Flexible Relational Data Model – A Common Ground for Schema-Flexible Database Systems

Hannes Voigt and Wolfgang Lehner

Database Technology Group,
Technische Universität Dresden,
01062 Dresden, Germany
{firstname.lastname}@tu-dresden.de
http://wwwdb.inf.tu-dresden.de/

Abstract. An increasing number of application fields represent dynamic and open discourses characterized by high mutability, variety, and pluralism in data. Data in dynamic and open discourses typically exhibits an irregular schema. Such data cannot be directly represented in the traditional relational data model. Mapping strategies allow representation but increase development and maintenance costs. Likewise, NoSQL systems offer the required schema flexibility but introduce new costs by not being directly compatible with relational systems that still dominate enterprise information systems. With the Flexible Relational Data Model (FRDM) we propose a third way. It allows the direct representation of data with irregular schemas. It combines tuple-oriented data representation with relation-oriented data processing. So that, FRDM is still relational, in contrast to other flexible data models currently in vogue. It can directly represent relational data and builds on the powerful, well-known, and proven set of relational operations for data retrieval and manipulation. In addition to FRDM, we present the flexible constraint framework FRDM-C. It explicitly allows restricting the flexibility of FRDM when and where needed. All this makes FRDM backward compatible to traditional relational applications and simplifies the interoperability with existing pure relational databases.

Keywords: data model, flexibility, relational, irregular data.

1 Introduction

Today's databases are deployed in diverse and changing ecosystems. An increasing number of application fields is characterized by high mutability, variety, and pluralism in the data. High mutability is caused by the persistent acceleration of society [16] and technological development [19]. Variety appears in database discourses because information systems extending their scope and strive to cover every aspect of the real world. Pluralism is inevitable with the onging cross linking of information systems and the consolidation of data from different stackholders in a single database. Particular drivers of these developments are end

Y. Manolopoulos et al. (Eds.): ADBIS 2014, LNCS 8716, pp. 25–38, 2014.

user empowerment [20], agile software development methods [6], data integration [14,26], and multi-tenancy [17]. The once stable and closed discourses of databases are rather dynamic and open today.

In such dynamic and open discourses, data often has a irregularly structured schema. Data with an irregular schema exhibits four characteristic traits: (1) multifaceted entities that cannot be clearly assigned to a single entity type, (2) entities with varying sets of attributes regardless of their entity type, (3) attributes occuring completely independent of particular entity types, and (4) attribute-independent technical typing of values. All four traits cannot be directly represented in the relational data model. As a natural reaction, many developers perceive traditional relational data management technologies as cumbersome and dated [18]. Various mapping strategies [15,1,4] allow a presentation but they imply additional costs of implementation and maintenance. Further, the inherent logical schema of the data is not complete visible on the resulting relational data. This schema incompleteness particularly is a problem in the likely case that multiple applications access a database through different channels.

In recent years, the NoSQL movement has introduced a number of new data models, query languages, and system architectures that exhibit more flexibility regarding the schema. Many NoSQL systems allow the direct representation of data with irregular schema as well as the gradual evolution of the schema. Hence, NoSQL systems appear to be a very appealing choice for applications with a dynamic and open discourse. However, the introduction of new data models, query languages, and system architectures is not for free. Particularly in enterprise environments where 90 % of the databases are relational [9] new data models are often a bad fit. They imply additional costs for mapping and transforming data between different data models and require new database management and application development skills. These costs multiply if applications store the different parts of their data in the respectively ideal data model across different database management systems – a scenario often referred to as *polyglot persistency* in the NoSQL context.

With the Flexible Relational Data Model (FRDM) we propose a third way. It allows the direct representation of data with irregular schemas from dynamic and open discourses. This includes multifaceted entities, variable attributes sets, independent attributes, as well as independent technical types. At the same time FRDM remains 100 % backward compatible to the traditional relational data model. Purely relational data with regular schemas can also be represented and relationally processed in FRDM directly. FRDM achieves this centering the data representation around the individual tuples while maintaining the relation as the primary means of data processing.

Additionally, we present the flexible constraint framework FRDM-C that provides explicit restrictions to the flexibility of FRDM. Scope and range of the restriction can be tailored to any requirements ranging from the constraint-free, descriptive nature of pure FRDM to the strictly prescriptive nature of the traditional relational data model. FRDM-C helps to introduce rigidity exactly when and at which parts of data needed. FRDM-C constraints can vary in their effect

from simply informing to strictly prohibiting, so that they are not only a tool to maintain data quality but also help achieving data quality.

The remainder of this paper is structured as follows. Section 2 presents the FRDM data model, in particular how it represents data and how data is processed in FRDM. The constraint framework FRDM-C is discussed in Section 3. With both introduced, Section 4 shows how pure relational data can be represented directly in FRDM to demonstrate the backward compatibility of FRDM. This is followed by considerations regarding the implementation of FRDM within the architecture of relational database management systems in Section 5. In Section 6, we compare FRDM with other data models regarding the provided flexibility and backward compatibility. Finally, Section 7 concludes the paper.

2 FRDM

FRDM is a relational data model for structured data. It is free of the relational inflexibilities but remains directly compatible to the relational model. The most prominent feature of FRDM is that it separates the functionality of data representation, data processing, and constraints. Data representation and data processing are realized in separate, dedicated concepts. We detail the data representation of FRDM in Section 2.1 and discuss data processing in Section 2.2. Schema constraints are realized as explicit constraints outside of the core data model in the constraint framework FRDM-C, which is presented in Section 3.

2.1 Data Representation

The data representation of FRDM builds on four concepts. The central concept is the *tuple*:

Tuple. A tuple is the central concept of the flexible relational data model and represents an entity. It consists of values, each belonging to an attribute and is encoded according to a technical type.

The concepts *entity domain*, *attribute*, and *technical type* describe data represented in tuples and provide logical data handles:

Entity Domain. Entity domains are logical data handles allowing to distinguish logical groups of tuples within a database. Tuples belong to at least one entity domain and may belong to multiple entity domains, so that domains can intersect each other.

Attribute. Attributes are logical data handles allowing to distinguish values within a tuple. Each tuple can instantiate each attribute only once.

Technical Type. Technical types determine the physical representation of values. Value operations such as comparisons and arithmetic are defined on the level of technical types.

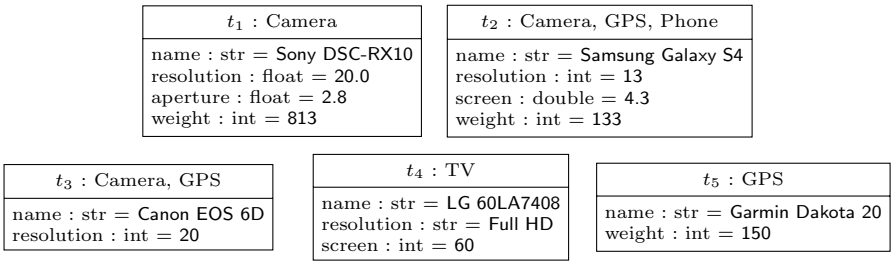

Fig. 1. Example entities representing electronic devices

Tuples:

$$\mathbb{D} = \left\{ \begin{array}{l} t_1 = [\text{Sony DSC-RX10}, 20.0, 2.8, 813]\,, \\ t_2 = [\text{Samsung Galaxy S4}, 13, 4.3, 133]\,, \\ t_3 = [\text{Canon EOS 6D}, 20]\,, \\ t_4 = [\text{LG 60LA7408}, \text{Full HD}, 60]\,, \\ t_5 = [\text{Garmin Dakota 20}, 150] \end{array} \right\}$$

Description elements:

$$\mathbb{A} = \{aperture, name, resolution, screen, weight\}$$

$$\mathbb{T} = \{float, int, str\}$$

$$\mathbb{E} = \{Camera, GPS, Player, Phone, TV\}$$

Schema function:

$$f_s = \left\{ \begin{array}{l} t_1 \to [name, resolution, aperture, weight]\,, \\ t_2 \to [name, resolution, screen, weight]\,, \\ t_3 \to [name, resolution]\,, \\ t_4 \to [name, resolution, screen]\,, \\ t_5 \to [name, weight] \end{array} \right\}$$

Membership function:

$$f_m = \left\{ \begin{array}{l} t_1 \to \{Camera\}\,, \\ t_2 \to \{Camera, GPS, Phone\}\,, \\ t_3 \to \{Camera, GPS\}\,, \\ t_4 \to \{TV\}\,, \\ t_5 \to \{GPS\} \end{array} \right\}$$

Typing function:

$$f_t = \left\{ \begin{array}{l} (t_1, name) \to str, (t_1, resolution) \to float, (t_1, aperture) \to float, (t_1, weight) \to int, \\ (t_2, name) \to str, (t_2, resolution) \to int, (t_2, screen) \to double, (t_2, weight) \to int, \\ (t_3, name) \to str, (t_3, resolution) \to int, \\ (t_4, name) \to str, (t_4, resolution) \to str, (t_4, screen) \to int, \\ (t_5, name) \to str, (t_5, weight) \to int \end{array} \right\}$$

Fig. 2. Example entities in the flexible relational data model

Formally, a flexible relational database is a septuple $(\mathbb{D}, \mathbb{A}, \mathbb{T}, \mathbb{E}, f_s, f_t, f_m)$. The payload data $\mathbb{D}$ is a set of tuples. A tuple is an ordered set of values $t = [v_1, \ldots, v_m]$. Let $\mathbb{A}$ be the set of all attributes available in the database. Then the tuple schema function $f_s : \mathbb{D} \to \mathcal{P}(\mathbb{A}) \setminus \emptyset$ denotes the schema of each tuple, i.e., the set of attributes a tuple instantiates. $f_s(t) = [A_1, \ldots, A_m]$ if t instantiates the attributes $A_1, \ldots, A_m$ so that $t \in A_1 \times \cdots \times A_m$. For convenience, we denote with $t[A] = v$ that tuple t instantiates attribute A with value v. $\mathbb{T}$ is the set of all available technical types T. The typing function $f_t : \mathbb{D} \times \mathbb{A} \to \mathbb{T}$ shows the encoding of values, with $f_t(t, A) = T$ if the value $t[A]$ is encoded according to the technical type T. Finally, $\mathbb{E}$ is the set of all available entity domains E, while the membership function $f_m : \mathbb{D} \to \mathcal{P}(\mathbb{E}) \setminus \emptyset$ denotes which tuples belong to these domains. $f_m(t) = \{E_1, \ldots, E_k\}$ if t belongs to the entity domains $E_1, \ldots, E_k$.

As an example, Figure 1 shows six entities in a UML object diagram like notation. The entities represent electronic devices as they could appear in a product catalog. Note that this small example exploits all the flexibilities of FRDM.

Relation *GPS*

name	resolution	screen	weight
Samsung Galaxy S4	13	4.3	133
Canon EOS 6D	20	∄	∄
Garmin Dakota 20	∄	∄	150

Fig. 3. Relation by entity domain

Relation *Camera* ∩ *GPS*

name	resolution	screen	weight
Samsung Galaxy S4	13	4.3	133
Canon EOS 6D	20	∄	∄

Fig. 4. Relation from relational operator

All six entities are self-descriptive and have their individual set of attribute. The order of the attributes within an entity differs, too. Entities t_2 and t_3 belong to multiple entity domains. Attributes, such as *name*, appear independently from entity domains. The technical typing of values, for instance of the attribute *resolution*, varies independently from the attribute. In the flexible relational data model these six entities can be represented directly as shown in Figure 2.

2.2 Data Processing

For data processing, FRDM builds on the well-known concept of a *relation*. It allows processing tuples in a relational manner:

Relation. Relations serve as central processing containers for tuples. FRDM queries operate on relations; query operations have relations as input and produce relations as output. The tuples in a relation determine the schema of the relation. Each attribute instantiated by at least a single tuple in the relation is part of the relation's schema.

Let t be a tuple in relation R, then R has the schema $S_R = \bigcup_{t \in R} f_s(t)$ so that $S_R \subseteq \mathbb{A}$. A relation R with schema S_R does not have to instantiate each tuple in every attribute, rather it is $R \subseteq \bigcup_{S_i \in \mathcal{P}(S_R) \setminus \emptyset} \left(\bigtimes_{A \in S_i} A \right)$. In other words, tuples may only instantiate a subset of a relation's schema, except the empty set. While $t[A] = v$ denotes that tuple t instantiates attribute A with value v, $t[A] = \nexists$ indicates that tuple t does not instantiate attribute A.

Mass operations address tuples by means of entity domains. Hence, each entity domain denotes a relation containing all tuples that belong to this domain. Specifically, an entity domain E denotes a relation R so that $E \in f_m(t)$ holds for all $t \in R$. In the following, we refer to a relation representing tuples of domain E simply as E where unambiguously possible. Figure 3 shows the relation denoted by the entity domain *GPS* in the electronic device example.

The well-known relational operators are applicable directly to FRDM relations. However, the descriptive nature of a FRDM relation requires two minor modifications to their semantics. First, the logic of selection predicates and projection expressions has to take into account that attributes may not be instantiated by a tuple. An appropriate evaluation function for such predicates and expressions is described in [28]. In a nutshell, tuples that do not instantiate an attribute used in a selection predicate are not applicable to the predicate and

do not qualify. Tuples that do not instantiate an attribute used in a projection expression do not instantiate the attribute newly defined by the expression. Second, all operations have a strictly tuple-oriented semantics, i.e., the schema of the relation resulting from an operation is solely determined by the qualifying tuples. In consequence, the schema resulting from a selection can differ from the schema of the input relation. More specifically, the resulting schema of a selection is equal to or a subset of the input schema depending on which tuples qualify, so that $S_{\sigma_P(A)} \subseteq S_A$. Likewise, the schemas of the operand relations do not matter for set operations. Tuples are equal if they instantiate the same attributes with equal values. For a union, the resulting schema is the union of the schemas of the operands, so that $S_{A \cup B} = S_A \cup S_B$. For set difference, the resulting schema is equal to or a subset of the left operand's schema, again, depending on which tuples qualify, so that $S_{A \setminus B} \subseteq S_A$. Derived operators, such as join or intersection, are affected similarly. As an example, Figure 4 shows the relation resulting from the intersection of the relation *GPS* (cf. Figure 3) and the relation denoted by entity domain *Camera*.

3 FRDM-C

FRDM-C is a flexible constraint framework meant to accompany FRDM. The flexibility of FRDM originates from its lack of implicit constraints. Nevertheless, constraints are a powerful feature if their effect is desired by the user. For the user, constraints are the primary means to obtain and maintain data quality. Each constraint is a proposition about data in the database. Data either complies to or violates this proposition, i.e., every constraint categorizes data into two disjoint subsets. It is up to the user how to utilize this categorization. At least, constraints inform about which data is compliant and which is violating. At most, constraints prohibit data modifications that would result in violating data. Constraints present themselves as additional schema objects, attached to the schema elements of the data model. The user can add and remove constraints any time.

Formally, constraints take the general form of a triple (q, c, o). q is the qualifier; c is the condition compliant data has to fulfill; o is the effect (or the outcome) the constraint will have. The qualifier determines to which tuples the constraint applies. It is either an entity domain $E_q \in \mathbb{E}$, a attribute $A_q \in \mathbb{A}$, or a pair of both (E_q, A_q). Correspondingly, a constraint applies to all tuples t with $E_q \in f_m(t)$, with $A_q \in f_s(t)$, or with $(E_q, A_q) \in f_m(t) \times f_s(t)$, respectively. We denote the set of tuples a constraint C applies to as $\mathbb{D}_C$. Conditions are either tuple conditions or key conditions, depending on whether they affect individual tuples or groups of tuples. The effect determines the result of the operations that lead to violating data and what happens to the violating data itself. In the following, we will detail conditions and effects.

3.1 Conditions

The first group of conditions is tuple conditions. Tuple conditions restrict data on the level of individual tuples, e.g., by mandating to which entity domains a tuple can belong. Formally, a tuple condition is a function $c : \mathbb{D} \to \{\top, \bot\}$. Then, $\mathbb{D}_C^{\top} = \{t \mid t \in \mathbb{D}_C \wedge c(t)\}$ are the complying tuples and $\mathbb{D}_C^{\bot} = \{t \mid t \in \mathbb{D}_C \wedge \neg c(t)\}$ are the violating tuples. Tuple conditions are:

Entity Domain Condition. An entity domain condition requires tuples $t \in \mathbb{D}_C$ to belong to an entity domain E_c so that $E_c \in f_m(t)$. We denote a specified entity domain condition as *entity-domain*(E_c).

Attribute Condition. A attribute condition requires tuples $t \in \mathbb{D}_C$ to instantiate a attribute A_c so that $A_c \in f_s(t)$. We denote a specified attribute condition as *value-domain*(A_c).

Technical Type Condition. A technical type condition limits values of tuples $t \in \mathbb{D}_C$ in attribute A_c to a specified technical type T_c so that $T_c = f_t(t, A_c)$. We denote a technical type condition as *tech-type*(A_c, T_c).

Value Condition. A value condition requires values of tuples $t \in \mathbb{D}_C$ in attribute A_c to fulfill a specified predicate p so that $p(t[A_c])$ holds. We denote a value condition as *value*(A_c, p).

The second group of conditions is key conditions. Key conditions restrict data on the level of tuple groups. Formally, a key condition is a function $c : \mathcal{P}(\mathbb{D}) \to \{\top, \bot\}$. Key conditions are:

Unique Key Condition. A unique key condition requires tuples to instantiate a set of attributes $\mathbb{A}_K \subseteq \mathbb{A}$ uniquely so that $t_i[\mathbb{A}_K] \neq t_j[\mathbb{A}_K]$ holds for all $t_i, t_j \in \mathbb{D}_C$ with $t_i \neq t_j$. As a result, all complying tuples are unambiguously identifiable on $\mathbb{A}_K$. We denote a unique key condition as *unique-key*$(\mathbb{A}_K)$.

Foreign Key Condition. A foreign key condition requires tuples to instantiate attributes $\mathbb{A}_F \subseteq \mathbb{A}$ with values referencing at least one tuple on attributes $\mathbb{A}_R \subseteq \mathbb{A}$ so that for every $t_F \in \mathbb{D}_C$ there is one $t_R \in \mathbb{D}_R$ so that $t_F[\mathbb{A}_F] = t_R[\mathbb{A}_R]$. Similarly to $\mathbb{D}_C$, the set of referenceable tuples $\mathbb{D}_R \subseteq \mathbb{D}$ is identified by either an entity domain $E_R \in \mathbb{E}$, a attribute $A_R \in \mathbb{A}$, or a pair of both (E_R, A_R). We denote a foreign key condition as *foreign-key*$(\mathbb{A}_F, \mathbb{A}_R, q_R)$ where q_R is the qualifier of $\mathbb{D}_R$.

If a group of tuples does not fulfill a key condition, not all tuples of the group are considered to be violating. We have to distinguish two cases. In the first case, a constraint already exists in the database and a modification of tuples results in a violation. Here, only the modified tuples become violating tuples. In the second case, the constraint is added to the database and the tuples already existing in the database violate this constraint. Here the smallest subset of tuples that violates the condition becomes the set of violating tuples. For a unique key constraint, these are all duplicates. For a foreign key constraint, these are all tuples with a dead reference.

All conditions can be negated in a constraint. Negation swaps the set of violating tuples with the set of complying tuples. For instance, the negated entity

domain condition $\neg$*entity-domain*(E_c) prohibits the entity domain E_c instead of requiring it. For two constraints $C = (q, c, o)$ and $C' = (q, \neg c, o)$, it holds that $\mathbb{D}^{\top}_{C'} = \mathbb{D}^{\bot}_{C}$ and $\mathbb{D}^{\bot}_{C'} = \mathbb{D}^{\top}_{C}$. Which tuples violate a constraint is crucial for the effect of the constraint.

3.2 Effects

We distinguish four types of effects constraints can have. They vary in the rigor the constraint will exhibit.

Informing. Allows all operations. The complying tuples and the violating tuples can be queried by using the constraint as a query predicate.

Warning. Allows all operations and issues a warning upon operations that lead to violating tuples. The creation of the constraint results in a warning about already existing violating tuples.

Hiding. Allows all operations and issues a warning upon operations that lead to violating tuples and hides violating tuples from all other operations. The creation of the constraint results in hiding already existing violating tuples except for operations that explicitly request to see violating tuples by using the constraint as predicate.

Prohibiting. Prohibits operations that lead to violating tuples and issues an error. The creation of the constraint is prohibited in case of already existing violating tuples.

4 Presentation of Purely Relational Data

The presented flexible relational data model is a superset of the traditional relational model. Traditional relations can be represented directly in the flexible model. A relational database is a septuple $(\mathbb{D}, \mathbb{A}, \mathbb{T}, \mathbb{R}, f_\sigma, f_\theta, f_\mu)$, where $\mathbb{R}$ is the set of relations, $\mathbb{A}$ is the set of domains, $\mathbb{T}$ is the set of technical types, $\mathbb{D}$ is the set of tuples, f_σ is the schema function $\mathbb{R} \to \mathcal{P}(\mathbb{A})$, f_θ is the typing function $\mathbb{A} \to \mathbb{T}$, and f_μ is the membership function $\mathbb{D} \to \mathbb{R}$. The corresponding flexible relational database is $(\mathbb{D}, \mathbb{A}, \mathbb{T}, \mathbb{E}, f_s, f_t, f_m)$ with

$$
\begin{aligned}
\mathbb{E} &= \{\textit{name-of}(R) \mid R \in \mathbb{R}\}\,, \\
f_s &= \{t \to f_\sigma(f_\mu(t)) \mid t \in \mathbb{D}\}\,, \\
f_t &= \{(t, A) \to f_\theta(A) \mid A \in f_\sigma(f_\mu(t)) \wedge t \in \mathbb{D}\}\,, \text{ and} \\
f_m &= \{t \to \{\textit{name-of}(f_\mu(t))\} \mid t \in \mathbb{D}\}\,.
\end{aligned}
$$

To emulate the model-inhernt constraints of the relational model the flexible relational database has to be supplemented with explicit constraints. For each relation $R \in \mathbb{R}$ we add the following prohibitive ($P \hat{=} \textit{prohibiting}$) constraints:

- Entity domains have to mutually exclude each other, so that tuples can be only part of one entity domain. This can be achieved with constraints of the form $(\textit{name-of}(R), \neg \textit{entity-domain}(E), P)$ where $\textit{name-of}(R) \neq E$ and $R \in \mathbb{R}$.

- Entity domains prescribe the attributes of their corresponding relation. This can be achieved with constraints of the form $(\textit{name-of}(R), \textit{value-domain}(A), P)$ for $A \in f_\sigma(R)$ and $R \in \mathbb{R}$.
- Entity domains forbid all other attributes. This can be achieved with constraints of the form $(\textit{name-of}(R), \neg\textit{value-domain}(A'), P)$ for $A' \notin f_\sigma(R)$ and $R \in \mathbb{R}$.
- Attributes prescribe the technical type as defined by the corresponding relation. This can be achieved with constraints of the form $(A, \textit{tech-type}(A, f_\theta(A)), P)$ for $A \in f_\sigma(R)$ and $R \in \mathbb{R}$.

5 Implementation Consideration

The FRDM data model is positioned as a flexible descendant of the relational model. Therefore it is suitable to be implemented within the existing and established relational database system architecture. In this section, we briefly discuss how this can be done. The characteristics of FRDM require four main changes to existing relational database system code.

First, plan operators and query processing have to be adapted to handling descriptive relations. More specifically, plan operators must reflect the adapted semantics of their logical counterparts. Logically, operators have to remove attributes from the schema of a relation if no tuple instantiates them. With a tuple-at-a-time processing model, this orphaned attribute elimination is a blocking operation, since the system can determine the schema only after all tuples are processed. Implicit duplicate elimination is similarly impractical and thus it was not implemented in relational database systems. Likewise a practical solution for the elimination of orphaned attributes is that plan operators determine the schema of the resulting operation as narrow as they safely can before the actual tuple processing and accept possible orphaned attributes in the result relation. Similar to the DISTINCT clause, SQL can be extended with a, say, TRIM clause that allows the user to explicitly request orphaned attribute elimination.

Second, the physical storage of tuples has to be adapted to the representation of entity domains. For tuple storage, the existing base table functionality can be reused but needs to be extended to handle uninstantiated attributes. Solutions for such an extension are manifold in literature, e.g., interpreted record [7,11], vertical partitioning [1], and pivot tables [3,13]. Another reasonable approach is a bitmap as it is used for instance by PostgreSQL [24] to mark NULL values in records. Tuples can appear in multiple entity domains. However, for storage economy and update efficiency, tuples should only appear in a single physical table. Replication should be left to explicit replication techniques. Consequently, the database system has to assign each tuple to a single physical table and maintain its logical entity domain membership somehow. In principle, there are two ways how this can be done. One is to encode the domain membership in the physical table assignment. Here, the system would create a physical table for each combination of entity domains occurring in a tuple and store tuples in the corresponding table. The mapping is simple and easy to implement.

The downside is that it may lead to a large number of potentially small physical tables (at worst 2^E tables where E is the number of entity domains in a database) and tuples need to be physically moved if their domain membership is changed. The other way is storing the domain membership, e.g., with a bitmap, directly in a tuple itself. This gives liberty regarding the assignment of tuples to physical tables, up to using a single (universal) table for all tuples. With many tuples having the same domain membership, it comes to the price of storage overhead – negligible in most cases, though.

Third, the physical tuple layout has to be extended to also represent the technical type of values directly in the tuple. This is necessary for independent technical types. To reduce storage needs and decrease interpretation overhead, the system can omit the technical type in the tuple where explicit constraints prescribe a technical type. However, creating and dropping such explicit constraints becomes expensive as the physical representation of the affected tuples has to be changed.

Fourth, independent attributes require a modification of the system catalog. In most system catalogs, attributes have a reference to the base table they belong to. This reference has to be removed to make attributes available to all tuples regardless of their entity domain membership.

6 Related Work

Over decades, research and development have created numerous data models and approaches to represent data. Obviously, we can concentrate only on the most prominent ones used for representing structured data. Data models worth considering can be grouped in four main categories: (1) relational models, (2) software models, (3) document models, (4) tabular models, (5) graph data models, and (6) models from the data modeling theory. In the following, we will briefly discuss these categories with regard to the flexibility to directly represent data of dynamic and open discourses.

Relational models are extensions of the traditional relational model [28,7,2,5]. These extensions intend to free the relational model from one or more implicit constraints. Hence, these extended relational models allow additional flexibility compared to the pure relational model. Specifically, reasonable extensions exist to support variable attribute sets. Besides, all these extensions preserve 100 % compatibility with the relational model. To the best of our knowledge, there are no extensions that add support for multifaceted entities, independent attributes, and independent technical types to the relational model.

Software models originate from programming languages and other software development technologies. Generally, software models consist of elements to structure operations and elements to structure data. The elements to structure data resemble a data model. Two popular software models are object orientation and role modeling [27]. Both build on the notion of an object and encompass a dedicated association element to represent relationships. Accordingly, they provide no direct compatibility with the relational model, a fact also well known as

Table 1. Flexible Data Models vs. Requirements

Category	Data Model	Multi-faceted entities	Variable attribute sets	Independent attributes	Independent technical types	Relational representation and processing
Relational	Pure relational					✓
	Extended NULL semantic [28]		(✓)[1]			✓
	Interpreted column [2]		(✓)[2]			✓
	Interpreted record [7]		✓			✓
	Polymorphic table [5]	(✓)[3]	(✓)[2]			✓
	FRDM	✓	✓	✓	✓	✓
Programming	Object orientation	(✓)[4]			(–)[6]	
	Role modeling [27]	(✓)[5]			(–)[6]	
Document	XML, well-formed [31]		✓	✓	(✓)[7]	
	XML, valid [32]	(✓)[3]	✓			
	JSON [12]	(✓)[8]	(✓)[8]	(✓)[8]	✓	
	OEM [22]	(✓)[8]	(✓)[8]	(✓)[8]	✓	
Tabular	Bigtable [10]	✓	✓		✓	
Graph	Property graph [25]		✓	✓	(–)[6]	
	Neo4J [21]	✓	✓	✓	✓	
	Freebase [8]	✓	(✓)[1]			
	RDF [30]	(✓)[8]	(✓)[8]	(✓)[8]	✓	
	RDF w/ RDF Schema [29]	✓	✓	✓	✓	
Theory	Intensional classification [23]	✓	✓	✓	(–)[6]	

[1] only generalization [2] only specialization [3] extensions [4] inheritance
[5] roles [6] not specified [7] no technical types [8] no entity types

object-relational impedance mismatch. With inheritance and the notion of roles, these two software models offer limited support for multifaceted entities. Particularly the role concept allows the dynamic leaving and joining of entity types. Nevertheless, which combination of entity types an entity can join has to be modeled upfront.

Document models [31,32,12,22] have been developed for representing documents, e.g., web pages. Typically, document models represent data as a hierarchy of entities, where entities nest other entities. Nesting is the only or the primary means of entity referencing. The identity of an entity solely or primarily depends on the position of an entity within the hierarchy. In consequence, document models offer direct relational compatibility. Document models offer more flexibility than most relational systems or software models. However, most of their flexibility originates from completely omitting entity types. Where document models have schema information, such as DTD or XML Schema, they are similarly strict.

A tabular data model also organizes data in tables like the relational data model but in a significantly different way. The data model of Google's Bigtable system [10] defined the category of tabular data models. Because of its success, it has also remained the only model of its kind that draws considerable attention. Bigtable organizes data in large, distributed, sparse tables. The columns of such a table are grouped in column families. Rows can stretch across multiple column families and are free to instantiate any column in a column family, so that the Bigtable data model supports multifaceted entities as well as variable attribute sets. The Bigtable model also supports independent technical types. However, the row identity is restricted to a user-given row key and the processing is limited to put and get operations on row level. Hence, the Bigtable model cannot be considered completely relational compatible.

Graph data models [25,21,8,30,29] build on the mathematical definition of a graph. They represent data as vertices and edges, where vertices represent entities and edges represent relationships, i.e., references to other entities. In practice, graph models differ in how data is represented in a graph. Beside vertices and edges, graphs can have labels and attribute–value pairs attached to the vertices and even to the edges. [25] distinguishes nine types of graphs. Most prominent are the property graph and the RDF graph. All graph models emphasize the representation of data rather than modeling of schema. Graph models have a descriptive nature and allow in most cases the direct representation of data from dynamic and open discourses. In all graph models, however, entities have an object identity and edges are an explicit representation of references. Consequently, graph models are not directly compatible to relational data.

Finally in the theory of data modeling, intensional classification was proposed to allow for more schema flexibility [23]. Here, entity domains are defined intensionally, i.e., by a set of attributes. All entities that instantiate the set of attributes defining an entity domain belong to that domain. Accordingly, the intensional classification builds on independent attributes and allows multi-faceted entities as well as variable attribute sets. Technical types are not considered in the approach. While intensional classification is appling, it is less flexible than extensional classification used in FRDM, since entities are required to instantiate an defined attribute set to belong to a domain. They cannot be explicitly added to a domain regardless their intension. In that sense, intensional classification is a useful complement to extensional classification.

As a summary, Table 1 shows which flexibilities sample data models in the discussed categories do allow. We can see that none of these models fulfills all flexibility requirements. Graph models, particularly as in Neo4j, are free of implicit constraints regarding entity domains, attributes and technical types, while the relational approaches are the only ones to offer value-based identity and value-based references. FRDM integrates the level of flexibility graph models provide with value-based identity and value-based references, as indicated in Table 1, in a super-relational fashion.

7 Conclusion

As an evolutionary approach to meet the need for more flexible database systems and to build on the still existing dominance of relational database systems we proposed the flexible super-relational data model FRDM. FRDM is entity-oriented instead of schema-oriented. It is designed around self-descriptive entities, where schema comes with the data and does not have to be defined up front. Additionally, FRDM allows multi-faceted entities where entities can belong to multiple entity domains. Attributes can exist independently from entity domains in FRDM. Similarly, FRDM allows technically typing values independently from their attributes. FRDM can express irregular data as well as regular relational data. We demonstrated both by examples. For data retrieval, FRDM builds on the powerful, well-known, and proven set of relational operations. Compared to the relational data model, FRDM is free of implicit constraints. Nevertheless, where these constraints are needed and welcome, the presented constraint framework FRDM-C allows formulating explicit restrictions to the flexibility of FRDM. A lot of technological expertise, knowledge, and experience have accumulated in and around relational database management systems over the last three decades. We are convinced FRDM contributes to the use of that also in the more flexibility-demanding areas of data management.

References

1. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: VLDB 2007 (2007)
2. Acharya, S., Carlin, P., Galindo-Legaria, C.A., Kozielczyk, K., Terlecki, P., Zabback, P.: Relational support for flexible schema scenarios. The Proceedings of the VLDB Endowment 1(2) (2008)
3. Agrawal, R., Somani, A., Xu, Y.: Storage and Querying of E-Commerce Data. In: VLDB 2001 (2001)
4. Aulbach, S., Grust, T., Jacobs, D., Kemper, A., Rittinger, J.: Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques. In: SIGMOD 2008 (2008)
5. Aulbach, S., Seibold, M., Jacobs, D., Kemper, A.: Extensibility and Data Sharing in evolving multi-tenant databases. In: ICDE 2011 (2011)
6. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development (2001), `http://agilemanifesto.org/`
7. Beckmann, J.L., Halverson, A., Krishnamurthy, R., Naughton, J.F.: Extending RDBMSs To Support Sparse Datasets Using An Interpreted Attribute Storage Format. In: ICDE 2006 (2006)
8. Bollacker, K.D., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: A Collaboratively Created Graph Database For Structuring Human Knowledge. In: SIGMOD 2008 (2008)
9. Brodie, M.: OTM"10 Keynote. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 2–3. Springer, Heidelberg (2010)

10. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: Bigtable: A Distributed Storage System for Structured Data. In: OSDI 2006 (2006)
11. Chu, E., Beckmann, J.L., Naughton, J.F.: The Case for a Wide-Table Approach to Manage Sparse Relational Data Sets. In: SIGMOD 2007 (2007)
12. Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (July 2006), http://tools.ietf.org/html/rfc4627
13. Cunningham, C., Graefe, G., Galindo-Legaria, C.A.: PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS. In: VLDB 2004 (2004)
14. Franklin, M.J., Halevy, A.Y., Maier, D.: From Databases to Dataspaces: A New Abstraction for Information Management. SIGMOD Record 34(4) (2005)
15. Friedman, C., Hripcsak, G., Johnson, S.B., Cimino, J.J., Clayton, P.D.: A Generalized Relational Schema for an Integrated Clinical Patient Database. In: SCAMC 1990 (1990)
16. Gleick, J.: Faster: The Acceleration of Just About Everything. Pantheon Books, New York (1999)
17. Jacobs, D.: Enterprise Software as Service. ACM Queue 3(6) (2005)
18. Kiely, G., Fitzgerald, B.: An Investigation of the Use of Methods within Information Systems Development Projects. The Electronic Journal of Information Systems in Developing Countries 22(4) (2005)
19. Kurzweil, R.: The Law of Accelerating Returns (March 2001), http://www.kurzweilai.net/the-law-of-accelerating-returns
20. Nagarajan, S.: Guest Editor's Introduction: Data Storage Evolution. Computing Now, Special Issue (March 2011)
21. Neo Technology: Neo4j (2013), http://neo4j.org/
22. Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object Exchange Across Heterogeneous Information Sources. In: ICDE 1995 (1995)
23. Parsons, J., Wand, Y.: Emancipating Instances from the Tyranny of Classes in Information Modeling. ACM Transactions on Database Systems 25(2) (2000)
24. PostgreSQL Global Development Group: PostgreSQL 9.2.4 Documentation, chap. 56.6: Database Page Layout (2013)
25. Rodriguez, M.A., Neubauer, P.: Constructions from Dots and Lines. Bulletin of the American Society for Information Science and Technology 36(6) (August 2010)
26. Sarma, A.D., Dong, X., Halevy, A.Y.: Bootstrapping Pay-As-You-Go Data Integration Systems. In: SIGMOD 2008 (2008)
27. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. Data & Knowledge Engineering 35(1) (2000)
28. Vassiliou, Y.: Null Values in Data Base Management: A Denotational Semantics Approach. In: SIGMOD 1979 (1979)
29. W3C: RDF Vocabulary Description Language 1.0: RDF Schema (February 2004), http://www.w3.org/TR/2004/REC-rdf-schema-20040210/
30. W3C: Resource Description Framework (RDF): Concepts and Abstract Syntax (February 2004), http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
31. W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition). (November 2008), http://www.w3.org/TR/2008/REC-xml-20081126/
32. W3C: XML Schema Definition Language (XSD) 1.1 Part 1: Structures. (July 2011), http://www.w3.org/TR/2011/CR-xmlschema11-1-20110721/