1. Introduction

The data dictionary pattern is a broad view of what a database is. Interestingly, this pattern itself can be expressed as a database. A slide deck on the subject, written by Peter Gates, can be found here at: [http://www.uoregon.edu/~dspivak/cs/db_dic.pdf](http://www.uoregon.edu/~dspivak/cs/db_dic.pdf).

In this note, I will give a category theoretic description of this idea.

I will use the following idea for labeling data columns.

Remark 1.1. One can imagine each data type as a table with one column and a row for each possible value of that data type. For example the data type char(1) can be represented as a table $T_{\text{char}(1)}$ with one column and 26 rows. Similarly the data type char(2) can be represented as a table $T_{\text{char}(2)}$ with one column and $26 \times 26 = 676$ rows. The table for integers would be huge or theoretically infinite, but at least one can consider it abstractly in the same way as char(10).

In this short note, we attempt to formulate Gates’s ideas in a categorical way. We do this by drawing each table as a node and each foreign-key column as an arrow from the source table to the target table. We also draw each data column as an arrow from the table to the 1-column table as in Remark 1.1.

The resulting figures below appear as graphs. Since we assume that every table has a distinguished primary key, each is in fact a reflexive graph (meaning each node in it has an arrow pointing to itself). We do not draw these reflexive arrows for typographical reasons.

Finally, we sometimes want to ensure that two different paths through a graph agree. For example, each cell in a database appears in a unique column and row. Each column and each row appear in a unique table. For each cell, the table of its column must be the same as the table of its row. We would say that the diagram

\[
\begin{array}{c}
\text{Cell} \\
\downarrow \\
\text{Column}
\end{array} \quad \begin{array}{c}
\rightarrow \\
\downarrow \\
\rightarrow
\end{array} \quad \begin{array}{c}
\text{Row} \\
\downarrow \\
\text{Table}
\end{array}
\]

commutes.

2. The dictionary pattern

The following is a reformulation of slide 2. What Gates calls “Value” I am calling “Cell”. There are two paths from Cell to Table – these commute. There are also two paths from Column to $T_{\text{String}}$ – these do not commute.

This project was supported in part by a grant from the Office of Naval Research: N000140910466.
2.1. **Typing the columns.** Let us add information about the type of a column as follows. Let $\mathbf{DT}$ denote the set of datatypes, $\mathbf{U}$ the disjoint union of their domains, and $\pi: \mathbf{U} \to \mathbf{DT}$ the function that associates each instance to its type. We can think of $\mathbf{DT}$ as a table of data types, think of $\mathbf{U}$ as a table of their instances (which of course is impractical given its size, but in theory fine), and think of $\pi$ as a foreign key with source $\mathbf{U}$ and target $\mathbf{DT}$. We now have:

![Diagram showing relationships between tables, rows, columns, and cells with annotations for Table, Name, DT, String, Row, Column, Cell, fk_row_table, fk_column_table, fk_cell_row, fk_cell_column, fk_cell_u, T_{String}, and U.]

The seven arrows above correspond to Gates’s seven non-id columns.

**Remark 2.1.** Given a foreign key with source $A$ and target $B$, Gates names it $\text{fk}_{A,B}$. In my opinion this naming system has several drawbacks.

In category theory, every arrow has a source and target, just like a foreign key does. Since each arrow is typically drawn from the source to the target, there is no need to include the name of the source and target of an arrow when naming the arrow. In other words, if an arrow goes from $A$ to $B$, we can call it $f: A \to B$, rather than something like $\text{fk}_{A,B}$. This frees us to name the arrow by its meaning rather than its syntax. Moreover, we may also have multiple arrows from $A$ to $B$, which Gates’s naming system may have a harder time handling.

We shall therefore adopt the convention that an arrow may have any name, such as $f: A \to B$. This arrow will represent a foreign key column in table $A$ that refers to the primary key column of table $B$, and this foreign key will be named $f$.

We also draw data columns of table $A$ (i.e. columns of $A$ that are not foreign keys) as arrows out of $A$. The target of such an arrow will be the data type of that column. This is in keeping with [Spivak. A simple model of databases]. See Remark [1.1]. The idea then is that a column of $A$ with data type char(1) can be represented as a foreign key $A \to T_{\text{char}(1)}$. With this convention, both keys and data are represented in the same way.
The following is a redrawing of the above diagram using arrows decorated with their “meaning” rather than their source and target. An arrow whose meaning is “obvious” will be left unlabeled. I will refer to the following as the “dictionary pattern”:

(Dictionary Pattern)

3. Dictionary: Referential Integrity

We now refer to slide 3 in Gates’s deck. We make the following changes: we drop the “Schema” table for simplicity (although to include it is not hard) and we change the naming system as per Remark 2.1.
4. Refactoring

We now refer to slide 6 in Gates’s deck. Again, we do not include the table called “Schema” for simplicity. We can draw this slide categorically as follows:

5. The categorical viewpoint

In the paper [A simple model of databases], we assume that every table has a chosen primary key. The idea of that paper is that a column representing pure data (e.g. a date-column or name-column) acts just like a foreign key column: we can think of such a column as a foreign key mapping to a 1-column table of data (e.g. just a date or just a name). We call such a 1-column table a terminal table.

We now write down this idea using two different representations: the usual schematic view and our graph view.

5.1. Unnamed tables and columns. Suppose we have a set of unnamed tables and a set of unnamed columns. This would be represented graphically by the
CATEGORICAL DESCRIPTION OF PETER GATES'S DICTIONARY PATTERN

schema:

\[
\begin{array}{c}
\text{Table} \\
\text{primary} \\
\text{Column}
\end{array}
\]

Categorically, we demand that for any table \( T \), the source of the primary column and the target of the primary column are both \( T \). Call this category \( P \).

A database on this schema is a functor \( P \rightarrow \text{Sets} \). It consists of a set of tables, a set of columns, for each table a primary column, and for each column both a source and a target table. But whatever we call such a functor, it is the same as a reflexive graph. In other words, a schema for a database is a reflexive graph.

Each vertex represents a table. Vertices that don’t have outgoing arrows are “terminal tables,” representing pure data. The identity arrow on a vertex represents its primary key, every other outgoing arrow is a foreign key (or, if its target is a terminal table, then it is pure data).

5.2. Conclusion. A database as in Gates’s deck is simply a reflexive graph with named nodes and named edges. The difference between a reflexive graph and a category is simply the ability to declare that two different paths through the graph should result in the same output. In other words, we can declare that transitively following one path of foreign keys from table \( A \) to table \( B \) will result in the same mapping as following a different path of foreign keys from \( A \) to \( B \). See Diagram 1 for an example of this.