Functorial Data Migration: From Theory to Practice

Ryan Wisnesky\textsuperscript{1}, David I. Spivak\textsuperscript{1}, Patrick Schultz\textsuperscript{1}, and Eswaran Subrahmanian\textsuperscript{2}

\textsuperscript{1} Massachusetts Institute of Technology
\textsuperscript{2} Carnegie Mellon University

Abstract. In this paper we describe a functorial data migration scenario about the manufacturing service capability of a distributed supply chain. The scenario is a category-theoretic analog of an OWL ontology-based “semantic enrichment” scenario developed at the National Institute of Standards and Technology (NIST). The scenario is presented using, and is included with, the open-source FQL tool, available for download at categoricaldata.net/fql.html.

1 Introduction to Functorial Data Migration

In the functorial data model, which originated with Rosebrugh and others in the late 1990s \textsuperscript{3}, a database schema is a finitely presented category (essentially, a directed multi-graph and path equality constraints) and a database instance on a schema \( S \) is a set-valued functor from \( S \) (essentially, a set of tables). The database instances on a schema \( S \) constitute a category, denoted \( S\text{-Inst} \), and a functor \( F: S \rightarrow T \) between schemas \( S \) and \( T \) induces three adjoint data migration functors, \( \Sigma_F: S\text{-Inst} \rightarrow T\text{-Inst} \), \( \Pi_F: S\text{-Inst} \rightarrow T\text{-Inst} \), and \( \Delta_F: T\text{-Inst} \rightarrow S\text{-Inst} \). These functors provide a category-theoretic alternative to traditional, set-theoretic operations for information integration such as SQL and the chase \textsuperscript{2}.

We have developed a simple algebraic query language for the functorial data model, FQL, as well as a corresponding integrated development environment (IDE), the FQL IDE. The FQL IDE is an FQL code editor, a FQL ↔ SQL translator, a FQL execution engine, and a data visualization tool designed in the spirit of the schema-mapping tool Clio \textsuperscript{4}. The FQL IDE is open source, written in java, and available for download at categoricaldata.net/fql.html. In this paper, we demonstrate how the FQL IDE is used in practice by describing an example data migration scenario developed in collaboration with the National Institute of Standards and Technology (NIST).

Remark. Rosebrugh et al’s original model \textsuperscript{3} has a number of theoretical issues that prevent it from being used directly as a basis for information integration. Hence, FQL is actually based on an extension of Rosebrugh’s model, described in \textsuperscript{7}. The exact definition of this extension does not matter for the purposes of this paper.

\textsuperscript{**} Work performed while visiting the National Institute of Standards and Technology.
2 An Enrichment Scenario

The example described in this paper is an FQL analog of a “semantic enrichment” scenario developed at NIST and published as [5]. In this scenario, a database (called Portal A in [5]) contains information about equipment, including the capabilities of such equipment; for example, that a particular machine \( m \) can drill holes as small as .5cm in metal. The goal of the scenario is to “enrich” Portal A’s data with additional 3rd party information about materials, so that, for example, Portal A’s data also contains the fact that \( m \) can drill holes in iron, because iron is a kind of metal.

In [5], Portal A’s database is a Microsoft Access database, the 3rd party enriching information about materials is an OWL ontology, and enrichment is done by invoking a black-box OWL reasoner on an input query, Portal A’s data, the OWL ontology about materials, and an OWL ontology relating portal A’s vocabulary (e.g., “iron”) and the material ontology vocabulary (e.g., “ferrous”). In this paper, we simplify this scenario as follows: we assume Portal A’s data is given as a SQL database, that the ontology about materials is simply an “is-a” parenthood function, and that the correspondence between Portal A’s vocabulary and the is-a hierarchy vocabulary is a “synonyms” relation between sets of words.

Our FQL development consists of three main steps: first, we import Portal A’s data, the is-a hierarchy, and the synonyms into FQL. Second, we transitively close the is-a hierarchy, join it with the synonyms relation, and then join the result to Portal A’s data. Finally, we test the result of our enrichment on a particular query (query 1 from [5]). This query gives additional results on the enriched data, which demonstrates that FQL can be used to do semantic enrichment along the lines described in [5]. Although we only have space to sketch the outline of the development, the entire development – about 2000 lines of FQL code, 1800 lines of which are schema and data definitions – is included as a built-in example in the FQL IDE.

2.1 Step 1: Import relational data

The schema for Portal A’s data is a SQL schema in categorical normal form [6]: every table consists of a distinguished (primary key) ID column, a set of “attribute” columns whose values contain strings or integers, and a set of foreign key columns whose values contain IDs that refer to other tables. Consequently, Portal A’s schema can be regarded as the presentation of a category: the objects of the category are the table names and type names, and the arrows between objects are the foreign key or attribute columns in the schema. An instance on Portal A’s schema, which physically is a set of relations, can then be regarded as a set-valued functor. The actual Portal A schema as visualized in Microsoft Access is shown in Figure 1 and a snippet of the SQL commands defining the Portal A data are shown in Figure 2. The FQL IDE imports these SQL commands and emits corresponding FQL code that defines an FQL schema and an FQL instance on that schema. A portion of Portal A’s data, as displayed in the FQL IDE, is shown in Figure 3.
CREATE TABLE unitcode (  
id INT PRIMARY KEY, Code VARCHAR(255), Description VARCHAR(255)  );
INSERT INTO unitcode VALUES  
(1,"EA","Each part/piece count"),
(2,"Thousands","1000 parts/pieces count"),
(3,"Inch","Length measure in inches"),
(4,"mm","Length measure in millimeters"),
(5,"cm","Length measure in centimeters");

Fig. 2. Snippet of SQL for Portal A
Two additional inputs are specified in the original scenario: an OWL ontology \( X \) containing myriad facts about materials (e.g., steel is a metal), and an OWL ontology relating the vocabulary used by \( X \) (e.g., “ferrous”) to the vocabulary used by portal \( A \) (e.g., “iron”). At present we do not have a good understanding about how OWL relates to FQL. So, we went through these ontologies by hand and stripped out relevant data. The result was 1) a (total) function \( \text{parent} : O \rightarrow O \), where set \( O \) is the set of words from the ontology, and 2) a “synonyms” relation \( \text{syn} \subset O \times N \) where \( N \) is the set of words from portal \( A \). The FQL schemas for functions and relations (also called spans) are shown in Figure 4. The function \( \text{parent} \), as displayed in the FQL IDE, is shown in Figure 5.

### 2.2 Step 2: Process imported data

We enrich Portal \( A \)’s data as follows. First, we compute the reflexive, transitive closure of the parenthood function, resulting in an \( \text{isa} \) relation. To do this, we define, in Figure 4, for each natural number \( n \), a functor \( F^n : T \rightarrow S \) from the schema for a relation (called \( T \)) to the schema for a function (called \( S \)). Given an \( S \) instance (e.g., the \( \text{parent} \) function) \( I \), \( \Delta F^n(I) \) computes, as a \( T \) instance (i.e., relation), the \( n \)-ary composition of \( I \), i.e., \( F^n \), with the 0-th composition being the reflexive closure of \( I \). The reflexive transitive closure of \( I \) is then a union \( \Delta F^0(I) \cup \Delta F^1(I) \cup \ldots \). For this example, we used \( n = 3 \). Taking the union of two instances on the same schema is a built-in FQL primitive. A portion of the resulting \( \text{isa} \) relation, as displayed in the FQL IDE, is shown in Figure 6.

We now have a relation (\( T \) instance) \( \text{isa} \subset O \times O \), where \( O \) is the set of words from the OWL materials ontology, and we have a relation (\( T \) instance) \( \text{syn} \subset O \times N \), where \( N \) is the set of words from Portal \( A \). We next compute a “translation” of \( \text{isa} \) to use words from Portal \( A \) by joining \( \text{isa} \) with \( \text{syn} \) resulting in a new relation (\( T \) instance) \( \text{isa}' \subset N \times N \). To specify this join we use FQL’s “select/from/where” syntax; an example of this syntax is shown in Figure 7. Note that FQL’s select/from/where syntax is syntactic sugar: the select/from/where syntax is equivalent to a data migration of the form \( \Sigma \circ \Pi \circ \Delta \).

Now that we have the \( \text{isa}' \) relation (\( T \) instance) on Portal \( A \)’s vocabulary, we enrich Portal \( A \)’s data by joining it and the \( \text{isa}' \) relation together. Because Portal \( A \)’s schema is large, it is impractical to write the join by hand, even using FQL’s select/from/where syntax. Hence, we developed an FQL extension to generate this join for us from the definition of Portal \( A \)’s schema. The result of the join is a new, larger instance on Portal \( A \)’s schema.

![Fig. 4. The reflexive transitive closure of a function \( I \) is \( \Delta F^0(I) \cup \Delta F^1(I) \cup \ldots \)](image-url)
Fig. 5. Initial “is-a” parent function displayed in the FQL IDE

Fig. 6. Transitivity closed “is-a” relation displayed in the FQL IDE
2.3 Step 3: Query processed data

Having enriched Portal A’s data, we can query it, using query 1 from [5]. The
query we are using is written in FQL’s select/from/where syntax and is shown
in Figure 7. Before enrichment, this query returns only two rows (Figure 8).
After enrichment, this query returns many more rows (Figure 9), because the
isa’ relation contains many kinds of pre-hardened stainless steel. (Note that “Pre-
hardened stainless steel” does not appear in Figure 5 because that term is used
by Portal A but not by the 3rd party OWL materials ontology).

3 Conclusion

We conclude with a functorial query language design and implementation les-
on learned by developing this example. Not only does FQL’s select/from/where
query syntax save time and effort compared to writing $\Sigma \circ \Pi \circ \Delta$ migrations,
in many cases we were able to write select/from/where queries when we had
no idea how to write the corresponding $\Sigma \circ \Pi \circ \Delta$ migration. Moreover, FQL’s
select/from/where queries can be executed directly in a more efficient manner
than by translation to a migration of the form $\Sigma \circ \Pi \circ \Delta$. The reason is that
many techniques from relational database theory, such as join re-ordering, can
be applied directly to select/from/where syntax. Hence we conclude that se-
lect/from/where syntax should be primitive in any functorial query language.

```
select
  m.material_Material_Name as mn,
  c.capability_Capability_Name as ccn,
  c.capability_Max_Length as ml,
  uc.unitcode_Code as ucc,
  posc.productorservicecategory_Category_Name as pcn
from
  productorservicecategory as posc,
  material as m,
  unitcode as uc,
  capability as c,
  capabilitymaterials as cmX,
  capabilitycategories as cc
where
  c = cmX.capabilitymaterials_Capability_id and
  uc = c.capability_Max_Length_Unit and
  uc.unitcode_Code="cm" and
  m = cmX.capabilitymaterials_Material_id and
  c = cc.capabilitycategories_Capability_id and
  posc = cc.capabilitycategories_ProductOrServiceCategory_id and
  (m.material_Material_Name="Pre-hardened Stainless Steel" or
   m.material_Material_Name="17-4 Stainless Steel") and
  (posc.productorservicecategory_Category_Name="Sinker EDM" or
   posc.productorservicecategory_Category_Name="Ram EDM")
```

Fig. 7. FQL syntax for Query 1 [5], translated from SQL
Fig. 8. Query result on initial data displayed in the FQL IDE

Fig. 9. Query result on enriched data displayed in the FQL IDE

References

1. Michael Barr and Charles Wells, editors. *Category theory for computing science, 2nd ed.* 1995.
2. Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
3. Michael Fleming, Ryan Gunther, and Robert Rosebrugh. A database of categories. *J. SYMBOLIC COMPUT.*, 35:127–135, 2002.
4. Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD ’05*.
5. Boonserm Kulvatunyou, Nenad Ivezic, and Yunsu Lee. On enhancing communication of the manufacturing service capability information using reference ontology. *International J. of Computer Integrated Manufacturing*, 27(12):1105–1135, 2014.
6. David I. Spivak. Functorial data migration. *Inf. Comput.*, 217:31–51, August 2012.
7. David I Spivak and Ryan. Wisnesky. Relational foundations for functorial data migration. [http://arxiv.org/abs/1212.5303](http://arxiv.org/abs/1212.5303) 2013.