections are "flights with layover," i.e. pairs of flights with the destination city of the first flight equal to the departing city of the second flight. Similarly, the database of multi-city trips of a given length n is simply the union (colimit) of n copies of the database of flights $\mathcal{X}$ in this way.

Moreover, if we want to use $\mathcal{X}$ to find the set of available round-trips, we simply join the ends of the schemas in Diagram 6.1.1 to make a circle

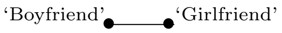


This is not just heuristic; we have literally taken the indicated limit of databases. The result is a new database whose global sections are precisely the pairs of flights which constitute a round-trip.

The point is that one can intuit this result by visualizing round-trips as circles, and then applying that vision to the schemas themselves.

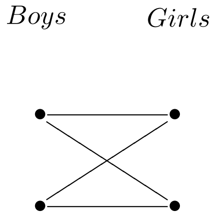
Example 6.1.2. In 2004, Bearman et al. [BMS04] present data which shows that at a certain high school called "Jefferson High," there is a statistically small number of sexual couples that later switch partners. That is, if B_1 and G_1 are sexual partners and B_2 and G_2 are sexual partners, then it rarely happens that later B_1 mates with G_2 and B_2 mates with G_1. As they say "...we find many cycles of length 4 in the simulated networks, but few in Jefferson..."

Suppose then that we take their raw data and put it on the schema



which we denote X. Visually, we represent two boys and two girls who switch partners as follows:

(7)



(where, say, horizontal lines represent the original partnerships and diagonal lines represent the new partnerships). And indeed, we can take the union of four copies of X along various vertices to obtain a database with the above 4-cycle schema.

In other words, there is a way to take raw data over a line segment, representing partnerships, and automatically generate data over the "switch schema," Diagram (7), just by taking the indicated limit of databases. The global sections of this new "switched partners" database are precisely what is being studied in Bearman's paper.

As in Example 6.1.1, the point is that the shape of the schema is intuitive. Using schemas that are geometrically intuitive may enhance the ability of users to manipulate and make sense out of the raw data.

6.2. **Query equivalences.** It is well known that joining tables together is very costly. If one only wishes to consider certain rows or columns of a join, he or she should isolate those rows or columns *before* performing the join, not after. For that reason, one is taught to "push selects and projects," i.e. to do these operations first.

How does one prove that projecting first and then joining will result in the same database as will joining first and then projecting? The proofs of results like these are generally tedious. In this section, we do not claim any new results. We merely show that these simple query equivalences are obvious when one uses the language of simplicial databases and knows basic category theory.

For example, it is a standard category-theoretic fact that, in *any* category $\mathcal{C}$ with limits, there is a natural isomorphism

$$(A \times_B C) \times_D E \cong (C \times_D E) \times_B A. \tag{8}$$

Note that both joins and selects are examples of such limits (see Sections 5.7 and 5.8). The formula (8) in particular applies to the category $\mathbf{DB}$ of databases and proves that "selecting E from a join of A and C gives the same result as first selecting E from C and then joining the result with A.

Projecting a database to a subschema is easy to describe in the theory of simplicial databases: one simply restricts the sheaf $\mathcal{K}$ and the map τ to that subschema (see Section 5.5). The fact that projects commute with joins follows from basic sheaf theory, e.g. that the limit of a diagram of sheaves is the same as the limit of the underlying diagram of presheaves.

6.3. **Privileges.** The sheaf-theoretic nature of our conception of databases lends itself nicely to the idea of privileges. It often happens that one wishes to give a particular user the ability to modify certain sections of the database but not others.

If X is the schema for a database $\mathcal{X}$, perhaps we wish to give a particular user the ability to modify data on the subschema $i\colon X' \subset X$.

To accomplish this, note that there is a map of databases

$$\mathcal{X} = (X, \mathcal{K}_X, \tau_X) \longrightarrow (X', i^*\mathcal{K}_X, i^*\tau_X) = \mathcal{X}'$$

We allow the user to see $\mathcal{X}'$ as a database and make changes to it (we could also limit the ways in which this user can modify $\mathcal{X}'$ – only allow insertions, for example). At any given time, the user only sees the sub-database $\mathcal{X}'$.

Suppose he or she adds a few lines to the sheaf $i^*\mathcal{K}_X$ to make it $i^*\mathcal{K}_X \cup \mathcal{L}$. To update the main database, we take the colimit of the diagram of sheaves

$$\begin{tikzcd}
i_!i^*\mathcal{K}_X \arrow[r] \arrow[d] & i_!(i^*\mathcal{K}_X \cup \mathcal{L}) \\
\mathcal{K}_X &
\end{tikzcd}$$

and the result will be a new sheaf on X with the appropriate insertions.

Deletions are handled in a somewhat different way, but the idea is the same. If the user deletes data from the sheaf $i^*\mathcal{K}_X$ to obtain the sheaf $i^*\mathcal{K}_X \backslash \overline{\mathcal{D}}$, then to update the main database may require us to delete entries from larger schemas (see Section 5.9). The updated sheaf on X will be the limit of the diagram

$$\begin{tikzcd}
 & \mathcal{K}_X \arrow[d] \\
i_+(i^*\mathcal{K}_X \backslash \overline{\mathcal{D}}) \arrow[r] & i_+i^*\mathcal{K}_X.
\end{tikzcd}$$

Again, we are not claiming that privileges of this type are anything new. We are claiming that they are naturally phrased in this categorical language, thus bringing a new and powerful mathematical tool to bear on the problems of the subject.

6.4. **Comparison to other categories of databases.** As mentioned in the introduction, many other categorifications of databases have been presented over the years. One of the nice features of category theory is that one can compare various categories using functors. Given another categorical formulation of databases, we could try to produce a functor from it to **DB** and from **DB** back to it. The way that these functors behave (e.g. if they are adjoint, or if one or the other is fully faithful) will tell us about the relative expressive power of the models, as well as help us to understand how to translate between them. We hope to work on such a comparison in the future.

6.5. **Further research.** The category-theoretic and also geometric nature of simplicial databases opens up many directions for future research. We present a few in this subsection that we intend to pursue. Many of these ideas were suggested to us by Paea LePendu.

6.5.1. *Topological methods.* First, we would like to consider how we might use methods from algebraic topology to study databases. Recall from Example 4.1.5 that there is a functor **Sch** → **Top** called *topological realization* that allows one to naturally view any schema as a topological space. Furthermore, we already saw in Example 6.1.2 that importing topological ideas can have real world meaning: topological 4-cycles represented pairs of mating couples that switched partners.

Another example of the usefulness of topological methods is given by “lifting problems.” Problems of this sort include the famous question “are there three foods, each pair of which taste good when eaten together, but the threesome of which tastes bad when eaten together?”

To phrase this in terms of social networks, suppose that for any n people, either this group is said to be a friendship group or it is not. The above lifting problem becomes: “are there three people, each pair of which is a friendship group, but the triple is not?” These types of phenomena can be represented geometrically, so having simplicial sets as schemas may be useful for their study.

Homotopical methods from algebraic topology may also be useful. When one object “morphs” into another over the course of time (such as a child becoming an adult), it is difficult to know how to treat that object in a database. Homotopy theory is the study of gradual transformation through time, and the author sees some potential for using it to study real-world phenomena.

Finally, the geometric nature of our schemas may be useful for query optimization. Schemas can be classified according to their geometric structure. It may be that in performing many queries, a database management system learns that some geometric structures are being used more often than others. The patterns which emerge may be only visible when one uses schemas that have this higher dimensional geometric nature.

6.5.2. *Functional dependencies and normal forms.* In this paper we have not discussed functional dependencies or normal forms. It is appealing to ask the following question:

Question 6.5.1. Let $X \in \mathbf{Sch}$ denote a schema; it should be thought of as having a shape (again, via the topological realization functor $\mathbf{Sch} \to \mathbf{Top}$), namely a union of tetrahedra. We wonder:

(1) Given a set of functional dependencies, is there a natural way to annotate the shape X so that these dependencies are made visual?
(2) Given a schema X that has been annotated in this way, can one easily determine whether it is in a certain normal form?
(3) If an annotated schema is not in normal form, do the annotations help in finding the normalization?

If the answer to these questions is affirmative, we will have more evidence that the geometric nature of our schemas is useful for database design and management.

We hope to address these questions in the near future.

6.5.3. *Database integration.* We believe that having a rigorous definition for *morphisms of databases* (see Definition 4.3.7) will be of use in the problem of database integration. The morphisms of databases can account for simultaneous changes in schemas and in data. It is also easy to allow changes in data types as well, a topic we will address in later work.

Also, as mentioned in Remark 4.3.4 and Section 5.3, the use of internal keys should prove immensely valuable. Instead of including an arbitrarily chosen identifier for an object as part of the data for that object, as required in the theory of relational databases, our theory keeps these arbitrary identifiers separate. When attempting to integrate databases, it is imperative that one know which sections of the data are *observed and invariant properties* of the objects being classified, and which sections of the data are *arbitrarily assigned* for management reasons. Our theory keeps these sections of the data distinct, by use of a sheaf of keys $\mathcal{K}$ that is not considered part of the data.

In future research, we hope to show that database integration is made substantially easier when one works with a rigorous and geometric model like the one we present here. Before we do so, we need to explain how to work with a change in type specifications, which is not hard, and how to deal with constraints in the data. See Section 6.5.5 for our plans in this direction.

6.5.4. *Ontologies and networks.* One intuitively knows that there is a connection between databases and ontologies. An ontology is meant for organizing knowledge, a database is meant for organizing information, and there is a strong correlation between the two. In order to make this correlation precise, one must first find precise definitions of ontologies and databases. Further, these definitions should be phrased in the same language so that they can be compared. Category theory was invented for the purposes of comparing different mathematical structures, and as such provides a good setting for this project.

Our plan (see [Spi09])) for a categorical definition of communication networks involves annotating the simplices of a simplicial set with databases. That is, each node in a network has access to a database of "what it knows," and connections between nodes allow communication via a common language and set of shared knowledge. In order to make this precise, we need a precise definition for a category of databases, for which Definition 4.3.7 suffices.

6.5.5. *More exotic types.* Throughout this paper, we have fixed a type specification $\pi\colon U \to \mathbf{DT}$, where $\mathbf{DT}$ is a set of data types, and U is the disjoint union of the corresponding domains. This allows for types like strings, characters, dates, integers, etc. It also allows for more general types like "functions from A to B" or "probability distributions on a space."

However, as flexible as our type specifications may be, the situation can be generalized considerably by allowing π to be a functor between categories, rather than a function between sets. The simplest application is one that is already implicitly used, namely sorting data. The set of strings is in fact an ordered set, and so can be represented as a category (with a morphism from A to B if B is lexicographically larger than A). Another application comes from putting constraints in the data, for example if we were only to allow (city, state) pairs for which the city is within the state.

By generalizing type specifications to include categories rather than sets, we open up many new possibilities for making sense of data. Causal relationships can be represented, as can processes. In short, morphisms make the theory more dynamic.

References

[Ber01] Philip A. Bernstein, *Generic model management: A database infrastructure for schema manipulation*, pp. 1–6, Springer Berlin/Heidelberg, 2001.

[BMS04] P.S. Bearman, J. Moody, and K. Stovel, *Chains of affections: the structure of adolescent romantic and sexual networks*, American Journal of Sociology **110** (2004), 44–91.

[Bor94a] Francis Borceux, *Handbook of categorical algebra. 1*, Encyclopedia of Mathematics and its Applications, vol. 50, Cambridge University Press, Cambridge, 1994, Basic category theory. MR MR1291599 (96g:18001a)

[Bor94b] ———, *Handbook of categorical algebra. 3*, Encyclopedia of Mathematics and its Applications, vol. 52, Cambridge University Press, Cambridge, 1994, Categories of sheaves. MR MR1315049 (96g:18001c)

[BW90] Michael Barr and Charles Wells, *Category theory for computing science*, Prentice Hall International Series in Computer Science, Prentice Hall International, New York, 1990. MR MR1094561 (92g:18001)

[Cod70] E.F. Codd, *A relational model of data for large shared data banks*, Communications of the ACM **13** (1970), 377–387.

[Dat05] C.J. Date, *Database in depth*, O'Reilly, 2005.

[Dis96] Zinovy Diskin, *Databases as diagram algebras: Specifying queries and views via the graph-based logic of sketches*, Tech. report, Frame Inform Systems, 1996.

[DK94] Zinovy Diskin and Boris Kadish, *Algebraic graph-oriented=category-theory-based manifesto of categorizing data base theory*, Tech. report, Frame Inform Systems, 1994.

[EN07] Ramez Elmasri and Shamkant Navathe, *Fundamentals of database systems*, 5th ed., Pearson; Addison Wesley, San Francisco, 2007.

[Fri08] Greg Friedman, *An elementary illustrated introduction to simplicial sets*, ePrint available at http://arxiv.org/pdf/0809.4221.pdf, 2008.

[GB92] Joseph A. Goguen and Rod M. Burstall, *Institutions: abstract model theory for specification and programming*, J. Assoc. Comput. Mach. **39** (1992), no. 1, 95–146. MR MR1147298 (93h:03056)

[GJ99] Paul G. Goerss and John F. Jardine, *Simplicial homotopy theory*, Progress in Mathematics, vol. 174, Birkhäuser Verlag, Basel, 1999. MR MR1711612 (2001d:55012)

[Gra01] Marco Grandis, *Finite sets and symmetric simplicial sets*, Theory Appl. Categ. **8** (2001), 244–252 (electronic). MR MR1825431 (2002c:18010)

[Joh02] Peter T. Johnstone, *Sketches of an elephant: a topos theory compendium. Vol. 2*, Oxford Logic Guides, vol. 44, The Clarendon Press Oxford University Press, Oxford, 2002. MR MR2063092 (2005g:18007)

[JR03] M. Johnson and R. Rosebrugh, *Three approaches to partiality in the sketch data model*, Electronic Notes in Theoretical Computer Science **78** (2003), 1–18.

[JRW02] Michael Johnson, Robert Rosebrugh, and R. J. Wood, *Entity-relationship-attribute designs and sketches*, Theory Appl. Categ. **10** (2002), 94–112 (electronic). MR MR1883480 (2002m:18004)

[ML98] Saunders Mac Lane, *Categories for the working mathematician*, second ed., Graduate Texts in Mathematics, vol. 5, Springer-Verlag, New York, 1998. MR MR1712872 (2001j:18001)

[MLM94] Saunders Mac Lane and Ieke Moerdijk, *Sheaves in geometry and logic*, Universitext, Springer-Verlag, New York, 1994, A first introduction to topos theory, Corrected reprint of the 1992 edition. MR MR1300636 (96c:03119)

[PS95] Frank Piessens and Eric Steegmans, *Categorical data-specifications*, Theory Appl. Categ. **1** (1995), No. 8, 156–173 (electronic). MR MR1356700 (97b:18001)

[RW92] Robert Rosebrugh and R. J. Wood, *Relational databases and indexed categories*, Category theory 1991 (Montreal, PQ, 1991), CMS Conf. Proc., vol. 13, Amer. Math. Soc., Providence, RI, 1992, pp. 391–407. MR MR1192160 (93i:68054)

[Spi08] David Spivak, *Geometric databases*, Algebraic Topological Methods in Computer Science, application pending, 2008.

[Spi09] ———, *Geometric networks: A higher-dimensional approach to networks and databases.*, Technical Proposal for ONR grant, available at http://www.uoregon.edu/∼dspivak/technical.pdf, 2009.