A Global Model-Driven Denormalization Approach for Schema Migration

Jihane Mali[1,2], Shohreh Ahvar[2], Faten Atigui[3], Ahmed Azough[1], and Nicolas Travers[1,3]

[1] Léonard de Vinci Pôle Universitaire, Research Center, Paris La Défense, France
firstname.lastname@devinci.fr
[2] ISEP firstname.lastname@isep.fr
[3] CEDRIC, Conservatoire National des Arts et Métiers (CNAM), Paris, France
firstname.lastname@cnam.fr

Abstract. With data's evolution in terms of volume, variety, and velocity, Information Systems (IS) administrators have to steadily adapt their data model and choose the best solution(s) to store and manage data in accordance with users' requirements. In this context, many existing solutions transform a source data model into a target one, but none of them leads the administrator to choose the most suitable model by offering a limited solution space automatically calculated and adapted to his needs. We propose `ModelDrivenGuide`, an automatic global approach for leading the model transformation process. It starts by transforming the conceptual model into a logical model, and it defines refinement rules that help to generate all possible data models. Our approach then relies on a heuristic to reduce the search space by avoiding cycles and redundancies. We also propose a formalisation of the denormalization process and we discuss the completeness and the complexity of our approach.

Keywords: NoSQL, MDA, Denormalization, Model Refinement, Heuristic,

1 Introduction

For several decades, the storage and the exploitation of data have mainly relied on relational databases. But the explosion of data, especially characterised by the `3V` (Volume, Variety & Velocity), has opened up major research issues related to modeling, manipulating, and storing massive amounts of data [20].

The resulting so-called NoSQL systems correspond to *four families* of data structures: key-value oriented (`KVO`), column oriented (`CO`), document oriented (`DO`), and graph oriented (`GO`). These systems have raised new problems of transforming traditional databases into non-relational databases.

Choosing the most suitable data model family (*i.e.,* relational and NoSQL) and its implementation is a crucial task. Thus, this highly distributed context emphasizes the problem of integrating an Information System (IS) and a proper data model is essential for such systems.

Existing methods produce a single targeted data model while missing important considerations that could favor other data models with better trade-off between requirements Hence we need to produce a set of data models in order to allow the IS designer to choose the best compromise. However, generating data models that are useful for the use case among all possible ones is a hard issue. Thus, producing a reduced search space of relevant data models to help this choice becomes a real challenge.

Based on Model-Driven Architecture (MDA[4]), we propose the `ModelDriven-Guide` approach which starts from the conceptual model, then goes from one logical model to another thanks to refinement rules. It provides a functional decision-making process by integrating the use case composed of a set of queries, to guide the implementation choice of the SQL and NoSQL solution(s). In previous work [16], we presented an overview of our model transformation approach by focusing on the *5 families* common meta-model to allow the specification of any data model. This work especially focuses on the following contributions:

- A complete formalization of the search space generation of data models with transformation rules and completeness, allowing to get *all* the possible models going from one logical model to another;
- A full formalism of the data model refinement problem with complexity bounds for both merge and split rules, first time in the literature;
- A recursive heuristic to reduce the search space of generated data models, based on redundancies and use cases;
- A thorough study of data model transformation by simulating schemas and use cases, and the TPC-C benchmark.

This paper is organized as follows: the related work is presented in Section 2, our approach is formalized in Section 3 and the models' generation heuristic is developed in Section 4. Then, we validate our approach in Section 5. Finally, our conclusion opens the research perspectives in Section 6.

2 Related Works

Model Transformation and NoSQL DB. Some of the existing approaches suggest a set of mapping rules that transform a relational model into a NoSQL one like for `CO` families [21] or `DO` families with *MongoDB* [18].

Moreover, other studies have focused on the transformation of a conceptual model into a NoSQL data model. Chebotko et al. [6] present a query-driven approach for modeling *Cassandra* starting from an ER model. They define dedicated transformation rules between these logical and physical models. De Freitas et al. [11] propose a conceptual mapping approach which transforms ER models into one of the 4 NoSQL families with an abstract formalization of the mapping rules. Also, schema evolution over time has been studied, and its relation to NoSQL data migration [19].

[4] https://www.omg.org/mda/index.htm

Table 1. NoSQL transformation approaches

	Relational	DO	CO	KVO	GO	Approach	
[6],[14],[21]			✓			Dedicated	Manual
[18]		✓				Dedicated	Manual
[7]					✓	Dedicated	Manual
[1],[11]		✓	✓	✓	✓	Dedicated	SemiAutomated
[8]		✓	✓			Semi-Global	SemiAutomated
`ModelDrivenGuide`	✓	✓	✓	✓	✓	Global	Automated

The following studies adopt a MDA to transform a UML class diagram into NoSQL DB. Li Y. et al. [14] propose to transform a class diagram into a `CO` DB with *HBase*, and Daniel G. et al. [7] into a `GO` DB.

Abdelhedi et al. [1] propose to transform a class diagram into a common logical model that describes the four families of NoSQL DB. Then, it's transformed into physical models related to the four families. Alfonso et al. propose the system *Mortadelo* [8] that defines a model-driven design process that generates implementations for `CO` and `DO` starting from the conceptual model.

Table 1 summarizes the main transformation strategies with dedicated conceptual/relational to NoSQL solutions or semi-global (rule-driven choice) compared to our global approach non-limited by a mapping between a source and a target concept, but covers all the possibilities using models transformation.

Reducing the Search Space. Adopting a global approach inevitably leads to the generation of a multitude of possible choices. It is therefore useful to perform a search space reduction that leads to a limited solution space from which the most optimal solution can be chosen. Statistical sampling [9] can be used to reduce the solution space, but results in approximations that may miss the optimal solution regarding the use case. Machine Learning (ML) approaches can also be useful in order to perform this task [10]. However, it requires a large amount of training data models classified according to user use case. Furthermore, ML-based approaches can be seen as a black box for IS designers.

Instead, rule-based approaches can be a good alternative to lead the reduction task in an explainable way. In our case, we have used Heuristics that have been used in the context of NoSQL DB generation. Atzeni et al. provide in [3] a set of heuristics for transforming object-oriented models into a common model before transforming it into a platform-dependent model. However those heuristics are rather driven by a query use case expressed through an affinity matrix, which makes it an approximate and probabilistic approach.

3 ModelDrivenGuide Approach

Our approach `ModelDrivenGuide` aims to produce a set of data models giving choices to the IS designer instead of focusing on dedicated solutions which

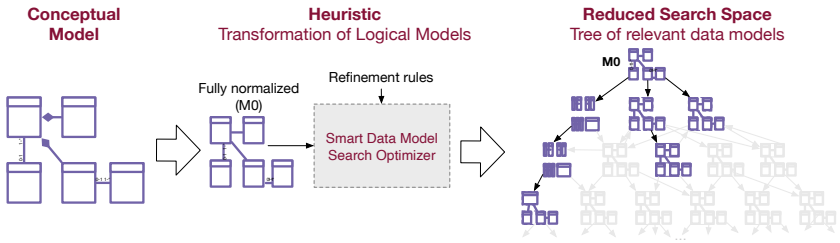


Fig. 1. ModelDrivenGuide: A Heuristic to Generate Data Models

prevents any trade-off. It is based on logical models and refinement rules as illustrated in Figure 1.

`ModelDrivenGuide` begins with the fully normalized data model ($\mathcal{M}_0$) (*i.e.,* the relational model).

Then, the *Smart Data Model Search Optimizer* which is at the heart of this paper, relies on data models' denormalization with two refinement rules (Merge & Split) applied recursively on logical models (Definition 1) to obtain all possible solutions.

Especially, it is based on a heuristic that optimizes the denormalization process. This process applies refinement rules recursively without conditions to generate all models and is called *Naïve* in the following. The heuristic reduces the search space to relevant data models since some of the generated models can be redundant and even useless for the associated use case. So far, no other approach combines merge and split rules to generate *all data models* suitable for a given context of use.

Definition 1. *Let $\mathcal{M}$ be a data model conform to the 5Families meta-model [16] where $\mathcal{M} = (\mathcal{C}, \mathcal{R}, \mathcal{L}, \mathcal{K}, \mathcal{E}, \kappa)$ is composed of* `concepts` $c(r_1, ..., r_m) \in \mathcal{C} | r_1, ..., r_m \in \mathcal{R}$, `rows` $r(k_1, ..., k_n) \in \mathcal{R} | k_1, ..., k_n \in \mathcal{K}$, `key values` $\mathcal{K}$ *(Atomic Values or Complex Values),* `references` $ref_{i \to j} \in \mathcal{L} | k_i, k_j \in \mathcal{K}$, `edges` $\mathcal{E} : \mathcal{C} \times \mathcal{C}$ *and* `constraints` $cons(k) \in \kappa | k \in \mathcal{K}$.

To simplify the formalism of manipulations on data models $\mathcal{M}$, we will focus mostly on rows' transformations and references. Thus, a data model $\mathcal{M}$ can be denoted as a graph $\mathcal{M}(\mathcal{R}, \mathcal{L})$ where nodes are rows such that: $\forall i \in \{1, .., n\}, r_i \in \mathcal{R}$, and edges are references such that: $\forall j, k \in \{1, .., n\}, ref_{j \to k} \in \mathcal{L} | r_j, r_k \in \mathcal{R}, r_k = ref_{j \to k}(r_j)$. Our goal in this paper is to start from an initial $\mathcal{M}_0(\mathcal{R}, \mathcal{L})$ and to generate a set of optimal models $\mathcal{M}^{opt}$ compatible with the use case.

Driving Example: Figure 2 gives a logical representation of a data model $\mathcal{M}$ with 4 rows `Warehouse`, `Customer`, `StoredIn` and `Order` (resp. W, C, S, and O) of 4 corresponding concepts. Four references link keys from source row (*i.e.,* primary key - bold red) to the target row (*i.e.,* foreign key).

3.1 Logical Model Refinement Rules

The Logical Model refinement is based on three transformation rules:

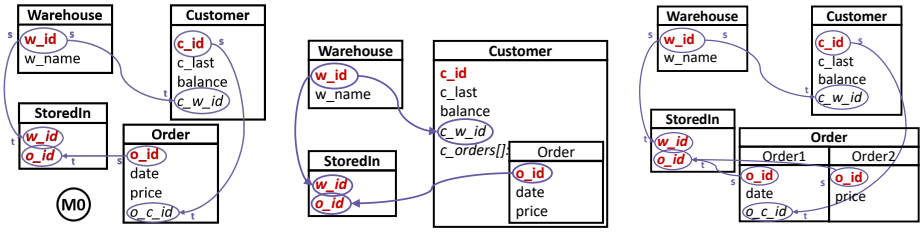


Fig. 2. Driving Example **Fig. 3.** Merge: $m(C, O, ref_{C \to O})$ **Fig. 4.** Split: $s(O, price)$

1) **merge** *rows* to produce *complex values* for nesting (`CO`, `DO`) or to merge keys for values concatenation (`KVO`),

2) **split** a *row* to produce two rows in a same *concept* (`CO`),

and 3) transform *references* into the equivalent *edges* (`GO`).

At logical level, models' refinement uses endogenous transformations where both source and target models are conformed to the same meta-model, here the `5Families` meta-model. We detail especially *merge* and *split* refinement rules which are the basis of the data model generation framework.

Merge is a refinement rule applied between two rows referring to each other (*i.e.,* associated classes), where the result is a single row with a complex value.

Definition 2. *Let m:* $\mathcal{M} \to \mathcal{M}$ *be an endogenous function that merges rows from a 5Families model* $\mathcal{M}$*. The merge function* $m(r_i, r_j, ref_{i \to j})$ *is applied on two rows* $r_i, r_j \in \mathcal{R}$ *(source and target rows) linked by a reference* $ref_{i \to j} \in \mathcal{L}$ *with corresponding keys* $k_i, k_j \in \mathcal{K}$*. The merge function produces a new model* $\mathcal{M}'$ *where* r_j *is a complex value of* r_i*, and removes* $ref_{i \to j}$*, denoted by:*

$$m(r_i, r_j, ref_{i \to j}) = r_i\{r_j\}$$

The merge function is a bijective function $m^{-1}(r_i\{r_j\}) = (r_i, r_j, ref_{i \to j})$ *which rebuilds* $ref_{i \to j}$ *and non-nested rows.*

This rule corresponds to the merge of two nodes in graph $\mathcal{M}$ where node j (row) is embedded in node i. Notice that $m(r_i, r_j, ref_{i \to j}) \neq m(r_j, r_i, ref_{i \to j})$ since the nested row is done in the opposite way. From reference $ref_{i \to j}$, when the target row r_j is nested into the source r_i, r_j is embedded into a list of values.

Example: applying a merge on the driving example, like $m(Customer, Order, ref_{C \to O})$ results in the data model shown in Figure 3. We notice that after applying this merge rule, the reference $ref_{C \to O}$, linking `Customer` row and `Order` row was transformed into `c_orders` containing a list of nested `Order[]`.

Cycles issue. By recursive application of merges, a problem occurs in case of data models containing cycles through references like in our driving example. Figure 5 illustrates recursive application of four merges to obtain successively

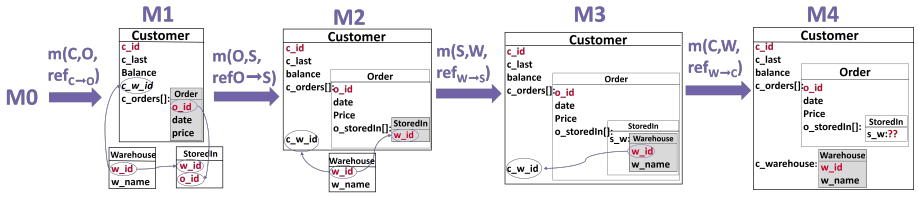


Fig. 5. Particular Case: Cycle

M_0 (of Figure 2) to M_4 data models. Since the merge function removes references (Definition 2), the M_4 data model doesn't contain references anymore. However, we must notice that the M_0 contains a cycle with a reference $ref_{W \to S}$ between *Warehouse* and *StoredIn*. During the 4^{th} merge, this reference has been transformed into a *ComplexValue* into *StoredIn[]*. But the remaining reference $ref_{C \to W}$ between *Customer* and *Warehouse* is used to merge them (data model M_4) which gets rid of the information $ref_{W \to S}$. This removal makes it impossible to get backward in the data model's generation process. This case occurs during *merges* on nested rows. We apply m^{-1} on the nested row to rebuild the reference, and apply m on the other reference.

Split is a refinement rule applied on a row containing several keys, associating several rows for the same concept (*i.e.,* `CO`'s column family).

Definition 3. *Let s:* $\mathcal{M} \to \mathcal{M}$ *be an endogenous function that splits rows from a 5Families model* $\mathcal{M}$*. The split function* $s(r_i, k)$ *is applied on a row* $r_i \in \mathcal{R}$ *and a key value* $k \in keys(r_i)$ *not linked to a constraint. The split function produces a new model* $\mathcal{M}'$ *with two rows* $\overline{r_{i_k}}$ *and* r_{i_k} *with the same constraint key* $pk \in keys(r_i)$ *(i.e., primary key) where* $\overline{r_{i_k}} = (pk, k_i) | \forall k_i \in keys(r_i) \wedge k_i \neq k$*, and* $r_{i_k} = (pk, k)$*. The split function s is denoted by:*

$$s(r_i, k) = (\overline{r_{i_k}}, r_{i_k})$$

s is bijective $s^{-1}(\overline{r_{i_k}}, r_{i_k}) = (r_i)$ *and merges common constraints and keys.*

Example: let's take the example of the model depicted in Figure 2, if we apply a split on `Warehouse` using key *price*, we obtain the model in Figure 4. Notice that rows *Order1* and *Order2* are linked to the same concept *Order* (mixed up when a concept contains only one row).

Although splits are usually used for the `CO` family, they can also be useful to optimize documents size `DO` and thus computation time when combining only useful keys from two rows (split+merge) from useless keys (stored apart).

3.2 Inverse functions

We should notice that *merge* and *split* refinement rules can be considered inverse functions for specific cases. In fact, if a split is applied on a key k_j as a *complex*

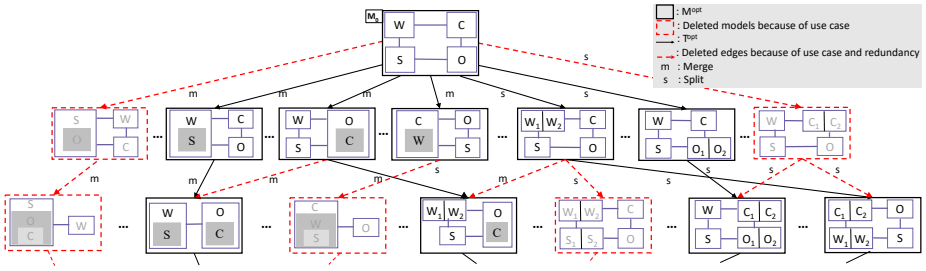


Fig. 6. Two Levels of the Graph of Possible Solutions

value r_j, it rebuilds a row with a single key containing the reference information $ref_{i \to j}$ and keys from row r_j. Then:

$$s(r_i\{r_j\}, k_j) = (r_i, r_j) = m^{-1}(r_i\{r_j\}) \tag{1}$$

At the opposite, a *merge* between two rows $\overline{r_{i_k}}$ and r_{i_k} with a same constraint pk_i can be considered to be the inverse of a *split*. Then:

$$m(\overline{r_{i_k}}, r_{i_k}, pk_i) = (r_i) = s^{-1}(\overline{r_{i_k}}, r_{i_k}) \tag{2}$$

3.3 Completeness of our Approach

Our approach based on the previously defined refinement rules generates all possible data models by combining recursively several merges and splits. Before defining a generation strategy of data models, it is necessary to prove the completeness of our approach. It states that splits and merges are sufficient to generate all possibilities without modifying keys (*e.g.,* materialization, replication). Moreover, there is always a path between two data models.

Theorem 1. *Completeness. Merge and Split refinement rules are sufficient to generate all denormalized data models* $\mathcal{M}^*$ *without key modification, conformed to the 5Families meta-model, beginning with the fully normalized data model.*

Proof. We want to prove that there is at least one *path* (sequence of rules) between two denormalized models, which means there is a path between a denormalized model M_1 and the fully normalized data model M_0.

Let $step(M_1, M_2, r)$ be a rule applied between two models M_1 and M_2, where $r \in \{merge, split\}$ and $M_2 = r(M_1)$. And let $path(M_1, M_n)$ be a sequence of steps between two models M_1 and M_n, such that:
$path(M_1, M_n) = [step(M_1, M_2, r_1), ..., step(M_{n-1}, M_n, r_{n-1})]$

Suppose that M_0 and M_n share the same set of rows and keys and there isn't any *path* between a denormalized model M_n and the normalized one M_0.

$\implies path(M_0, M_n) = \emptyset$

$\implies \exists M, path(M, M_n) \neq \emptyset, path(M_0, M) = \emptyset$

This implies that:

1) this model M is not conformed to the *5Families* meta-model (Absurd since rules are endogenous), or

2) the applied rules can't be inverse (Absurd, from definition 2 and 3).

Of course, it exists more than one path from a data model to another since there are several possible compositions of functions that produce a given data model. Thus, we obtain a *lattice* that represents the generated solutions where nodes are data models, and edges are applied refinement rules. The lattice reduction problem is discussed in the following section.

4 Smart Data Model Search Optimizer

Models refinement leads to the production of plenty of data models in a form of a graph. Its aim is to generate the most suitable data models and thus allow to choose the targeted solution. But the number of possibilities explodes as splits on $\mathcal{M}$ can be applied on each key and merges can be done in both ways.

Starting from $\mathcal{M}$ as $\mathcal{M}(\mathcal{R}, \mathcal{L})$, applying the i^{th} transformation rule on the rows in R, leads to $\mathcal{M}_i$. If the rule is split, it divides one row to several columns. If the rule is merge, it merges 2 rows which are a set of columns themselves.

Considering $\mathcal{M}$ as a set of rows, all generated $\mathcal{M}_i$ by the naïve process, form a graph $G(\mathcal{M}^*, \mathcal{T}^*)$ where $\mathcal{M}^*$ is the set of generated data models and $\mathcal{T}^*$ the transformations that link data models. This problem is a reduced problem of generating all partitioning of a set.

The total number of partitions of the set $\mathcal{M}$ can be calculated as the *Bell* number. *Bell* number grows exponentially by increasing the size of the set, and set partitioning has been proved to be NP-hard [13]. Moreover, splits on data models are combined with merges making the growth far more computational.

Therefore, we need a heuristic to explore only an optimal subset of state space. Considering the initial $\mathcal{M}$ as a state space graph of $\mathcal{M}_0(\mathcal{R}, \mathcal{L})$, and transformed graph after applying the i^{th} transformation rule as $\mathcal{M}_i$, $T(\mathcal{M}^{opt}, \mathcal{T}^{opt})$ is the reduced search space tree growing through transformations done by our heuristic. To obtain T we apply 1) a Depth First Search (*DFS*) strategy to traverse G at each step i and 2) a reduction of the graph based on the use case.

4.1 Complexity of the *Naïve* Denormalization Process

The generation of data models $\mathcal{M}^{opt}$ by applying refinement rules recursively on various combinations of model transformations $\mathcal{M}_i$, produces a hierarchy of target data models T. At the top level of this hierarchy is the fully normalized relational model $\mathcal{M}_0$ and at the lowest level are completely denormalized models.

This hierarchy's size $|T|$ depends on the size of the input data model with the number of rows $|\mathcal{R}|$ and references $|\mathcal{L}|$. Figure 6 shows the first two levels of the data models' generation graph (G) for our driving example, where W, C, O and S stand for *Warehouse*, *Customer*, *Order* and *StoredIn* respectively. The dashed edges show the edges that were deleted by the heuristic to avoid

redundancies, cycles and transformations that do not take into consideration the use case. Consequently, dashed squares represent the lonely nodes (models) which cannot be produced since no more transformation can reach them.

The set of `merges` depends on the number of linked rows in $\mathcal{M}$ and the number of generated models is given by a *Fubini number* or *Ordered Bell number* [4]: $Fn_{|\mathcal{L}|} = \sum_{k=0}^{|\mathcal{L}|} k! \times \binom{|\mathcal{L}|}{k}$
This number is an upper bound since the number of generated data models can be less depending on references' organization between rows. Our driving example can produce up to 75 models considering just merges $Fn_4 = \sum_{k=0}^{4} k! \times \binom{4}{k} = 75$.

Moreover, the number of `splits` on a row depends on the number of keys in a row $|keys(r)|$ (without primary keys) which is given by the *Bell's number* [5]: $B_{|keys(r)|} = \sum_{k=0}^{|keys(r)|} \binom{|keys(r)|}{k} \times B_{|keys(r)|-1}$

Thus, in our example rows *Customer* and *Order* contain 3 non-primary keys $B_3 = 5$, for *Warehouse* $B_1 = 1$ and *StoredIn* $B_0 = 1$. And those solutions combine together producing $B_{|keys(C)|} \times B_{|keys(O)|} \times B_{|keys(W)|} \times B_{|keys(S)|} = 25$. To finish with, split and merged rows can be combined to produce all possible solutions in T. Each split row can be merged and adds new nested solutions.

Thus, we can formalize the complete denormalization problem with the combination of rows' merges and splits all together as a product. The total number of possible data models gives the following complexity measure:

$$|\mathcal{M}^*| = Fn_{|\mathcal{L}|} \times \prod_{k=1}^{|\mathcal{R}|} B_{|keys(r_k)|}$$

Our example will generate $75 \times 25 = 1,875$ data models.

As said previously, since there are several ways to produce a single $\mathcal{M}_i \in \mathcal{M}^{opt}$, it requires to prune the lattice G of solutions $\mathcal{M}^*$.

4.2 Application of the Refinement Rules

Due to the aforementioned problems, we need to provide a heuristic in order to reduce the search space and avoid cycles. The target is a subset of $\mathcal{M}^*$ called $\mathcal{M}^{opt}$ after pruning edges and nodes from T. The heuristic avoids to produce different paths between two data models. In fact, applying splits and merges in different orders will produce the same effects on the resulting data models. Moreover, every merge can be reversed by a split and produce a cycle in the generation of data models (not shown in Figure 6). Thus, four main rules have to be adopted and are presented in the following.

A row with complex values Applying a split on a row with complex values (merged rows) gives the following result:

$$s(m(r_i, r_j, ref_{i \to j}), k_j) = s(r_i\{r_j\}, k_j) = (r_i, r_j, ref_{i \to j})$$

where r_i and r_j are two rows that are first merged and represented as $r_i\{r_j\}$, $ref_{i \to j}$ is the reference linking the two rows and used for the merge and k_j is the key on which we apply the split. We notice that this transformation takes the models' generation back in the hierarchy with the initial model (r_i, r_j) of two rows, hence generates a cycle. Thus, the first rule avoids applying a split on a row with complex values.

A row linked to two or more rows When a row r_i is linked to two rows r_j and r_k with two references $ref_{i\to j}$ and $ref_{i\to k}$, a redundancy will occur, since:

$$\begin{aligned} m(m(r_j, r_i, ref_{j\to i}), r_k, ref_{j\to k}) &= m(m(r_j, r_k, ref_{j\to k}), r_i, ref_{j\to i}) \\ &= r_j\{r_i, r_k\} \end{aligned}$$

where $r_i, r_j, r_k \in \mathcal{R}$ and r_j is linked both to r_i ($ref_{j\to i}$) and r_k ($ref_{j\to k}$).

Since both transformations are equivalent (the result is the same data model), the heuristic discards one of the two merges.

The use case By considering the use case, we can reduce the number of joins only to those used in queries that combine rows through references. Also splits should not separate keys if queries from the use case combine them. It avoids solutions which require instance reconstruction with costly joins.

Moreover, splits applied thanks to queries can generate redundancy. If queries $q(K)$ and $q(\overline{K})$ applied on r use complement keys, they will produce the same data model (*i.e.,* $s(r, K) = s(r, \overline{K})$). To avoid this issue, we compute complementary queries to prune redundant splits.

We must notice that Theorem 1 on the completeness of our approach remains true according to the two first simplification rules. In fact, only duplicate edges in $\mathcal{M}^*$ are removed producing a DFS keeping all the nodes. Thus, it always exists a *single* path, to obtain a data model. However, this last simplification rules removes nodes since we remove edges from $\mathcal{M}^*$ if the refinement rule do not rely on a query from the use case. Commonly said in the literature [6, 11, 1], we assume the fact that those transformations will lead to a data model *useless* for the use case as well as the whole corresponding subtree. Consequently, the completeness remains valid for data models relying on the use case.

4.3 Heuristic of Model Generation

To formalize the generation of data models with our heuristic, Algorithm 1 gives the recursive function `5FMHeuristic` which generates a list of data models $\mathcal{M}^{opt}$. It takes a relational model as an entry and produces all possible models (of all families). Starting from $i = 0$, at stage i, it processes a current data model $\mathcal{M}_i$ on which splits and merges will be applied by taking into account a set of queries $\mathcal{Q}$ from the use case. For simplicity, we consider that a query is a set of involved keys within the data model (for filters, joins, projects, aggregates, etc.).

Each time the `5FMHeuristic` function is called with a new data model from the *5Families* meta model, it is added to the global output list $\mathcal{M}^{opt}$ (line 2). Then, all keys from the data model ($K \in \mathcal{K}$) are tested except those which are involved in a Primary Key constraint (line 3). To test the keys, the first check (lines 5-7) verifies if the key is used in a query from $\mathcal{Q} \cup \mathcal{Q}'$ (remaining and processed queries). It also checks if the key is the last remaining key from the current row R except the Primary Key (needed for instance reconstruction). In that case, the key is put in a new row by applying the rule `Split` (Definition 3) and then

Algorithm 1 5FMHeuristic

global: A list of data models $\mathcal{M}^{opt}$ from *5FamiliesModel*
input: a data model $\mathcal{M}_i$ from *5FamiliesModel*, a list of input queries $\mathcal{Q}$ (a query is a set of keys from $\mathcal{M}_i$), a list of used queries $\mathcal{Q}'$
init: $\mathcal{M}_i = \mathcal{M}_0$, the relational data model, $\mathcal{M}^{opt} = \emptyset$, $\mathcal{Q}' = \emptyset$

1: **procedure** 5FMHeuristic($\mathcal{M}_i, \mathcal{Q}, \mathcal{Q}'$)
2: $\mathcal{M}^{opt} := \mathcal{M}^{opt} \cup \mathcal{M}_i$
3: **for all** key $K \in \mathcal{K} \wedge$!*Constraint*(K, CType=PK) **do**
4: $r := Row(K)$
5: **if** $K \notin \mathcal{Q} \cup \mathcal{Q}'$ **then**
6: **if** $\exists k \in r | k \neq K \wedge$!*Constraint*($k$, Ctype=PK) **then**
7: 5FMHeuristic(*Split*($\mathcal{M}$, K), $\mathcal{Q}$, $\mathcal{Q}'$)
8: **else**
9: **for all** $q \in \mathcal{Q}$ **do**
10: **if** $\forall k \in q | Row(k) = r, \exists k \in R | k \notin q \wedge$!$Constraint(k, Ctype = PK) \wedge \forall q' \in \mathcal{Q} | \overline{q} \neq q'$ **then**
11: 5FMHeuristic(*Split*($\mathcal{M}_i$, q), $\mathcal{Q} - q, \mathcal{Q}' \cup q$)
12: **else if** $\exists k \in q | Row(k) \neq r \wedge Concept(k) = Concept(K)$ **then**
13: continue
14: **else if** $\exists k_1, k_2 \in q | ref_{k_1 \to k_2} \vee ref_{k_2 \to k_1} \in \mathcal{L}$ **then**
15: 5FMHeuristic(*Merge*($\mathcal{M}_i$, q, k_1, k_2), $\mathcal{Q} - q, \mathcal{Q}' \cup q$)
16: 5FMHeuristic(*Merge*($\mathcal{M}_i$, q, k_2, k_1), $\mathcal{Q} - q, \mathcal{Q}' \cup q$)
17: 5FMHeuristic(*ReferenceToEdge*($\mathcal{M}_i$, q, k_1, k_2), $\mathcal{Q} - q, \mathcal{Q}' \cup q$)

recursively call `5FMHeuristic` on this new data model ($\mathcal{M}_{i+1}$). If the key K belongs to at least one remaining query q from $\mathcal{Q}$ (line 9), we check the rules from the heuristic. First (lines 10-11), if all keys from q belong to the same row r and the latter can be split (except the Primary Key and non-complementary keys), we split it by taking all the keys from q in a new row, instead of the single key K. The generated data model is then recursively denormalized (line 11) without the query q (moved to $\mathcal{Q}'$).

The second rule concerns the keys from a query q (line 12) that belong to the same concept but in different rows, we then avoid applying a merge between those rows and continue to the next query (line 13).

The third rule (line 14) applies the `Merge` rule when a query q uses two keys linked by a *Reference*. In this case, we apply merges in both directions (lines 15-16). Finally, to produce `GO` data models, references are transformed into *Edge*s (line 17) with the `ReferenceToEdge` rule (not presented in this article).

Thanks to this heuristic, the application of refinement rules will generate a tree of possibilities whose first nodes try splits and finish with merges (DFS). In addition, the use case reduces drastically the number of nodes for queries using few keys, and drives towards merges for multi-concept queries.

From our example of 4 concepts, the number of solutions, initially equal to 1,875, is reduced to only 125. This number will be reduced even more by taking into account multi-concept queries impacting the *Fubini*'s number, and far more with queries involving several keys in a same concept (*Bell*'s number).

4.4 Complexity of the Heuristic

Our problem is abstracted by a graph of solution $G(\mathcal{M}^*, \mathcal{T}^*)$, on which our heuristic has two reduction effects. First, it reduces the graph's size by pruning useless data models since it focuses only on the use case $\mathcal{Q}$, obtaining $\mathcal{M}^{opt}$. Second, it applies a DFS approach to avoid redundancies and removes all links in order to produce a tree of solutions called $\mathcal{T}^{opt}$.

Since the heuristic takes into account the distinct references used by queries $|refs(\mathcal{Q})|$, the *Fubini* number is impacted. *Bell* number with splits is also modified based on the size of distinct sets of keys from queries for a given row k: $|KeySet(\mathcal{Q}_k)|$. Thus, the upper bound of the estimated number of solutions is:

$$|\mathcal{M}^{opt}| = Fn_{|refs(\mathcal{Q})|} \times \prod_{k=1}^{|\mathcal{R}|} |KeySet(\mathcal{Q}_k)|$$

And the number of transformations to apply is: $|\mathcal{T}^{opt}| = |\mathcal{M}^{opt}| - 1$

5 Experiments

5.1 Implementation

Our `ModelDrivenGuide` approach is implemented in *Java*. It starts with a UML model automatically transformed into a relational model ($\mathcal{M}_0$).

Then, refinement rules are applied recursively on data models using the *Naïve* (generate all solutions) and the `5FMHeuristic` strategies.

In order to get the number of generated models and to see the gain obtained by our heuristic, a *signature* file has been implemented; a sorted list of concepts, rows, keys, nesting and references. It ensures their uniqueness and detects identical data models produced by two distinct paths. It will help to cut cycles in the *Naïve* strategy and to count the number of duplicated data models.

5.2 TPC-C

To illustrate our approach, we used the TPC-C[5] benchmark giving a full use case mixing at the same time transactions, joins and aggregations. For this test, we focus on the three concepts (classes in the UML model): *Customer*, *Order* and *OrderLine*.

Table 2 shows the number of generated data models, splits, merges and unique data models (thanks to signatures). The *Naïve* approach on TPC-C benchmark, produces 2,313 data models by applying 1,646 splits and 666 merges (1,445 distinct data models and thus 868 redundancies).

While the `5FMHeuristic` generates only 27 data models with only 2 splits (few splitting queries) and 24 merges (impacted by splits). The number of distinct data models is equal to the total number of generated models (no redundancies) which is the goal of our heuristic (DFS approach).

[5] `http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf`

Table 2. Generated Models

Strategy	Nb models	Nb Splits	Nb Merges	Nb Unique Models
Naïve	2,313	1,646	666	1,445
5FMHeuristic	27	2	24	27

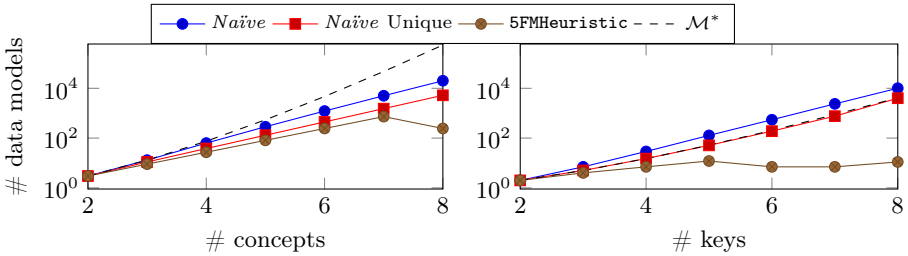


Fig. 7. Nb of data models with merges wrt. nb concepts (5 join queries)

Fig. 8. Nb of data models with splits wrt. nb keys (1 concept & 5 queries)

To compare our approach with competitors [1, 8], it is important to remind that they both generate a **unique** data model that corresponds to their targeting approach optimizing one criteria. In our case, we produce this data model among 27 others offering a large choice of possible models which contains this solution. After decreasing the number of generated models noticeably, it becomes an easier task for the IS administrator to choose the most convenient one by taking into consideration either the ease of implementation, NoSQL compatibility, security policy, storage or environmental impact, etc. We must notice that among those solutions it is possible to select a data model which can be implemented in several families of databases at the same time called Polystores [2].

5.3 Data Models Generation

In order to study the behavior of the *Naïve* and the `5FMHeuristic`, a generator of data models has been implemented in order to simulate the impact of denormalization strategies. It produces various data models by varying the number of concepts and keys, and also the topology of linked concepts (lines *vs* stars). We will produce the average number of generated data models per strategy. Moreover, random queries are generated to see the effect of use cases by varying the number of joins and involved keys.

Our code is available on Github[6] with a configuration file to parameter the generator.

To study the impact of the heuristic on the applied rules (Merge, Split and both of them), thanks to the aforementioned signatures, we will count the average number of generated data models from the initial data model (relational) and the generated models before and after applying the heuristic.

[6] `https://github.com/leonard-de-vinci/5FM_generator/`

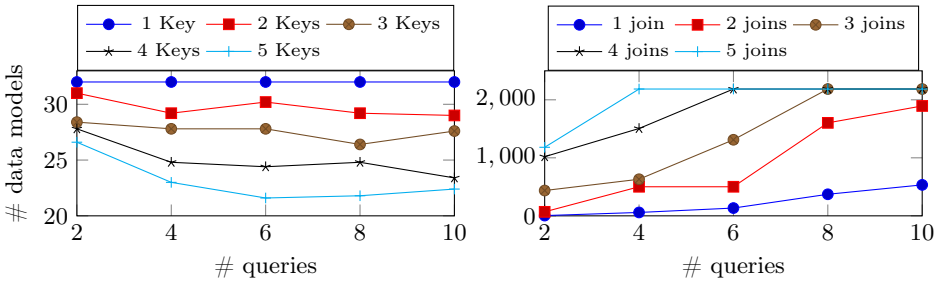


Fig. 9. `5FMHeuristic` - Nb of data models (splits) wrt. Nb of filter queries (8 concepts of 5 keys each)

Fig. 10. `5FMHeuristic` - Nb of data models (merges) wrt. nb joins per query (8 concepts of 5 keys each)

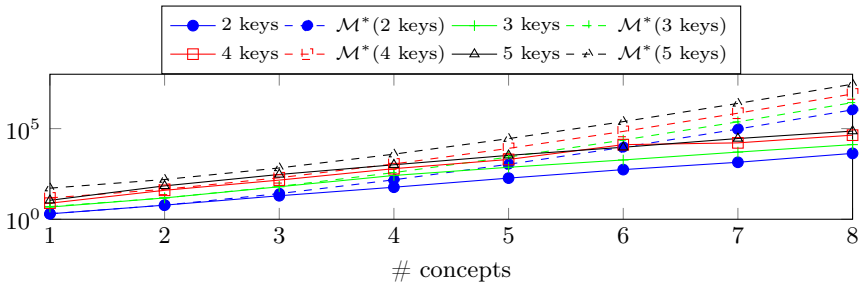


Fig. 11. `5FMHeuristic` - Nb of data models wrt. nb concepts varying keys (5 queries)

Figure 7 shows the evolution of the number of generated data models with the *Naïve* approach and our `5FMHeuristic`. We focus only on merges by limiting keys to primary and foreign keys. The *Naïve* approach follows an exponential growth (dashed curve) but slightly lower than the *Fubini* number since simulated data models do not contain cycles (with references) nor star-shaped data models which lead to more combinations and reach this upper-bound. The *Naïve* unique approach avoids cycles and duplicates which reduces significantly the number of solutions. Our heuristic targets only required merges which can witness a decrease when it reaches bigger data models (here 7 concepts). In fact, since the number of queries/joins are a constant, their impact on the number of merges becomes noticeable. Thus the number of possibilities decreases rapidly.

Figure 8 focuses on splits varying number of keys on a single concept. The *Naïve* approach shows the distribution of the *Bell* number with duplicates (dashed curve & *Naïve* unique). The number of data models produced by `5FMHeuristic` is reduced sharply for splits. Here also the number of possible splits is bounded by the number of involved queries reaching a threshold based on the size of the KeySet (simplified Bell number to the KeySet).

The impact of the use case on the `5FMHeuristic` is shown in Figure 9 by varying the number of keys per query. It's applied on 8 concepts of 5 keys,

each with different numbers of queries (no joins - no merges). We notice that only single-key queries produce the maximum number of splits since it offers all possibilities, while this number decreases with bigger queries. When the KeySet size is high, it implies that splits cannot be applied since all keys must be put together in a same row. Thus, we can conclude that small queries produce more data models to find more adapted solutions.

Figure 10 focuses on join queries by varying the number of joins per query. Contrary to the splits, the bigger the joins are the more data models are generated. It is due to the fact that queries allow merging more concepts with each other allowing to produce data models with a single concept.

Figure 11 plots the global impact of the `5FMHeuristic` with both merges and splits by varying the number of concepts and the number of keys for each. With 5 different queries (filters and joins), it is interesting to see that the distribution remains similar for all cases. The number of keys per concept has an impact on possible splits and consequently on merges. For 8 concepts, it varies from 726 for 2 keys/concept to 16,767 data models for 4 keys/concept. We plotted the $|\mathcal{M}^*|$ complexity (dashed curves) for each key size which shows the exponential growth of the produced data models, at most 28,383,420 solutions for 8 concepts of 5 keys. We can see that the `5FMHeuristic` drastically reduces the search space.

To conclude, queries with more joins and fewer keys have more impact on the number of generated data models than other ones. Moreover, our heuristic implies that splits impact possible merges due to denormalizations' combination.

6 Conclusion and Future Work

In this paper, we have proposed an IS modeling and implementation MDA-based approach that generates *all* data models by recursive denormalization to find suitable solutions. We propose the first full formalization of the problem and issues with refinement rules and data models manipulation. Then a heuristic of generations based on DFS and the use case, allows avoiding the explosion of solutions in terms of the amount of data models and paths to obtain them.

Our goal is to offer the IS designer a limited solution space of which he can choose the best model that will be adapted to his constraints (*e.g.,* efficiency, green computing, integration, polystores, security). To guide the choice of implementation, we can find performance comparisons of NoSQL DB for dedicated needs [12, 17, 20], studies of applicability to specific software quality choices [15].

In a future work, we would like to enrich this choice process by offering a cost model that can assist the designer in his choice by associating each model of the solution space with an implementation cost and then sort them accordingly.

We also work on the definition of the eligibility of a data model in the target DB. Indeed, the generated models must be compatible and it is necessary to define mapping rules; a key & a value for `KVO`, Concepts & Links for `GO`, etc.

References

1. Abdelhedi, F., Brahim, A.A., Atigui, F., Zurfluh, G.: MDA-based Approach for NoSQL Databases Modelling. In: DAWAK'17. pp. 88–102. Springer (2017)
2. Alotaibi, R., Cautis, B., Deutsch, A., Latrache, M., Manolescu, I., Yang, Y.: Estocada: Towards scalable polystore systems. VLDB Endow. **13**(12), 2949–2952 (2020)
3. Atzeni, P., Bugiotti, F., Cabibbo, L., Torlone, R.: Data modeling in the NoSQL world. Int. J. CSI **67**, 103–149 (2020)
4. Belbachir, H., Djemmada, Y., Németh, L.: The deranged Bell numbers. arXiv preprint arXiv:2102.00139 (2021)
5. Bell, E.T.: Exponential polynomials. Annals of Mathematics **35**(2), 258–277 (1934)
6. Chebotko, A., Kashlev, A., Lu, S.: A Big Data Modeling Methodology for Apache Cassandra. In: ICBD'15. pp. 238–245. IEEE (2015)
7. Daniel, G., Sunyé, G., Cabot, J.: UMLtoGraphDB: mapping conceptual schemas to graph databases. In: Conceptual Modeling (ER'16). pp. 430–444 (2016)
8. de la Vega, A., García-Saiz, D., Blanco, C., Zorrilla, M., Sánchez, P.: Mortadelo: Automatic generation of nosql stores from platform-independent data models. Future Generation Computer Systems **105**, 455–474 (2020)
9. Doshi, P., Gmytrasiewicz, P.J.: Monte carlo sampling methods for approximating interactive pomdps. J. Artif. Int. Res. **34**(1), 297–337 (Mar 2009)
10. Eggermont, J., Kok, J.N., Kosters, W.A.: Genetic Programming for Data Classification: Partitioning the Search Space. In: SAC'04. pp. 1001–1005. NY (2004)
11. de Freitas, M.C., Souza, D.Y., Salgado, A.: Conceptual mappings to convert relational into nosql databases. In: ICEIS'16. pp. 174–181 (2016)
12. Jatana, N., Puri, S., Ahuja, M., Kathuria, I., Gosain, D.: A survey and comparison of relational and non-relational database. IJERT **1**(6), 1–5 (2012)
13. Lewis, M., Kochenberger, G., Alidaee, B.: A new modeling and solution approach for the set-partitioning problem. COR **35**(3), 807–813 (2008)
14. Li, Y., Gu, P., Zhang, C.: Transforming UML class diagrams into HBase based on meta-model. In: ISEEE'14. vol. 2, pp. 720–724 (2014)
15. Lourenço, J.R., Cabral, B., Carreiro, P., Vieira, M., Bernardino, J.: Choosing the right NoSQL database for the job: a quality attribute evaluation. Journal of Big Data **2**(1), 18 (2015)
16. Mali, J., Atigui, F., Azough, A., Travers, N.: ModelDrivenGuide: An Approach for Implementing NoSQL Schemas. In: DEXA'20. pp. 141–151. Springer (2020)
17. Raut, A.: NoSQL database and its comparison with RDBMS. International Journal of Computational Intelligence Research **13**(7), 1645–1651 (2017)
18. Rocha, L., Vale, F., Cirilo, E., Barbosa, D., Mourão, F.: A framework for migrating relational datasets to NoSQL. Procedia Computer Science **51**, 2593–2602 (2015)
19. Störl, U., Klettke, M., Scherzinger, S.: NoSQL Schema Evolution and Data Migration: State-of-the-Art and Opportunities. In: EDBT'20. pp. 655–658 (2020)
20. Tauro, C.J., Aravindh, S., Shreeharsha, A.: Comparative study of the new generation, agile, scalable, high performance NoSQL databases. International Journal of Computer Applications **48**(20), 1–4 (2012)
21. Vajk, T., Fehér, P., Fekete, K., Charaf, H.: Denormalizing data into schema-free databases. In: CogInfoCom'13. pp. 747–752. IEEE (2013)