




# Data Integration in a Multi-model Environment

Jaroslav Pokorný<sup>1,2</sup> 

<sup>1</sup> Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic  
jaroslav.pokorny@matfyz.cuni.cz, pokorny@savs.cz  
<sup>2</sup> Škoda Auto University, Mladá Boleslav, Czech Republic

**Abstract.** A multi-model approach to heterogeneous database (DB) integration requires a more user-friendly solution, i.e., a possibility to see various conceptual or data schemas in a unified way. We use a functional approach based on so-called attributes named by short natural language expressions with associated expressions describing their type. Attributes are functions that can be manipulated by a version of typed lambda calculus, which with arithmetic and aggregation functions enables to build a powerful query language. We consider the relational, E-R, JSON, and graph data/conceptual models. A query over such integrated DB can be expressed by a term of the typed lambda calculus. A more user-friendly version of such language can serve as a powerful query tool in practice.

**Keywords:** functional approach · typed lambda calculus · multi-model approach

## 1 Introduction

Historically, data integration is associated with distributed databases (DB) developed mainly in 80ties. These DBs used mostly the relational DB model, a global schema and local schemas for DBs were placed in multiple DB nodes in a network. Then, two approaches based on DB schemas management occurred:

- top-down – starting with a global schema to design schemas for particular data stores in network sites,
- bottom-up – i.e., to use a schema mapping for schemas of data stores in sites with a middleware (e.g., JDBC). The process consists of integrating local DBs with their (local) schemas into a global DB with its global schema.

We remind that the former concerns rather homogenous DB models used in integrated data stores, using usually relational DBs, while the latter supports various DB models and consequently heterogeneous database systems (DBS).

Now, systems that store and process Big Data have become a common component of data management architectures. Generally, Big Data can be a combination of (i) structured data in DBs and data warehouses based on SQL, (ii) semi-structured data, such as web server logs or streaming data from sensors, organized by the means of, e.g., RDF graphs or XML documents, or (iii) unstructured data, such as document (or text) collections. Here, we will consider categories (i) and (ii).

A traditional problem how to approach data in such environment is the way how data is integrated. The remainder of the paper is organized as follows. Section 2 presents a functional modelling of conceptual and DB structures including tools appropriate for their querying, i.e., typed functions and a typed lambda calculus. Section 3 explores some approaches to data integrations. Section 4 presents also functional querying integrated data. Finally, Sect. 5 provides conclusions and topics for future works.

## 2 Functional Data Modelling

We start from classic approaches to functional DBs, that use a version of functional typing and a typed lambda calculus in Sects. 2.1 and Sect. 2.2. (for more details, e.g., [6]). In Sect. 2.3, we present how functional conceptual structures *attributes* can be described by expressions of a natural language. Combining attributes and typed lambda calculus we obtain a powerful query language (QL) presented in Sect. 2.4.

### 2.1 Functional Data Types

We assume the existence of some *elementary types*  $S_1, \dots, S_k$  ( $k \geq 1$ ) constituting a *base*  $\mathbf{B}$ . More complex types are constructed in the following way:

If  $S, R_1, \dots, R_n$  ( $n \geq 1$ ) are types, then

- (i)  $(S:R_1, \dots, R_n)$  is a (*functional*) *type*,
- (ii)  $(R_1, \dots, R_n)$  is a (*tuple*) *type*.

The set of *types*  $\mathbf{T}$  over  $\mathbf{B}$  is the least set containing all types from  $\mathbf{B}$  and those given by (i)–(ii). When  $S_i$  in  $\mathbf{B}$  are interpreted as non-empty sets, then  $(S:R_1, \dots, R_n)$  denotes the set of all (total or partial) functions from  $R_1 \times \dots \times R_n$  into  $S$ ,  $(R_1, \dots, R_n)$  denotes the Cartesian product  $R_1 \times \dots \times R_n$ . Elementary type *Bool* = {TRUE, FALSE} is also in  $\mathbf{B}$ . It allows to model sets (resp. relations) as unary (resp. n-ary) characteristic functions. An object  $o$  of the type  $T$  is called a *T-object*. We denote it  $o/T$ . Logical connectives, quantifiers, and predicates are typed functions, e.g., **and**/(*Bool*: *Bool*, *Bool*) and **implies**/(*Bool*: *Bool*, *Bool*). Arithmetic operations are (*Number*: *Number*, *Number*)-objects. The aggregation functions have also associated types, e.g., **SUM**/(*Real*: (*Bool*: *Real*)). We use the infix notation for functions and arithmetic operations. We write ‘ $\forall x\dots$ ’ and ‘ $\exists x\dots$ ’, for application of the universal and existential quantifier, respectively. Relations are (*Bool*:  $S_1, \dots, S_m$ )-objects, where  $S_i$  are descriptive elementary types.

### 2.2 Typed Lambda Calculus

Let  $\mathbf{F}$  be a collection of constants, each having a fixed type, and suppose to have a denumerable set of variables of each type at disposal. The *language of lambda terms*  $\mathbf{LT}$  is defined as follows:

Let types  $R, S, R_1, \dots, R_n$  ( $n \geq 1$ ) be elements of  $\mathbf{T}$ . Then

- (1) Every variable of type  $R$  is a term of type  $R$ . (*variable*)
- (2) Every constant (a member of  $\mathbf{F}$ ) of type  $R$  is a term of type  $R$ . (*constant*)

- (3) If  $M$  is a term of type  $(S:R_1, \dots, R_n)$ , and  $N_1, \dots, N_n$  are terms of types  $R_1, \dots, R_n$ , respectively, then  $M(N_1, \dots, N_n)$  is a term of type  $S$ . (application)
- (4) If  $x_1, \dots, x_n$  are distinct variables of types  $R_1, \dots, R_n$ , respectively, and  $M$  is a term of type  $S$ , then  $\lambda x_1, \dots, x_n(M)$  is a term of type  $(S:R_1, \dots, R_n)$ . ( $\lambda$ -abstraction)
- (5) If  $N_1, \dots, N_n$  are terms with types  $R_1, \dots, R_n$ , respectively, then  $(N_1, \dots, N_n)$  is a term of type  $(R_1, \dots, R_n)$ . (tuple)
- (6) If  $M$  is a term of type  $(R_1, \dots, R_n)$ , then  $M[1], \dots, M[n]$  are terms of respective types  $R_1, \dots, R_n$ . (components)

Terms can be interpreted by an interpretation assigning to each function from  $\mathbf{F}$  an object of the same type, and a semantic mapping from  $\mathbf{LT}$  into all functions and Cartesian products given by the type of system  $\mathbf{T}$ . Briefly, an application is evaluated as the application of an associated function to its arguments, the  $\lambda$ -abstraction “constructs” a new function. A tuple is a member of the Cartesian product of sets of typed objects.

### 2.3 Conceptual Modelling with Attributes

In general, attributes are parametrized by *possible worlds* (elementary type  $w$ ) and *time moments* (elementary type  $t$ ). Mathematical/logical functions are not dependent on  $w$  and  $t$ . For simplicity, we will not assume either possible worlds or time moments in the paper. For example,  $\text{ACTORS}/(\text{Bool}:\text{Name}, \text{Title}, \text{Role})$  and  $\text{MOVIES}/(\text{Bool}:\text{Title}, \text{Released}, \text{Director}, \text{Genre})$  represent named attributes - relations.

$\text{A\_JOURNAL\_TO\_WHICH\_THE\_USER\_CONTRIBUTES}/(\text{Journal}:\text{User}),$   
 $\text{MOVIES\_RATED\_BY\_A\_USER}/((\text{Bool}:\text{Stars}, \text{Movie}):\text{User}),$

are rather functional attributes. We will denote them  $\text{JU}$  and  $\text{SMU}$ , respectively.

Other conceptual constructions are *propositions* of type  $\text{Bool}$ . Attributes generate certain basic propositions, e.g., “Mr. Baker contributes to the journal Computer Reviews”. It is generated by the  $\text{JU}$  attribute. A *conceptual schema* is a tuple of attribute specifications and, possibly, a set of integrity constraints, i.e., certain propositions giving explicitly some information about attributes. An *information base* is a set of TRUE-propositions induced by attributes in an actual world and in a given time moment. Obviously, all known conceptual constructs used in conceptual modelling are cases of attributes. In [4] and [7] we applied this approach to XML and JSON data, respectively.

### 2.4 Querying with Attributes

The  $\mathbf{LT}$  language can be used as a theoretical tool for building a functional  $\mathbf{QL}$ . The choice of functions determines the expressive power of  $\mathbf{QL}$ . A query in such language is expressed by a  $\mathbf{LT}$  term, e.g.,

$$\lambda u^{User}, n^{Number} (n = \text{COUNT}(\lambda m^{Movie} (\exists s^{Stars} \text{SMU}(u)(s, m))))$$

of type  $(\text{Bool}:\text{User}, \text{Number})$ . Indexes of variables denote their types. The query means “Find for each user the number of rated movies”.

A more complex example of a term uses a universal quantifier and implication:

$$\lambda n^{Name}(\forall t^{Title}(\exists re^{Rolasede}, g^{Genre} \text{MOVIES}(t^{Title}, re^{Released}, 'Spielberg', Director, g^{Genre}) \text{implies } \exists ro^{Role} \text{ACTORS}(n^{Name}, t^{Title}, ro^{Role})))$$

expressing the query “Find the names of actors, who play in each Spielberg film.”

We gain a tool for common manipulation of relations and other typed functions. Then, the query results can be relations, nested relations, typed functions, etc. For Boolean queries, YES/NO can be a query result. It is important that there is no sharp line between conceptual and DB modelling with the functional approach. An application of the typed lambda calculus with equality is used in the approach of Hillebrand [2].

### 3 Multi-model Approach to Data Integration

Today, polystores and multi-model DBs are considered for DBs with multiple data stores [3]. In a *polystore* multiple storage engines are distinct and accessed separately through their own query engines. A more user-friendly solution of heterogeneous DB integration, is referred to as *multi-model DBs*. Typically, the relational data model can be one of them [1]. The query is then executed on more data sources, but an additional layer is often used to enable data integration.

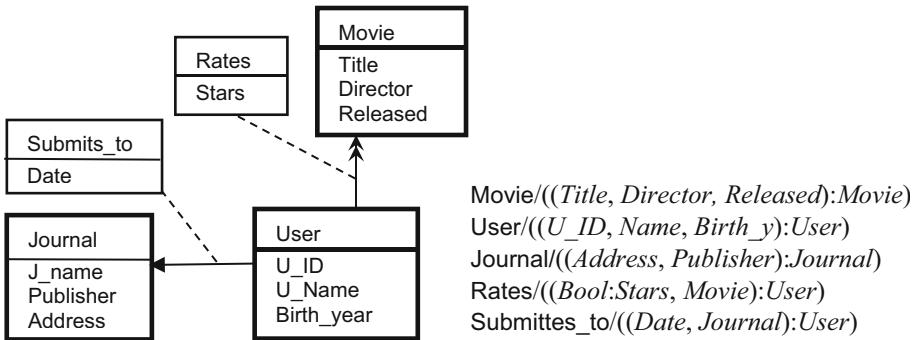


Fig. 1. GDB conceptual schema Movies and its functional version

The notion of attribute applied in GDBs can be restricted to attributes of types (R:S), (Bool(R):S), or (Bool:R, S), where R and S are entity types. This strategy simply covers binary functional types, binary multivalued functional types, and binary relationships described as binary characteristic functions. The last option corresponds to M:N relationship types. For modelling directed graphs, the first two types are sufficient, because M:N relationship types can be expressed by two “inverse” binary multivalued functional types. For graphical expressing a graph conceptual schema, we use two types of arrows according to associated binary functional types (see Fig. 1).

Properties describing entity types can be of types (S<sub>1</sub>, ..., S<sub>m</sub>:R), where S<sub>i</sub> are descriptive elementary types and R is an entity type. They are of types (S<sub>1</sub>, ..., S<sub>m</sub>, R<sub>1</sub>:R<sub>2</sub>) and ((Bool:S<sub>1</sub>, ..., S<sub>m</sub>, R<sub>1</sub>):R<sub>2</sub>) for binary functional and binary multivalued functional types, respectively. Functional querying in GDBs is described, e.g., in [5].

For relational DBs, we can assume the existence of an E-R schema describing the semantics of relations. Here we use attributes for conceptual schemas based on E-R

models and sufficiently structured approach for expressing semantics of data in particular NoSQL DBs. The database schemas of these DBs are then described by sets of attributes, i.e. rather as *local conceptual schemas* (LCSs), a global schema is obtained by union of these LCSs. Such approach can be generalized to most NoSQL DBs [8].

In the case of NoSQL, even more than one data model is often included in one DB architecture. For example, the distributed DB Cassandra combines column-based and key-value data models, DynamoDB combines document-oriented and key-value data models. ArangoDB also represents a multi-model approach, meaning that it can address JSON documents, graphs, and key-values. OrientDB is a multi-model DB including geospatial, graph, fulltext, and key-valued data models. MarkLogic enables to store and search JSON and XML documents and RDF triples. In [10] the gap between SQL and NoSQL is solved via an abstraction level in which the NoSQL data are transformed to triples incorporated into SQL DB as virtual relations.

## 4 Querying Multi-model Data

In literature, we can find two basic general frameworks for unified modelling and management of multi-model data. The categorical approaches described [3, 9] use category theory for transformations between models and are usable also for conceptual querying. Querying multi-model data by a functional approach means to describe DB structures in particular DBs functionally by attributes. It means, in principle, that LCSs are specified. Since sets (relations) are modelled as their characteristic functions, we gain a tool for common manipulation of relations and functional data from NoSQL DBs. In consequence, the query results can be relations, nested relations or XML [4], JSON [7], graph data [5] as well, again expressed by **LT** terms.

Another approach uses a global schema similarly to the ANSI/SPARC approach. In such logical integration, the *global conceptual (or mediated) schema* (GCS) is entirely virtual and not materialized. The bottom-up design involves both the generation of the GCS and the mapping of individual LCSs to this GCS. In any case, there are difficulties in schema integration, because of different structures and semantics among local DBs. Details of integration of relational DBs and GDBs functionally are described in [6]. Data selection is performed in the source systems using SQL and Cypher. The results are mapped into data structures associated with the source query term.

*Example 1:* Suppose the relational attributes {ACTORS, MOVIES} from Sect. 2.3 and GDB described in Fig. 1, i.e., attributes {Movie, User, Journal, Rates, Submittes\_to}. In the integrated DB, i.e., the multi-model system, the term in the simplified notation

$$\lambda u^{User} \lambda g^{Genre}, n^{Number} \\
(n^{Number} = \text{COUNT}_{Movie} (\lambda m^{Movie} (\text{Rates}(u^{User})(m^{Movie}) \text{ and} \\
\exists t^{Title} s^{Title} \text{Movie}(m^{Movie}), t^{Title} = s^{Title} \text{ and MOVIES}(s^{Title}, g^{Genre}))) \\
))$$

expresses the query “Find for each user the genres and the number of reviews he/she made in them”. The answer will be of type  $((\text{Bool}: \text{Genre}, \text{Number}), \text{User})$ , i.e. a new multivalued attribute assigning to each user a binary relation with tuples containing a

genre and the number of the rates created for this genre by a given user. The query term is decomposed and transformed into a query program that requires evaluation of the included attributes, e.g., by SQL and Cypher expressions, respectively. These partial results serve to the integration that generates the query result.

## 5 Conclusions

In the paper, we have focused on integration of relational and NoSQL DBs. Even a variant of the E-R model can be used in without problems. Formally, we used a functional typing system serving for specification of so-called attributes. The attributes can be named with expressions of a natural language, bringing database querying closer to conceptual querying. A typed lambda calculus can be used as a manipulation language.

The presented tools create a formal background covering querying an integrated multi-model DB. Such a language could be based on SQL-like syntax, in principle. Another interesting topic for research is the expressive power of the subsets of **LT** considered, the solution of their user variants, and the complexity of formulating queries in such apparatus. In general, the expressive power of a user QL depends on a choice of constant functions included into the QL. These are themes for future work.

## References

1. Candel, C.J.F., Sevilla Ruiz, D., García-Molina, J.J.: A unified metamodel for NoSQL and relational databases. *Inf. Syst.* **104**, 101898 (2022)
2. Hillebrand, G., Kanellakis, P.C., Mairson, H.G.: Database query languages embedded in the typed lambda calculus. *Inf. Comput.* **127**(2), 117–144 (1996)
3. Lu, J., Holubová, I., Cautis, B.: Multi-model databases and tightly integrated polystores: current practices, comparisons, and open challenges. In: Proceedings of the CIKM 2018, 27th ACM International Conference on Information and Knowledge Management, pp. 2301–2302 (2018)
4. Pokorný, J.: XML functionally. In: Desai, B.C., Kioki, Y., Toyama, M. (eds.) Proceedings of the IDEAS2000, pp. 266–274. IEEE Computer Society (2000)
5. Pokorný, J.: Functional querying in graph databases. In: Nguyen, N., Tojo, S., Nguyen, L., Trawiński, B. (eds.) ACHIDS 2017, Part I. LNCS, vol. 10191, pp. 291–301. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-54472-4\\_28](https://doi.org/10.1007/978-3-319-54472-4_28)
6. Pokorný, J.: Integration of relational and NoSQL databases. *Vietnam J. Comput. Sci.* **6**(4), 389–405 (2019)
7. Pokorný, J.: JSON functionally. In: Proceedings of ADBIS 2020, Lyon, France, August 25–27, pp. 139–153 (2020)
8. Pokorný, J., Richta, K.: Towards conceptual and logical modelling of NoSQL databases. In: Insfran, E., et al. (eds.) Advances in Information Systems Development. LNISO, vol. 55, pp. 255–272. Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-95354-6\\_15](https://doi.org/10.1007/978-3-030-95354-6_15)

9. Svoboda, M., Čontoš, P., Holubová, I.: Categorical modelling of multi-model data: one model to rule them all. In Proceedings of the 10th International Conference on MEDI 2021, Tallin, pp. 190–198 (2021)
10. Thant, P.T., Naing, T.T.: Hybrid query processing system (HQPS) for heterogeneous databases (relational and NoSQL). In: Proceedings of the International Conference on Computer Networks and Information Technology, pp. 53–58 (2014)