Information Fusion 134 (2026) 104337

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

# Normalization-driven optimization of knowledge graph creation for semantically grounded information fusion

Enrique Iglesias [a,b,*], David Chaves-Fraga [c], Sören Auer [a,b,d], Maria-Esther Vidal [a,b,d]

[a] L3S Research Center, Hannover, Germany
[b] Leibniz University of Hannover, Hannover, Germany
[c] CiTIUS, Universidade de Santiago de Compostela, Santiago de Compostela, Spain
[d] TIB Leibniz Information Centre for Science and Technology, Hannover, Germany

ARTICLE INFO

*Keywords:*
Data integration system
RDF mapping languages
Functional dependencies
Normalization theory

ABSTRACT

The growing demand for reliable decision-making systems has made Knowledge Graphs (KGs) a cornerstone in the field of information fusion. As data structures capable of unifying heterogeneous information sources through semantically enriched representations, KGs facilitate interoperability and contextual integration across domains. However, constructing such KGs remains a challenging task due to the high resource demands in terms of memory, execution time, and complexity of data mappings. This work addresses the challenge of creating scalable KGs by introducing optimization techniques that minimize memory usage and execution time through data partitioning and dependency-aware integration planning. The proposed approach leverages functional dependencies (FDs) to streamline input mappings and data sources, ensuring that only essential attributes are included in each transformation step while independently processing non-dependent attributes. To evaluate the effectiveness of our approach, we conducted an extensive empirical study using state-of-the-art KG creation engines and benchmarking frameworks. A total of 552 experimental configurations were executed. Our results demonstrate memory consumption reductions of up to a factor of 1221.71 and execution time improvements by a factor of 1112.97. These findings underscore the importance of data source characteristics, such as functional dependencies, in optimizing KG creation performance and highlight the potential of FD-based planning in enhancing existing KG engines.

## 1. Introduction

Knowledge Graphs (KGs) are expressive data structures that model entities and their relationships as directed, edge-labeled graphs [1]. By providing a shared semantic model over heterogeneous data sources, KGs enable advanced data integration, contextualization, and reasoning [2]. Their adoption spans a wide range of application domains, including digital humanities [3], medicine [4,5], transportation systems [6], and open government data [7]. In these settings, KGs serve as semantic backbones for information fusion, supporting the integration of data expressed in diverse formats into coherent and interpretable knowledge representations. By enabling dataset linking, inconsistency resolution, and inference, KGs have become essential components of decision support systems that require semantic interoperability, transparency, and scalability.

Despite their potential to support information fusion, constructing high-quality KGs from heterogeneous data sources remains a complex and resource-intensive task [8]. KG creation is commonly formalized as a *data integration system* (DIS) [9], defined as a tuple $\text{DIS} = (O, S, M)$, where $O$ is an ontology capturing domain semantics, $S$ is a set of structured or semi-structured data sources, and $M$ is a set of mapping assertions specifying how source data is transformed into RDF triples. In practice, these mappings are expressed using declarative W3C standards such as R2RML [10] and RML [11], which promote transparency, traceability, and maintainability. While this declarative abstraction enables engine-independent KG creation, it also conceals execution-level behavior, making performance issues difficult to anticipate and control. As a consequence, practical KG creation pipelines frequently suffer from performance bottlenecks, even when mature engines [12–15] and benchmarking frameworks [16] are employed. These bottlenecks are often not caused by the ontology or the intended KG schema, but by structural and semantic anomalies in the mappings and in the way source data is accessed. Common issues include redundant attribute replication, unnecessary joins, repeated scans of large data sources, and violations of functional dependencies that lead to duplicate triple generation and costly post-processing steps. Such inefficiencies are particularly problematic in

* Corresponding author.
*E-mail addresses:* iglesias@l3s.de (E. Iglesias), david.chaves@usc.es (D. Chaves-Fraga), auer@tib.eu (S. Auer), maria.vidal@tib.eu (M.-E. Vidal).

https://doi.org/10.1016/j.inffus.2026.104337
Received 23 April 2025; Received in revised form 13 February 2026; Accepted 30 March 2026
Available online 1 April 2026

data-intensive and dynamic domains, such as biomedicine [17], where large volumes of heterogeneous data must be integrated repeatedly under strict memory and time constraints.

A significant source of these inefficiencies lies in how source data is fragmented and accessed during KG creation. Fragmentation—well studied in database systems [18]—refers to partitioning a relation $R$ into non-overlapping fragments, either by rows (horizontal fragmentation), by attributes (vertical fragmentation), or by a combination of both (hybrid fragmentation). In the context of DISs, fragmentation directly influences the structure of mappings and the execution plans generated by KG creation engines. While horizontal fragmentation may affect semantic completeness by separating related entities, appropriate fragmentation strategies can reduce redundancy, limit unnecessary joins, and improve execution efficiency. Consequently, fragmentation plays a central role in balancing semantic correctness and computational efficiency during KG materialization.

Several RML-compliant KG creation engines implicitly exploit fragmentation or mapping decomposition as part of their internal optimization strategies. For example, Morph-KGC [19,20] applies hybrid strategies that combine vertical projection with horizontal deduplication to reduce redundant data processing. Similarly, SDM-RDFizer [14,21] optimizes execution by detecting predicate overlaps and reusing intermediate results. While effective in practice, these optimizations are tightly coupled to engine-specific execution models and rely on heuristics, which limit their portability and make their behavior difficult to analyze or generalize across engines and datasets.

To overcome these limitations, engine-independent optimization techniques have been proposed. Mapping partitioning approaches [22] decompose complex mappings into smaller units to reduce join complexity and enable parallel execution, while systems such as MapSDI [23] apply vertical fragmentation and deduplication to minimize intermediate data size. However, these approaches are largely heuristic-driven and lack a formal foundation that explains *when* and *why* such transformations are correct and beneficial. In particular, they do not systematically exploit semantic properties of the data, such as functional dependencies, nor do they provide guarantees about redundancy elimination or semantic preservation. Motivated by these limitations, this work introduces a normalization theory for DISs grounded in classical relational dependency theory [24]. By treating KG creation as a transformation problem over DISs, rather than as an engine-specific optimization task, our approach provides a principled and general framework for restructuring mappings and source access patterns. The resulting normalized DISs are semantically equivalent to the original systems but are designed to support more efficient execution during KG materialization.

**Problem Statement.** The problem addressed in this work concerns optimizing the *KG creation process* from heterogeneous data sources without altering the resulting KG. In current practice, declarative mappings are evaluated by RML-compliant engines over relational or semi-structured data sources. While these mappings are semantically correct, their structure often yields inefficient execution plans with redundant data access, excessive joins, repeated computations, and costly duplicate elimination during materialization. In this context, a *more efficient representation* of a DIS is defined as one that produces a KG that is *semantically equivalent* to the original (i.e., isomorphic up to blank node renaming), but with lower computational cost—measured in terms of reduced memory consumption, fewer joins, and shorter execution time when evaluated by KG creation engines. These inefficiencies stem from structural and semantic anomalies in mappings and source access patterns, rather than from the ontology or the target KG itself. As KGs scale in size and complexity, particularly in data-intensive information fusion scenarios, systematically addressing these inefficiencies becomes essential.

**Proposed Solution.** A normalization-based approach is introduced to optimize KG creation from DISs. Inspired by classical relational database normalization theory, a set of mapping-level normal forms—referred to as *Triples Map Normal Forms*—is defined to characterize structural and semantic anomalies that lead to inefficient KG creation. Based on these normal forms, a transformation algorithm rewrites a given DIS into a semantically equivalent but normalized system. The normalization process exploits functional dependencies to eliminate redundancy, reduce unnecessary joins, and reorganize mapping structures, thereby improving execution efficiency and memory usage while guaranteeing that the resulting KG is semantically equivalent to the original (i.e., isomorphic up to blank node renaming).

NormaKG is introduced as an engine-independent normalization theory for DISs. NormaKG enforces a set of mapping-level normal forms—TM-1NF, TM-2NF, TM-3NF, and TM-JNF—through transformation rules that ensure attribute atomicity, respect functional dependencies, and optimize join execution. The resulting normalized DIS produces a KG that is semantically equivalent to the original one, while significantly reducing computational and memory overhead during materialization.

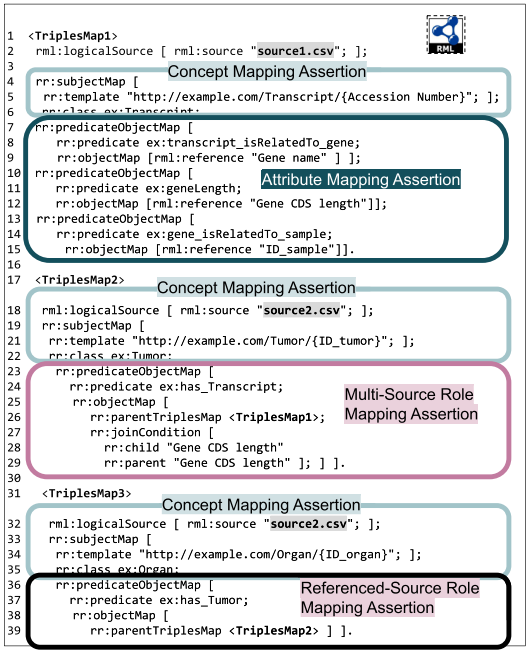
**Contributions.** This paper extends prior work presented in a short article [25], which introduced an initial normalization theory for DISs. The theory is expanded by introducing additional normal forms for triples maps, together with a normalization algorithm that rewrites DISs to satisfy these forms. The main contributions are as follows:

- A formal normalization theory for DISs is developed through the definition of four mapping-level normal forms—TM-1NF, TM-2NF, TM-3NF, and TM-JNF—each addressing specific classes of structural and semantic anomalies that negatively affect KG creation efficiency.
- A set of normalization rules and a transformation algorithm are formalized to restructure DISs while preserving the semantics of the generated KG; the correctness of both the rules and the algorithm is formally proven.
- The theory is realized in NormaKG, a modular and engine-agnostic framework that produces normalized DISs executable by any RML-compliant engine, eliminating the need for runtime duplicate removal.
- NormaKG is positioned as a data fusion method applicable to the data-level preprocessing and feature-level fusion stages.
- An empirical evaluation is conducted on state-of-the-art benchmarks—GTFS-Madrid-Bench [16], SDM-Genomic-Datasets [26], and KROWN [27]—using existing RML engines. The results demonstrate consistent and significant reductions in execution time and memory consumption.

A total of 552 test configurations were executed. Overall, the experimental results confirm that NormaKG substantially improves KG creation performance while preserving the semantics of the output KG. By automating DIS normalization in a principled and engine-independent manner, this work enables scalable and reliable KG generation for real-world information fusion scenarios.

NormaKG is positioned as a data fusion technique operating at the data-level preprocessing and feature-level fusion stages. Rather than introducing a new fusion architecture, NormaKG optimizes intermediate representations by normalizing heterogeneous data sources and eliminating structural redundancies before downstream fusion. By consolidating semantically equivalent attributes into a single KG, it reduces inconsistencies in the feature space and improves the efficiency of subsequent fusion steps, including state estimation and decision-making. This positioning enables NormaKG to complement centralized, decentralized, and distributed fusion architectures.

The remainder of the paper is structured as follows. Section 2 defines DISs and introduces the anomalies that motivate our normalization theory. Section 3 reviews related work on mapping optimization and data fragmentation. Section 4 presents the normalization theory, including mapping-level normal forms, transformation rules, and the normalization algorithm. Section 5 describes the NormaKG framework, detailing its architecture, properties, and implementation. Section 6 reports the results of our empirical evaluation across benchmarks and engines. Finally, Section 7 concludes the paper and outlines future research directions.




(a) Mapping Assertions in RML.

| Name | Description |
|---|---|
| Concept MAs | Conjunctive rules over the predicate symbols of data sources in $S$ to create the instances of a class $C$ in the ontology $O$. This mapping assertion corresponds to a `rr:subjectMap` where attributes in the logical source $S_i$, define the subject of the class $C$ |
| Single-Source Role MAs | Defines a role $P(.,.)$ regarding a source's attributes. In terms of RML, this *MA* is represented by `rr:template`, which is a simple projection of the attributes in the logical source $S_i$ |
| Referenced-Source Role MAs | In terms of RML, this assertion corresponds to a `rr:RefObjectMap` where a *MA* is referred to by using the predicate `rr:parentTriplesMap`. Both MAs are defined over the same logical source $S_i$. |
| Multi-Source Role MAs | This assertion corresponds to a `rr:RefObjectMap` including `rr:joinCondition`, where *MA* stands for the triples map referred by the predicate `rr:parentTriplesMap` |
| Attribute MAs | In terms of RML, this assertion corresponds defines the object value as a `rml:reference`. It projects the value as a string. |

(b) Types of Mapping Assertions.

**Fig. 1. Mapping Assertions: Overview and examples of mapping assertions in RML.** (a) illustrates concrete RML triples maps, highlighting concept, attribute, and role mapping assertions over logical sources, while (b) classifies these assertions according to their structural role in a data integration system (DIS). This decomposition makes explicit how different assertion types induce joins, dependencies, and redundancy, which form the basis for the normalization theory introduced in this paper.

## 2. Data integration systems and motivation for a normalization theory

This section defines Data Integration Systems (DIS), mapping assertions, anomalies in DIS, basic concepts, and a motivating scenario to illustrate the impact of these anomalies on the efficiency of the KG creation processes.

### 2.1. Data integration systems and mapping assertions

*Knowledge graph (KG)* [28] is an RDF graph and can be viewed as a directed edge-labeled graph $KG = (V, E, L)$:

- $I$ is a set of IRIs, $B$ a set of blank nodes, and $Li$ a set of literals, where $I$, $B$, and $Li$ are pairwise disjoint;
- $V \subseteq I \cup B \cup Li$ is a set of nodes (RDF terms);
- $L \subseteq I$ is a set of edge labels (RDF predicates are IRIs);
- $E \subseteq (I \cup B) \times I \times (I \cup B \cup Li)$ is a set of directed labeled edges (RDF triples).

A blank node $b \in B$ denotes an unnamed resource (an existential variable) whose identifier is scoped to a specific RDF graph. Hence, RDF graphs are considered equivalent up to renaming of blank nodes. In this paper, the terms *knowledge graph (KG)* and *RDF knowledge graph* are used interchangeably. Unless stated otherwise, every KG considered in this work is an RDF graph under the RDF data model.
The creation of a KG $G$ can be defined as a data integration system $DIS_G = \langle O, S, M \rangle$, where $O$ is a set of classes and properties of a unified ontology, $S$ is set of data sources, and $M$ corresponds to mapping rules or assertions that follow the concepts established in $O$ as conjunctive queries over sources in $S$ [9]. The execution of $M$ over the data sources in $S$ generates the instances in $G$. The rules in $M$ (Iglesias et al. [22]) can be presented as Horn clauses (i.e., $body(\overline{X}) : -head(\overline{Y})$) that follow the Global As View (GAV) approach. In other words, $body(\overline{X})$ is an association of predicates over data sources in $S$ and their attributes, and $head(\overline{X})$ is a predicate that represents classes and properties in $O$. $\overline{Y}$ is a subset of variables from $\overline{X}$, and the rule head may include functions. Mapping assertions (*MAs*) represent the [R2]RML triples maps; they are defined as follows (Table in Fig. 1(b) summarizes these definitions). R2RML [10] is a mapping language for defining custom RDF triples maps (TM) for the transformation of relational databases into RDF KG, and its extension, RDF Mapping language (RML) [11], introduces the possibility to transform data in various formats, e.g., CSV, JSON, and XML files.
**Concept Mapping Assertions** are conjunctive rules that define the instances of a class $C$ in the ontology $O$ using a logical $S_i$ source in $S$. In RML, a concept mapping assertion corresponds to a `rr:subjectMap` where attributes in the logical source $S_i$ define the subject of the class $C$; a predefined function allows for string concatenation; it is expressed with the RDF predicate `rr:template`. Fig. 1a presents the triples maps corresponding to concept mapping assertions; they define instances of `ex:Transcript` (line 6), `ex:Tumor` (line 22), and `ex:Organ` (line 35). The Horn clause that represents a concept *MA* is as follows:

$$S_i(\overline{X}) \rightarrow C(f(y))$$

Where $f(.)$ is a function for generating unique identifiers (e.g., concatenation), $\overline{X}$ is an array of attributes from source $S_i$, and $y$ is the set of inputs for $f(.)$. An example of the application of this function is as follows:

$$S_1(\text{Accession\_Number}, \text{Gene\_Name}, \text{Length}) \rightarrow$$
$$\text{ex:Transcript}(\text{concat}(\text{'ex:'}, \text{Accession\_Number}))$$

In the example, there is a source $S_1$ with the properties Accession_Name, Gene_Name, and Lenght. When applying the concept MA to $S_1$, it results in an entity of class ex:Transcript with the corresponding value from the attribute Accession_Number as an identifier.
**Referenced-Source Role Mapping Assertions** specify the object value

of a role $P(.,.)$ over a source $S_i$ that also defines the subject of a referred concept mapping assertion *MA*. In RML, this assertion corresponds to a `rr:RefObjectMap` where the mapping assertion *MA* is referred to using the predicate `rr:parentTriplesMap`. Both mapping assertions are defined over the same logical source $S_i$. In Fig. 1a, `TriplesMap3` defines the property `ex:hasTumor` as the subject of the *TM* `TriplesMap2`. Both `TriplesMap2` and `TriplesMap3` are defined over the same logical source. The Horn clause that defines a referenced-source role *MA* is as follows:

$$S_i(\overline{X}) \rightarrow P(f_1(y_1), f_2(y_2))$$

The subject and object entities IRIs are generated using the functions $f_1$ and $f_2$ over the attributes $y_1$ and $y_2$, respectively. The object value is defined by the role $P(.,.)$, which is based on the referenced concept *MA*. An example of applying the Horn clause is as follows:

$$S_2(\text{ID\_organ}, \text{ID\_tumor}) \rightarrow$$
$$\text{has\_Tumor}(f_1(\text{ID\_organ}), f_2(\text{ID\_tumor}))$$

In the example, the source $S_2$ has the attributes ID_organ and ID_tumor. When applying the referenced concept *MA* to $S_2$, it produces a triple with the predicate has_Tumor. The subject of this triple is derived by applying the corresponding concept MA to the value of ID_organ, while the object is obtained by applying the concept *MA* from the referred *TM* to the value of ID_tumor.

**Multi-Source Role Mapping Assertions** define a role $P(.,.)$ where the subject and object are drawn from different data sources $S_j$ and $S_i$, respectively. In this case, $S_j$ contributes the subject of a concept mapping assertion (*MA*), and $S_i$ defines the object. Because the data is distributed across sources, a join condition is required to relate values from both sources. In RML, this type of mapping corresponds to a `rr:RefObjectMap` that includes a `rr:joinCondition`, where the mapping assertion *MA* is referred to via `rr:parentTriplesMap`. As shown in Fig. 1a, `TriplesMap2` defines the role `ex:has_Transcript`, where the subject is based on `source2.csv` and the object is derived from `TriplesMap1`, which maps `source1.csv`. The join condition aligns the `Gene CDS length` attribute across both sources. The Horn clause that defines a multi-source role *MA* is:

$$S_i(\overline{X_{i,1}}), S_j^M(\overline{X_{j,2}}), \theta(\overline{X_{i,1}}, \overline{X_{j,2}}) \rightarrow P(f_1(y_1), f_2(y_2))$$
$$S_j^M(\overline{X_{j,2}}) \rightarrow C_k(f_1(y_1))$$

An example of applying this Horn clause using the sources in Fig. 1a is:

$$S_1(\text{Accession\_Number}, \text{Gene CDS length}),$$
$$S_2(\text{ID\_tumor}, \text{Gene CDS length}),$$
$$S_1.\text{Gene CDS length} = S_2.\text{Gene CDS length} \rightarrow$$
$$\text{has\_Transcript}(f_1(\text{ID\_tumor}), f_2(\text{Accession\_Number}))$$

This mapping states that if a tumor from $S_2$ shares a `Gene CDS length` with a transcript in $S_1$, then an `ex:has_Transcript` link is created between them, with IRIs generated using functions $f_1$ and $f_2$.

**Attribute Mapping Assertions** define a data property $A$ whose subject is an entity instance and whose object is a literal value from the data source. In RML, this corresponds to a `rr:predicateObjectMap` that includes a `rr:objectMap` with a `rml:reference` pointing to the source attribute. Fig. 1a illustrates several examples in `TriplesMap1`, where the properties `ex:geneLength`, `ex:transcript_isRelatedTo_gene`, and `ex:transcript_isRelatedTo_sample` are defined based on attribute values from `source1.csv`. The Horn clause is:

$$S_i(\overline{X}) \rightarrow A(f(y_1), y_2)$$

An example of the application of this Horn clause using `TriplesMap1` is:

$$S_1(\text{Accession\_Number}, \text{Gene\_CDS\_length}) \rightarrow$$
$$\text{geneLength}(concat('ex:', \text{Accession\_Number}), \quad \text{Gene\_CDS\_length})$$

This rule creates a data property assertion where the subject is generated from `Accession_Number` using a template (via `concat`) and the object is the literal value in the `Gene CDS length` field. The classification of mapping assertions into concept, role, and attribute types allows for a unified representation of DISs independent of a specific mapping language (e.g., R2RML or RML). By expressing these assertions as Horn clauses, we lay a logical foundation for identifying and reasoning about structural and semantic anomalies within DISs. This formalization enables the systematic application of our normalization theory, which aims to optimize KG creation by transforming DISs without compromising semantic equivalence.

### 2.2. Data integration systems and anomalies

Multiple factors influence the performance of KG creation, particularly in terms of memory usage and execution time. Chaves-Fraga et al. [8] identify key factors such as the volume and heterogeneity of data, the number of duplicates, and the complexity of the mappings. Various strategies have been proposed to mitigate these challenges, including mapping partitioning, execution planning, and data fragmentation. While effective in practice, these strategies often address symptoms rather than the underlying causes, which are rooted in anomalies present within the declarative components of a data integration system (DIS). We define a set of anomalies that commonly affect the efficiency, scalability, and semantic quality of DIS-based KG creation:

- **Redundancy Anomalies:** Arise when duplicate values in the data sources lead to the generation of repeated triples. Since KG creation engines must detect and eliminate duplicates to ensure correctness, this process can introduce significant computational and memory overhead [8].
- **Join Anomalies:** Occur when multi-source joins, e.g., many-to-many relationships, are executed without considering key constraints or join selectivity. This results in unnecessarily large intermediate results, increasing both execution time and memory usage. Such anomalies are particularly problematic in Multi-Source Role Mapping Assertions (MRMAs).
- **Dependency Violations:** Emerge when functional dependencies (FDs) and key constraints in the source data are not respected. As a result, derived attributes are redundantly included in mappings, leading to semantic inconsistency, redundant triple generation, and inflated processing costs.
- **Structural Mapping Anomalies:** Happen when mapping assertions include irrelevant or redundant source attributes that are not semantically required. This increases the size and complexity of RDF triples, reduces mapping clarity, and introduces unnecessary overhead in transformation pipelines.
- **Execution Plan Anomalies:** Result from suboptimal evaluation orders of mapping assertions. When overlapping triples are generated repeatedly due to unoptimized execution, the system incurs additional costs in duplicate elimination and materialization [14].

These anomalies contribute directly to the inefficiencies observed in KG creation, leading to excessive memory consumption, longer runtimes, and poor scalability. Addressing them requires a principled approach to restructuring DISs and enforcing semantic correctness, which motivates the normalization theory proposed in this work.

### 2.3. Preliminary concepts

**Functional Dependencies.** It is presented in [24,29] that functional dependencies, FD, are integrity constraints developed in the real world, which possible legal relations for a relation scheme will be indicated. In other words, not every possible relation can be an instance of a particular scheme unless the FD holds for that relation. An FD $f$ is a function $f_1 : X \longrightarrow Y$, where each $X$ and $Y$ are subsets of a set of attributes in

a relation. Given a specific value of $X$, $Y$ has only one value associated with $X$. Therefore, there can only be one in $Y$ that corresponds to one value in $X$ so that the conditions of an FD can be fulfilled. Another FD comes up in the form of $f_2 : Y \longrightarrow X$ when there exists a one-to-one relationship between $X$ and $Y$. If there is no FD between $X$ and $Y$, it can be inferred that at least one value of $X$ is associated with one or more values in $Y$ and vice versa. A relation $r$ satisfies an FD $f : X \longrightarrow Y$, when each tuple $\mu$ and $\upsilon$ in $r$ fulfilled one of the following conditions:

- $\mu[X] = \upsilon[X]$ then $\mu[Y] = \upsilon[Y]$
- $\mu[X] \neq \upsilon[X]$

Further, if $\mu[X] = \upsilon[X]$ then $\mu[Y] \neq \upsilon[Y]$ then relation $r$ violates FD. To investigate whether an FD holds within an existing scheme, it is necessary to verify each possible relation to determine if the dependency was violated.
Consider $S_1$ with attributes `Accession_Number`, `Gene_Name`, and `Gene_CDS_length`. Suppose the functional dependency:

`Accession_Number` → `Gene_Name`, `Gene_CDS_length`

This means that for every unique `Accession_Number`, there should be exactly one `Gene_Name` and one `Gene_CDS_length`. The following source satisfies the dependency:

| Accession_Number | Gene_Name | Gene_CDS_length |
|---|---|---|
| NM_001 | BRCA1 | 1782 |
| NM_002 | TP53 | 1176 |

In this instance of the tabular data source, each value of `Accession_Number` is associated with exactly one value of `Gene_Name` and `Gene_CDS_length`, satisfying the FD. Now, consider another instance of the same tabular data source:

| Accession_Number | Gene_Name | Gene_CDS_length |
|---|---|---|
| NM_001 | BRCA1 | 1782 |
| NM_001 | BRCA1 | 1920 |

Here, the second row violates the FD because `Accession_Number` = NM_001 is associated with two different values for `Gene_CDS_length`. This violates the constraint that a single determinant value (in this case, `Accession_Number`) must map to a unique dependent value. Such violations in a DIS lead to redundant or conflicting mappings in KG creation and may affect semantic consistency and performance.
**Key and prime attributes.** Given a relation $R$, a *key* is a set $K$ of attributes of $R$ that functionally determines all the attributes in $R$, and $K$ is minimal, i.e., no proper subset of $K$ meets this condition. $R$ can have several keys, and any attribute of $R$ that belongs to any of the key of $R$ is called a *prime attribute* [24]. In Fig. 1a, `source1.csv` corresponds to a relation comprising four attributes `Accession Number`, `Gene Name`, `Gene CDS length`, and `ID_sample`. Since {`Accession Number`} ⟶ {`Gene Name`,`Gene CDS length`}, there is only one key, {`Accession Number`,`ID_sample` }; both `Accession Number` and `ID_sample` are prime attributes.
**Closure of Attributes based on Functional Dependencies.** A data source $R$ includes attributes in set $Y$ with functional dependencies in $F$. For $X \subseteq Y$, the set $X^+$ represents the *closure* of $X$ based on $F$: $X^+$ comprises all the attributes in $Y$ that are functionally determined by $X$ based on $F$.
**Example:** Consider the data source `source1.csv` with attributes `Accession_Number`, `Gene_Name`, and `Gene_CDS_length`, and the functional dependency:

$F$ = {`Accession_Number` → `Gene_Name`, `Gene_CDS_length`}

The closure of {`Accession_Number`} under $F$, {`Accession_Number`}$^+$, is: {`Accession_Number`, `Gene_Name`, `Gene_CDS_length`}

**Cover of a Set of Functional Dependencies.** A set of functional dependencies $G$ *covers* another set $F$ iff every FD in $F$ can be inferred from $G$, i.e., $F^+ \subseteq G^+$.
**Example:** Consider the sets of functional dependencies:

$F$ = {`Accession_Number` → `Gene_Name`,
`Accession_Number` → `Gene_CDS_length`}

and

$G$ = {`Accession_Number` → `Gene_Name`, `Gene_CDS_length`}

Then $G$ covers $F$ since `Accession_Number` → `Gene_Name` and `Accession_Number` → `Gene_CDS_length` can both be inferred from the single FD in $G$.
**Minimal Cover.** A set $E$ of functional dependencies is the *minimal cover* of a set $F$ if:

1. $E$ covers $F$, i.e., $F^+ \subseteq E^+$
2. $E$ is minimal: if any dependency $Z \rightarrow Q$ is removed from $E$, then $E$ no longer covers $F$

**Example:** Consider the functional dependencies:

$F$ = {`Accession_Number` → `Gene_Name`, `Gene_CDS_length`}

A minimal cover $E$ of $F$ is:

$E$ = {`Accession_Number` → `Gene_Name`,
`Accession_Number` → `Gene_CDS_length`}

Each dependency in $E$ has a singleton attribute on the right-hand side, and the set contains no redundant FDs. Removing any one of them would mean $E$ no longer covers $F$.
**Mapping Rule Planning.** The purpose of planning the execution of mapping rules is to group triples into partitions and determine an order of execution, which will aid the KG creation engines in the generation process. Iglesias et al. [22] determines an execution plan for the TMs by first defining $GP_M$, a set of sets of MAs in $M$ (inter- and intra-source) so that the union of all the sets of $GP_M$ equals $M$ and the pair-wise intersection of the sets in $GP_M$ is empty. Afterward, a plan $\overline{GP}_M$ is defined for $GP_M$ since the order of execution of the sets in affects the creation process. $\overline{GP}_M$ is a bushy tree plan, where the leaves are the groups of MA from $GP_M$. The inner nodes of the bushy tree represent union operators that combine the resulting RDF triples generated from executing the sets of $GP_M$. If there are overlapping triples between sets of $GP_M$, a duplicate removal process is applied at the level of the inner node. An execution plan is defined by which groups of $GP_M$ require duplicate removal; inner nodes that require duplicate removal are executed first, and those that do not have overlapping triples will be executed at the end. These plans of mapping rules minimize execution time but not necessarily memory consumption.
**Data Source Fragmentation.** Data fragmentation divides a database into multiple smaller databases. Data fragmentation must be done to rebuild the original database from the pieces. Data fragmentation can be carried out in one of three ways: Horizontal, Vertical, and Hybrid [30]. Horizontal fragmentation divides the table horizontally by rows to create a subset of rows. Vertical fragmentation splits the table vertically so that the new tables comprise a subset of columns. Lastly, fragmentation is considered hybrid when both horizontal and vertical fragmentation are applied.

### 2.4. Motivating example

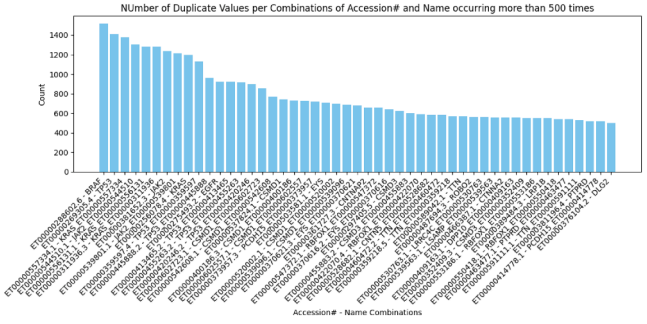
The motivating example for this work is a test case extracted from the SDM-Genomic-Datasets benchmark [26]. This benchmark was constructed by randomly sampling somatic mutation data from the COSMIC database.[1] It includes data sources of varying sizes (e.g., 10K, 100K,

[1] https://cancer.sanger.ac.uk/cosmic

Gene — 15 Attributes — 1M Rows

| Name | Accession# | CDS Length | HGNC Id | IDSample | IDTumour | Primary Site | Primary Histology | MutatioId | Mutation Descript | Mutation Gen Position | Mutation Somatic Status | Pubmed PMID | cFormat | pFormat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FOXP2_ET00000393494 | ET00000393494.2 | 2148 | 13875.0 | 2196248 | 2064526 | Ovary | Carcinoma | 37289045 | Unknown | 7:114161100-114161100 | Confirmed somatic variant | NULL | FOXP2_c.169-13572T>C | NULL |
| TAZ_ET00000351413 | ET00000351413.4 | 837 | 11577.0 | 2459924 | 2322761 | Biliary_tract | Carcinoma | 28894616 | Unknown | 23:153649092-153649092 | Confirmed somatic variant | NULL | TAZ_c.735+18T>C | NULL |
| CACNA1G | ET00000359106.5 | 7134 | 1394.0 | 1650962 | 1565749 | Large_intestine | Carcinoma | 31248983 | Substitution-missense | 17:48667837-48667837 | Confirmed somatic variant | 22810696.0 | CACNA1G_c.2307G>T | CACNA1G_E769D |
| RB1 | ET00000267163.4 | 2787 | 9884.0 | 2643737 | 2504024 | Stomach | Carcinoma | 20521458 | Substitution-missense | 13:49033829-49033829 | Reported in another cancer sample as somatic | 25656989.0 | RB1_c.1966C>T | RB1_R656W |
| RB1 | ET00000267163.4 | 2787 | 9884.0 | 809227 | 729428 | Cervix | Carcinoma | 152028398 | Unknown | NULL | Reported in another cancer sample as somatic | 1648218.0 | RB1_c.? | NULL |
| PCDH11X_ET00000373088 | ET00000373088.1 | 3933 | 8656.0 | 1913996 | 1802278 | Lung | Carcinoma | 33855333 | Substitution-missense | 23:91131801-91131801 | Reported in another cancer sample as somatic | | PCDH11X_c.562G>A | PCDH11X_D188N |
| YIPF7_ET00000415895 | ET00000415895.4 | 771 | 26825.0 | 2197818 | 2066096 | Kidney | NULL | 44707228 | Unknown | 4:44646971-44646971 | Confirmed somatic variant | | YIPF7_c.116+5031A>C | |
| AKT1 | ET00000554581.1 | 1443 | 391.0 | 1783437 | 1687436 | Endometrium | Carcinoma | 76516502 | Substitution-codingsilent | 14:105239693-105239693 | Variant of unknown origin | | AKT1_c.852G>A | AKT1_K284= |
| ASH1L_ET00000368346 | ET00000368346.3 | 8910 | 19088.0 | 2296168 | 2161771 | Large_intestine | Carcinoma | 35970166 | Substitution-codingsilent | 1:155340732-155340732 | Confirmed somatic variant | 25344691.0 | ASH1L_c.6390C>T | ASH1L_G2130= |
| JPH1 | ET00000342232.4 | 1986 | 14201.0 | 1650994 | 1565781 | Large_intestine | Carcinoma | 27657925 | Insertion-frameshift | 8:75157285-75157285 | Confirmed somatic variant | 22810696.0 | JPH1_c.1388dup | JPH1_R464Kfs*4 |
| JPH1 | ET00000342232.4 | 1986 | 14201.0 | 1650994 | 1565781 | Large_intestine | Carcinoma | 27657925 | Insertion-frameshift | 8:75157285-75157285 | Confirmed somatic variant | 22810696.0 | JPH1_c.1388dup | JPH1_R464Kfs*4 |
| GABRE | ET00000370328.3 | 1521 | 4085.0 | 2144505 | 2013723 | Lung | Carcinoma | 35099844 | Substitution-missense | 23:151131055-151131055 | Confirmed somatic variant | 22696596.0 | GABRE_c.403G>A | GABRE_D135N |

(a) Original data source from SDM-Genomic-Datasets with 1M rows, 15 attributes, 25% of the rows are duplicates




(b) Number of repeated instances per combinations of values of Accession# and Name extracted from SDM-Genomic-Datasets

| Engine | Time (sec) | Memory Usage (MB) |
|---|---|---|
| SDM-RDFizer | 395.53 sec | 4,709.76 MB |
| Morph-KGC | 222.97 sec | 23,371.90 MB |
| RMLMapper | TimeOut (five hours) | 33,170.98 MB ≤ |
| FlexRML | 920.33 sec | 2,220.37 MB |

(c) Results for the Execution Existing KG Creation Engines

**Fig. 2. Motivating Example Baseline 1.** a) The original data source used for the motivating example. (b) Bar graph showing the number of repeated combinations of the Accession# and Name attributes. (c) Results of the performance of the KG creation engines. This figure uses a DIS from SDM-Genomic-Datasets, employed in Baseline 1, to illustrate the impact of a non-normalized DIS—specifically data redundancy—on the performance of KG creation engines. As shown in (c), some engines, particularly RMLMapper, exhibit longer execution times and higher memory usage due to the overhead of processing duplicate data.

1M, and 10M rows) and different levels of duplication (e.g., 25% and 75%). Moreover, the benchmark provides multiple mapping configurations with varying structural and semantic complexity.

The specific test case used in this study involves two data sources, each containing 1 million rows and a duplication rate of 25%. These sources consist of 15 attributes in total. The corresponding mapping includes three concept mapping assertions (*MA*s), twelve attribute *MA*s, one referenced-source role *MA*, and one multi-source role *MA*. The resulting knowledge graph has a size of 4.6 GB.

Two data normalization approaches are applied: MapSDI [23] and NormaKG. MapSDI isolates multi-source role *MA*s by creating two separate triples maps: one containing only the multi-source role *MA* (child *TM*) and another containing only the corresponding concept *MA* (parent *TM*). In cases where a referenced-source or multi-source role *MA* uses the same data source for both parent and child *TM*s, and the join is performed over a common key attribute, MapSDI replaces the join with an equivalent single-source role *MA*. Additionally, MapSDI projects the data sources per *TM*, ensuring that each *TM* accesses only the necessary attributes.

NormaKG builds upon MapSDI by further normalizing the mapping structure. It groups single-source role *MA*s and attribute *MA*s that are functionally dependent on their concept *MA*, while isolating those that are not. A functional dependency is a relationship between two sets of attributes in a data source, where one set (the determinant) uniquely determines the other (the dependent). Leveraging functional dependencies enables the identification and elimination of data redundancy, improving both efficiency and semantic clarity during KG creation.

Four KG creation engines are used in the experimental evaluation: SDM-RDFizer [14], Morph-KGC [19], RMLMapper [31], and FlexRML [32]. The execution timeout is set to five hours for all engines.

Fig. 2a illustrates the original data source used in the motivating example. This configuration is denoted as *Baseline 1*. This scenario reflects a **Redundancy Anomaly**, where the duplicate data in the sources leads to repeated triples in the KG that must be eliminated. Fig. 2a highlights the frequency of repeated value combinations in the `Accession#` and `Name` attributes, with many combinations exceeding 500 occurrences. As Chaves-Fraga et al. [8] noted, high duplicate rates significantly impact KG creation by increasing resource consumption and execution time. To generate a duplicate-free KG, engines must apply duplicate removal mechanisms, which can be computationally expensive. To assess the impact of duplicates on KG creation, we evaluated several engines using this dataset. Fig. 2c compares execution time and memory usage across engines.

Among the tested engines, RMLMapper failed to complete execution within the five-hour limit. This failure stems from two key anomalies in the data integration system:

- **Redundancy Anomalies:** RMLMapper materializes all triples (including duplicates) before applying sorting-based duplicate elimination. This leads to memory overload and excessive execution time.
- **Join Anomalies:** The presence of an N-M join between two 1M-row sources (due to a multi-source role MA) without considering selectivity causes the generation of massive intermediate results, further increasing execution time and memory usage.

7 Attributes

| Name | Accession# | CDS Length | HGNC Id | IDSample | IDTumour | Primary Site |
|---|---|---|---|---|---|---|
| FOXP2_ET00000393494 | ET00000393494.2 | 2148 | 13875.0 | 2196248 | 2064526 | Ovary |
| TAZ_ET00000351413 | ET00000351413.4 | 837 | 11577.0 | 2459924 | 2322761 | Biliary_tract |
| CACNA1G | ET00000359106.5 | 7134 | 1394.0 | 1650962 | 1565749 | Large_intestine |
| RB1 | ET00000267163.4 | 2787 | 9884.0 | 2643737 | 2504024 | Stomach |
| RB1 | ET00000267163.4 | 2787 | 9884.0 | 809227 | 729428 | Cervix |
| PCDH11X_ET00000373088 | ET00000373088.1 | 3933 | 8656.0 | 1913996 | 1802278 | Lung |
| YIPF7_ET00000415895 | ET00000415895.4 | 771 | 25825.0 | 2197818 | 2066096 | Kidney |
| AKT1 | ET00000554581.1 | 1443 | 391.0 | 1783437 | 1687436 | Endometrium |
| ASH1L_ET00000368346 | ET00000368346.3 | 8910 | 19088.0 | 2296168 | 2161771 | Large_intestine |
| JPH1 | ET00000342232.4 | 1986 | 14201.0 | 1650994 | 1565781 | Large_intestine |
| GABRE | ET00000370328.3 | 1521 | 4085.0 | 2144505 | 2013723 | Lung |

761,938 rows

7 Attributes

| Name | Accession# | IDSample | IDTumour | Primary Site | Mutatiold | pFormat |
|---|---|---|---|---|---|---|
| FOXP2_ET00000393494 | ET00000393494.2 | 2196248 | 2064526 | Ovary | 37289045 | NULL |
| TAZ_ET00000351413 | ET00000351413.4 | 2459924 | 2322761 | Biliary_tract | 28894616 | NULL |
| CACNA1G | ET00000359106.5 | 1650962 | 1565749 | Large_intestine | 31248983 | CACNA1G_E769D |
| RB1 | ET00000267163.4 | 2643737 | 2504024 | Stomach | 20521458 | RB1_R656W |
| RB1 | ET00000267163.4 | 809227 | 729428 | Cervix | 152028398 | NULL |
| PCDH11X_ET00000373088 | ET00000373088.1 | 1913996 | 1802278 | Lung | 33855333 | PCDH11X_D188N |
| YIPF7_ET00000415895 | ET00000415895.4 | 2197818 | 2066096 | Kidney | 44707228 | |
| AKT1 | ET00000554581.1 | 1783437 | 1687436 | Endometrium | 76516502 | AKT1_K284= |
| ASH1L_ET00000368346 | ET00000368346.3 | 2296168 | 2161771 | Large_intestine | 35970166 | ASH1L_G2130= |
| JPH1 | ET00000342232.4 | 1650994 | 1565781 | Large_intestine | 27657925 | JPH1_R464Kfs*4 |
| GABRE | ET00000370328.3 | 2144505 | 2013723 | Lung | 35099844 | GABRE_D135N |

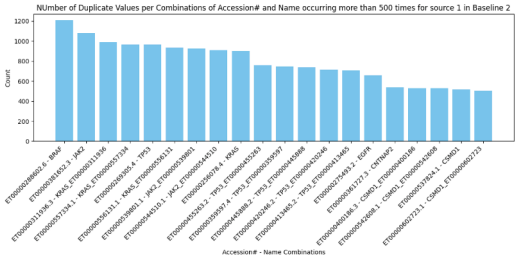
774,966 rows

2 Attributes

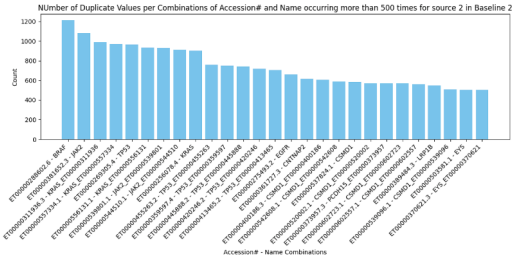
| CDS Length | IDTumour |
|---|---|
| 2148 | 2064526 |
| 837 | 2322761 |
| 7134 | 1565749 |
| 2787 | 2504024 |
| 2787 | 729428 |
| 3933 | 1802278 |
| 771 | 2066096 |
| 1443 | 1687436 |
| 8910 | 2161771 |
| 1986 | 1565781 |
| 1521 | 2013723 |

716,012 rows

(a) Three projected data sources following the simple projection method




(b) Repeated instances source 1




(c) Repeated instances source 2

| Engine | Time (sec) | Improvement (%) | Memory Usage (MB) | Improvement (%) |
|---|---|---|---|---|
| SDM-RDFizer | 276.55 sec | 30.08% | 376.08 MB | 91.99% |
| Morph-KGC | 205.58 sec | 7.8% | 23,053.28 MB | 1.36% |
| RMLMapper | 10,635.00 sec | 0% | 15,651.80 MB | 52.83% ≤ |
| FlexRML | 106.72 sec | 88.4% | 2,113.19 MB | 4.83% |

(d) Results of Executing Existing KG Creation Engines

**Fig. 3. Motivating Example Baseline 2.** (a) Resulting data sources after applying the MapSDI method. (b) Bar graph showing the number of repeated combinations of the Accession# and Name attributes in projected data source 1. (c) Same as (b), but for projected data source 2. (d) Performance results of the selected engines on the Baseline 2 DIS. This figure illustrates the outcome of applying MapSDI to the DIS shown in Fig. 2a and its effect on engine performance. As shown in (a), the original data source was split into three smaller, targeted sources—each corresponding to a specific triples map and containing only the necessary data for its execution. These data sources are approximately 25% smaller than the original. The bar graphs in (b) and (c) demonstrate a reduction in repeated value combinations, indicating decreased redundancy. Finally, (d) shows that this reduction improved performance, with shorter execution times and lower memory usage.

Other engines like Morph-KGC, SDM-RDFizer, and FlexRML completed the test but exposed additional weaknesses:

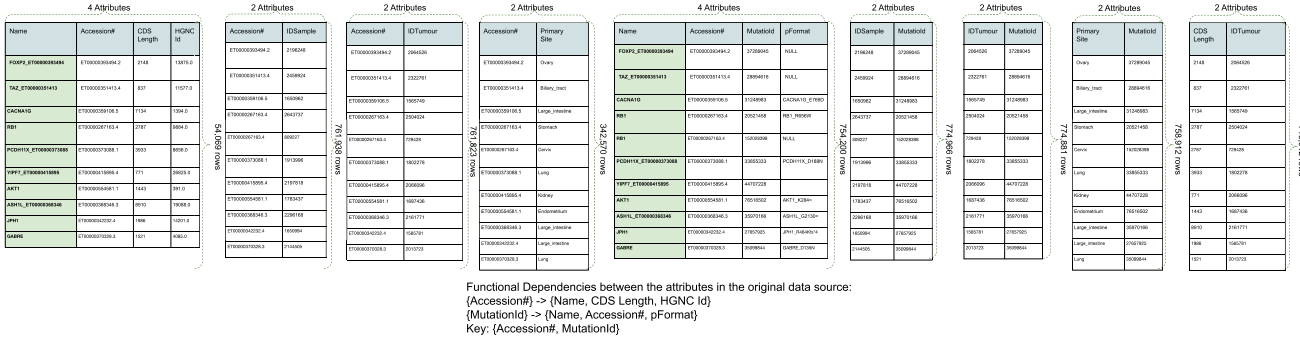
- **Execution Plan Anomalies:** SDM-RDFizer must store intermediate results for duplicate removal. Its execution strategy does not fully leverage join selectivity or plan optimization.
- **Structural Mapping Anomalies:** All mappings are executed over the full data source, even if many attributes are irrelevant to a given TM.
- **Dependency Violations:** No use of functional dependencies to reduce redundancy. In particular, attribute *MA*s whose object attributes do not fully functionally depend on the subject of the *MA* cause duplicated triples in the resulting KG during execution.

Two normalization strategies were evaluated to address these anomalies: *Baseline 2: MapSDI Projection* and *Baseline 3: NormaKG Normalization.*

*Baseline 2: MapSDI Projection.* Redundant columns are removed (addressing **Structural Mapping Anomalies**), and duplicate rows are partially filtered out (reducing **Redundancy Anomalies**). Fig. 3a illustrates the resulting data sources after applying the MapSDI method to the original data source from Fig. 2a. As shown, the original data source has been divided into three smaller data sources, each corresponding to a TM in the test case. While duplicate removal is still required, performance improves—particularly for engines with high memory usage like SDM-RDFizer and RMLMapper (see Fig. 3). However, Morph-KGC sees minimal benefit due to its internal data projection process. The MapSDI approach reduces data sources by removing unnecessary columns (vertical fragmentation) and eliminating duplicate rows (horizontal fragmentation). This results in approximately a 25% reduction in the number of rows, leading to an overall smaller dataset. Additionally, Fig. 3b and c show a decrease in the number of `Accession#` and `Name` value combinations exceeding 500 repetitions. However, duplicate removal is still required to generate a duplicate-free KG.

The table in Fig. 3d presents the performance of KG creation engines when using the data sources generated by the MapSDI method. RMLMapper presented performance improvements since it could complete the generation of the KG. This improvement can be attributed to the reduction in the number of duplicate values in the data source due to the simple reduction method, which reduced the amount of time and memory needed to generate the KG. Unfortunately, RMLMapper presented the longest execution time, and Morph-KGC presented the highest memory usage.

SDM-RDFizer improved its performance by an order of magnitude in memory savings and a shorter execution time. These improvements stem from the smaller data sources, which reduced iteration time, limited the number of duplicate checks, and lowered resource requirements for data loading. However, since the number of unique triples remained unchanged, the memory required to store them for duplicate removal did not decrease.




(a) Resulting data sources after applying the functional projection method.

| Engine | Time (sec) | Improvement (%) | Memory Usage (MB) | Improvement (%) |
|---|---|---|---|---|
| SDM-RDFizer | 259.96 sec | 34.28% | 135.82 MB | 97.12% |
| Morph-KGC | 198.76 sec | 10.86% | 22.855.22 MB | 2.21% |
| RMLMapper | 9,811.62 sec | 45.56% ≤ | 14,790.74 MB | 55.41 % ≤ |
| FlexRML | 74.48 sec | 91.91% | 9.44 MB | 99.57% |

(b) Performance of Existing KG Creation Engines

**Fig. 4. Motivating Example Baseline 3.** (a) Resulting data sources after applying the NormaKG approach. (b) The table shows the performance of the engines when transforming the normalized DIS. This figure presents the DIS resulting from applying NormaKG to the dataset shown in Fig. 2a, along with the corresponding engine performance. As shown in (a), NormaKG transforms the original DIS into nine smaller, normalized data sources, ranging in size from approximately 25% smaller to two orders of magnitude smaller. The normalization follows the principle of grouping functionally dependent attributes while isolating those that are not. The impact of this normalization is evident in (b), where engines show significantly reduced execution times and memory usage, especially FlexRML, which has a reduction of three orders of magnitude in memory usage.

Due to the simplified join operation, FlexRML showed notable execution time reductions and moderate memory savings. Although the number of triples generated from the join remained the same as in Baseline 1, the smaller data sources reduced the cost function computation time, improving overall execution efficiency.

Morph-KGC displayed only slight improvements in execution time and memory usage compared to Baseline 1. This is because Morph-KGC inherently applies its own data projection by decomposing each TM into multiple smaller TMs, each containing a single property. While Baseline 2's smaller data sources slightly reduced resource requirements, the major contributors to Morph-KGC memory usage—parallel processing—remained largely unaffected.

Baseline 2 results show the positive impact of DIS normalization on RML engines' performance. However, the extent of improvement depends on the architecture of the engine. Engines that store and process large amounts of duplicate data, such as RMLMapper and SDM-RDFizer, benefit the most. Meanwhile, engines like Morph-KGC, which internally optimize data projection, see limited gains.

*Baseline 3: NormaKG Normalization.* Attributes functionally dependent on the concept MA are grouped, and non-functionally dependent attributes are isolated. This eliminates redundant records (**Dependency Violations**) and removes the need for duplicate checking (**Redundancy Anomalies**). Additionally, multi-source joins are simplified or replaced with projections, mitigating **Join Anomalies**. The updated mappings avoid overlapping triple generation, thus reducing **Execution Plan Anomalies**.

Fig. 4a presents the resulting data sources after applying the NormaKG approach to the original source shown in Fig. 2a. As illustrated, the original data source is decomposed into nine smaller, semantically coherent sources. The size of the resulting fragments varies: some are approximately 25% smaller than the original, while others are reduced by up to two orders of magnitude. The NormaKG approach applies data normalization based on the functional dependencies (FDs) defined over the attributes of the data source used in the concept *MA* of each *TM*. Attributes that are functionally dependent on the subject of the concept *MA* remain in the same data source alongside the key attribute. In contrast, attributes that are not functionally dependent are isolated into separate data sources, each retaining the key attribute for referencing. The corresponding *TMs* are updated to reflect the new data source structure. By applying this normalization method, the data sources are transformed into a form that eliminates redundant records. This enables RML engines to generate the knowledge graph without requiring duplicate removal operations, thereby improving performance during KG materialization.

Fig. 4b presents the performance of the KG creation engines when transforming Baseline 3. All the engines presented reductions in memory usage and execution time.

RMLMapper presented improved execution time and memory usage. Compared to Baseline 1, it has at least 45.49% savings in execution time and at least 55.41% in memory usage. These savings can be attributed to the fact that RMLMapper does not have to apply duplicate removal, thus reducing the execution time. Unfortunately, RMLMapper still takes significantly longer than the other engines. This poor performance is due to how RMLMapper executes joins.

SDM-RDFizer improved its performance, especially its memory usage, which was reduced by one order of magnitude. Compared to Baseline 1, it saved 97.12% in memory usage and 34.28% in execution time. This significant savings in memory usage is due to not having to remove duplicates. SDM-RDFizer removes duplicates by storing all generated triples so that any new triples are compared to what was generated before. If a triple is determined to have been generated previously, it will be discarded. On the contrary, if a triple is determined not to have been generated previously, it is stored and added to the KG. SDM-RDFizer does have memory flushing policies that remove from memory triples that are not needed anymore. Still, they are not applied until all the TMs that might generate a particular triple are executed. This process increases memory usage. Therefore, by not removing duplicates, SDM-RDFizer does not have to store any generated triples, thus reducing its memory usage.

FlexRML presented the most significant improvement among all the engines. It offers savings of one order of magnitude in execution time and three orders of magnitude in memory usage. In comparison with Baseline 1, it saved 91.91% in execution time and 99.57% in memory usage. These significant savings are thanks to not having to remove duplicates. FlexRML removes duplicates very similarly to SDM-RDFizer. It stores all generated triples in a hash function so that any future triples are

compared to them to determine if they are duplicates. By not having to remove duplicates, FlexRML does not have to store any triples, thus reducing its memory usage. FlexRML is optimized to keep memory usage to a minimum; by having duplicate-free and smaller data sources, FlexRML could maintain its memory usage to the absolute minimum.
The Morph-KGC performance slightly improved compared to Baseline 1. Memory usage and execution time were reduced by 2.21% and 10.86%, respectively. Unfortunately, this lack of improvement can be attributed to the nature of Morph-KGC. As mentioned previously, Morph-KGC partitioned all TMs by their properties, and the data sources are projected accordingly. When projecting the data sources, all duplicate records are removed from the projected data sources. Since the data sources in Baseline 3 have been projected already, the small savings that Morph-KGC has are because most data sources cannot be projected any further and are much smaller than the data source in Baseline 1. Therefore, Morph-KGC requires fewer resources for the projection. Unfortunately, nothing else in the process in which Morph-KGC generates a KG is affected in using NormaKG.
The results highlight the significant benefits of using FDs and data normalization to optimize KG creation. As with Baseline 2, the extent of improvement depends on the engine's nature. Morph-KGC saw limited benefits due to its built-in projection approach, which already removes duplicates. RMLMapper, despite now completing KG creation, remains inefficient due to its slow join execution. SDM-RDFizer and FlexRML experienced the most dramatic improvements due to their reliance on in-memory duplicate checking, which Baseline 3 eliminated.
The motivating example illustrates how each of the anomalies described in Section 2 contributes to performance degradation during KG creation. Normalization methods systematically eliminate or mitigate these anomalies, leading to significant performance and scalability improvements.

## 3. Related work

Knowledge Graph (KG) creation from structured data relies on the definition and execution of mapping rules that specify how data from heterogeneous sources should be semantically transformed into RDF triples. Declarative mapping languages such as R2RML [33] and RML [31,34] are widely used for this purpose, and several engines have been developed to interpret and execute these mappings. However, the design and organization of mappings and data sources significantly affect the performance and scalability of the KG creation process. In this section, we review existing work on optimizing KG creation, grouping it into four categories: (i) mapping partitioning, (ii) data source fragmentation, (iii) engine-specific optimization, and (iv) semantic integration frameworks. We position NormaKG as a novel contribution that introduces a normalization theory for data integration systems and provides formal, semantics-preserving transformations to eliminate inefficiencies that existing techniques overlook.

### 3.1. Mapping partitioning techniques

Mapping partitioning techniques aim to reduce the complexity of KG creation pipelines where data sources or mapping assertions are partitioned. These techniques improve modularity, execution planning, or memory usage. Existing methods can be broadly categorized into two groups: *engine-specific* and *engine-agnostic* partitioning.
*Engine-specific partitioning strategies* are tightly coupled to the internal execution model of particular KG engines. For instance, Morph-KGC [19,20] applies syntactic mapping decomposition, splitting each triples map (*TM*) into multiple simpler *TM*s, each focused on a single predicate-object pair. This approach facilitates parallelism and reduces execution times but applies horizontal and vertical fragmentation indiscriminately, regardless of whether the fragmentation is semantically warranted. Morph-KGC does not exploit functional dependencies (FDs) or consider join semantics during partitioning, and thus may generate redundant triples when attributes not functionally dependent on the subject are included in multiple *TM*s. Importantly, Morph-KGC applies fragmentation, but not semantic normalization.
*Engine-agnostic partitioning strategies*, on the other hand, aim to provide general-purpose solutions independent of a specific engine. Iglesias et al. [22] propose partitioning complex mappings by grouping *TM*s based on their logical sources and organizing execution as a bushy tree. Intermediate union operations combine the outputs of different *TM* groups. This method improves execution planning and enables partial parallelization. However, it does not formally address semantic redundancy, partial or transitive dependencies, or the structural integrity of the resulting mappings.
Both approaches offer practical improvements, yet they rely on heuristic or structural criteria and do not incorporate semantic constraints. In contrast, *NormaKG* introduces a language-independent normalization theory grounded in relational dependency theory [24]. By enforcing a hierarchy of mapping-level normal forms (*TM*-NFs), NormaKG guarantees that the partitioned mappings are minimal, semantically coherent, and free of redundancy. It uses functional dependencies to guide the decomposition process, avoiding unnecessary subject duplication and ensuring that predicate-object pairs are only defined when functionally sound. To evaluate the effectiveness of NormaKG, we compare it with the methods proposed by Iglesias et al. [22] and Morph-KGC [19] in Section 6. The experiments assess execution time, memory usage, and the number of redundant triples produced, using benchmark scenarios from SDM-Genomic [26]. This comparison demonstrates the benefits of normalization-based partitioning for efficient KG creation.

### 3.2. Data fragmentation techniques

Data fragmentation techniques seek to reduce redundancies and remove unnecessary data to minimize the cost of the KG creation process. These techniques aim to improve memory usage and execution time. These methods, similar to mapping partitioning techniques, can be classified as either *engine-specific* or *engine-agnostic*.
*Engine-specific fragmentation strategies* are directly associated with specific KG engines. Morph-KGC [19,20] applies both mapping partitioning and data fragmentation techniques for the KG creation process. It partitions each TM into a single predicate-object pair. Afterward, each data source is vertically fragmented to retain the columns required by the corresponding partitioned TM, and then horizontally fragmented to remove duplicate rows. The data fragmentation technique employed by Morph-KGC focuses on removing unnecessary data but does not employ formal criteria for fragmentation. This method, alongside mapping partitioning, enables the parallel execution of TMs; however, because it does not leverage any criteria for data fragmentation, it can produce redundant triples when attributes are not functionally dependent on the subject.
*Engine-agnostic fragmentation strategies* seek to provide an engine-independent approach. MapSDI [23] employs data fragmentation for the optimization of the KG creation process. MapSDI focuses on applying data fragmentation to remove data that the TM does not need for its execution. MapSDI applies vertical fragmentation to project the pertinent columns and horizontal fragmentation to remove duplicate rows. Unlike Morph-KGC, MapSDI does not apply mapping partitioning; thus, its fragmentation is determined solely by the TM's requirements. This method improves engine performance with respect to memory usage and execution time. Unfortunately, MapSDI does not use FDs for data fragmentation; therefore, KG engines must still apply duplicate removal to ensure a duplicate-free KG.
Both approaches provide practical improvements, yet they do not adhere to any criteria beyond those required by the TM. Therefore, a duplicate-removal step remains necessary for KG creation. NormaKG applies data fragmentation following the FDs of the data sources. This method focuses on keeping functionally dependent properties together while isolating those that are not. This ensures a duplicate-free KG cre-

ation process. In Section 6, NormaKG is empirically compared with MapSDI [23].

### 3.3. Engine-specific optimization strategies

Engine-specific optimization strategies are embedded in the execution models of particular KG creation engines and are typically designed to improve runtime efficiency rather than to prevent structural anomalies in mapping assertions expressed using [R2]RML.

SDM-RDFizer [14,21] introduces several physical-level optimizations. It uses custom data structures for RDF data compression and fast in-memory access. The engine incorporates a mapping execution planner that reorders rule evaluation to maximize reuse of intermediate results. It also provides physical operators optimized for processing complex RML mappings involving multi-way joins, large datasets, and high duplicate rates. While effective at runtime, these optimizations do not address the structural design of mappings. In particular, SDM-RDFizer does not use functional dependencies to guide join planning or attribute selection, and it handles duplicate elimination as a post-processing step. In contrast, NormaKG applies semantic normalization prior to execution. By enforcing mapping-level normal forms, it eliminates structural redundancies and duplicate-generating joins in an engine-agnostic manner.

Morph-KGC [19] is a scalable engine that supports R2RML, RML, and RML-star [35]. It improves performance by introducing the concept of mapping partitions, which decomposes each mapping into smaller, independently executable units. This partitioning helps reduce memory consumption, supports parallel execution, and enables more localized duplicate elimination. However, Morph-KGC's partitioning strategy is syntactic and not guided by semantic considerations. It applies horizontal and vertical fragmentation uniformly across mappings, without analyzing whether the fragmentation is necessary. Moreover, it does not consider functional dependencies or attribute determinacy when creating partitions. In contrast, NormaKG performs dependency-aware normalization that preserves semantic correctness and avoids redundant subject generation, ensuring that each mapping is minimal and functionally sound.

FlexRML [32] introduces further runtime optimizations with a focus on performance predictability and resource management. It uses an estimation function to approximate the number of triples that will be generated, enabling better memory management during execution. Additionally, FlexRML applies several mapping-level optimizations, including self-join elimination and mapping normalization, to reduce execution cost. However, these normalization steps are applied during execution and are primarily designed to enhance runtime efficiency rather than ensure structural soundness of the input mappings. In contrast, NormaKG applies normalization transformations before execution, making the optimization process engine-independent and systematically improving mapping quality and performance without requiring runtime analysis.

### 3.4. Semantic integration frameworks and mapping tools

Several frameworks have been developed to support the specification and transformation of semantic mappings, with a focus on usability and integration flexibility rather than structural optimization.

Karma [36] provides a graphical interface that allows users to define semantic mappings by aligning data source attributes with ontology terms. It includes features such as semantic type suggestion and rule-based transformations, enabling domain experts to model mappings without requiring deep technical knowledge. However, Karma does not enforce structural constraints or verify the semantic minimality of the generated mappings. It assumes the correctness and performance of the mappings are managed elsewhere and therefore lacks mechanisms to detect redundant joins, unnecessary attribute usage, or violations of functional dependencies during mapping definition.

LDIF [37] implements an integration pipeline that combines mapping execution with post-processing tasks. These include entity resolution through Silk [38] and data fusion using Sieve [39]. While LDIF supports flexible integration workflows, it treats mappings as fixed inputs and does not analyze or optimize them for redundancy, dependency compliance, or join efficiency. As a result, it assumes that mappings are semantically valid and execution-ready, without providing tools to ensure or improve their structural quality.

In contrast, NormaKG serves as a pre-processing step that enhances mapping quality prior to execution. It systematically detects and eliminates anomalies in data integration systems, restructures mappings based on functional dependencies, and ensures that all mappings are structurally sound and semantically minimal. As a result, tools like Karma and LDIF can benefit from reduced execution overhead and redundancy-free mappings without requiring changes to their existing workflows.

### 3.5. Information fusion and neural fusion architectures

Information fusion aims to combine heterogeneous information sources to improve the robustness, accuracy, and interpretability of downstream tasks. A widely adopted taxonomy distinguishes three principal levels of fusion: *data-level*, *feature-level*, and *decision-level* fusion [40]. This classification provides a structured framework for analyzing both classical sensor fusion methods and recent neural fusion architectures.

**Data-level fusion** integrates raw or minimally processed data prior to feature extraction or learning. It is commonly applied in sensor networks, condition monitoring, and multi-modal perception systems, where multiple measurements describing the same phenomenon are combined to reduce uncertainty and redundancy. Recent approaches employ deep learning models to jointly process synchronized multi-sensor signals, often requiring carefully aligned and temporally consistent inputs [41]. While data-level fusion can exploit fine-grained correlations, it is highly sensitive to inconsistencies, schema heterogeneity, redundancy, and join-induced multiplicity in the underlying data. As a result, structural data integration becomes a prerequisite for effective fusion.

**Feature-level fusion** operates on intermediate representations learned independently from different modalities or sources. Neural architectures typically implement this level via concatenation, attention mechanisms, embedding aggregation, or cross-modal interaction layers. Hybrid CNN–RNN or attention-based models have been proposed to fuse vibration, acoustic, and visual features in industrial fault diagnosis and monitoring scenarios [42,43]. Feature-level fusion provides greater architectural flexibility than raw data fusion but remains dependent on the quality, consistency, and semantic alignment of the upstream feature representations.

**Decision-level fusion** combines outputs of multiple models using voting, weighting, ensemble aggregation, or probabilistic combination strategies [40]. This level is comparatively robust to heterogeneity in model architectures and input modalities, and is frequently used in safety-critical applications. However, because interactions between sources are considered only after independent processing, decision-level fusion may overlook fine-grained structural dependencies across data sources.

From the perspective of knowledge graphs (KGs) and data integration systems (DISs), KG creation can be interpreted as a form of *semantic data-level fusion*. Heterogeneous sources are aligned through declarative mappings and unified into a structured representation governed by an ontology. Unlike neural fusion models, KG-based fusion does not rely on learned embeddings but on explicit schema alignment, functional dependencies, and mapping rules. Consequently, structural anomalies in mappings—such as partial dependencies, transitive dependencies, or join-induced multiplicities—directly propagate into the fused representation and may degrade downstream fusion efficiency.

**Table 1**
Comparison of Related Work with NormaKG.

| Approach | Optimization Type | Strengths | Limitations / Contrast with NormaKG |
|---|---|---|---|
| Iglesias et al. [22] | Mapping Partitioning | Modular execution plan, union-aware | No dependency handling, may retain redundancy |
| Morph-KGC [19] | Mapping + Source Partitioning | Parallelization, simple TM decomposition | Redundant output; syntactic partitioning only; no semantic guarantees |
| MapSDI [23] | Source Projection | Attribute-level pruning | Manual configuration; no normalization or FD analysis |
| SDM-RDFizer [14] | Execution Scheduling | Predicate-aware prioritization | No mapping normalization; engine-specific strategy |
| Karma, LDIF [36,37] | Integration Tools | Schema alignment, post-processing support | No optimization of mappings or source structures |
| **NormaKG (This Work)** | Mapping + Source Normalization | FD-aware, join-free, duplication-free, engine-independent | Requires FD input; introduces normalization step |

The present work does not introduce a new neural fusion architecture. Instead, it addresses a complementary and largely orthogonal problem: the *structural optimization of data-level fusion prior to learning*. By introducing a normalization theory for DISs, NormaKG eliminates redundancy, unnecessary joins, and duplicate-generating dependencies during preprocessing. The resulting KG serves as a structurally consistent and duplicate-free intermediate representation, thereby strengthening subsequent feature-level and decision-level fusion processes.
Moreover, NormaKG can support *feature-level fusion* by transforming heterogeneous raw and semi-structured sources into a unified KG that functions as a normalized feature space. In this representation, semantically equivalent attributes are explicitly aligned rather than implicitly reconciled by downstream learning models. This reduces representational redundancy, mitigates schema heterogeneity, and improves robustness of subsequent fusion architectures.
In summary, while neural fusion models focus on representation learning and aggregation strategies, NormaKG operates at the structural layer of the fusion pipeline. It enhances the integrity and efficiency of intermediate fused representations, thereby complementing data-level, feature-level, and decision-level fusion methods without replacing them.

### 3.6. Novel features in NormaKG

NormaKG distinguishes itself from prior work by introducing a formal normalization theory for DISs. It defines a hierarchy of four mapping-level normal forms—TM-1NF, TM-2NF, TM-3NF, and TM-JNF—each addressing a specific class of structural or semantic anomalies. The associated transformation algorithm guarantees semantic preservation while eliminating redundant joins and restructuring mappings to ensure key-based subject generation and functional dependency compliance.
Unlike existing systems that rely on heuristic or engine-specific optimizations, NormaKG produces normalized DISs that can be executed by any RML-compliant engine without requiring runtime duplicate elimination. This engine-agnostic design ensures broad applicability and consistent performance improvements across diverse execution environments. Empirical results demonstrate that NormaKG significantly outperforms state-of-the-art systems in execution time, memory consumption, and output correctness. A summary of these comparisons is provided in Table 1.

## 4. A normalization theory for data integration systems

This section describes the normalization theory proposed to transform DISs suffering from the anomalies presented in Section 2 into equivalent anomaly-free DISs; Section 2 summarizes the notation used throughout the paper to define this theory.

**Definition 1.** Knowledge Graph Isomorphism. Let $G$ and $G'$ be knowledge graphs. We say that $G$ and $G'$ are *isomorphic up to blank node renaming* if there exists a bijection: $\mu : \text{BNodes}(G) \rightarrow \text{BNodes}(G')$ such that for every triple $(s, p, o) \in G$, the triple $(\mu(s), p, \mu(o)) \in G'$, where:

- $\mu(x) = x$ if $x$ is an IRI or literal,
- $\mu(x)$ is the corresponding blank node under the bijection if $x$ is a blank node.

We write $G \cong G'$ to denote that $G$ and $G'$ are isomorphic up to blank node renaming.

Intuitively, $G$ and $G'$ contain exactly the same triples, except possibly for the identifiers assigned to blank nodes.

**Definition 2.** Semantic Equivalence. Let $DIS_G = (O, S, M)$ and $DIS'_G = (O, S', M')$ be two data integration systems defined over the same ontology $O$. Let $G$ and $G'$ be the KGs generated by executing $DIS_G$ and $DIS'_G$, respectively, under set semantics. We say that $DIS_G$ and $DIS'_G$ are *semantically equivalent*, written $DIS_G \equiv DIS'_G$, if $G$ and $G'$ are isomorphic up to blank node renaming, i.e., $G \cong G'$.

**Problem Statement** Given a data integration system $DIS_G = (O, S, M)$, we define an evaluation function $\phi(.)$ that generates a KG $G$ from $DIS_G$. A utility function $f(.)$ quantifies the cost of executing $\phi(.)$; $f(.)$ captures execution time, memory consumption, and redundancy elimination. The problem of *optimizing KG creation* consists of finding a normalized data integration system $DIS'_G = (O, S', M')$ such that: i) $DIS_G$ and $DIS'_G$ are *semantically equivalent*, i.e., $\phi(DIS_G)$ and $\phi(DIS'_G)$ are isomorphic (up to blank node renaming); ii) Execution cost minimization, i.e., $f(\phi(DIS'_G))$ is minimal.

$$DIS'_G = \underset{DIS'_G \in \mathcal{B}^G}{\arg\min}\ f(\phi(DIS'_G)) \tag{1}$$

where $\mathcal{B}^G$ represents the set of normalized data integration systems that are semantically equivalent to $DIS_G$, i.e., that generate KGs isomorphic to $\phi(DIS_G)$ up to blank node renaming.
**Proposed Solution** We propose a normalization theory for data integration systems that establishes conditions for ensuring the correctness and efficiency of knowledge graph (KG) creation. Our approach addresses two critical issues:

- Identifying when the data sources in $S$ are free of anomalies and duplicate values, allowing us to determine triples maps whose evaluation will not generate duplicate triples during the KG creation process.
- Defining criteria for normalized triples maps, transforming original data sources and triples maps into normalized ones to minimize duplicate triples.

Since duplicate removal is a computationally expensive task during the KG creation process, our approach aims to reduce duplicate generation. This significantly lowers the consumption of computational resources such as memory and execution time, improving the overall efficiency.
We introduce a normalization process that transforms a given $DIS_G$ into a normalized $DIS'_G$, optimizing execution efficiency while preserving the semantics of the original data integration system. The normalization process ensures:

**Table 2**
Notation Summary.

| Notation | Explanation |
|---|---|
| $DIS_{\mathcal{G}} = (O, S, M)$ | Data Integration System, where $O$ is a unified ontology, $S$ is a set of data sources, and $M$ is a set of mapping assertions. Evaluating $DIS_{\mathcal{G}}$ generates a knowledge graph $\mathcal{G}$. |
| $body(\overline{X})$ :- $head(\overline{Y})$ | Mapping assertion defined as a Horn clause; $body(\overline{X})$ is a conjunction of predicates over sources in $S$, and $head(\overline{Y})$ is a class or property predicate in $O$. |
| $S_i(\overline{X})$ | Predicate symbol representing data source $S_i \in S$. |
| $C(f(y))$ | Class predicate in $O$; $f(y)$ is a functional symbol generating an IRI. |
| $P(f_1(y_1), f_2(y_2))$ | Role (object property) predicate in $O$. |
| $A(f(y_1), y_2)$ | Data property predicate in $O$. |
| $\theta(\overline{X_{i,1}}, \overline{X_{i,2}})$ | Join condition between attributes of logical sources. |
| $KG = (V, E, L)$ | Knowledge Graph, where $V$ is a set of vertices, $E$ is a set of edges, and $L$ is a labeling function. |
| $\phi(DIS_G)$ | Evaluation function producing an RDF KG from $DIS_G$. |
| $f(\phi(DIS_G))$ | Utility function measuring execution cost (time, memory consumption, redundancy elimination). |
| $G \cong G'$ | Knowledge graphs $G$ and $G'$ are isomorphic up to blank node renaming (RDF KG isomorphism). |
| $DIS_G \equiv DIS'_G$ | Data integration systems $DIS_G$ and $DIS'_G$ are semantically equivalent, i.e., $\phi(DIS_G) \cong \phi(DIS'_G)$. |
| $\mathcal{B}^G$ | Set of normalized data integration systems semantically equivalent to $DIS_G$. |
| $\mu[X]$ | Value of attribute set $X$ in tuple $\mu$. |
| $\upsilon[X]$ | Value of attribute set $X$ in tuple $\upsilon$. |
| $FD$ | Set of functional dependencies (minimal cover). |
| $TM$-1NF, $TM$-2NF, $TM$-3NF, $TM$-JNF | First, Second, Third, and Join Normal Forms for Triples Maps. |
| $DIS_G^{\text{norm}}$ | Normalized data integration system obtained by Algorithm 1. |

- The transformation of sources and mapping assertions to minimize duplicate generation during KG creation.
- The preservation of all relevant information in $DIS_G$.
- The reduction of computational complexity in evaluating mapping assertions $M$.

Our approach establishes a formal framework to identify when a data source $S$ is in a normalized state and provides transformation rules for mapping assertions to guarantee efficient KG generation. By differentiating between normalized triples maps and non-normalized ones, we ensure that the execution of mappings in $M$ results in an optimized KG creation process with minimal duplicate removal overhead.

### 4.1. Normal forms

We define a set of normal forms, each progressively reducing the redundancy and anomalies discussed in Section 2.2. These normal forms are inspired by classical database normalization theory [24], and are adapted in our context to address the structural and semantic characteristics of each component in a data integration system $DIS_G$. Without loss of generality, we assume that the normalization theory is applied over tabular data sources, i.e., the schemas of sources in $S$ conform to the relational data model. In cases where the original data sources are in semi-structured formats such as XML or JSON, we assume that these sources have been transformed into flat, tabular representations suitable for relational-style processing [44,45].

A **triples map** $tm$ is a mapping rule that includes: (a) A *Concept Mapping Assertion (CMA)*: Defines instances of a class in $O$ from a data source in $S$. (b) Zero or more *Role Mapping Assertions (RMA)*: Define relationships between instances. (c) Zero or more *Attribute Mapping Assertions (AMA)*: Define data properties.

**Prime Concept Mapping Assertion**. Consider a *Concept Mapping Assertion* $m_i : S_i(X_i) : -C(Y_i)$, where variables in $Y_i$ represent attributes in $X_i$ from $S_i$. $m_i$ is called a *Prime Concept Mapping Assertion* for $S_i$ iff all the attributes in $Y_i$ correspond to prime attributes of $S_i$. As shown in Fig. 1a, the `Accession Number` is a prime attribute of `source1.csv` and is used to define the class `ex:Transcript`, making it a *Prime Concept Mapping Assertion*.

**Example:** Consider the data source `source1.csv` (as shown in Fig. 1a) with attributes `Accession_Number`, `Gene_Name`, `Gene_CDS_length`, and `ID_sample`.

Suppose that {`Accession_Number`, `ID_sample`} is a key for the relation defined by $S_1$ from the file source1.csv, and thus, both attributes are considered prime. Now, consider the concept mapping assertion:

$$S_1(\text{Accession\_Number}, \text{Gene\_Name}, \text{Length}) \rightarrow$$
$$\text{ex:Transcript}(\text{concat}(\text{'ex:'}, \text{Accession\_Number}))$$

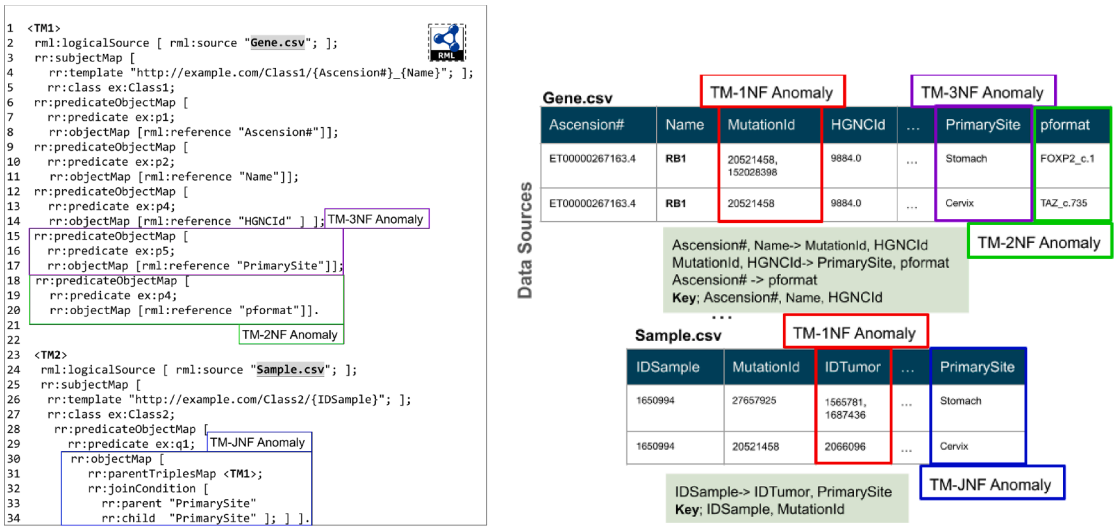
Since the subject of the resulting class instance is built using only `Accession_Number`, which is a prime attribute of $S : 1$, thus, this is a *Prime Concept Mapping Assertion*.

**Subject Attribute Set of a Triples Map.** Let $tm$ be a triples map in a data integration system, and let $S_i$ be its associated logical source. The *subject attribute set* of $tm$, denoted $\texttt{SubjAtt}(tm)$, is the minimal set of attributes $\{A_1, A_2, \ldots, A_k\} \subseteq \texttt{Attr}(S_i)$ used in the subject template of $tm$ to generate RDF subject IRIs or blank nodes.

**Example:** If the subject of a triples map $tm$ is constructed using the template `http://example.org/Gene/{geneID}_{species}`, then $\texttt{SubjAtt}(tm) = \{\texttt{geneID}, \texttt{species}\}$.

**TM-1NF (First Normal Form for Triples Maps)**. A triples map $tm$-defining correspondences over the attributes in $X_i$- is in TM-1NF iff: (a) Every attribute in the set of attributes $X_i$ is atomic (i.e., it does not contain multi-valued attributes or nested structures). (b) Each mapping assertion refers to a single for every subject-predicate pair.

**TM-1NF** addresses *structural mapping anomalies* by enforcing the use of atomic attributes, ensuring that no multi-valued or complex fields are




(a) RML Mapping with abnormalities.

(b) Data sources containing abnormalities violating TM-1NF, TM-2NF, and TM-3NF.

**Fig. 5. Running Example with original DIS used in the Example.** This figure illustrates the mapping and corresponding data sources for the running example. (a) shows a mapping with an N–M join between $TM1$ and $TM2$ that violates TM-JNF, as the join results in duplicate triples. Property $p5$ violates TM-2NF due to a partial dependency on the subject, while property $p4$ violates TM-3NF because $PrimarySite$ introduces a transitive dependency. (b) depicts a set of data sources reflecting the anomalies in (a), along with a violation of TM-1NF: properties $MutationId$ and $IDTumor$ contain multi-valued entries.

included in mapping assertions. This restriction ensures that each attribute used in a mapping refers to a single, indivisible value, preventing the generation of multiple RDF triples from nested or composite fields. As a result, it avoids the need for data preprocessing or special parsing logic during the KG creation process.

**TM-2NF (Second Normal Form for Triples Maps).** Let $tm$ be a triples map defined over a logical source $S'$ with attribute set $X'$, and let $T$ be the subject attribute set of $tm$. The triples map $tm$ is in **TM-2NF** iff: (a) $tm$ meets the **TM-1NF**, (b) $T$ is a **key** of $S'$, and (c) If there is at least one **partial dependency** of the form $P \rightarrow A$ with $P \subset T$ exists in the functional dependencies of $S'$, then the set of attributes from $A$ used in the mapping assertions of $tm$ contains at most **one non-prime attribute**.

This definition ensures that a triples map over a partially normalized source may tolerate a single property derived from a partially dependent attribute, while still avoiding redundancy captured by the relational 2NF.

**TM-3NF (Third Normal Form for Triples Maps).** Let $tm$ be a triples map defined over a logical source $S_i$ with attribute set $X_i$, and let $FD$ be the set of functional dependencies defined over $S_i$. Let SubjectAtt($tm$) $\subseteq X_i$ be the *Subject Attribute Set of* $tm$, i.e., the set of attributes used to construct the subject of the triples map. Then, $tm$ is in **TM-3NF** iff: (a) $tm$ is in **TM-2NF**; (b) SubjectAtt($tm$) forms a **superkey** of $S_i$; (c) Every non-prime attribute used in any *Attribute Mapping Assertion (AMA)* or *Role Mapping Assertion (RMA)* is not transitively dependent on any non-key attribute (i.e., it is **prime** or fully dependent on a key).

This ensures that:

- The subject of every generated RDF triple is uniquely identifiable through a superkey;
- All object values are determined without redundancy, preventing transitive dependency anomalies;
- The mapping structure avoids semantic inconsistency and unnecessary duplication during KG materialization.

In Fig. 1a, `TriplesMap1` comprises one *Concept Mapping Assertion* defining class `ex:Transcript` using `Accession Number` and three *Attribute Mapping Assertions*. Since `Accession Number` is not a superkey of `source1.csv`, `TriplesMap1` is not in TM-3NF. In Fig. 1a, `TriplesMap1` comprises one *Concept Mapping Assertion* defining class `ex:Transcript` using `Accession Number` and three *Attribute Mapping Assertions*. Since `Accession Number` is not a superkey of `source1.csv`, `TriplesMap1` is not in 3NF.

**TM-3NF** avoids *redundancy anomalies*, *dependency violations*, and *execution plan anomalies*. By requiring that the *subject attribute set of a triples map* corresponds to a superkey of the logical source, and that all attributes used in *Attribute* and *Role Mapping Assertions* are prime, ensures that no redundant triples are generated, eliminates inconsistencies arising from transitive dependencies, and minimizes the need for duplicate elimination during KG creation.

**TM-JNF (Join Normal Form for Triples Maps).** Let $tm_1, tm_2$ be two triples maps such that $tm_2$ is a *child* of $tm_1$ via a join condition $C$ over a set of attributes `ATT`.

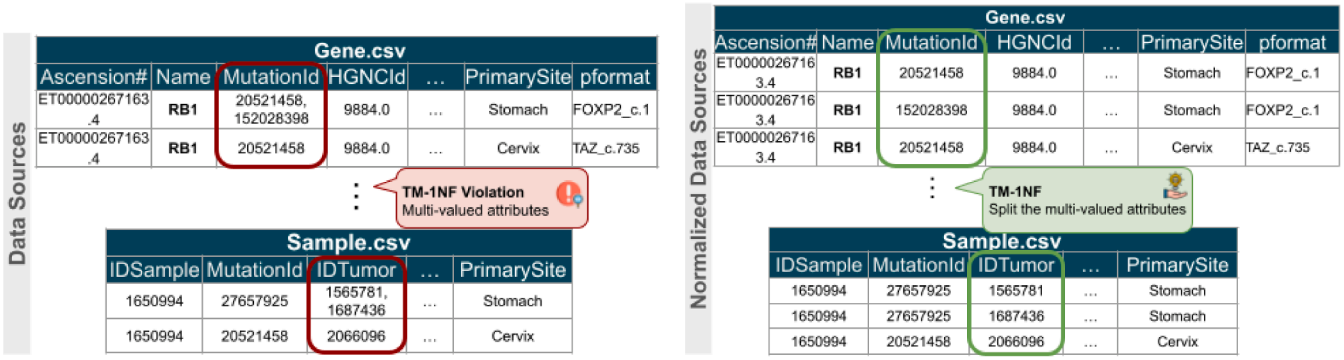
The triples map $tm_2$ satisfies **TM-JNF** iif: (a) Let $J = S_1 \bowtie_C S_2$ be the logical join of the sources referenced by $tm_1$ and $tm_2$ under condition $C$. (b) The set of attributes used to construct the *subject of* $tm_2$ must form a **superkey** of the join result $J$, i.e., they functionally determine all other attributes in $J$.

**TM-JNF** ensures that no duplicate RDF subjects (or subject-object pairs) are generated from join operations during KG materialization. When the subject in the child triples map is derived from a superkey of the join result, the mapping will yield uniquely identifiable RDF resources, thereby avoiding duplication due to join multiplicity.

**Normalized Data Integration System (NDIS).** Let $DIS_G = \langle O, S, M \rangle$ be a data integration system and $FD$ be the set of functional dependencies among the attributes of sources in $S$. $DIS_G$ is a *Normalized Data Integration System (NDIS)* iff: (a) All the sources in $S$ are in 3NF according to classical relational normalization; (b) All triples maps in $M$ are in TM-3NF; and (c) Any triples map involving Multi-Source Role Mapping Assertions is also in TM-JNF.

### 4.2. Transforming data integration systems

To normalize a data integration system into TM-NF, we define a set of transformation rules that guide the restructuring of mapping assertions and data sources. These transformations are aligned with the structural requirements of TM-1NF, TM-2NF, TM-3NF, and TN-JNF. We now define a set of transformation rules to normalize a data integration system




(a) Data sources violating TM-1NF.

(b) Data sources respecting TM-1NF.

| Engine | Time | Memory Usage | TM-1NF Time | TM-1NF Memory Usage |
|---|---|---|---|---|
| SDM-RDFizer | 152.67 sec | 957.99 MB | 131.15 sec | 646.98 MB |
| Morph-KGC | 67.33 sec | 3451.97 MB | 45.70 sec | 3106.88 MB |
| RMLMapper | TimeOut | 17856.85 MB ≤ | TimeOut | 17692.62 MB ≤ |
| FlexRML | 819.98 sec | 755.96 MB | 798.90 sec | 404.89 MB |

(c) Performance Existing KG Creation Engines

**Fig. 6. TM-1NF Normalization.** Normalized data sources that satisfy TM-1NF. This figure builds on the data sources from Fig. 5, highlighting and addressing the anomaly that violates TM-1NF. (a) shows the original data sources, where attributes $MutationID$ and $IDTumor$ are multi-valued, thus violating **TM-1NF**. (b) presents the result of normalizing these data sources by splitting the multi-valued attributes into separate rows, thereby ensuring compliance with **TM-1NF**. (c) presents a table that shows the performance of the KG creation engines when running the original DIS and the normalized one.

$DIS_G = \langle O, S, M \rangle$ according to proposed normal forms. Each rule targets a specific class of anomalies toward a 3NF data integration system. More importantly, the transformation rules assume that the set of functional dependencies defined over each data source $S_i \in S$ corresponds to a *minimal cover*, i.e., the dependencies are non-redundant and satisfy both minimality and coverage with respect to the original set of functional dependencies of $S_i$. The algorithm described by Ullman [24] can be used to compute the minimal cover for each data source prior to applying the transformation rules.

**TM-1NF – Enforcing Attribute Atomicity**

- **Rule 1.1:** For each data source $S_i$, identify attributes in $X_i$ that are multi-valued (e.g., lists, arrays) or nested. Generate a new source $S_i'$ by unnesting or flattening these attributes such that each atomic value appears in a separate row. This transformation results in an equivalent source where all attributes are atomic.
- **Rule 1.2:** For each triples map $tm$ that references a multi-valued or nested attribute $a \in X_i$: (a) Update the logical source of $tm$ to point to the transformed source $S_i'$ where $a$ has been flattened into atomic values. (b) Replace any mapping assertion that uses $a$ with an assertion over the atomic version of $a$, such that each RDF triple is generated for exactly one atomic value of $a$, ensuring semantic consistency with the original mapping intent.

Fig. 5 presents a DIS that exhibits multiple anomalies. Both data sources shown in Fig. 5b contain multi-valued properties (*MutationId* and *IDTumor*), which violate the principle of atomicity and, thus, TM-1NF. Fig. 6 shows the normalization of the data sources *Gene.csv* and *Sample.csv* to comply with TM-1NF. This normalization involves splitting the values of the multi-valued properties into separate rows to ensure each field contains atomic values.

Fig. 6 presents a table comparing the performance of KG creation engines when transforming the original DIS from Fig. 5 and the TM-1NF-compliant DIS shown in Fig. 6b. All engines show reduced execution time and memory usage with the TM-1NF-compliant DIS, as the original DIS requires additional preprocessing to handle multi-valued properties. This overhead—splitting rows containing multiple values—accounts for the performance gap. Particularly, RMLMapper fails to complete the transformation of either DIS within a reasonable time frame.

**TM-2NF – Eliminating Partial Dependencies**

- **Rule 2.1:** Identify all composite keys $K_i$ in the logical source $S_i$ and determine subsets $P \subset K_i$ such that there exist attributes $A \subseteq X_i$ with $P \rightarrow A$ and $A \nrightarrow K_i$.
- **Rule 2.2:** For each such partial dependency $P \rightarrow A$ where $A$ contains more than one non-prime attribute: (a) Create a new logical source $S_j$ over $K_i \cup A$. (b) Define a new triples map $tm'$ over $S_j$ with subject attribute set $K_i$. (c) Move to $tm'$ a single attribute mapping assertion or role mapping assertion that uses a non-prime attribute in $A$. This guarantees that the resulting triples map $tm'$ does not violate TM-2NF.
- **Rule 2.3:** In the original triples map $tm$, remove the AMA or RMA that was moved to $tm'$. If this assertion affects a semantically relevant triple (i.e., the object position of a triple in $tm$), then add a referenced object mapping assertion (ROMA) from $tm$ to $tm'$ using the join condition $K_i$, thereby preserving semantic connectivity.

The *Gene.csv* data source from Fig. 5b contains a property that violates TM-2NF. In particular, the property *pformat* is only dependent on *Ascension#* and not on the entire composite key (*Ascension#*, *Name*, and *HGCNId*), thereby violating TM-2NF. Fig. 7 presents the normalization of *Gene.csv* and its corresponding triples map *TM1* to ensure compliance with TM-2NF. In this normalization, *pformat* is projected into a separate data source. The triples map *TM1* is also partitioned accordingly, with the property *p3* moved into its own dedicated triples map.

Fig. 7 presents a table showing the performance of KG creation engines when transforming the TM-1NF-compliant DIS from Fig. 6 and the TM-2NF-compliant DIS from Fig. 7. All engines demonstrate improved performance with the TM-2NF-compliant DIS. This improvement is primarily due to the isolation of *p3*, a partially dependent property in *Gene.csv*, which reduces redundancy and decreases the overall data size. RMLMapper remains unable to complete the transformation of either DIS.

**TM-3NF – Eliminating Transitive Dependencies and Enforcing Superkey-Based Subject Construction**

- **Rule 3.1:** Let $tm$ be a triples map defined over a logical source $S_i$ with attribute set $X_i$, and let $FD$ be the set of functional dependencies defined over $S_i$. If there exists a transitive dependency of the form $K_i \rightarrow A$ and $A \rightarrow B$ such that: (a) $K_i$ is the key of $S_i$, (b) $A$ and $B$ are disjoint subsets of non-key attributes in $X_i$, (c) $B$ is referenced in the

```
<TM1>
  rml:logicalSource [ rml:source "Gene.csv"; ];
  rr:subjectMap [
     rr:template "http://example.com/Class1/{Ascension#}_{Name}";];
   rr:class ex:Class1;
rr:predicateObjectMap [
   rr:predicate ex:p1;
    rr:objectMap [rml:reference "Ascension#"]];
rr:predicateObjectMap [
  rr:predicate ex:p2;
    rr:objectMap [rml:reference "Name"]];
rr:predicateObjectMap [
  rr:predicate ex:p4;
   rr:objectMap [rml:reference "HGNCId" ] ];
rr:predicateObjectMap [
  rr:predicate ex:p5;
  rr:objectMap [rml:reference "PrimarySite"]];
rr:predicateObjectMap [
  rr:predicate ex:p3;
  rr:objectMap [rml:reference "pformat"]].
```

TM-2NF Anomaly

(a) RML mapping with TM-2NF anomaly.

```
<TM1.1>
 rml:logicalSource [ rml:source "Gene_proy_1.csv"; ];
 rr:subjectMap [
    rr:template "http://example.com/Class1/{Ascension#}_{Name}";
  rr:class ex:Class1;];
rr:predicateObjectMap [
  rr:predicate ex:p1;
   rr:objectMap [rml:reference "Ascension#"]];
rr:predicateObjectMap [
  rr:predicate ex:p2;
    rr:objectMap [rml:reference "Name"]];
rr:predicateObjectMap [
   rr:predicate ex:p4;
    rr:objectMap [rml:reference "HGNCId" ] ];
rr:predicateObjectMap [
  rr:predicate ex:p5;
  rr:objectMap [rml:reference "PrimarySite"]].
<TM1.2>
 rml:logicalSource [ rml:source "Gene_proy_2.csv"; ];
 rr:subjectMap [
    rr:template "http://example.com/Class1/{Ascension#}_{Name}";
  rr:class ex:Class1;];
rr:predicateObjectMap [
  rr:predicate ex:p3;
 rr:objectMap [rml:reference "pformat"]].
```

TM-2NF Normalized

(b) RML mapping normalized TM-2NF.

**Gene.csv**

| Ascension# | Name | MutationId | HGNCId | ... | PrimarySite | pformat |
|---|---|---|---|---|---|---|
| ET0000026716 3.4 | RB1 | 20521458 | 9884.0 | ... | Stomach | FOXP2_c.1 |
| ET0000026716 3.4 | RB1 | 152028398 | 9884.0 | ... | Stomach | FOXP2_c.1 |
| ET0000026716 3.4 | RB1 | 20521458 | 9884.0 | ... | Cervix | TAZ_c.735 |

TM-2NF Violation
Partially dependent attribute

(c) Data source with TM-2NF anomaly.

**Gene_proy_1.csv**

| Ascension# | Name | MutationId | HGNCId | PrimarySite |
|---|---|---|---|---|
| ET0000026716 3.4 | RB1 | 20521458 | 9884.0 | Stomach |
| ET0000026716 3.4 | RB1 | 152028398 | 9884.0 | Stomach |
| ET0000026716 3.4 | RB1 | 20521458 | 9884.0 | Cervix |

TM-2NF Normalization
pformat is isolated

**Gene_proy_2.csv**

| Ascension# | Name | pformat |
|---|---|---|
| ET0000026716 3.4 | RB1 | FOXP2_c.1 |
| ET0000026716 3.4 | RB1 | TAZ_c.735 |

(d) Data source normalized TM-2NF.

| Engine | TM-1NF Time | TM-1NF Memory Usage | TM-2NF Time | TM-2NF Memory Usage |
|---|---|---|---|---|
| SDM-RDFizer | 131.15 sec | 646.98 MB | 130.73 sec | 620.47 MB |
| Morph-KGC | 45.70 sec | 3106.88 MB | 34.95 sec | 2725.83 MB |
| RMLMapper | TimeOut | 17692.62 MB ≤ | TimeOut | 17221.58 MB ≤ |
| FlexRML | 798.90 sec | 404.89 MB | 545.41 sec | 372.29 MB |

(e) Performance Existing KG Creation Engines

**Fig. 7. TM-2NF Normalization.** Normalized DIS that satisfy TM-2NF. This figure illustrates triples map $TM1$ from Fig. 5 and the normalized Gene.csv data source from Fig. 6b, highlighting the TM-2NF violations and the resulting DIS after normalization. (a) shows triples map $TM1$, where property $p3$ ($pformat$) violates TM-2NF due to a partial dependency on the subject attributes. (b) presents the normalized triples maps, where $p3$ is isolated into its own map. (c) shows the normalized data source from Fig. 6b, in which $pformat$ still violates TM-2NF. (d) displays the fully normalized data source, where $pformat$ is moved to a separate file, consistent with the mapping in (b); unnecessary attributes are also removed. (e) presents a comparison table showing the performance of KG creation engines when using the TM-1NF versus TM-2NF DIS.

object position of at least one AMA or RMA in $tm$, then: (i) Create a new logical source $S_j$ containing attributes $A \cup B$, (ii) Define a new triples map $tm'$ over $S_j$ where the subject is generated from $A$ (which becomes the key of $S_j$), (iii) Migrate all AMAs and RMAs in $tm$ that refer to $B$ to $tm'$.

- **Rule 3.2:** In the original triples map $tm$, remove all mapping assertions (AMAs and RMAs) that involve attributes in $B$. If $\texttt{SubjAttrs}(tm) \nrightarrow B$ under $FD$ and the removed assertions are necessary to preserve the semantics of the original mapping, then introduce a *Referenced Object Mapping Assertion (ROMA)* from $tm$ to $tm'$ using a join condition $\theta(K_i, A)$, expressing the semantic relationship between entities identified by $tm$ and those by $tm'$.
- **Rule 3.3:** For each triples map $tm$, verify that the subject attribute set $\texttt{SubjAttrs}(tm)$ forms a **superkey** of its logical source. If this condition is not satisfied, update the logical source (e.g., through restructuring or projection) or revise the subject construction in $tm$ so that $\texttt{SubjAttrs}(tm) \rightarrow X_i$ holds.

The *Gene.csv* data source from Fig. 5b contains a property that violates TM-3NF. Specifically, *PrimarySite* introduces transitive dependencies among the functional dependencies (FDs). It is functionally dependent on *MutationId* and *HGCNId*, which are themselves dependent on *Ascension#* and *Name*. Fig. 8 shows the normalization of *Gene.csv* and its corresponding triples map *TM1* to achieve compliance with TM-3NF. This normalization involves projecting *PrimarySite* into a separate table and partitioning *TM1* by moving the property *p5* into its own triples map. As a result, the transitive dependencies are eliminated.

Fig. 8 presents a table showing the performance of the KG creation engines when transforming the TM-2NF-compliant DIS from Fig. 7 and the TM-3NF-compliant DIS introduced in Fig. 8. All engines show improved performance with the TM-3NF DIS. This improvement can be attributed to the further normalization of the DIS, resulting in smaller and more manageable data sources. RMLMapper is still unable to transform either DISs within a reasonable time, primarily due to how it handles join execution, which significantly increases its processing time.

**TM-JNF – Eliminating Join Anomalies**

- **Rule J.1:** For every pair of triples maps $tm_1$ and $tm_2$ such that $tm_2$ is a child of $tm_1$ via a join condition $\theta(\overline{X_i}, \overline{X_j})$, construct a new logical

```
<TM1>
  rml:logicalSource [ rml:source "Gene_proy_1.csv"; ];
  rr:subjectMap [
     rr:template
"http://example.com/Class1/{Ascension#}_{Name}";];
    rr:class ex:Class1;
  rr:predicateObjectMap [
    rr:predicate ex:p1;
      rr:objectMap [rml:reference "Ascension#"]];
  rr:predicateObjectMap [
    rr:predicate ex:p2;
      rr:objectMap [rml:reference "Name"]];
  rr:predicateObjectMap [
    rr:predicate ex:p4;
     rr:objectMap [rml:reference "HGNCId" ] ];
rr:predicateObjectMap [
  rr:predicate ex:p5;
  rr:objectMap [rml:reference "PrimarySite"]].
```

TM-3NF Anomaly

(a) RML mapping with TM-3NF anomaly.

```
<TM1.1>
  rml:logicalSource [ rml:source "Gene_proy_1.csv"; ];
  rr:subjectMap [
     rr:template "http://example.com/Class1/{Ascension#}_{Name}";
    rr:class ex:Class1;];
  rr:predicateObjectMap [
    rr:predicate ex:p1;
      rr:objectMap [rml:reference "Ascension#"]];
  rr:predicateObjectMap [
    rr:predicate ex:p2;
      rr:objectMap [rml:reference "Name"]];
  rr:predicateObjectMap [
    rr:predicate ex:p4;
     rr:objectMap [rml:reference "HGNCId" ] ].
<TM1.3>
 rml:logicalSource [ rml:source "Gene_proy_3.csv"; ];
 rr:subjectMap [
    rr:template "http://example.com/Class1/{Ascension#}_{Name}";
    rr:class ex:Class1;];
 rr:predicateObjectMap [
  rr:predicate ex:p5;
 rr:objectMap [rml:reference "PrimarySite"]].
```

TM-3NF Normalized

(b) RML mapping normalized TM-3NF.

**Gene_proy_1.csv**

| Ascension# | Name | MutationId | HGNCId | PrimarySite |
|---|---|---|---|---|
| ET00000267163.4 | RB1 | 20521458 | 9884.0 | Stomach |
| ET00000267163.4 | RB1 | 152028398 | 9884.0 | Stomach |
| ET00000267163.4 | RB1 | 20521458 | 9884.0 | Cervix |

TM-3NF Violation
Transitive attribute

(c) Data source with TM-3NF anomaly.

**Gene_proy_1.csv**

| Ascension# | Name | HGNCId |
|---|---|---|
| ET00000267163.4 | RB1 | 9884.0 |

TM-3NF Normalization
PrimarySite is isolated

**Gene_proy_3.csv**

| Ascension# | Name | PrimarySite |
|---|---|---|
| ET00000267163.4 | RB1 | Stomach |
| ET00000267163.4 | RB1 | Cervix |

(d) Data source normalized TM-3NF.

| Engine | TM-2NF Time | TM-2NF Memory Usage | TM-3NF Time | TM-3NF Memory Usage |
|---|---|---|---|---|
| SDM-RDFizer | 130.73 sec | 620.47 MB | 111.94 sec | 561.88 |
| Morph-KGC | 34.95 sec | 2725.83 MB | 30.86 sec | 2720.43 MB |
| RMLMapper | TimeOut | 17221.58 MB ≤ | TimeOut | 17065.31 MB ≤ |
| FlexRML | 545.41 sec | 372.29 MB | 22.21 sec | 275.39 MB |

(e) Performance Existing KG Creation Engines

**Fig. 8. Normalized TM-3NF.** Normalized DIS that satisfy TM-3NF. This figure builds on the DIS shown in Fig. 7, highlighting the anomalies that violate TM-3NF and presenting the resulting normalized DIS. (a) shows the triples map from Fig. 7b, where property $p5$ violates TM-3NF due to a transitive dependency introduced by the attribute *PrimarySite*. (b) presents the normalized triples maps, where $p5$ is isolated into its own map. (c) shows the TM-2NF data source, highlighting *PrimarySite* as the source of the transitive dependency. (d) displays the normalized TM-3NF data sources, where *PrimarySite* has been moved to a separate data source to resolve the transitive dependency. (e) presents a comparison table showing the performance of KG creation engines using the TM-2NF versus TM-3NF DIS.

source $S_k$ as the relational join of the sources referenced by $tm_1$ and $tm_2$ under $\theta$.

- **Rule J.2:** Let $J$ be the result of the join. Identify the set of attributes used to construct the subject in $tm_2$. If these attributes form a *superkey* of $J$ (i.e., they functionally determine all other attributes in $J$), then define a new logical source $S_k$ as the projection of $J$ to only the attributes required to generate triples in $tm_2$.
- **Rule J.3:** Replace the original triples map $tm_2$ with a new triples map $tm'_2$ that uses $S_k$ as its logical source and defines the same triple patterns over the projected attributes; $tm'_2$ does not include any join.

**TM-JNF** addresses *join anomalies* by eliminating costly and potentially spurious N-M joins during runtime. By transforming the join into a precomputed logical source—ensuring that the subject of the triples map is constructed from a superkey of the join result—duplicate generation is structurally avoided. This transformation: *(i)* reduces runtime memory usage, *(ii)* avoids unnecessary computation, and *(iii)* guarantees correctness by preventing overlapping or duplicate triples.

The triples map *TM2* from Fig. 5 and its corresponding data source *Sample.csv* contain an anomaly that violates TM-JNF. The join defined in *TM2* uses the property *PrimarySite* as a join condition, generating duplicate triples. To eliminate this redundancy, the DIS is normalized to preemptively resolve the join at the data level. Fig. 9 illustrates this normalization process. Since the join is a many-to-many (N-M) relationship, it is executed in advance, producing a new data source for *TM2*. Consequently, the join condition is removed from *TM2* and replaced with a `rr:template` operation, which is simpler to execute.

Fig. 9 presents a table showing the performance of the KG creation engines when transforming the TM-3NF-compliant DIS from Fig. 8 and the TM-JNF-compliant DIS introduced in Fig. 9. All engines show improved performance, with SDM-RDFizer and FlexRML, in particular, significantly reducing memory usage. This improvement is due to the TM-JNF-compliant DIS enabling KG generation without requiring du-

```
<TM2>
 rml:logicalSource [ rml:source "Sample.csv"; ];
 rr:subjectMap [
  rr:template "http://example.com/Class2/{IDSample}";
];
  rr:class ex:Class2;
   rr:predicateObjectMap [
     rr:predicate ex:q1;
        rr:objectMap [
        rr:parentTriplesMap <TM1>;
        rr:joinCondition [
          rr:parent "PrimarySite"
          rr:child  "PrimarySite" ]; ] ].
```

TM-JNF Anomaly

(a) RML mapping with TM-JNF anomaly.

```
<TM2>
 rml:logicalSource [ rml:source "Sample_proy_1.csv"; ];
 rr:subjectMap [
  rr:template "http://example.com/Class2/{IDSample}"; ];
  rr:class ex:Class2;
   rr:predicateObjectMap [
     rr:predicate ex:q1;
        rr:objectMap [
        rr:template "http://example.com/Class1/{Ascension#}_{Name}"
   ;]].
```

TM-JNF Normalized

(b) [RML mapping normalized TM-JNF.

**Sample.csv**

| IDSample | MutationId | IDTumor | ... | PrimarySite |
|---|---|---|---|---|
| 1650994 | 27657925 | 1565781 | ... | Stomach |
| 1650994 | 27657925 | 1687436 | ... | Stomach |
| 1650994 | 20521458 | 2066096 | ... | Cervix |

TM-JNF Violation
Redundant results

(c) Data source with TM-JNF anomaly.

**Sample_proy_1.csv**

| IDSample | Ascension# | Name |
|---|---|---|
| 1650994 | ET00000267163.4 | RB1 |

TM-JNF Normalization
Join is executed between the sources

(d) Data source normalized TM-JNF.

| Engine | TM-3NF Time | TM-3NF Memory Usage | TM-JNF Time | TM-JNF Memory Usage |
|---|---|---|---|---|
| SDM-RDFizer | 111.94 sec | 561.88 | 65.65 sec | 109.75 MB |
| Morph-KGC | 30.86 sec | 2720.43 MB | 23.47 sec | 2622.12 MB |
| RMLMapper | TimeOut | 17065.31 MB ≤ | 156.04 sec | 16483.92 MB |
| FlexRML | 22.21 sec | 275.39 MB | 17.69 sec | 4.44 MB |

(e) Performance Existing KG Creation Engines

**Fig. 9. Normalized TM-JNF.** Normalized DIS that satisfies TM-JNF. This figure illustrates triples map $TM2$ and its corresponding data source from Fig. 5, highlighting the violation of TM-JNF and the resulting normalization. (a) shows triples map $TM2$ from Fig. 5a, which violates TM-JNF because its join operation generates duplicate triples. (b) presents the normalized version of $TM2$, where the join is removed since it is executed at the data source level, making the join condition unnecessary in the mapping. (c) shows the original data source for $TM2$, where the join on $PrimarySite$ leads to redundancy. (d) presents the resulting data source after performing the join directly at the data level. With this final normalization step, the DIS from Fig. 5 is fully normalized. (e) provides a comparison table showing the performance of KG creation engines when using the TM-3NF versus the fully normalized DIS.

plicate removal. SDM-RDFizer and FlexRML handle duplicate removal by storing all generated triples and comparing each new triple to previous ones—a process that consumes substantial memory. RMLMapper, which previously struggled with join operations, can execute the TM-JNF-compliant DIS efficiently because the join has been replaced with a simpler operation that it can handle.

With the normalization that was performed to comply with TM-JNF, the DIS introduced in Fig. 5 is completely normalized and complies with all normal forms.

Comparing the performance of the engines when executing the original DIS from Fig. 5 and the fully normalized version from Fig. 9 reveals a significant improvement for the engines in both execution time and memory usage. SDM-RDFizer and FlexRML, in particular, show substantial reductions in memory consumption—with FlexRML reducing usage by two orders of magnitude. Notably, RMLMapper can now complete the execution of the fully normalized DIS, which it previously could not. While Morph-KGC also demonstrated improved performance, its gains were more modest compared to the other engines. These results underscore the importance of applying data normalization in the KG creation process.

**Mapping Refinement Rules**

- **Rule 4.1:** Decompose each *Multi-Source Role Mapping Assertion* (MRMA) into separate triples maps, each with a logical source and explicit join condition. If necessary, define a new logical source as the relational join of the participating sources and reassign the corresponding mapping assertions to the new TM.
- **Rule 4.2:** For every triples map $tm$, remove from its mapping assertions any attribute that is not a **prime attribute** of the logical source or is not involved in a join condition required to evaluate a Role Mapping Assertion (RMA). This ensures that only functionally valid attributes are preserved.
- **Rule 4.3:** Group Attribute Mapping Assertions (AMAs) and Role Mapping Assertions (RMAs) in a triples map $tm$ only if they lie within the same **functional dependency closure** of the attributes used in the Concept Mapping Assertion (CMA). That is, let $X_C$ be the subject attribute set of the CMA; then, only include AMAs and RMAs defined over attributes $A$ such that $X_C \rightarrow A$ holds in the dependency set $FD$.

### 4.3. Correctness of the transformation rules

This section establishes the correctness of the transformation rules defined in Section 4.2.

**Definition 3.** Rule Correctness. A transformation rule $R$ is *correct* if, for every data integration system $DIS_G$, its application produces a system $DIS'_G$ such that:

1. $DIS_G \equiv DIS'_G$ (semantic preservation),
2. the resulting logical sources satisfy the structural condition of the target normal form,
3. no new violations of functional dependencies are introduced.

The correctness of the Transformation Rules is demonstrated for each class of rules based on Definitions 1–3.

*TM-1NF Correctness.*

**Proposition 1.** *Rules 1.1–1.2 preserve semantic equivalence and enforce attribute atomicity.*

**Proof.** Let $DIS_G = (O, S, M)$ and $DIS'_G = (O, S', M')$ be data integration systems such as $DIS'_G$ is obtained after applying Rules 1.1–1.2 to a source $S_i$ that contains a multi-valued (or nested) attribute $a \in X_i$. Denote by $G$ and $G'$ the RDF graphs generated by executing $DIS_G$ and $DIS'_G$, respectively, under set semantics.

**Attribute atomicity.** By Rule 1.1, $S_i$ is transformed into $S'_i$ by unnesting/flattening $a$ so that each value of $a$ appears in a separate tuple. Hence, in $S'_i$, every attribute value is atomic. Therefore, $DIS'_G$ satisfies TM-1NF.

**Semantic equivalence.** We prove $DIS_G \equiv DIS'_G$ by contradiction, i.e., assume that $G$ and $G'$ are not isomorphic up to blank node renaming. Then, there exists an RDF triple $t$ such that either (i) $t \in G$ and $t \notin G'$, or (ii) $t \in G'$ and $t \notin G$ (modulo blank node renaming).

*Case (i): $t \in G$ and $t \notin G'$.* Since $t \in G$, it is generated by some triples map $tm \in M$ from some tuple $\tau$ of its logical source. If $t$ does not depend on $a$ (i.e., $a$ does not appear in the object position of the corresponding AMA/RMA, nor in any expression used to construct it), then Rules 1.1–1.2 do not modify the attributes used to generate $t$, and the subject construction of $tm$ is unchanged. Hence the same valuation exists in $DIS'_G$, implying $t \in G'$, a contradiction.

Otherwise, $t$ depends on $a$, i.e., $t = (s, p, o)$ is generated from a value $v \in \tau[a]$ (where $\tau[a]$ denotes the set/list of values of $a$ in tuple $\tau$). By Rule 1.1, $\tau$ is replaced in $S'_i$ by tuples $\tau_v$ such that $\tau_v[a] = v$ and all other attributes coincide with $\tau$. By Rule 1.2, $tm$ is updated to read $a$ from $S'_i$ and to emit exactly one triple per atomic value. Therefore, evaluating $tm$ over $\tau_v$ produces the same triple $(s, p, o)$ in $G'$. Thus $t \in G'$, a contradiction.

*Case (ii): $t \in G'$ and $t \notin G$.* Since $t \in G'$, it is generated by some updated triples map $tm' \in M'$ from some tuple $\tau_v$ of the transformed source $S'_i$. If $t$ does not depend on $a$, then $tm'$ coincides with $tm$ on the attributes used for $t$, and $\tau_v$ agrees with the original tuple $\tau$ on those attributes; hence $t$ is also generated in $G$, contradicting $t \notin G$.

If $t$ depends on $a$, then $\tau_v[a] = v$ for some value $v$ that originated from a tuple $\tau$ in $S_i$ with $v \in \tau[a]$ by construction of Rule 1.1. In the original system, evaluating $tm$ over $\tau$ with the value $v$ in the expansion of $a$ generates the same triple $t$. Hence $t \in G$, contradicting $t \notin G$.

Since both cases lead to contradictions, our assumption was false. Therefore, $G$ and $G'$ are isomorphic up to blank node renaming, i.e., $DIS_G \equiv DIS'_G$. □

*TM-2NF Correctness.*

**Proposition 2** (TM-2NF Correctness). *Rules 2.1–2.3 eliminate partial dependencies while preserving semantic equivalence.*

**Proof.** Let $DIS_G = (O, S, M)$ and $DIS'_G = (O, S', M')$ be data integration systems such as $DIS'_G$ is obtained after applying Rules 2.1–2.3 to a triples map $tm$ defined over a logical source $S_i$ with composite key $K_i$ and attribute set $X_i$. Assume that Rule 2.1 identifies a partial dependency $P \to A$ with $P \subset K_i$, $A \subseteq X_i$, and $A \nrightarrow K_i$. Let $G$ and $G'$ be the KGs generated by executing $DIS_G$ and $DIS'_G$, respectively, under set semantics.

**Elimination of partial dependencies.** By contradiction, assume that after applying Rules 2.2–2.3, the resulting system still violates TM-2NF, i.e., there exists a logical source in $S'$ with a composite key $K$ and a non-prime attribute $B$ such that $Q \to B$ for some proper subset $Q \subset K$. Consider the step that handles the corresponding partial dependency identified by Rule 2.1. Rule 2.2 creates a new logical source $S_j$ over $K_i \cup A$ and a triples map $tm'$ with subject attribute set $K_i$, and moves exactly one mapping assertion that uses a non-prime attribute in $A$ to $tm'$. After this move, the moved non-prime attribute is no longer generated from $tm$ and is generated only from $tm'$. Moreover, in $S_j$, the key is $K_i$, hence no proper subset of $K_i$ can determine any non-prime attribute in $A$ without contradicting the assumption that $K_i$ is a key. Therefore, the moved assertion cannot participate in a TM-2NF violation in $tm'$. In $tm$, Rule 2.3 removes the moved assertion, hence the identified violation cannot remain in $tm$ either. This contradicts the assumption that a partial dependency violation persists after applying the rules. Hence, Rules 2.1–2.3 eliminate the identified partial dependencies and $DIS'_G$ complies with TM-2NF.

**Semantic equivalence.** We now prove $DIS_G \equiv DIS'_G$ by contradiction. Assume that $G$ and $G'$ are not isomorphic up to blank node renaming. Then there exists an RDF triple $t$ such that either (i) $t \in G$ and $t \notin G'$, or (ii) $t \in G'$ and $t \notin G$ (modulo blank node renaming).

*Case (i): $t \in G$ and $t \notin G'$.* Since $t \in G$, it is generated by some triples map from some tuple $\tau \in S_i$. If $t$ is produced by an AMA/RMA not moved by Rule 2.2, then the corresponding mapping assertion remains in the same triples map and is evaluated over a logical source that still contains all attributes required for $t$; subject construction is unchanged. Hence $t$ is also generated in $G'$, a contradiction.

Otherwise, $t$ is produced by the specific AMA/RMA moved from $tm$ to $tm'$ by Rule 2.2, and thus depends on some non-prime attribute $b \in A$. Let $\tau[K_i] = \kappa$ be the key value of $\tau$. By Rule 2.2(a), $S_j$ contains (at least) the tuple $\tau'$ with $\tau'[K_i] = \kappa$ and $\tau'[A] = \tau[A]$. By Rule 2.2(b)–(c), $tm'$ generates the same predicate-object pair for $b$ and uses the same subject attributes $K_i$. Therefore, the triple $t$ is generated by $tm'$ in $DIS'_G$, contradicting $t \notin G'$.

If the moved assertion affects a semantically relevant triple (object position in $tm$), Rule 2.3 adds a ROMA from $tm$ to $tm'$ using the join condition on $K_i$, which preserves the corresponding links between the entities generated by $tm$ and those generated by $tm'$. Thus, no link triple required by the original mapping is lost. Hence Case (i) cannot occur.

*Case (ii): $t \in G'$ and $t \notin G$.* Since $t \in G'$, it is generated either by an assertion that already existed in $tm$ and was not moved, or by the moved assertion now in $tm'$. In the former case, the same assertion evaluated over the corresponding tuple in $S_i$ generates $t$ in $G$, contradicting $t \notin G$. In the latter case, $t$ is generated by $tm'$ over some tuple $\tau' \in S_j$ with $\tau'[K_i] = \kappa$ and $\tau'[A] = \alpha$. By construction of $S_j$ (Rule 2.2(a)), $\tau'$ originates from some tuple $\tau \in S_i$ with the same key value $\tau[K_i] = \kappa$ and the same attribute values on $A$. Since the moved mapping assertion existed in $tm$ before the transformation, evaluating $tm$ over $\tau$ generates the same triple $t$ in $G$. This contradicts $t \notin G$.

Since both cases lead to contradictions, our assumption was false. Therefore, $G$ and $G'$ are isomorphic up to blank node renaming, i.e., $DIS_G \equiv DIS'_G$. □

*TM-3NF Correctness.*

**Proposition 3** (TM-3NF Correctness). *Rules 3.1–3.3 eliminate transitive dependencies and enforce superkey-based subject construction while preserving semantic equivalence.*

**Proof.** Let $DIS_G = (O, S, M)$ and $DIS'_G = (O, S', M')$ be data integration systems such as $DIS'_G$ is obtained after applying Rules 3.1–3.3. Let $tm$ be a triples map defined over a logical source $S_i$ with attribute set $X_i$, key $K_i$, and functional dependency set $FD$. Assume Rule 3.1 detects a transitive dependency of the form $K_i \to A$ and $A \to B$ such that $A$ and $B$ are disjoint subsets of non-key attributes and some attribute in $B$ appears in the object position of at least one AMA or RMA in $tm$. Let $G$ and $G'$ be the KGs generated by executing $DIS_G$ and $DIS'_G$, respectively, under set semantics.

**Elimination of transitive dependencies.** By contradiction, assume that after applying Rules 3.1–3.2 the resulting system still violates TM-3NF, i.e., there exists a logical source $S^\star \in S'$ with key $K^\star$ and non-key attributes $Y^\star$ such that a non-key attribute $y \in Y^\star$ depends transitively on $K^\star$ via some non-key attribute $z \in Y^\star$ (equivalently, $K^\star \to z$ and $z \to y$ with $y$ non-prime). Consider the transformation triggered by this transitive pattern.

Rule 3.1(i) creates a new logical source $S_j$ containing $A \cup B$ and Rule 3.1(ii) makes $A$ the key of $S_j$. Thus, within $S_j$, every attribute in $B$ is functionally determined by $A$ and the dependency chain $K_i \to A \to B$ no longer exists inside a single source because $K_i$ is not an attribute set of $S_j$. Rule 3.1(iii) migrates to $tm'$ all mapping assertions that refer to attributes in $B$. Rule 3.2 removes from $tm$ all mapping assertions that

involve $B$. Hence, no triple pattern in $tm$ is any longer evaluated over $B$, and every triple pattern that uses $B$ is evaluated only in $tm'$ over $S_j$, where the determinant is the key $A$.

Therefore, the original transitive dependency is structurally eliminated from the mapping layer: $B$ is no longer generated in a triples map whose subject is built from $K_i$. This contradicts the assumption that a transitive dependency violation persists after applying Rules 3.1–3.2. Hence, Rules 3.1–3.2 eliminate the identified transitive dependencies and enforce TM-3NF with respect to attributes that participate in RDF triple generation.

**Superkey-based subject construction.** Rule 3.3 explicitly enforces that for every triples map $tm$ in the resulting system, $\texttt{SubjAttrs}(tm)$ is a superkey of its logical source, i.e., $\texttt{SubjAttrs}(tm) \to X$ holds for the attributes $X$ of that source. By contradiction, assume that after applying Rule 3.3 there exists a triples map $tm^\star$ whose subject attribute set is not a superkey of its logical source. Then Rule 3.3 would have applied to $tm^\star$ and either (i) updated the logical source (via restructuring/projection) or (ii) revised subject construction until $\texttt{SubjAttrs}(tm^\star) \to X$ holds. Thus, such $tm^\star$ cannot exist, contradicting the assumption. Therefore, the resulting system enforces superkey-based subject construction.

**Semantic equivalence.** We now prove $DIS_G \equiv DIS'_G$ by contradiction. Assume that $G$ and $G'$ are not isomorphic up to blank node renaming. Then there exists an RDF triple $t$ such that either (i) $t \in G$ and $t \notin G'$, or (ii) $t \in G'$ and $t \notin G$ (modulo blank node renaming).

*Case (i): $t \in G$ and $t \notin G'$.* Since $t \in G$, it is generated by some triples map (in particular, possibly $tm$) from some tuple $\tau \in S_i$. If $t$ does not depend on any attribute in $B$, then Rules 3.1–3.2 do not change the mapping assertions needed to generate $t$ and do not change the subject construction of the generating triples map (except possibly to enforce superkey-ness, which does not remove bindings). Hence the same triple is generated in $G'$, contradicting $t \notin G'$.

Otherwise, $t$ depends on some attribute $b \in B$ that appears in the object position of an AMA/RMA in $tm$. By Rule 3.1(iii), that mapping assertion is migrated to $tm'$ and evaluated over $S_j$. Because $K_i \to A$ and $A \to B$, for every tuple $\tau \in S_i$ there exists a corresponding tuple $\tau' \in S_j$ with $\tau'[A] = \tau[A]$ and $\tau'[B] = \tau[B]$ (by construction of $S_j$ as a projection containing $A \cup B$). Moreover, the subject of $tm'$ is generated from $A$, and since $A$ is the key of $S_j$, it identifies the same $A$-entity consistently. Therefore, the triple $t$ is generated by $tm'$ in $G'$, contradicting $t \notin G'$.

If, in addition, the original semantics required a link between the entity generated by $tm$ (keyed by $K_i$) and the entity generated by $tm'$ (keyed by $A$), Rule 3.2 introduces a ROMA with join condition $\theta(K_i, A)$, ensuring that the intended semantic relationship is preserved. Thus, no semantically required triple is lost, and Case (i) cannot occur.

*Case (ii): $t \in G'$ and $t \notin G$.* Since $t \in G'$, it is generated either by: (a) a mapping assertion that remained in some original triples map, or (b) a migrated mapping assertion now evaluated in $tm'$ over $S_j$, or (c) a ROMA introduced by Rule 3.2. For (a), the corresponding assertion and the required attributes existed in $DIS_G$, thus $t$ would also be generated in $G$, contradicting $t \notin G$. For (b), $t$ uses only attributes from $A \cup B$. By construction of $S_j$, each tuple $\tau' \in S_j$ originates from some tuple $\tau \in S_i$ with the same values on $A \cup B$. Since the migrated assertion existed in $tm$ before the transformation, evaluating $tm$ over $\tau$ generates the same triple $t$ in $G$, contradicting $t \notin G$. For (c), a ROMA is introduced only to preserve semantic connectivity that was already implied by the original mapping patterns; thus, the corresponding relationship is already entailed by $DIS_G$ via the original assertions linking the same entities through the shared determinant structure. Hence, $t$ cannot be genuinely new with respect to the intended semantics, contradicting $t \notin G$. Both cases yield contradictions. Therefore, our assumption was false and $G$ and $G'$ are isomorphic up to blank node renaming, i.e., $DIS_G \equiv DIS'_G$. □

*TM-JNF Correctness.*

**Proposition 4** (TM-JNF Correctness)**.** *Rules J.1–J.3 eliminate join anomalies while preserving semantic equivalence.*

**Proof.** Let $DIS_G = (O, S, M)$ and $DIS'_G = (O, S', M')$ be data integration systems such as $DIS'_G$ is obtained after applying Rules J.1–J.3 to a parent–child pair of triples maps $(tm_1, tm_2)$, where $tm_2$ is a child of $tm_1$ via a join condition $\theta(\overline{X_i}, \overline{X_j})$. Let $S_i$ and $S_j$ be the logical sources referenced by $tm_1$ and $tm_2$, respectively. Let $J = S_i \bowtie_\theta S_j$ denote the relational join result, and let $S_k$ be the logical source constructed by Rules J.1–J.2. Let $tm'_2$ be the join-free triples map produced by Rule J.3 with logical source $S_k$. Let $G$ and $G'$ be the RDF KGs generated by executing $DIS_G$ and $DIS'_G$, respectively, under set semantics.

**Elimination of join anomalies.** By contradiction, assume that after applying Rules J.1–J.3, the resulting system still exhibits a join anomaly for the original pattern, i.e., evaluating $tm'_2$ produces duplicate RDF triples due to multiple bindings generating the same subject–predicate–object. Rule J.2 applies only when the subject attribute set of $tm_2$ (denoted $\texttt{SubjAttrs}(tm_2)$) forms a superkey of $J$; hence

$$\texttt{SubjAttrs}(tm_2) \to \texttt{Attrs}(J).$$

For any two tuples $\tau_1, \tau_2 \in J$, if $\tau_1[\texttt{SubjAttrs}(tm_2)] = \tau_2[\texttt{SubjAttrs}(tm_2)]$ then $\tau_1 = \tau_2$. Since $tm'_2$ constructs the subject from $\texttt{SubjAttrs}(tm_2)$ and uses $S_k$ as a projection of $J$ that retains exactly the attributes needed to generate the triples of $tm_2$, each subject in $tm'_2$ corresponds to at most one tuple in $S_k$. Hence, there cannot exist two distinct bindings producing the same subject together with the same predicate and object in $tm'_2$. This contradicts the assumption that duplicates are generated in $DIS'_G$. Thus, Rules J.1–J.3 eliminate join anomalies for the transformed pattern.

**Semantic equivalence.** We prove $DIS_G \equiv DIS'_G$ by contradiction. Assume that $G$ and $G'$ are not isomorphic up to blank node renaming. Then there exists an RDF triple $t$ such that either (i) $t \in G$ and $t \notin G'$, or (ii) $t \in G'$ and $t \notin G$ (modulo blank node renaming).

*Case (i): $t \in G$ and $t \notin G'$.* If $t$ is not generated by $tm_2$ (nor by any mapping assertion affected by the transformation of $(tm_1, tm_2)$), then Rules J.1–J.3 do not modify the corresponding triples maps, and $t$ is generated unchanged in $G'$, a contradiction. Otherwise, $t$ is generated by $tm_2$ in $DIS_G$ from a binding produced by executing the join between $S_i$ and $S_j$ under $\theta$. Thus, there exists a tuple $\tau \in J$ such that evaluating the triple patterns of $tm_2$ over $\tau$ produces $t$. By Rule J.1, $J$ is materialized (conceptually) and by Rule J.2, $S_k$ is defined as a projection of $J$ onto the attributes required by $tm_2$. Hence, the projection $\pi(\tau)$ of $\tau$ onto $\texttt{Attrs}(S_k)$ belongs to $S_k$. Rule J.3 defines $tm'_2$ over $S_k$ with the same triple patterns as $tm_2$ (restricted to the projected attributes), and without requiring a runtime join. Therefore, evaluating $tm'_2$ over $\pi(\tau)$ produces the same triple $t$ in $G'$, contradicting $t \notin G'$.

*Case (ii): $t \in G'$ and $t \notin G$.* If $t$ is generated by a triples map unaffected by Rules J.1–J.3, then it is also generated in $G$, contradicting $t \notin G$. Otherwise, $t$ is generated by $tm'_2$ from some tuple $\tau_k \in S_k$. By construction, $S_k$ is a projection of $J$, so there exists a tuple $\tau \in J$ such that $\tau_k = \pi(\tau)$. In $DIS_G$, executing the join $\bowtie_\theta$ yields $\tau$, and evaluating $tm_2$ over $\tau$ produces the same triple patterns (since $tm'_2$ preserves them). Hence, $t$ is generated in $G$, contradicting $t \notin G$.

Since both cases lead to contradictions, our assumption was false. Therefore, $G$ and $G'$ are isomorphic up to blank node renaming, i.e., $DIS_G \equiv DIS'_G$. □

*Mapping Refinement Rules Correctness.*

**Proposition 5** (Mapping Refinement Rules Correctness)**.** *Rules 4.1–4.3 preserve semantic equivalence and enforce dependency-consistent mapping structure.*

**Proof.** Let $DIS_G = (O, S, M)$ and $DIS'_G = (O, S', M')$ be data integration systems such as $DIS'_G$ is obtained after applying Rules 4.1–4.3. Let $G$ and $G'$ be the KGs generated by executing $DIS_G$ and $DIS'_G$, respectively, under set semantics.

**Dependency-consistent mapping structure.** We show that after applying Rules 4.1–4.3, each triples map in $M'$ satisfies the intended dependency-consistency constraints.

*Rule 4.1 (MRMA decomposition).* By contradiction, assume that after applying Rule 4.1 there exists a multi-source role mapping assertion (MRMA) in $M'$ that still implicitly combines multiple logical sources without an explicit join. Rule 4.1 applies to *each* MRMA and replaces it by separate triples maps with explicit join conditions, or, if needed, by a new logical source defined as the relational join of the involved sources. Hence, an MRMA without explicit join cannot remain in $M'$, contradicting the assumption.

*Rule 4.2 (Removal of non-prime / join-irrelevant attributes).* By contradiction, assume there exists a triples map $tm^\star \in M'$ that still contains in one of its mapping assertions an attribute $a$ that is neither a prime attribute of the logical source of $tm^\star$ nor used in any join condition required to evaluate an RMA. Rule 4.2 applies to *every* triples map and removes exactly such attributes. Therefore, $a$ cannot remain in $tm^\star$, contradicting the assumption.

*Rule 4.3 (Grouping by FD-closure of subject attributes).* Let $tm$ be any triples map in $M'$ with CMA subject attribute set $X_C = \texttt{SubjAttrs}(tm)$ and logical source dependencies $FD$. By contradiction, assume that $tm$ contains an AMA/RMA over some attribute $A$ such that $X_C \nrightarrow A$ under $FD$ (i.e., $A \notin \mathrm{cl}_{FD}(X_C)$). Rule 4.3 enforces that only assertions over attributes $A$ with $X_C \rightarrow A$ are grouped in $tm$. Hence, an assertion over such an $A$ cannot remain grouped in $tm$, contradicting the assumption. Therefore, in $M'$, each triples map contains only assertions within the functional dependency closure of its subject attributes, which ensures dependency-consistent mapping structure.

**Semantic equivalence.** We now prove $DIS_G \equiv DIS'_G$ by contradiction. Assume that $G$ and $G'$ are not isomorphic up to blank node renaming. Then there exists an RDF triple $t$ such that either (i) $t \in G$ and $t \notin G'$, or (ii) $t \in G'$ and $t \notin G$ (modulo blank node renaming).

*Case (i): $t \in G$ and $t \notin G'$.* Since $t \in G$, it is produced by some triples map $tm \in M$ under some binding of variables to attribute values from one or more logical sources.

- If $t$ is produced by a mapping assertion that is unchanged by Rules 4.1–4.3, then it remains present in $M'$, and the same binding exists in $DIS'_G$; hence $t \in G'$, contradicting $t \notin G'$.
- If $t$ is produced by an MRMA, then Rule 4.1 replaces that MRMA by (i) explicit joins across newly introduced triples maps and/or (ii) an equivalent logical source defined as a relational join. In both cases, the set of satisfying join bindings is preserved (materialized joins and runtime joins yield the same satisfying tuple pairs). Therefore, $t$ is generated in $G'$, contradicting $t \notin G'$.
- If $t$ depends on an attribute removed by Rule 4.2, then that attribute is neither prime nor required for join evaluation. Such an attribute cannot be required to produce a *semantically relevant* triple under the dependency-consistency assumptions of the mapping: it is not needed to uniquely identify the subject (non-prime) and does not affect join satisfaction (join-irrelevant). Hence, either $t$ was redundant (duplicate under non-unique bindings) or not functionally determined by the subject, i.e., it was not a valid consequence of the intended mapping semantics. In particular, if $t$ were semantically required, then the generating assertion would have to refer only to attributes preserved by Rule 4.2, contradicting the assumption that it depends on a removed attribute.

Thus, Case (i) cannot occur.

*Case (ii): $t \in G'$ and $t \notin G$.* Since $t \in G'$, it is produced by some triples map $tm' \in M'$.

- If $tm'$ corresponds to an unchanged triples map from $M$, then the same assertion and bindings exist in $DIS_G$, implying $t \in G$, contradicting $t \notin G$.
- If $tm'$ is introduced by Rule 4.1 (MRMA decomposition), then $tm'$ either evaluates an explicit join or reads from a materialized join source. In both cases, each binding used to generate $t$ corresponds to a binding that satisfied the original MRMA join condition. Hence, the original MRMA in $M$ also generated $t$, contradicting $t \notin G$.
- Rules 4.2–4.3 do not introduce new attributes or new value combinations; they only remove attributes (Rule 4.2) or redistribute assertions to triples maps whose subject closure determines their attributes (Rule 4.3). Therefore, they cannot enable the generation of a triple that was impossible in the original system. Hence, $t$ must have been generatable already in $DIS_G$, contradicting $t \notin G$.

Since both cases lead to contradictions, our assumption was false. Therefore, $G$ and $G'$ are isomorphic up to blank node renaming, i.e., $DIS_G \equiv DIS'_G$. □

### 4.4. Normalization algorithm

This section presents Algorithm 1 that transforms $DIS_G = (O, S, M)$ into a normalized $DIS_G^{\text{norm}} = (O, S', M')$ that satisfies TM-1NF, TM-2NF, TM-3NF, and TM-JNF. Algorithm 1 performs multiple transformations to eliminate anomalies that degrade performance or violate semantics in KG creation. It normalizes data sources based on relational normal forms using a *minimal cover set* of functional dependencies *FD*, and restructures mapping assertions by factoring in join patterns, redundancy, and semantic alignment. As a result, the output system is optimized for execution in any RML-compliant engine without the need for post-hoc duplicate removal or special join optimization.

**Algorithm 1** Normalize Data Integration System into TM-NF.

**Require:** Data Integration System $DIS_G = (O, S, M)$, Functional Dependencies $FD$ (Minimal Cover set)
**Ensure:** Normalized Data Integration System $DIS_G^{\text{norm}} = (O, S', M')$

```
Initialize S' ← ∅, M' ← ∅
for all source S_i ∈ S do
    Apply Rule 1.1 to transform S_i into atomic form (TM-1NF)
    Extract candidate keys, prime/non-prime attributes, and attribute closures using FD
    Apply Rules 2.1–2.3 to identify and decompose partial dependencies (TM-2NF)
    Apply Rules 3.1–3.3 to eliminate transitive dependencies and enforce superkey-based subjects (TM-3NF)
    Add the resulting sources to S'
end for
for all triples map tm ∈ M do
    if tm contains a Multi-Source Role Mapping Assertion (MRMA) then
        Apply Rule J.1 to compute the relational join of the input sources
        Apply Rule J.2 to check if the subject attribute set of tm is a superkey of the join result
        Apply Rule J.3 to construct a new triples map tm' over the materialized and projected join (TM-JNF)
    end if
    Apply Rules 4.1–4.3 to refine attribute and role mapping assertions according to FD closures
    Add all resulting triples maps (original or transformed) to M'
end for
return DIS_G^norm = (O, S', M')
```

### 4.5. Correctness of the normalization process

We now prove that Algorithm 1 preserves the semantics of the original data integration system. That is, given a data integration system $DIS_G = (O, S, M)$ and its normalized version $DIS_G^{\text{norm}} = (O, S', M')$, both systems generate KGs that are isomorphic up to blank node renaming when evaluated over the same data extensions.

**Theorem 1** (Correctness of the Normalization Algorithm). *Let $DIS_G = (O, S, M)$ be a data integration system, and let $DIS_G^{norm} = (O, S', M')$ be*

*the normalized system obtained through Algorithm 1. Assume that the set of functional dependencies defined over each data source $S_i \in S$ corresponds to a minimal cover. Then:*

$\phi(DIS_G) \cong \phi(DIS_G^{norm})$

**Proof (by contradiction).** Let $G = \phi(DIS_G)$ and $G^{\text{norm}} = \phi(DIS_G^{\text{norm}})$. Assume, for contradiction, that the two KGs are not isomorphic: $G \ncong G^{\text{norm}}$. Then there exists a triple $t = (s, p, o)$ such that either:

- $t \in G$ but no corresponding triple exists in $G^{\text{norm}}$ under any bijection between blank nodes, or
- $t \in G^{\text{norm}}$ but no corresponding triple exists in $G$ under any bijection between blank nodes.

However, Algorithm 1 is composed exclusively of transformation rules (TM-1NF, TM-2NF, TM-3NF, TM-JNF, and Mapping Refinement Rules), each of which has been proven (Propositions 1–5) to preserve RDF KG isomorphism. Under the assumption that the functional dependencies form a minimal cover:

- No determinant attributes required for subject construction are lost.
- No join bindings are altered.
- No functionally determined attribute values are removed.
- No new triple patterns are introduced.

Each transformation step individually preserves KG isomorphism. Since KG isomorphism is transitive, the composition of these transformations also preserves isomorphism. Thus, for every triple in $G$ there exists a corresponding triple in $G^{\text{norm}}$ (possibly differing only in blank node identifiers), and vice versa. This contradicts the assumption that $G \ncong G^{\text{norm}}$. Therefore,

$\phi(DIS_G) \cong \phi(DIS_G^{\text{norm}})$.

□

### 4.6. Robustness and structural properties of the normalization algorithm

Algorithm 1 exhibits the following structural robustness properties:

*Determinism.* Given a data integration system $DIS_G = (O, S, M)$ and a minimal cover of functional dependencies $FD$, Algorithm 1 is deterministic. That is, for fixed $(DIS_G, FD)$, the algorithm always produces the same normalized system $DIS_G^{\text{norm}} = (O, S', M')$. No stochastic choices, heuristic ranking, or cost-based search procedures are involved in the transformation process.

*Structure-Preserving Rewriting.* All normalization steps are structure-preserving rewritings of logical sources and mapping assertions. The transformations rely exclusively on:

- functional dependency reasoning,
- key identification and attribute closure computation,
- explicit restructuring of join conditions and mapping assertions.

No heuristic pruning or approximate optimization techniques are applied. Each transformation is syntactic and dependency-driven.

*Formal Semantic Guarantees.* The correctness of each class of transformation rules (TM-1NF, TM-2NF, TM-3NF, TM-JNF, and Mapping Refinement Rules) has been formally proven. Consequently, the normalization algorithm preserves semantic equivalence, defined as RDF graph isomorphism up to blank node renaming:

$\phi(DIS_G) \cong \phi(DIS_G^{\text{norm}})$.

*Data Independence.* The normalization process operates exclusively at the schema and mapping level. It does not rely on instance-level statistics, sampling, or data distribution assumptions. All transformations depend only on:

- the structure of the logical sources,
- the mapping definitions,
- the declared functional dependencies.

Therefore, the behavior of Algorithm 1 is independent of data size and value distributions, which makes the approach robust under scaling and heterogeneous data characteristics.

### 4.7. Complexity analysis

This section analyzes (i) the computational complexity of Algorithm 1 and (ii) its impact on the execution complexity of KG creation.

*Complexity of Algorithm 1.* Let $n$ be the number of data sources, $m$ the number of triples maps, $a$ the maximum number of attributes per source, and $|FD|$ the number of functional dependencies expressed in minimal cover form. Algorithm 1 performs structural transformations over $DIS_G = (O, S, M)$ without exploring combinatorial search spaces. The most expensive operations involve reasoning over functional dependencies, including candidate key identification and attribute closure computation. The computation of attribute closures under a set of functional dependencies can be performed in polynomial time using the standard closure algorithm (e.g., [24,46]).
Each normalization step (TM-1NF, TM-2NF, TM-3NF, TM-JNF, and mapping refinement) requires only inspection of attributes, functional dependencies, and join conditions. No exponential enumeration of attribute subsets is performed. Consequently, the overall normalization process runs in time polynomial in $n$, $m$, $a$, and $|FD|$.
Let the *size* of $DIS_G = (O, S, M)$ be

$$|DIS_G| = |S| + |M| + \sum_{S_i \in S} |X_i| + \sum_{S_i \in S} |FD_i|,$$

where $X_i$ is the attribute set of source $S_i$ and $FD_i$ is the (minimal-cover) set of functional dependencies defined over $S_i$. Therefore, Algorithm 1 terminates in time polynomial in $|DIS_G|$.

*Impact on KG Materialization Complexity.* While Algorithm 1 runs in polynomial time with respect to the size of $DIS_G = (O, S, M)$, its primary contribution lies in improving the computational behavior of KG materialization.
Let $T$ denote the total number of input tuples across all data sources. In non-normalized systems, triples maps may involve runtime joins whose intermediate result size depends on the cardinalities of the participating sources. If join conditions are not based on superkeys, such joins may generate intermediate bindings with unintended multiplicities. These multiplicities lead to duplicate triple generation and require explicit duplicate elimination during materialization.
Duplicate elimination in RML-compliant engines is typically implemented using in-memory data structures (e.g., hashing or buffering), incurring additional memory consumption proportional to the number of generated triples and increasing execution time due to repeated comparisons. Algorithm 1 mitigates these structural inefficiencies by:

- eliminating partial and transitive dependencies before materialization,
- enforcing superkey-based subject construction, ensuring unique entity bindings,
- and restructuring join patterns (TM-JNF) to avoid duplicate-generating configurations.

Although the worst-case asymptotic complexity of relational joins remains dependent on input data size, the normalization removes structural causes of unnecessary multiplicative blow-up. As a result, the number of intermediate bindings is reduced, leading to lower execution time

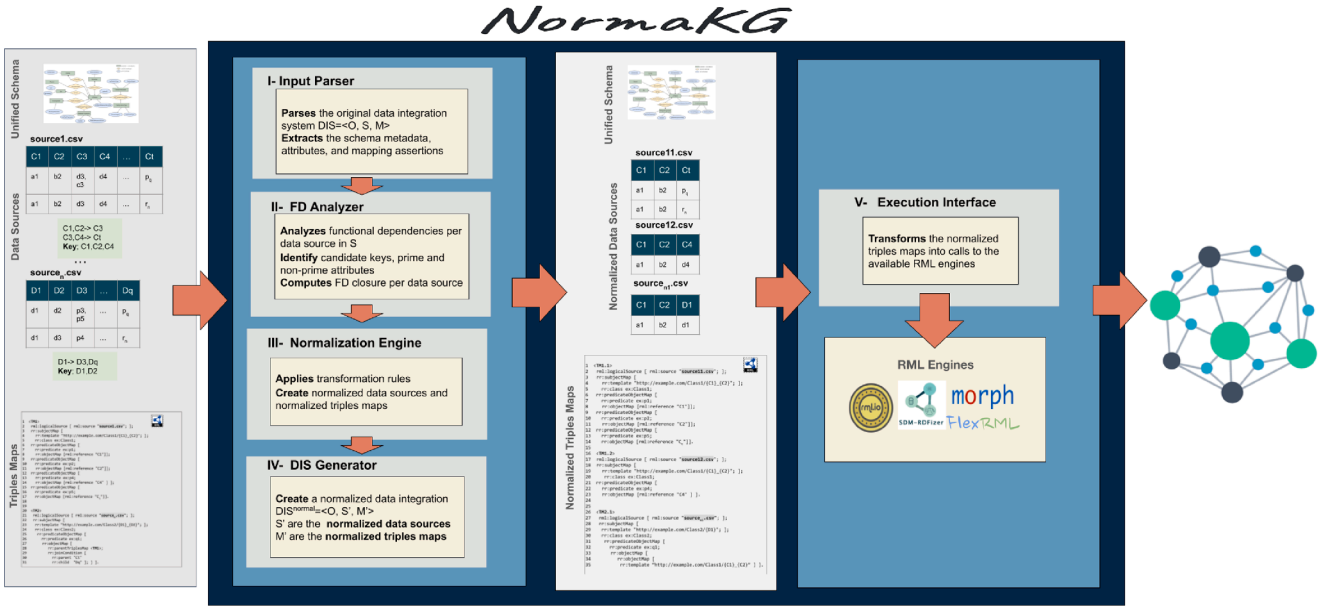



**Fig. 10. NormaKG Architecture: Normalizing Data Integration Systems for Efficient KG Creation.** The figure presents the architecture of NormaKG, a framework that transforms a data integration system $DIS_G = (O, S, M)$ into its normalized form $DIS_G^{\text{norm}} = (O, S', M')$. The pipeline includes five components: (I) the Input Parser extracts schema metadata and mapping logic; (II) the FD Analyzer identifies keys, FDs, and prime attributes; (III) the Normalization Engine applies transformation rules; (IV) the DIS Generator reconstructs the normalized system; and (V) the Execution Interface materializes the KG using external RML engines. Input/output examples illustrate how normalization improves data quality and mapping structure. This figure highlights how NormaKG eliminates anomalies and enables efficient, duplicate-free KG creation across RML engines.

and memory usage in practice, as demonstrated in Section 6. Importantly, these improvements are achieved without altering semantics, since the generated RDF KGs remain isomorphic.

## 5. NormaKG - A framework for KG creation from normalized DISs

### 5.1. Architecture of NormaKG

NormaKG is a framework designed to optimize KG creation by transforming a data integration system into its normalized form. The framework is **engine-agnostic**, i.e., it is independent of the RML implementations and can interface with any RML-compliant engine. NormaKG receives:

- A data integration system $DIS_G = (O, S, M)$, where $O$ is the ontology, $S$ is a set of data sources, and $M$ is a set of mapping assertions.
- A set of functional dependencies $FD$ defined over the attributes of the sources in $S$.

NormaKG relies on Algorithm 1 to produce a normalized $DIS_G^{\text{norm}} = (O, S', M')$, such that sources in $S'$ and triples maps in $M'$ conform TM-1NF, TM-2NF, TM-3NF, and TM-JNF. The resulting $DIS_G^{\text{norm}}$ can be executed in any standard RML engine without requiring runtime duplicate removal or complex join planning. This design ensures improved performance and scalability. Fig. 10 depicts the NormaKG architecture, comprising the following components:

- **Input Parser:** Parses the original data integration system $DIS_G = (O, S, M)$ and extracts:
  - the list of data sources $S$ along with their schema metadata (e.g., attribute names and data types),
  - the set of classes and properties defined in the ontology $O$, and
  - the mapping assertions from the triples maps in $M$, including subject templates, predicate-object maps, join conditions, and transformation functions.
- **FD Analyzer:** Takes the parsed source schemas and the user-defined functional dependencies *FD* (minimal cover) to:
  - infer candidate keys for each source,
  - identify prime and non-prime attributes, and
  - compute attribute closures for every $S_i \in S$ based on $FD$.
- **Normalization Engine:** Receives the mapping assertions, schema metadata, and functional dependency analysis results. It applies the normalization rules (TM-1NF to TM-JNF) to:
  - produce a set of normalized data sources $S'$, and
  - construct a corresponding set of normalized triples maps $M'$ that satisfy TM-NF.
- **DIS Generator:** Reassembles the normalized data integration system:

  $$DIS_G^{\text{norm}} = (O, S', M')$$

  by integrating the normalized sources and mapping assertions while preserving semantic consistency with the original ontology $O$.
- **Execution Interface:** Takes the normalized DIS and triggers the execution of $M'$ over $S'$ to generate the target KG $\mathcal{G}$. It interfaces with external KG creation engines (e.g., SDM-RDFizer, Morph-KGC), optionally adapting to engine-specific execution parameters or optimization strategies.

### 5.2. Properties and theoretical bounds

Let $DIS_G = (O, S, M)$ be a data integration system and $DIS_G^{\text{norm}} = (O, S', M')$ be its normalized form produced by the NormaKG framework. The following properties and bounds characterize the result of the normalization process:

1. **Dependency Preservation:** As established in Theorem 1, the normalized system preserves the dependency-driven semantics of the original DIS:

   $$\phi(DIS_G) \cong \phi(DIS_G^{\text{norm}})$$

2. **Normal Form Satisfaction:**
   - Each logical source $S_i' \in S'$ satisfies the *Third Normal Form (3NF)* under classical relational normalization theory.

- Each triples map $m' \in M'$ satisfies all four proposed normalization levels: **TM-1NF**, **TM-2NF**, **TM-3NF**, and **TM-JNF**.

3. **Cardinality Bounds:**
   - The number of logical sources may increase due to vertical decomposition required to satisfy TM-2NF and TM-3NF:

     $$|S'| \geq |S|$$

   - The number of triples maps may increase due to decomposition of join-heavy mappings and partitioning of functionally independent assertions:

     $$|M'| \geq |M|$$

   - Let $FD_i$ be the number of non-trivial functional dependencies identified in data source $S_i \in S$. Then the total number of logical sources in the normalized system is bounded by:

     $$|S'| \leq \sum_{S_i \in S} \left(1 + |FD_i|\right)$$

     Each FD may lead to a vertical fragmentation during TM-2NF or TM-3NF.
   - Let JoinDecomposition($m$) denote the number of join operations in mapping $m \in M$ requiring flattening via TM-JNF. Then the number of triples maps in $M'$ is bounded by:

     $$|M'| \leq \sum_{m \in M} (1 + \text{JoinDecomposition}(m))$$

     Each such join is replaced by a flattened source and a join-free triples map.
4. **Tractability and Scalability:** The base term 1 in the upper bounds for $|S'|$ and $|M'|$ accounts for the original source or mapping. The additive terms estimate the worst-case number of new sources or mappings introduced during normalization. Therefore, the growth in size of $S'$ and $M'$ is polynomial with respect to the number of FDs and join conditions in the original DIS. This ensures that the normalization process remains tractable and scalable for large data integration settings.

### 5.3. Duplicate-free execution in RML engines

We now show that normalized data integration systems generated by NormaKG do not require runtime duplicate elimination when executed in any RML engine.

**Theorem 2** (Duplicate-Free Execution). *Let $DIS_G^{norm} = (O, S', M')$ be the normalized system produced by Algorithm 1, where each data source $S'_i \in S'$ is in Third Normal Form (3NF), and each mapping $m' \in M'$ satisfies TM-1NF, TM-2NF, TM-3NF, and TM-JNF. Then, the evaluation of $M'$ over $S'$ by any RML-compliant engine produces a knowledge graph $\mathcal{G}$ that contains no duplicate RDF triples.*

**Proof (by contradiction).** Assume, for contradiction, that evaluating $DIS_G^{\text{norm}}$ using any RML engine generates duplicate triple bindings during materialization. Then there exists a triple $t = (s, p, o)$ such that $t$ is generated at least twice during evaluation. Let $\phi_{m'}(\mu)$ denote the set of RDF triples generated by evaluating a triples map $m' \in M'$ over an input tuple $\mu$. Then there exist two distinct tuples $\mu_1 \neq \mu_2$ (from one or more logical sources in $S'$) and a triples map $m' \in M'$ such that:

$$t \in \phi_{m'}(\mu_1) \cap \phi_{m'}(\mu_2).$$

That is, the same triple $t$ belongs to the sets of triples generated from both $\mu_1$ and $\mu_2$ by the same triples map $m'$.
Such a situation would require that two structurally distinct input tuples produce the same subject–predicate–object combination under evaluation. However, this cannot occur in $DIS_G^{\text{norm}}$ due to the following guarantees:

- **TM-3NF:** The subject of every triples map is generated from a superkey of the corresponding logical source. Therefore, if $\mu_1 \neq \mu_2$, then their subject bindings must differ.
- **TM-2NF and TM-3NF:** All object values used in attribute and role mapping assertions are functionally determined by the subject attributes. Hence, a given subject cannot yield conflicting or repeated object bindings unless functional dependencies are violated.
- **TM-JNF:** Join patterns are structurally materialized into flattened logical sources constructed from key–foreign key relationships. This prevents multiplicative join effects that could otherwise generate identical subject–object combinations from distinct intermediate bindings.
- **Relational 3NF in $S'$:** Each logical source $S'_i$ satisfies 3NF, eliminating redundancy caused by partial or transitive dependency violations.

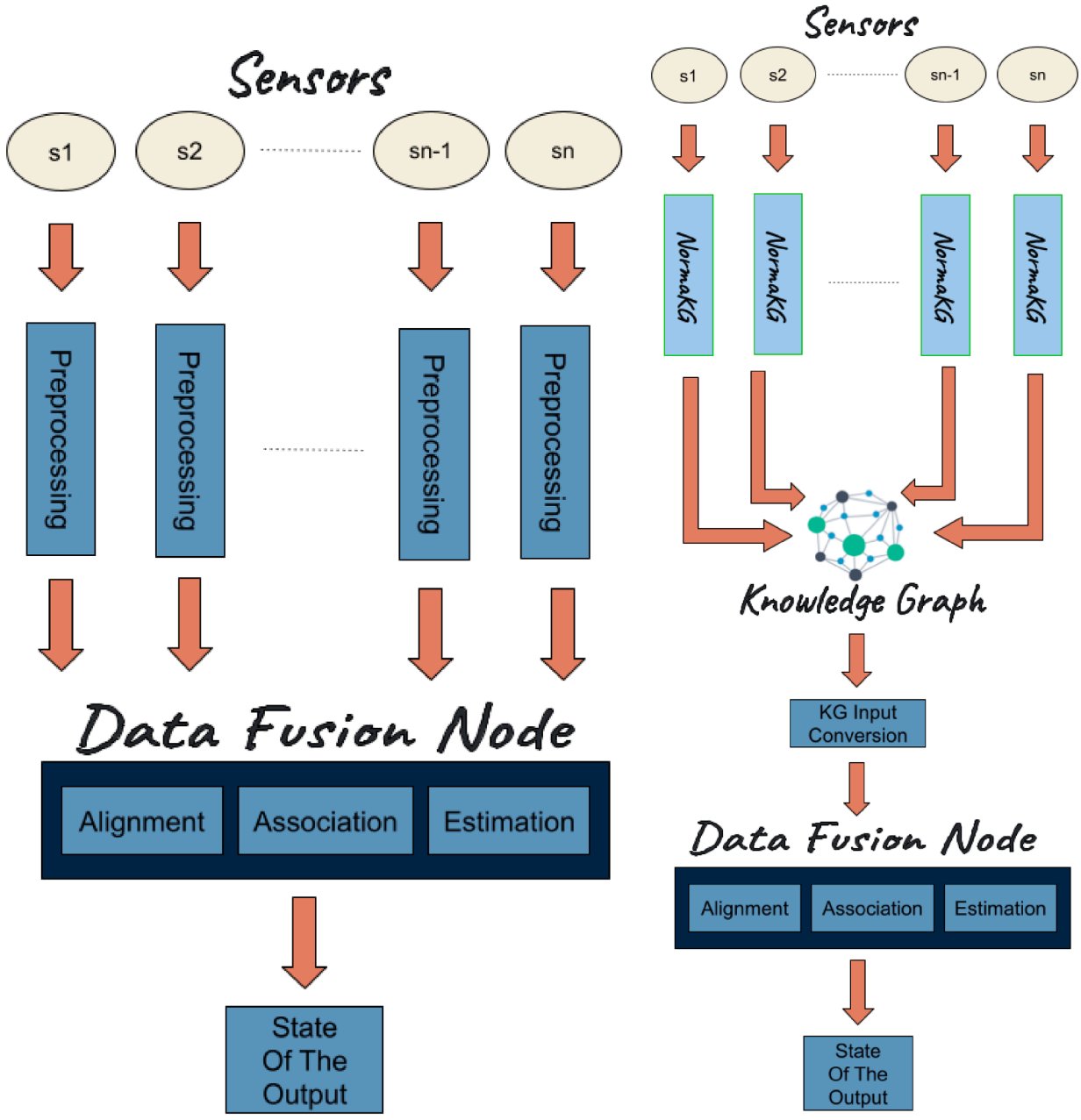
Consequently, no two distinct tuples $\mu_1$ and $\mu_2$ can generate the same triple $t$. This contradicts the assumption that duplicate triple bindings are produced during evaluation. Therefore, evaluation of $DIS_G^{\text{norm}}$ is duplicate-free. □

**Corollary 1** (No Duplicate Elimination). *Let $DIS_G^{norm} = (O, S', M')$ be the normalized DIS produced by Algorithm 1. Then, evaluation of $M'$ over $S'$ by any RML-compliant engine does not require runtime duplicate elimination to ensure correctness of the resulting RDF graph.*

**Proof.** By Theorem 2, evaluation of $DIS_G^{\text{norm}}$ does not generate duplicate triple *bindings* during materialization, i.e., no RDF triple is produced more than once by the evaluation process. Therefore, any runtime duplicate elimination performed by an RML engine would be superfluous for this input: it would not remove any additional triples, and skipping it does not change the resulting KG (under the set-based KG/RDF semantics adopted in this paper). □

### 5.4. NormaKG as an information fusion approach

NormaKG, as illustrated in Section 5, is a structural optimization approach that enhances the KG creation process through normalization-driven data integration. Its objective is to eliminate redundancy, unnecessary joins, and duplicate-generating dependencies while preserving semantic equivalence. Unlike classical information fusion methods, NormaKG does not propose a new fusion architecture. Instead, it operates as a preprocessing layer that optimizes structural data integration prior to fusion and learning stages.
According to the taxonomy presented by Castanedo [40], fusion architectures can be categorized as *centralized*, *decentralized*, or *distributed*. In centralized architectures, raw data from multiple sources are transmitted to a single fusion node responsible for preprocessing, alignment, and state estimation. In decentralized architectures, multiple fusion nodes independently process local data while exchanging intermediate information. In distributed architectures, sensors perform local processing and transmit partial results to higher-level fusion components.
NormaKG is architecture-agnostic and can be integrated into any of these settings. In centralized architectures (see Fig. 11a), NormaKG operates at the data-level preprocessing stage, consolidating heterogeneous raw sources into a normalized KG before they are forwarded to the fusion node. In decentralized and distributed architectures, NormaKG can be applied locally at each fusion node or sensor-processing unit to ensure structural consistency prior to aggregation. In all cases, NormaKG improves the integrity of intermediate representations without altering the downstream fusion logic. Beyond data-level preprocessing, NormaKG also supports *feature-level fusion*. Feature-level fusion combines structured attributes extracted from multiple sources into a unified representation. However, heterogeneity in schemas, naming conventions, granularity, and dependency structures can introduce redundancy and inconsistency. Without normalization, concatenation or aggregation of features may inflate dimensionality and propagate structural noise into learning models.
NormaKG addresses these issues by enforcing semantic normalization




(a) Basic centralized data fusion architecture.

(b) Centralized data fusion architecture with NormaKG.

**Fig. 11. Data Fusion Architecture.** This figure presents two data fusion architectures. (a) shows a basic version of a centralized data fusion architecture (as presented in Castanedo [40]), in which data from various sensors is preprocessed before being sent to the architecture, which then generates the output. (b) presents a centralized data fusion architecture that utilizes NormaKG. In this architecture, NormaKG is employed during preprocessing. By applying NormaKG, redundancies in the data are eliminated, enabling the data to be unified as a KG. The KG is then given to the architecture to generate the output.

before fusion. By exploiting functional dependencies and eliminating structural anomalies, it consolidates heterogeneous features into a single KG that acts as a normalized feature space. In this representation, semantically equivalent attributes are explicitly aligned and merged, rather than implicitly reconciled by downstream algorithms. This reduces redundancy, improves structural coherence, and enhances robustness of subsequent fusion steps.

Importantly, NormaKG does not replace state estimation, situation assessment, or decision-level aggregation mechanisms. It does not generate fused outputs directly. Instead, it strengthens the structural foundations of intermediate representations used by these processes. Thus, NormaKG complements existing data-level, feature-level, and decision-level fusion strategies by ensuring that the fused knowledge representation is structurally consistent, dependency-compliant, and duplicate-free prior to downstream computation.

## 6. Empirical evaluation

This study aims to determine the impact of the normalization process on the KG creation process. For that reason, the RML-compliant engines RMLMapper, SDM-RDFizer, Morph-KGC, and FlexRML are used. The empirical evaluation seeks to answer the following research ques-

tions: **RQ1)** How does data normalization affect the performance of the state-of-the-art RML-compliant engines during KG creation? **RQ2)** What is the impact of the mapping assertions and volume of the data sources on memory consumed by KG creation engines? **RQ3)** What is the impact on the execution time of the execution of mapping assertions when applying normalization to the data source?

This section defines the experimental configuration to provide a comprehensive overview of the empirical assessment and the observed results. This configuration illustrates the chosen benchmark, metrics, and engines, along with the description of the environment used to evaluate the performance of state-of-the-art RML engines. Each experimental configuration is repeated five times, and the average execution time and maximum memory usage are reported as the outcome. The results are analyzed to determine the benefits and drawbacks of using data fragmentation during the KG creation process.

### 6.1. Experimental configuration

**Benchmarks.** Experiments are performed over datasets from **GTFS-Madrid-Bench**, **SDM-Genomic-Datasets**, and **KROWN**. Therefore, our experiments cover many parameters that affect the KG creation task, i.e., dataset size, TM type and complexity, selectivity of the results, and types of joins between the TMs.

**GTFS-Madrid-Bench** [16]: This benchmark enables the generation of different configurations that affect the characteristics of creating a KG. The dataset presents four scaling factors by default (1-csv, 10-csv, 100-csv, and 1000-csv) for the data sources. The scale value influences the size of the resulting KG. For example, the KG generated from 10-csv is ten times bigger than that generated from 1-csv. We consider mapping rules comprised of 13 TMs, with 13 concept MAs, 12 multi-source role MAs, 67 attribute MAs, and seven single-source role MAs involving ten data sources.

**SDM-Genomic-Datasets** [26]: This benchmark is created by randomly sampling data records from somatic mutation data collected from COSMIC.[2] SDM-Genomic-Datasets include eight different logical sources of various sizes, including 10k, 100k, 1M, and 5M rows. For every pair of sources of the same size, they differ in the percentage of data duplicate rate, which can be 25% or 75%, where each duplicate value is repeated 20 times. For example, a 10k logical source with a 25% data duplicate rate has 75% duplicate-free records (i.e., 7500 rows), and the rest of the 25% records (i.e., 2500 rows) correspond to 125 different records that are duplicated 20 times; in total, there are 7625 unique values. The SDM-Genomic-Datasets offer four different TMs configurations. **Conf1:** Set of three TMs, with three concept MAs, 12 attribute MAs, and six referenced-source role MAs. **Conf2:** Set of six TMs, with six concept MAs and five multi-source role MAs. Recreates a five-star join where five TMs refer to the same parent TM. **Conf3:** Set of five TMs, with five concept MAs, 12 attribute MAs, six referenced-source role MAs, and two multi-source role MAs. **Conf4:** Combination of Conf2 and Conf3.

**KROWN** [27]: This benchmark provides a data generator that allows a user to generate various benchmark scenarios with multiple scaling parameters, which a user can configure. The data generator can create synthetic data sources covering a wide range of test cases like different duplicate rates, empty values, TM with varying numbers of properties, joins with multiple join conditions with various levels of complexity, etc. For this empirical evaluation, three sets of data sources were created. The first set comprises two CSV files of 1000 rows and ten columns each. A Cartesian product is executed between the two data sources, resulting in a 1M-row data source with 20 columns. The second set comprises two CSV files, one with 10k rows and ten columns, and the other with 1000 rows and ten columns. A Cartesian product is applied to create a single data source that has 10M rows with 20 columns. The final set contains three CSV files; one of the files has 1000 rows and ten columns, and the other two have 100 rows and five columns. As in the other two sets, a Cartesian product creates 10M rows with 20 columns. Three experimental TMs configurations are used. **Conf1:** Set of two TMs, with two concept MAs, 20 attribute MAs, and two multi-source role MAs. **Conf1:** Set of two TMs, with two concept MAs, 20 attribute MAs, and five multi-source role MAs. **Conf1:** Set of three TMs, with three concept MAs, 20 attribute MAs, and three multi-source role MAs.

In total, 552 experimental configurations were executed.

**RML Engines.** These are the KG creation engines used for the experiments: RMLMapper v4.12,[3] Morph-KGC v2.6.2,[4] SDM-RDFizer v4.7.1.9,[5] and FlexRML v1.0.0.[6] RML-Planner is available on GitHub [7]. The proposed solution, NormaKG, is publicly available.[8]

**Metrics.** *Execution time* is the considered metric to determine the performance of the RML engines. Execution time is the elapsed time required to normalize the raw data and MAs and generate the intended KG. It is measured as the absolute wall-clock system time, as reported by the `time` command of the Linux operating system. The timeout is at 5 h. *Memory usage* measures the maximum memory used to generate a KG; it is measured using the Python library `malloc`. The `malloc` library measures the memory usage in Kilobytes; we convert the result into Megabytes. Each experimental configuration is executed five times, and the average Execution time and Memory usage are reported. Each experimental configuration is evaluated following six baseline approaches: **Baseline.** The KG is generated with no data normalization techniques. **Baseline + NormaKG.** The KG is generated by applying the proposed solution NormaKG. **MapSDI.** The KG is generated by applying the projection technique proposed in MapSDI. **MapSDI + NormaKG.** The KG is generated by applying both MapSDI and NormaKG projection methods. **RML-Planner**. The KG is generated by applying the mapping partitiong and execution plan proposed by the RML-Planner. **RML-Planner + NormaKG**. The KG is generated by applying the NormaKG projection and mapping partitioning technique.

The boxology pattern representation of these baseline approaches can be seen in Fig. 12. The experiments are executed in an Intel(R) Xeon(R) equipped with a Platinum 8160 CPU @ 2.10GHz 24 cores, 754 GB RAM, and with the O.S. Ubuntu 16.04 LTS.

### 6.2. Experimental results for SDM-genomic-datasets

This subsection presents the results of evaluating KG creation engines using the test cases from SDM-Genomic-Datasets. These test cases are designed to assess the engines' ability to manage key aspects of the KG creation process, including handling large, diverse data sources, removing duplicates, and executing complex joins. To this end, experiments were conducted with two data sizes (1M and 10M rows), two duplicate rates (25% and 75%), and multiple mapping configurations of varying complexity. This subsection addresses a total of 384 experimental test cases. Fig. 13 illustrates the evolution of the DIS in this benchmark, increasing from eight triples maps and corresponding data sources to 25 as the normalization rules were applied. Tables 3–6 summarize the results.

These tables show that all engines improved execution time and memory usage, mainly when applied data normalization methods. Notably, RMLMapper, which previously failed to complete several test cases, successfully executed them when applying NormaKG. In particular, RMLMapper can complete the creation process for more complex cases (i.e., Conf3 and Conf4) when used in conjunction with both NormaKG and RMLMapper. However, RMLMapper still experienced frequent

[2] https://cancer.sanger.ac.uk/cosmic GRCh37, version90, released August 2019

[3] https://github.com/RMLio/rmlmapper-java
[4] https://github.com/morph-kgc/morph-kgc
[5] https://github.com/SDM-TIB/SDM-RDFizer
[6] https://github.com/wintechis/flex-rml
[7] https://github.com/SDM-TIB/RML-Planner
[8] https://github.com/SDM-TIB/NormaKG

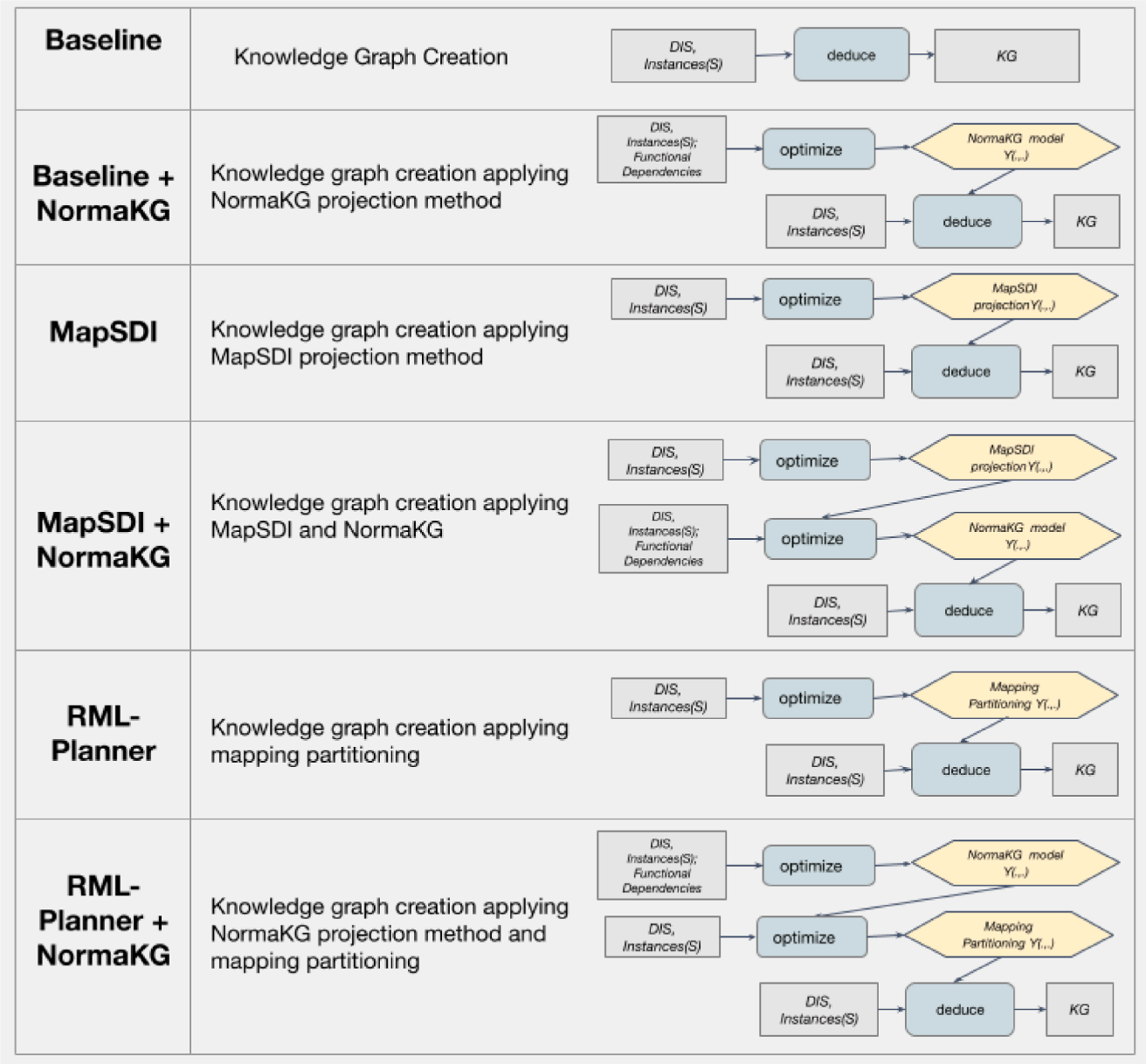



**Fig. 12. Design patterns for the KG creation approaches.** Boxology representation of the baseline approaches. This figure illustrates the design patterns of six KG creation pipelines. **Baseline** depicts the standard approach followed by most KG creation engines, where the DIS is directly provided to the engine, which then constructs the KG. **Baseline** + **NormaKG** introduces a normalization-based method that uses both the DIS and its FDs to normalize the data. The resulting normalized DIS is then provided to the engines for KG construction. **MapSDI** represents a projection-based approach in which unnecessary attributes and duplicate rows are removed from the DIS before being passed to the engines. **MapSDI** + **NormaKG** represents a combined approach that applies both projection methods (MapSDI and NormaKG). **RML-Planner** presents a mapping partitioning method in which mappings are grouped by data source, and an execution plan is then defined based on intra- and inter-mapping relations. **RML-Planner** + **NormaKG** represents a combined approach that applies the proposed solution NormaKG for data normalization and the mapping partitioning from RML-Planner.

timeouts, primarily due to its resource-intensive approach to duplicate removal and join execution.

The rest of the subsection will provide a detailed analysis of the performance of each KG creation engine.

SDM-RDFizer shows improved performance with NormaKG, MapSDI, RML-Planner, and the combination of these methods (i.e., MapSDI + NormaKG and RML-Planner + NormaKG). The MapSDI approach significantly reduces the execution time compared to the execution of the baseline mapping and data sources. This improvement is primarily due to the simplified TMs and the reduced size of the projected data sources, which lowers memory usage. RML-Planner also provides savings in terms of execution time, but at the cost of higher memory usage. This approach partitions the mapping into sub-mappings by data source, executes them in parallel, and then combines the resulting KGs into a single KG. This process is very memory intensive, especially in cases where it results in a large KG, like in the test cases with 10M rows and 25% duplicate rate.

When applying the NormaKG to the baseline configuration of the test cases, it presents the highest amount of savings in terms of memory usage, with savings of up to two orders of magnitude. This improvement results from isolating attributes that are not functionally dependent on the `rr:subjectMap` while retaining those necessary for KG generation without duplicate removal. When applying NormaKG alongside RML-Planner resulted in the shortest execution times, with

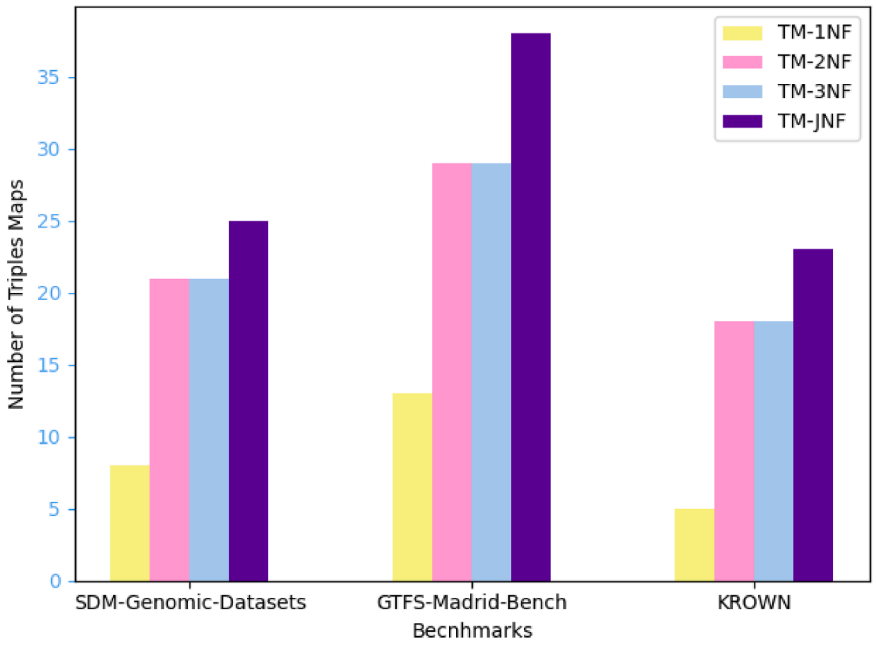



**Fig. 13. DIS Normalization.** Evolution of the benchmark DISs as normalization rules are applied. This figure illustrates how the number of triples maps—and, by extension, their associated data sources—changes as each normalization rule is applied. Since all benchmarks use valid CSV files as input, the TM-1NF stage reflects the original DIS configuration for each benchmark. Conversely, the TM-JNF stage represents the fully normalized DIS.

**Table 3**
Results of test cases on SDM-Genomic-Datasets with 1M and 25% duplicate rate. Best results in **bold**.

| Models | Baseline | SDM-RDFizer Time | SDM-RDFizer Memory | Morph-KGC Time | Morph-KGC Memory | RMLMapper Time | RMLMapper Memory | FlexRML Time | FlexRML Memory |
|---|---|---|---|---|---|---|---|---|---|
| | | | | **Conf1 + 1M rows** | | | | | |
| **Baseline** | **Original** | 360.18 sec | 1405.00 MB | 15.64 sec | 1001.612 MB | TimeOut | 39965.21 MB ≤ | 53.20 sec | 569.73 MB |
| | **NormaKG** | 261.45 sec | **107.03 MB** | **11.98 sec** | **938.86 MB** | 65.00 sec | 19902.34 MB | 35.14 sec | **4.66 MB** |
| **MapSDI** | **Original** | 344.51 sec | 1276.72 MB | 12.67 sec | 976.66 MB | TimeOut | 30229.65 MB ≤ | 39.07 sec | 568.05 MB |
| | **NormaKG** | 196.79 sec | 710.15 MB | 12.66 sec | 943.54 MB | 57.04 sec | 16780.01 MB | 33.83 sec | 4.68 MB |
| **RML-Planner** | **Original** | 209.83 sec | 2271.48 MB | 17.94 sec | 989.89 MB | TimeOut | 10562.89 MB ≤ | 46.80 sec | 568.69 MB |
| | **NormaKG** | **61.17 sec** | 2586.96 MB | 19.68 sec | 2597.14 MB | **28.82 sec** | **8430.09 MB** | **23.14 sec** | 59.72 MB |
| | | | | **Conf2 + 1M rows** | | | | | |
| **Baseline** | **Original** | 586.44 sec | 5483.80 MB | 158.44 sec | 18469.95 MB | TimeOut | 24326.98 MB ≤ | 3060.55 sec | 2163.93 MB |
| | **NormaKG** | **125.45 sec** | **190.89 MB** | **131.25 sec** | **18049.58 MB** | TimeOut | 9417.63 MB ≤ | 63.29 sec | **10.73 MB** |
| **MapSDI** | **Original** | 195.07 sec | 3553.38 MB | 132.28 sec | 18372.13 MB | TimeOut | 18423.24 MB ≤ | 90.25 sec | 1908.24 MB |
| | **NormaKG** | 128.75 sec | 1268.88 MB | 129.05 sec | 18150.06 MB | TimeOut | 16523.24 MB ≤ | 75.30 sec | 10.65 MB |
| **RML-Planner** | **Original** | 185.10 sec | 8942.61 MB | 147.79 sec | 18199.60 MB | TimeOut | 25887.63 MB ≤ | 1070.97 sec | 2498.30 MB |
| | **NormaKG** | 144.08 sec | 8215.42 MB | 132.47 sec | 18049.48 MB | TimeOut | 23698.76 MB ≤ | **56.19 sec** | 60.04 MB |
| | | | | **Conf3 + 1M rows** | | | | | |
| **Baseline** | **Original** | 554.80 sec | 4721.04 MB | 205.13 sec | 19972.11 MB | TimeOut | 28596.21 MB ≤ | 1542.65 sec | 2449.14 MB |
| | **NormaKG** | 389.52 sec | **135.69 MB** | **112.52 sec** | **18138.36 MB** | TimeOut | 3577.86 MB ≤ | 89.03 sec | **9.44 MB** |
| **MapSDI** | **Original** | 388.19 sec | 3761.33 MB | 179.39 sec | 18621.27 MB | TimeOut | 17652.32 MB ≤ | 120.12 sec | 2257.38 MB |
| | **NormaKG** | 291.04 sec | 635.05 MB | 168.35 sec | 19587.78 MB | TimeOut | 4692.72 MB ≤ | 99.09 sec | 10.83 MB |
| **RML-Planner** | **Original** | 234.19 sec | 10394.28 MB | 168.35 sec | 19587.78 MB | TimeOut | 27112.39 MB ≤ | 971.95 sec | 1875.31 MB |
| | **NormaKG** | **151.48 sec** | 9666.97 MB | 141.93 sec | 18049.67 MB | **278.24 sec** | **15975.36 MB** | **56.12 sec** | 59.85 MB |
| | | | | **Conf4 + 1M rows** | | | | | |
| **Baseline** | **Original** | 902.52 sec | 5780.99 MB | 214.46 sec | 20103.44 MB | TimeOut | 46998.78 MB ≤ | 3092.33 sec | 2582.94 MB |
| | **NormaKG** | 418.36 sec | **194.63 MB** | **113.37 sec** | **18197.54 MB** | TimeOut | 23749.54 MB ≤ | 90.95 sec | **10.86 MB** |
| **MapSDI** | **Original** | 434.23 sec | 3791.64 MB | 208.5 sec | 18786.4 MB | TimeOut | 38521.32 MB ≤ | 122.32 sec | 2363.10 MB |
| | **NormaKG** | 301.12 sec | 635.29 MB | 182.13 sec | 19774.18 MB | TimeOut | 25963.67 MB ≤ | 104.56 sec | 11.09 MB |
| **RML-Planner** | **Original** | 243.30 sec | 11070.22 MB | 151.06 sec | 18199.44 MB | TimeOut | 48957.12 MB ≤ | 1013.48 sec | 1875.45 MB |
| | **NormaKG** | **170.07 sec** | 10342.93 MB | 143.83 sec | 18054.80 MB | **338.06 sec** | **20046.70 MB** | **80.11 sec** | 60.03 MB |

savings of one order of magnitude. These saving are thanks to the parallel execution of the mapping partitions provided by RML-Planner. The only exception was Conf2 with 10M rows and 25% duplicate rate. This case presented a longer execution time than only applying NormaKG to the baseline. This increase in execution time is attributed to the process of combining the KG chunks, since the number of chunks is large, which lengthens the combination process.

The method of removing duplicates that SDM-RDFizer uses involves storing all generated triples and comparing each new triple with those stored, which consumes a significant amount of memory. By eliminating the need for duplicate removal, NormaKG reduces memory consumption to the levels required for the tool's core functionality.

Morph-KGC demonstrates reduced execution time when NormaKG, MapSDI, RML-Planner, and the combination of these methods, with

**Table 4**
Results of test cases on SDM-Genomic-Datasets with 1M and 75% duplicate rate. Best results in **bold**.

| Models | Baseline | SDM-RDFizer | | Morph-KGC | | RMLMapper | | FlexRML | |
|---|---|---|---|---|---|---|---|---|---|
| | | Time | Memory | Time | Memory | Time | Memory | Time | Memory |
| | | | | | **Conf1 + 1M rows** | | | | |
| **Baseline** | **Original** | 333.87 sec | 558.43 MB | 11.99 sec | 525.24 MB | TimeOut | 32658.11 MB ≤ | 45.96 sec | 235.53 MB |
| | **NormaKG** | 95.67 sec | **106.89 MB** | **5.91 sec** | **442.44 MB** | 26.92 sec | 11548.68 MB | 13.99 sec | **4.76 MB** |
| **MapSDI** | **Original** | 141.93 sec | 544.14 MB | 5.96 sec | 436.74 MB | TimeOut | 22348.25 MB ≤ | 14.86 sec | 234.99 MB |
| | **NormaKG** | 80.20 sec | 330.42 MB | 5.91 sec | 447.00 MB | 24.01 sec | 8574.40 MB | 13.34 sec | 4.63 MB |
| **RML-Planner** | **Original** | 85.22 sec | 940.83 MB | 12.85 sec | 540.96 MB | TimeOut | 30897.03 MB ≤ | 48.33 sec | 235.12 MB |
| | **NormaKG** | **24.92 sec** | 1013.20 MB | 9.99 sec | 1017.28 MB | **13.25 sec** | **4544.19 MB** | **5.06 sec** | 59.83 MB |
| | | | | | **Conf2 + 1M rows** | | | | |
| **Baseline** | **Original** | 418.51 sec | 2399.42 MB | 65.57 sec | 6570.22 MB | TimeOut | 37885.32 MB ≤ | 2971.93 sec | 978.69 MB |
| | **NormaKG** | 53.50 sec | **183.77 MB** | 47.32 sec | 6388.49 MB | TimeOut | 13882.26 MB ≤ | 27.19 sec | **10.05 MB** |
| **MapSDI** | **Original** | 82.94 sec | 1888.42 MB | 47.37 sec | 6498.49 MB | TimeOut | 28477.21 MB ≤ | 35.16 sec | 666.72 MB |
| | **NormaKG** | 63.16 sec | 1212.65 MB | 50.27 sec | 6869.07 MB | TimeOut | 11269.74 MB ≤ | 31.66 sec | 10.20 MB |
| **RML-Planner** | **Original** | 82.71 sec | 3339.25 MB | 53.14 sec | 6456.12 MB | TimeOut | 35997.01 MB ≤ | 1097.25 sec | 748.36 MB |
| | **NormaKG** | **53.18 sec** | 3045.99 MB | **41.68 sec** | **6388.14 MB** | TimeOut | 28614.58 MB ≤ | **22.05 sec** | 59.81 MB |
| | | | | | **Conf3 + 1M rows** | | | | |
| **Baseline** | **Original** | 492.65 sec | 1673.63 MB | 65.79 sec | 7285.02 MB | TimeOut | 19523.21 MB ≤ | 1537.45 sec | 998.79 MB |
| | **NormaKG** | 152.58 sec | **130.96 MB** | 42.77 sec | 6455.16 MB | 11676.00 sec | 3577.86 MB | 32.82 sec | **8.81 MB** |
| **MapSDI** | **Original** | 148.3 sec | 1247.22 MB | 65.09 sec | 6692.99 MB | TimeOut | 11213.54 MB ≤ | 42.54 sec | 879.44 MB |
| | **NormaKG** | 118.26 sec | 321.79 MB | 56.71 sec | 7079.27 MB | 10915.36 sec | **3364.23 MB** | 37.76 sec | 10.00 MB |
| **RML-Planner** | **Original** | 91.93 sec | 3771.13 MB | 60.31 sec | 6456.26 MB | TimeOut | 21188.74 MB ≤ | 917.56 sec | 1189.74 MB |
| | **NormaKG** | **57.95 sec** | 3771.13 MB | **42.52 sec** | **6388.24 MB** | **92.96 sec** | 6619.07 MB | **31.86 sec** | 59.87 MB |
| | | | | | **Conf4 + 1M rows** | | | | |
| **Baseline** | **Original** | 744.08 sec | 2781.71 MB | 76.27 sec | 7251.14 MB | TimeOut | 34630.01 MB ≤ | 3158.71 sec | 1080.36 MB |
| | **NormaKG** | 173.07 sec | **184.75 MB** | **43.21 sec** | 6473.05 MB | 17335.00 sec | **8221.85 MB** | **36.02 sec** | **10.14 MB** |
| **MapSDI** | **Original** | 176.70 sec | 1980.78 MB | 71.95 sec | 6707.14 MB | TimeOut | 21113.89 MB ≤ | 47.94 sec | 945.48 MB |
| | **NormaKG** | 125.37 sec | 322.44 MB | 60.41 sec | 7719.26 MB | 16822.47 sec | 7812.98 MB | 39.78 sec | 10.27 MB |
| **RML-Planner** | **Original** | 98.31 sec | 4179.07 MB | 59.17 sec | 6456.24 MB | TimeOut | 32174.34 MB ≤ | 1031.54 sec | 671.27 MB |
| | **NormaKG** | **60.41 sec** | 3885.78 MB | 58.19 sec | **6372.10 MB** | **130.10 sec** | 11069.54 MB | 74.49 sec | 59.81 MB |

**Table 5**
Results of test cases on SDM-Genomic-Datasets with 10M and 25% duplicate rate. Best results in **bold**.

| Models | Baseline | SDM-RDFizer | | Morph-KGC | | RMLMapper | | FlexRML | |
|---|---|---|---|---|---|---|---|---|---|
| | | Time | Memory | Time | Memory | Time | Memory | Time | Memory |
| | | | | | **Conf1 + 10M rows** | | | | |
| **Baseline** | **Original** | 3514.40 sec | 10708.64 MB | 150.96 sec | 8224.71 MB | TimeOut | OutOfMemory | 522.99 sec | 5109.69 MB |
| | **NormaKG** | 2646.84 sec | **107.46 MB** | 108.71 sec | 8080.07 MB | TimeOut | OutOfMemory | 252.82 sec | **4.68 MB** |
| **MapSDI** | **Original** | 3207.11 sec | 9933.39 MB | 111.99 sec | 8180.58 MB | TimeOut | OutOfMemory | 311.47 sec | 3636.44 MB |
| | **NormaKG** | 1631.33 sec | 5238.36 MB | 81.92 sec | 7280.56 MB | TimeOut | OutOfMemory | 293.86 sec | 4.70 MB |
| **RML-Planner** | **Original** | 1930.52 sec | 23720.43 MB | 124.93 sec | 7661.66 MB | TimeOut | 31897.03 MB ≤ | 531.43 sec | 5099.99 MB |
| | **NormaKG** | **849.19 sec** | 19162.38 MB | **77.13 sec** | **7198.05 MB** | **602.87 sec** | **23720.66 MB** | **71.30 sec** | 65.24 MB |
| | | | | | **Conf2 + 10M rows** | | | | |
| **Baseline** | **Original** | 5924.12 sec | 39604.83 MB | 2769.22 sec | 120963.98 MB | TimeOut | 151411.01 MB ≤ | TimeOut | 109530.00 MB ≤ |
| | **NormaKG** | **784.32 sec** | **194.44 MB** | **2408.54 sec** | **117720.09 MB** | TimeOut | 31093.40 MB ≤ | **416.27 sec** | **11.04 MB** |
| **MapSDI** | **Original** | 1225.31 sec | 27325.60 MB | 2477.06 sec | 120422.54 MB | TimeOut | 45655.20 MB ≤ | 595.21 sec | 11130.43 MB |
| | **NormaKG** | 756.99 sec | 11336.11 MB | 672.20 sec | 117150.51 MB | TimeOut | 47345.12 MB ≤ | 463.76 sec | 11.18 MB |
| **RML-Planner** | **Original** | 3056.56 sec | 52024.96 MB | 2371.42 sec | 117190.41 MB | TimeOut | 100623.78 MB ≤ | TimeOut | 13578.32 MB ≤ |
| | **NormaKG** | 2780.47 sec | 55342.53 MB | 2087.03 sec | 117177.27 MB | TimeOut | 96013.89 MB ≤ | 606.32 sec | 61.36 MB |
| | | | | | **Conf3 + 10M rows** | | | | |
| **Baseline** | **Original** | 5685.28 sec | 35903.94 MB | 4217.00 sec | 134700.82 MB | TimeOut | 38654.02 MB ≤ | TimeOut | 187431.02 MB ≤ |
| | **NormaKG** | 3324.54 sec | **137.05 MB** | **953.17 sec** | **118178.37 MB** | TimeOut | 14185.52 MB ≤ | **636.83 sec** | **9.56 MB** |
| **MapSDI** | **Original** | 3533.03 sec | 29033.52 MB | 1936.03 sec | 122633.62 MB | TimeOut | 27136.57 MB ≤ | 994.25 sec | 166643.27 MB |
| | **NormaKG** | 2241.51 sec | 4992.82 MB | 1742.17 sec | 128084.02 MB | TimeOut | 20122.68 MB ≤ | 699.53 sec | 11.31 MB |
| **RML-Planner** | **Original** | 2307.42 sec | 67623.09 MB | 1249.10 sec | 118179.00 MB | TimeOut | 40178.32 MB ≤ | TimeOut | 178937.84 MB ≤ |
| | **NormaKG** | **1398.56 sec** | 71915.96 MB | 1179.19 sec | 118069.70 MB | TimeOut | OutOfMemory | 696.36 sec | 59.79 MB |
| | | | | | **Conf4 + 10M rows** | | | | |
| **Baseline** | **Original** | 9733.07 sec | 47084.47 MB | 4682.00 sec | 137928.94 MB | TimeOut | 43126.97 MB ≤ | TimeOut | 23015.49 MB ≤ |
| | **NormaKG** | 3540.26 sec | **198.14 MB** | **1007.33 sec** | **118179.16 MB** | TimeOut | 16059.36 MB ≤ | **643.30 sec** | **11.07 MB** |
| **MapSDI** | **Original** | 3705.44 sec | 29065.04 MB | 4501.00 sec | 123680.24 MB | TimeOut | 31552.98 MB ≤ | 985.40 sec | 16877.17 MB |
| | **NormaKG** | 2177.89 sec | 4795.77 MB | 700.94 sec | 117152.21 MB | TimeOut | 18980.97 MB ≤ | 724.48 sec | 10.95 MB |
| **RML-Planner** | **Original** | 3868.69 sec | 74544.84 MB | 2882.32 sec | 117190.50 MB | TimeOut | 47561.87 MB ≤ | TimeOut | 27896.60 MB ≤ |
| | **NormaKG** | **3098.70 sec** | 71227.24 MB | 2864.18 sec | 117156.88 MB | TimeOut | OutOfMemory | 716.63 sec | 65.44 MB |

applying NormaKG to the baseline presented the shortest times. However, memory usage remains largely unchanged across normalization methods. This limited improvement in memory efficiency can be attributed to how Morph-KGC processes TMs and their associated data sources. Specifically, Morph-KGC divides each TM based on the number of `rr:predicateObjectMap` it contains. For instance, a TM with five `rr:predicateObjectMap` entries is split into five separate TMs, each retaining the same `rr:subjectMap` but containing only one of the original `rr:predicateObjectMap`. There is an increase in memory usage when using RML-Planner in the test cases. This increase in memory usage is attributed to the overhead produced by executing the mapping partitions in parallel and then combining the results. Additionally, Morph-KGC projects the data sources according to the new TMs and removes duplicates in the data source. Therefore, the data sources (regardless of the method) will be projected even further, and the amount of uploaded data will be the same. Projecting the

**Table 6**
Results of test cases on SDM-Genomic-Datasets with 10M and 75% duplicate rate. Best results in **bold**..

| Models | Baseline | SDM-RDFizer | | Morph-KGC | | RMLMapper | | FlexRML | |
|---|---|---|---|---|---|---|---|---|---|
| | | Time | Memory | Time | Memory | Time | Memory | Time | Memory |
| **Conf1 + 10M rows** | | | | | | | | | |
| **Baseline** | **Original** | 3325.31 sec | 4508.02 MB | 109.46 sec | 4379.70 MB | TimeOut | 49988.23 MB ≤ | 483.61 sec | 2088.75 MB |
| | **NormaKG** | 967.85 sec | **107.10 MB** | **42.39 sec** | **3226.41 MB** | **212.78 sec** | 32713.79 MB | 126.80 sec | **4.68 MB** |
| **MapSDI** | **Original** | 1294.46 sec | 4222.2 MB | 45.61 sec | 3243.01 MB | TimeOut | 43658.21 MB ≤ | 146.81 sec | 2082.48 MB |
| | **NormaKG** | 645.65 sec | 2090.51 MB | 31.96 sec | 2922.71 MB | 207.10 sec | 30731.25 MB | 116.01 sec | 4.66 MB |
| **RML-Planner** | **Original** | 771.49 sec | 7679.36 MB | 80.09 sec | 3355.10 MB | TimeOut | 47733.01 MB ≤ | 467.81 sec | 2085.37 MB |
| | **NormaKG** | **322.98 sec** | 7318.44 MB | 61.12 sec | 3070.09 MB | 232.50 sec | **10125.50 MB** | **39.14 sec** | 65.83 MB |
| **Conf2 + 10M rows** | | | | | | | | | |
| **Baseline** | **Original** | 4622.78 sec | 23297.66 MB | 942.20 sec | 59465.06 MB | TimeOut | 815246.45 MB ≤ | TimeOut | 57784.25 MB ≤ |
| | **NormaKG** | **374.90 sec** | **196.52 MB** | **707.12 sec** | **58218.18 MB** | TimeOut | 31093.40 MB ≤ | **221.25 sec** | **11.03 MB** |
| **MapSDI** | **Original** | 581.14 sec | 13739.04 MB | 712.49 sec | 59306.49 MB | TimeOut | 69635.27 MB ≤ | 297.73 sec | 5525.53 MB |
| | **NormaKG** | 458.78 sec | 11266.34 MB | 311.33 sec | 57912.00 MB | TimeOut | 45963.74 MB ≤ | 241.98 sec | 10.95 MB |
| **RML-Planner** | **Original** | 1433.91 sec | 27835.34 MB | 1157.21 sec | 57929.93 MB | TimeOut | 817559.36 MB ≤ | TimeOut | 62741.89 MB ≤ |
| | **NormaKG** | 1224.18 sec | 25807.93 MB | 960.59 sec | 57949.86 MB | TimeOut | 52691.74 MB ≤ | 484.65 sec | 64.85 MB |
| **Conf3 + 10M rows** | | | | | | | | | |
| **Baseline** | **Original** | 5049.04 sec | 16935.24 MB | 1108.72 sec | 65184.57 MB | TimeOut | 34175.07 MB ≤ | TimeOut | 67821.47 MB ≤ |
| | **NormaKG** | 1359.60 sec | **136.78 MB** | **471.72 sec** | **58340.32 MB** | TimeOut | 10163.63 MB ≤ | **277.47 sec** | **9.58 MB** |
| **MapSDI** | **Original** | 1412.41 sec | 14468.92 MB | 711.71 sec | 60317.82 MB | TimeOut | 21569.35 MB ≤ | 407.32 sec | 7788.42 MB |
| | **NormaKG** | 968.25 sec | 2023.06 MB | 593.80 sec | 63258.14 MB | TimeOut | 15823.74 MB ≤ | 325.16 sec | 11.22 MB |
| **RML-Planner** | **Original** | 941.11 sec | 31994.83 MB | 594.56 sec | 58340.75 MB | TimeOut | 35690.91 MB ≤ | TimeOut | 68851.87 MB ≤ |
| | **NormaKG** | **593.63 sec** | 34022.29 MB | 547.66 sec | 58244.36 MB | TimeOut | OutOfMemory | 353.3 sec | 59.70 MB |
| **Conf4 + 10M rows** | | | | | | | | | |
| **Baseline** | **Original** | 7987.25 sec | 24300.47 MB | 1298.57 sec | 65589.74 MB | TimeOut | 52693.75 MB ≤ | TimeOut | 13524.33 MB ≤ |
| | **NormaKG** | 1442.5 sec | **200.27 MB** | **447.77 sec** | **58340.70 MB** | TimeOut | 32911.60 MB ≤ | **297.45 sec** | **11.07 MB** |
| **MapSDI** | **Original** | 1489.25 sec | 14516.16 MB | 1286.76 sec | 60659.06 MB | TimeOut | 44683.78 MB ≤ | 440.40 sec | 7956.61 MB |
| | **NormaKG** | 934.34 sec | 1913.77 MB | 332.36 sec | 57913.41 MB | TimeOut | 33821.430 MB ≤ | 238.87 sec | 10.91 MB |
| **RML-Planner** | **Original** | 1682.43 sec | 35419.85 MB | 1359.55 sec | 57929.62 MB | TimeOut | 56179.02 MB ≤ | TimeOut | 14986.05 MB ≤ |
| | **NormaKG** | **1396.08 sec** | 33392.56 MB | 1301.07 sec | 57904.39 MB | TimeOut | OutOfMemory | 377.82 sec | 62.48 MB |

data sources and partitioning the mappings beforehand only helps the execution time.

Applying only the NormaKG method significantly improved RMLMapper's performance, enabling it to complete certain test cases, particularly simpler ones such as Conf1. On the other hand, using both NormaKG and RML-Planner allows RMLMapper to complete more complex test cases, such as Conf3 and Conf4. These improvements can be attributed to the fact that NomaKG permits RMLMapper to bypass duplicate removal, a process that otherwise increases execution time. RMLMapper handles duplicate removal by first generating the entire KG and then sorting and removing duplicate triples, which is computationally expensive. RML-Planner partitions the mapping based on data source, it defines an execution plan based on overlapping properties, executes each partition in parallel, and then combines all the KG chunks together. This distributes triple generation across the execution of the partitions. When executing RML-Planner alongside NormaKG, it increases the number of partitions, but distributes even further the amount of triples that are generated per partition. This combined method significantly reduces computational cost for the test cases, allowing RMLMapper to successfully complete them. Despite these optimizations, RMLMapper still failed to complete Conf2, primarily due to its inefficient join execution strategy. RMLMapper processes joins by iterating over all parent and child data sources multiple times, resulting in excessive resource consumption and prolonged execution times. Due to incomplete test runs, memory usage metrics for RMLMapper could not be fully analyzed. The reported values represent memory usage only up to the point of timeout, suggesting that actual consumption may have been significantly higher. As a result, definitive conclusions about memory efficiency cannot be drawn from the available data.

FlexRML showed significant improvements when using all approaches, particularly for complex cases (Conf2, Conf3, and Conf4) with 10M row data sources that it could not complete previously due to timeouts. This improvement stems from how FlexRML handles joins. FlexRML estimates the number of unique triples that will be generated from the join using a cost function, which determines the memory required for the join operation. The larger size of the baseline data sources demanded more resources, resulting in longer execution times and resource usage. Both normalization (i.e., NormaKG and MapSDI) approaches alleviated these issues by reducing the size of the data sources, with NormaKG delivering the most improvements between the two. When applying both alongside each other there are even more savings. Finally, using RML-Planner with NormaKG presented the shortest execution time in most test cases at the cost of memory usage.

FlexRML achieved considerable savings in execution time and memory usage, with NormaKG applied over the baseline outperforming MapSDI. MapSDI allowed FlexRML to complete all test cases without timing out, but resulted in higher memory consumption than when using NormaKG. This is due to FlexRML's approach to duplicate removal, which involves using a hash function to store all unique triples generated during KG creation. Each new triple is compared against this hash function: duplicates are discarded, while unique triples are added to both the hash and the KG. This process effectively creates a virtual copy of the KG in memory, inflating memory usage.

In contrast, the NormaKG approach prevents duplicates from being generated, allowing FlexRML to bypass duplicate removal entirely. Compared to the MapSDI approach, this results in substantial memory savings, up to two orders of magnitude. This efficiency is critical for handling large and complex datasets, as it eliminates the need to store or compare generated triples, thus optimizing resource usage.

When using RML-Planner with NormaKG, FlexRML presented the shortest execution time. This performance improvement is attributable to RML-Planner's support for parallel execution of mapping partitions. Unfortunately, this configuration presented a higher memory usage than only applying NormaKG to the baseline. This increase in memory usage is caused by the combination process that merges the KG chunks produced by the partitions.

### 6.3. Experimental results for GTFS-madrid-bench

This subsection presents the results of evaluating two experimental test cases using GTFS-Madrid-Bench, a benchmarking framework designed to assess KG creation engines with transportation datasets at

**Table 7**
Results of test cases on GTFS-Madrid-Bench. Best results in **bold**.

| Models | Baseline | SDM-RDFizer | | Morph-KGC | | RMLMapper | | FlexRML | |
|---|---|---|---|---|---|---|---|---|---|
| | | Time | Memory | Time | Memory | Time | Memory | Time | Memory |
| **Scale-10** | | | | | | | | | |
| **Baseline** | **Original** | 170.50 sec | 1069.52 MB | 41.20 sec | 1717.734 MB | Timeout | 15863.24 MB ≤ | 20.3 sec | 240.64 MB |
| | **NormaKG** | 134.00 sec | **108.96 MB** | **14.20 sec** | **704.04 MB** | 447.05 sec | **5751.07 MB** | 13.78 sec | **6.23 MB** |
| **MapSDI** | **Original** | 150.30 sec | 896.26 MB | 37.24 sec | 1699.66 MB | TimeOut | 12058.10 MB ≤ | 15.56 sec | 240.22 MB |
| | **NormaKG** | 70.46 sec | 595.74 MB | 34.83 sec | 1604.88 MB | 650.14 sec | 5915.40 MB | 13.64 sec | 145.45 MB |
| **RML-Planner** | **Original** | 78.09 sec | 1173.41 MB | 19.59 sec | 1177.26 MB | TimeOut | 10359.27 MB ≤ | 25.19 sec | 216.52 MB |
| | **NormaKG** | **44.90 sec** | 943.99 MB | 14.85 sec | 946.83 MB | **22.50 sec** | 6818.73 MB | **13.72 sec** | 59.98 MB |
| **Scale-100** | | | | | | | | | |
| **Baseline** | **Original** | 1683.60 sec | 8091.93 MB | 431.91 sec | 15772.40 MB | TimeOut | 41324.21 MB ≤ | 213.65 sec | 2287.87 MB |
| | **NormaKG** | 1402.04 sec | **340.38 MB** | **79.55 sec** | **6167.72 MB** | TimeOut | 20877.02 MB ≤ | 135.95 sec | **19.81 MB** |
| **MapSDI** | **Original** | 1601.9 sec | 7398.40 MB | 416.46 sec | 14581.51 MB | TimeOut | 32558.31 MB ≤ | 159.65 sec | 2280.70 MB |
| | **NormaKG** | 947.22 sec | 4742.51 MB | 445.72 sec | 14355.90 MB | TimeOut | 26241.95 MB ≤ | 134.54 sec | 2696.39 MB |
| **RML-Planner** | **Original** | 767.89 sec | 11735.26 MB | 125.96 sec | 11773.46 MB | TimeOut | 29688.23 MB ≤ | 248.94 sec | 2049.86 MB |
| | **NormaKG** | **401.89 sec** | 9602.11 MB | 121.58 sec | 9470.29 MB | **151.15 sec** | **16460.60 MB** | **22.31 sec** | 60.06 MB |

varying scales. A key challenge in this benchmark is the computationally expensive `rr:predicateObjectMap`, which generates the largest amount of triples through a self-join on the dataset's largest table. The NormaKG approach simplifies this process by transforming the self-join into an equivalent `rr:template` term, eliminating the need for an explicit join operation. This transformation is feasible because the join is one-to-one, with the parent and child conditions defined on a unique key attribute. This subsection addresses a total of 48 test cases. Fig. 13 illustrates the progression of the DIS in this benchmark, increasing from 13 triples maps and corresponding data sources to 38 as the normalization rules were applied. Table 7 shows the performance of RML engines. When evaluating the functionally projected DIS with RML-Planner, 29 partitions were executed, prioritizing triples maps that generated triples and ignoring those that existed only as parent sources for joins.

RMLMapper benefited from NormaKG, successfully completing the previously infeasible Scale-10 test case. It completed the execution of Scale-10 in all experimental configurations where NormaKG was applied (i.e., Baseline + NormaKG, MapSDI + NormaKG, and RML-Planner + NormaKG). Among these, Baseline + NormaKG has the best memory usage, while RML-Planner + NormaKG has the shortest execution time. However, RMLMapper could only execute Scale-100 when both NormaKG and RML-Planner were applied. The failure to execute Scale-100 under the other experimental configurations is due to the fact that Scale-100 produces a KG ten times larger than Scale-10; thus, RMLMapper's limited scalability remains a constraining factor despite the applied optimizations. The success of RML-Planner + NormaKG is due to two factors. First, NormaKG converts the most significant join. Second, RML-Planner partitions the input mapping, reducing the number of triple maps to be executed. By applying these two optimization techniques, significantly reduces computational cost for RMLMapper.

FlexRML achieved the most substantial memory savings when applying NormaKG to the Baseline, reducing memory consumption by 97.41% in Scale-10 and 99.13% in Scale-100. Two factors contributed to this: (1) NormaKG eliminates duplicate removal, allowing FlexRML to avoid the high memory cost of storing and comparing generated triples, and (2) the simplified mapping removes the need for FlexRML's cost function to estimate join results, further reducing resource usage. When executing NormaKG alongside MapSDI and RML-Planner, FlexRML achieves greater performance improvements than when executed with those methods alone. For Scale-10, execution time is comparable between Baseline + NormaKG and RML-Planner + NormaKG, whereas Baseline + NormaKG exhibits lower memory usage. For Scale-100, FlexRML achieves the greatest memory savings under Baseline + NormaKG, while the shortest execution time is observed under RML-Planner + NormaKG. This difference in memory usage can be attributed to the overhead introduced by executing RML-Planner, whereas the difference in execution time is due to RML-Planner enabling the parallel execution of TMs.

SDM-RDFizer also showed improvements in execution time and memory usage, reducing memory consumption by 89.81% in Scale-10 and 95.79% in Scale-100 in Baseline + NormaKG. SDM-RDFizer achieved the shortest execution time with RML-Planner + NormaKG, reducing execution time by 73.67% in Scale-10 and 76.13% in Scale-100. Similar to FlexRML, these gains can be attributed to the elimination of duplicate removal, reducing overall memory requirements. Additionally, since SDM-RDFizer executes joins by storing intermediate results, it benefits further from the functional projection method, as the most computationally expensive join has been simplified and no longer needs to be executed. SDM-RDFizer presented improved performance when executing MapSDI and RML-Planner alongside NormaKG.

Finally, Morph-KGC also demonstrated improved execution times and reduced memory usage with the NormaKG approach. These improvements stem from smaller projected data sources and reduced join complexity. These findings highlight the scalability advantages of NormaKG, particularly for complex mappings involving large-scale self-joins. Morph-KGC presented the best memory usage and execution time with the Baseline + NormaKG. These results indicate that for Morph-KGC executing with NormaKG alongside any other method caused additional overhead that negatively impacted its performance in this dataset.

### 6.4. Experimental results for KROWN

This subsection discusses the results of evaluating the five experimental configurations defined using the KROWN data generation benchmark. Both the MapSDI and RML-Planner approaches were applied across all test cases, either on their own or alongside NormaKG. Additionally, NormaKG is applied on its own to determine its benefits on the performance of the KG creation engines. The MapSDI approach yielded smaller subsets of the original data sources. Similarly, NormaKG normalization produced smaller data sources, excluding non-functional attributes stored separately. Both approaches aimed to simplify join execution. RML-Planner, on the other hand, partitions the mapping based on its data sources and then executes them based on an execution plan. This subsection addresses 120 test cases. Fig. 13 illustrates the evolution of the DIS in this benchmark, increasing from five triples maps and corresponding data sources to 23 as the normalization rules were applied. As shown in Table 8, all engines demonstrated reduced execution time and memory usage when transforming data sources projected using the normalized method. Notably, RMLMapper, which previously failed to complete the generation of the KG, succeeded with the normalized projection. Notably, having significant improvements in execution time and memory usage when running alongside RML-Planner and NormaKG (i.e., RML-Planner + NormaKG).

SDM-RDFizer successfully executed all test case configurations. Transforming data sources with only the proposed solution required the least memory (i.e., Baseline + NormaKG). Notably, in Conf3, SDM-

**Table 8**
Results of test cases on KROWN. Best results in **bold**.

| Models | Baseline | SDM-RDFizer | | Morph-KGC | | RMLMapper | | FlexRML | |
|---|---|---|---|---|---|---|---|---|---|
| | | Time | Memory | Time | Memory | Time | Memory | Time | Memory |
| **Conf1 + 1M rows** | | | | | | | | | |
| Baseline | **Original** | 423.52 sec | 327.48 MB | 91.99 sec | 3859.27 MB | TimeOut | 31755.26 MB ≤ | TimeOut | 233.34 MB ≤ |
| | **NormaKG** | **10.28 sec** | **111.83 MB** | **8.05 sec** | **697.14 MB** | 125.72 sec | **3667.78 MB** | **4.13 sec** | **4.74 MB** |
| MapSDI | **Original** | 12.99 sec | 324.05 MB | 8.45 sec | 697.62 MB | TimeOut | 5266.70 MB ≤ | 5.32 sec | 127.28 MB |
| | **NormaKG** | 10.95 sec | 143.78 MB | 6.03 sec | 698.94 MB | 103.86 sec | 3882.71 MB | 5.23 sec | 4.64 MB |
| RML-Planner | **Original** | 281.44 sec | 1198.26 MB | 11.44 sec | 816.31 MB | TimeOut | 9901.56 MB ≤ | TimeOut | 267.20 MB ≤ |
| | **NormaKG** | 31.24 sec | 539.27 MB | 9.99 sec | 815.59 MB | **20.00 sec** | 7820.59 MB | 6.78 sec | 59.59 MB |
| **Conf1 + 10M rows** | | | | | | | | | |
| Baseline | **Original** | 4255.42 sec | 2070.71 MB | 887.25 sec | 7254.48 MB | TimeOut | OutOfMemory | TimeOut | 2365.23 MB ≤ |
| | **NormaKG** | **72.61 sec** | **113.06 MB** | **58.85 sec** | **5898.51 MB** | 4856.71 sec | **4676.91 MB** | **39.56 sec** | **5.93 MB** |
| MapSDI | **Original** | 104.32 sec | 2041.23 MB | 60.87 sec | 5898.58 MB | TimeOut | OutOfMemory | 56.47 sec | 1180.14 MB |
| | **NormaKG** | 72.94 sec | 403.67 MB | 40.58 sec | 5898.75 MB | 4086.00 sec | 4870.29 MB | 51.26 sec | 4.63 MB |
| RML-Planner | **Original** | 2834.82 sec | 12581.02 MB | 96.08 sec | 7254.56 MB | TimeOut | OutOfMemory | TimeOut | 3784.23 MB ≤ |
| | **NormaKG** | 295.81 sec | 4736.78 MB | 80.04 sec | 7194.37 MB | **157.82 sec** | 16652.50 MB | 40.52 sec | 59.76 MB |
| **Conf2 + 1M rows** | | | | | | | | | |
| Baseline | **Original** | 430.06 sec | 325.73 MB | 86.67 sec | 3802.95 MB | TimeOut | 34889.62 MB ≤ | TimeOut | 327.79 MB ≤ |
| | **NormaKG** | **12.25 sec** | **112.65 MB** | **8.09 sec** | **685.53 MB** | 130.68 sec | **3553.15 MB** | **4.16 sec** | **4.79 MB** |
| MapSDI | **Original** | 13.00 sec | 324.58 MB | 8.75 sec | 685.88 MB | TimeOut | 6510.39 MB ≤ | 5.32 sec | 127.28 MB |
| | **NormaKG** | 13.21 sec | 142.37 MB | 6.19 sec | 685.75 MB | 103.79 sec | 3482.50 MB | 5.24 sec | 4.67 MB |
| RML-Planner | **Original** | 276.83 sec | 1187.32 MB | 10.92 sec | 803.50 MB | TimeOut | 9056.04 MB ≤ | TimeOut | 489.77 MB ≤ |
| | **NormaKG** | 31.56 sec | 539.27 MB | 11.71 sec | 802.70 MB | **19.47 sec** | 6663.18 MB | 7.95 sec | 59.77 MB |
| **Conf2 + 10M rows** | | | | | | | | | |
| Baseline | **Original** | 4348.92 sec | 2060.78 MB | 874.89 sec | 7253.33 MB | TimeOut | OutOfMemory | TimeOut | 2487.33 MB ≤ |
| | **NormaKG** | **75.73 sec** | **112.26 MB** | **58.82 sec** | **5898.39 MB** | 5188.42 sec | **6493.77 MB** | **40.27 sec** | **5.99 MB** |
| MapSDI | **Original** | 106.24 sec | 2041.29 MB | 61.08 sec | 5898.90 MB | TimeOut | OutOfMemory | 55.53 sec | 1180.24 MB |
| | **NormaKG** | 78.65 sec | 402.46 MB | 40.76 sec | 5899.04 MB | 3978.00 sec | 5012.57 MB | 51.65 sec | 4.60 MB |
| RML-Planner | **Original** | 2728.47 sec | 11628.34 MB | 95.63 sec | 7251.57 MB | TimeOut | OutOfMemory | TimeOut | 4098.25 MB ≤ |
| | **NormaKG** | 297.69 sec | 4747.86 MB | 80.85 sec | 7194.36 MB | **163.28 sec** | 16817.62 MB | 52.16 sec | 59.78 MB |
| **Conf3 + 10M rows** | | | | | | | | | |
| Baseline | **Original** | 5208.70 sec | 140.53 MB | 699.25 sec | 3039.76 MB | TimeOut | OutOfMemory | TimeOut | 895.97 MB ≤ |
| | **NormaKG** | 4.68 sec | **112.26 MB** | **3.32 sec** | **151.49 MB** | 11.96 sec | 1531.13 MB | **0.50 sec** | **4.81 MB** |
| MapSDI | **Original** | 4.99 sec | 139.66 MB | 4.00 sec | 152.84 MB | 911.92 sec | 2891.29 MB | 0.51 sec | 17.77 MB |
| | **NormaKG** | 5.58 sec | 116.71 MB | 2.87 sec | 152.61 MB | 5.96 sec | 1435.45 MB | 0.48 sec | 4.66 MB |
| RML-Planner | **Original** | 2868.14 sec | 846.36 MB | 72.85 sec | 3039.76 MB | TimeOut | OutOfMemory | TimeOut | 971.32 MB ≤ |
| | **NormaKG** | **2.51 sec** | 118.29 MB | 4.16 sec | 164.63 MB | **1.59 sec** | **213.30 MB** | 2.26 sec | 59.68 MB |

RDFizer presented the shortest execution time when applying RML-Planner + NormKG. Regarding execution time, SDM-RDFizer was the fastest with Baseline + NormaKG for the test cases Conf1 and Conf2 with 10M rows. For other configurations, the execution times were similar for both the MapSDI + NormaKG and Baseline + NormaKG approaches. However, memory usage was higher with MapSDI due to duplicate removal. This process requires the SDM-RDFizer to compare each new triple with all previously generated triples, storing these triples in memory for comparison. Comparing SDM-RDFizer's performance with and without normalization approaches reveals significant improvements. The proposed solution reduced execution time by up to two orders of magnitude and memory usage by up to one order of magnitude. Notably, SDM-RDFizer in Conf3, RML-Planner + NormaKG has the shortest execution time. Unfortunately, in the other test cases, SDM-RDFizer with RML-Planner + NormaKG presented longer execution times than Baseline + NormaKG. This can be attributed to the overhead produced by RML-planner when it partitions the mapping, executes the partitions, and then combines all the partial KGs.

Morph-KGC, plus its own data source partitioning, successfully executed all test case configurations. MapSDI + NormaKG and Baseline + NormaKG methods reduced memory usage and execution time, showing improvements of up to one order of magnitude. Baseline + NormaKG slightly outperformed the MapSDI + NormaKG, reflecting Morph-KGC's approach of partitioning TMs by properties. However, this partitioning method lacks specific optimization criteria, as TMs are divided until every property is isolated. Consequently, the normalized data sources—smaller than the originals—are further partitioned during KG generation. Similarly to what was seen with SDM-RDFizer, RML-Planner + NormaKG presented longer execution times and memory usage, which is attributed to the overhead produced by the execution of the RML-Planner.

In cases without normalization, Morph-KGC's performance was significantly worse due to the size of the baseline data sources, which were at least three orders of magnitude larger than the normalized ones. The high cost of join operations on these larger data sources and Morph-KGC's reliance on the Python library *pandas* for non-scalable operations, such as joins, further exacerbate resource consumption. For instance, joins executed over 1M or 10M rows are much more expensive than those performed on the smaller projected data sources. These factors highlight the critical role of normalization in improving Morph-KGC's performance.

RMLMapper could only complete test cases where the NormaKG approach was applied. All other cases either timed out or ran out of memory. The main reason RMLMapper can complete these test cases is that they do not require duplicate removal. RMLMapper removes duplicates by generating all possible triples, including duplicates, and then using a sorting algorithm to remove them. The more triples generated, the longer the sorting process will take.

Given the complexity of the joins defined in these test cases, RMLMapper used significantly more memory and took longer to execute. This could be attributed to the way RMLMapper performs joins. It opens both parent and child data sources and compares the values individually to determine the corresponding values. This process is applied each time a join is present. Repeatedly doing this significantly increases memory usage and execution time. Although the joins are simplified and the corresponding data sources are projected using the MapSDI approach, RMLMapper was still unable to complete these test cases because they require duplicate removal. Additionally, applying RML-Planner alone is

not sufficient to significantly improve RMLMapper's performance, since RML-Planner partitions the mapping but does not apply any transformations to its data sources. Thus, thanks to the application of NormaKG, duplicate removal becomes unnecessary, and RMLMapper can complete cases that it could not do before.

FlexRML could complete the test cases where the normalization approaches were used, but not the baseline ones (i.e., Baseline and RML-Planner). FlexRML exhibited the lowest memory usage when executing test cases employing the NormaKG approach. Memory usage is up to three orders of magnitude lower than the memory usage reported when FlexRML timed out while processing the baseline cases. Both the Baseline + NormaKG and MapSDI + NormaKG approaches present similar execution times in the test cases with 1M data sources. Still, MapSDI + NormaKG has higher memory usage in some cases, where the memory usage is three orders of magnitude larger than NormaKG. The savings achieved with the Baseline + NormaKG approach can be attributed to avoiding duplicate removal and executing simpler joins. FlexRML removes duplicates similarly to SDM-RDFizer, using a hash table where each triple generated is compared to the table. If the triple exists in the hash table, the triple is duplicated and discarded. If not, the triple is stored in the table for future comparisons; as with SDM-RDFizer, this method increases memory usage with the number of unique triples generated and stored. In the cases of executing joins, FlexRML applies a cost function to determine how many unique triples will be generated when executing a particular join. This function determines the amount of memory required to perform a join. In the case of the original test cases, the joins are performed over much larger data sources than those used in the normalization approaches, thus requiring substantially more resources to evaluate the cost function and apply the result. Therefore, implementing a more complex cost function and removing duplicates rendered FlexRML unable to complete execution of the baseline test cases. Unfortunately, RML-Planner + NormaKG produced an overhead that increased the execution time and memory usage for FlexRML.

### 6.5. Result analysis

The empirical study demonstrates that data normalization techniques significantly improve the performance of KG creation engines. Engines that leveraged the NormaKG approach showed marked improvements in execution time and memory consumption. Notably, RMLMapper and FlexRML completed test cases they could not before when applying NormaKG or NormaKG alongside RML-Planner, addressing **RQ1**. The NormaKG approach improves the engines' performance by normalizing data sources according to the functional dependencies of each TM's concept MA. This partitioning ensures that functional dependencies remain together while non-functional dependencies are isolated into separate TMs. As a result, the number of data sources decreases, making them more manageable and requiring fewer resources to process. This reduction in resource demands directly contributes to lower memory usage, particularly for engines such as SDM-RDFizer and FlexRML, which typically store all generated triples for duplicate removal. By eliminating the need for duplicate removal, memory usage was reduced by up to three orders of magnitude, answering **RQ2**. Additionally, the smaller, partitioned TMs accelerate the transformation process because less data must be processed. This directly results in shorter execution times, thereby addressing **RQ3**. Finally, combining NormaKG with other optimization methods (e.g., MapSDI and RML-Planner) leads to greater performance improvements than applying MapSDI and RML-Planner alone. These findings underscore the role of data normalization in optimizing KG creation, particularly for large and complex datasets, allowing for more scalable and efficient data integration.

RML-Planner and NormaKG are two approaches to KG creation optimization, each focusing on a different factor that impacts the creation process. RML-Planner applies mapping partitioning and parallel execution to reduce mapping complexity. It groups the TMs by data source and then determines an execution plan based on overlapping attributes between the partitions. Subsequently, the KG chunks are combined. This improves the engines' performance in terms of execution time. In this case, the engine's execution time is determined by the partition that takes the longest to execute and by the combination process that aggregates all the chunks. The main issue with this method is that the combination process can increase memory consumption and execution time when the chunks being combined are large. On the other hand, NormaKG focuses on data fragmentation, simplifying data sources and executing more complicated joins. This process reduces the memory usage and execution time of the engines. Notably, the application of both methods enables engines such as RML-Mapper to execute test cases that were previously infeasible, regardless of the optimization method used. Additionally, depending on the test case, employing both methods resulted in shorter execution times but higher memory usage than applying NormaKG alone. This increase in memory usage is due to the combination process used by RML-Planner. These findings indicate that optimal optimization depends on the nature of the test case.

## 7. Conclusions and future work

This paper addressed the problem of optimizing knowledge graph (KG) creation by transforming data integration systems (DISs) into normalized and semantically equivalent representations. Motivated by inefficiencies in existing RML-based pipelines—such as excessive memory usage, redundant triples, and costly join operations—we introduced a normalization theory for DISs, inspired by classical relational database normalization.

Our main contribution lies in defining a family of mapping-level normal forms—TM-1NF, TM-2NF, TM-3NF, and TM-JNF—that systematically eliminate structural, dependency-based, and execution-level anomalies in KG creation. We formalized a comprehensive set of normalization rules and proposed a transformation algorithm that restructures mapping assertions and logical sources based on functional dependencies. The approach guarantees semantic equivalence between the original and normalized DISs while improving runtime performance and eliminating the need for duplicate removal. These ideas are implemented in NormaKG, an engine-agnostic framework that produces optimized DISs compatible with any RML engine. Experimental results over established benchmarks confirm significant reductions in memory consumption and execution time.

By grounding mapping design in formal normalization principles, this work contributes a scalable and principled approach to efficient information fusion. It complements existing techniques for mapping partitioning and data source fragmentation by offering an automated and theory-driven alternative. Unlike ad hoc or engine-specific optimizations, our normalization framework provides a reproducible and portable foundation for transforming DISs into forms that are not only structurally sound but also highly efficient for integration tasks. By explicitly modeling functional dependencies and identifying mapping anomalies, our approach enables deeper insights into the structure and complexity of KG creation workflows and provides actionable guidance for their simplification. This supports not only runtime efficiency but also maintainability and extensibility in large-scale, multi-source information fusion settings.

NormaKG is positioned as a structural optimization layer operating at the data-level preprocessing and feature-level fusion stages of data integration pipelines. Rather than introducing a new fusion architecture, it enhances existing ones by strengthening the structural integrity of intermediate representations. By eliminating redundancy and preventing join-induced combinatorial blow-up, normalization improves the robustness and scalability of downstream processing stages, including analytics, reasoning, graph-based learning, inductive learning over KGs.

Future work will extend the normalization theory along several directions. First, we will generalize the framework to expressive and hierarchical data models such as JSON and XML, where nested and semi-structured dependencies introduce new forms of structural anomalies.

Second, we plan to incorporate automated functional dependency discovery and validation mechanisms to reduce reliance on manually specified dependency sets. Third, we will investigate cost-aware and adaptive normalization strategies that integrate runtime feedback and engine-specific characteristics while preserving semantic guarantees.
Additionally, we aim to integrate NormaKG into interactive mapping authoring environments and continuous KG construction pipelines, enabling real-time and incremental optimization in streaming and evolving data integration scenarios. This opens the door to adaptive and self-optimizing KG creation workflows capable of supporting large-scale, dynamic information fusion applications.
We envision normalization-driven KG construction as a foundational component of future scalable integration architectures, where structural correctness, semantic guarantees, and computational efficiency are treated as first-class design principles rather than post hoc optimizations.

## CRediT authorship contribution statement

**Enrique Iglesias:** Writing – review & editing, Writing – original draft, Software, Resources, Methodology, Investigation, Formal analysis, Conceptualization; **David Chaves-Fraga:** Writing – review & editing, Supervision, Methodology; **Sören Auer:** Methodology, Supervision, Funding acquisition, Writing - original draft, Writing - review & editing; **Maria-Esther Vidal:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

## Data availability

The data is available in GitHub

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:
Enrique Iglesias reports financial support was provided by Leibniz Association Consortia. Enrique Iglesias reports financial support was provided by Volkswagen Foundation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work was partially supported by the Lower Saxony Ministry of Science and Culture (MWK) with funds from the Volkswagen Foundation's zukunft.niedersachsen program (CAIMed - Lower Saxony Center for AI and Causal Methods in Medicine; GA No. ZN4257); the "Leibniz Best Minds: Programme for Women Professors", through funding of the "TrustKG-Transforming Data in Trustable Insights" project (Grant P99/2020); KISSKI, funded by the German Ministry of Education and Research (BmBF) for the project KISSKI AI Service Center (01IS22093C). Additionally, David Chaves-Fraga is funded by the Agencia Estatal de Investigación (Spain) (PID2023-149549NB-I00 & CPP2024-011786), the Xunta de Galicia - Consellería de Cultura, Educación, Formación Profesional e Universidades (Centro de investigación de Galicia accreditation 2024-2027 ED431G-2023/04) and the European Union (European Regional Development Fund - ERDF).

## References

[1] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, A.N. Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge Graphs, Synthesis Lectures on Data, Semantics, and Knowledge, Morgan & Claypool Publishers, 2021.
[2] A. Scherp, G. Gröner, P. Skoda, K. Hose, M. Vidal, Semantic web: past, present, and future, TGDK 2 (1) (2024) 3:1–3:37. https://doi.org/10.4230/TGDK.2.1.3
[3] V. de Boer, Knowledge graphs for cultural heritage and digital humanities, in: V. Gouet-Brunet, R. Kosti, L. Weng (Eds.), Proceedings of the 5th Workshop on analySis, Understanding and proMotion of heritAge Contents, SUMAC 2023, Ottawa, ON, Canada, 2 November 2023, ACM, 2023, p. 3. https://doi.org/10.1145/3607542.3617354. https://doi.org/10.1145/3607542.3617354
[4] F. Aisopos, S. Jozashoori, E. Niazmand, D. Purohit, A. Rivas, A. Sakor, E. Iglesias, D. Vogiatzis, E. Menasalvas, A.R. González, G. Vigueras, D. Gómez-Bravo, M. Torrente, R.H. López, M.P. Pulla, A. Dalianis, A. Triantafillou, G. Paliouras, M. Vidal, Knowledge graphs for enhancing transparency in health data ecosystems, Semant. Web 14 (5) (2023) 943–976.
[5] H. Turki, T. Shafee, M.A.H. Taieb, M.B. Aouicha, D. Vrandecic, D. Das, H. Hamdi, Wikidata: a large-scale collaborative ontological medical database, J. Biomed. Inf. 99 103292 (2019).
[6] M. Šidlovský, F. Ravas, Building knowledge graph in the transportation domain, in: 2023 Smart City Symposium Prague (SCSP), 2023, pp. 1–4. https://doi.org/10.1109/SCSP58044.2023.10146125
[7] E. Rajabi, R. Midha, J.F. de Souza, Constructing a knowledge graph for open government data: the case of Nova Scotia disease datasets, J. Biomed. Semantics 14 (1) (2023). http://dx.doi.org/10.1186/s13326-023-00284-w. https://doi.org/10.1186/s13326-023-00284-w
[8] D. Chaves-Fraga, K.M. Endris, E. Iglesias, Ó. Corcho, M. Vidal, What are the parameters that affect the construction of a knowledge graph?, in: OTM Confederated International Conferences, 2019.
[9] M. Lenzerini, Data integration: a theoretical perspective, in: ACM Symposium on Principles of Database Systems, 2002.
[10] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF mapping language, W3C recommendation 2012, W3C (2012).
[11] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: a generic language for integrated RDF mappings of heterogeneous data, in: Workshop on Linked Data on the Web, 2014.
[12] D. Chaves-Fraga, E. Ruckhaus, F. Priyatna, M. Vidal, Ó. Corcho, Enhancing virtual ontology based access over tabular data with Morph-CSV, Semant. Web 12 (6) 869–902 (2021).
[13] A. Dimou, T. De Nies, R. Verborgh, E. Mannens, R.V. de Walle, Automated metadata generation for linked data generation and publishing workflows, in: S. Auer, T. Berners-Lee, C. Bizer, T. Heath (Eds.), Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, Co-located with 25th International World Wide Web Conference (WWW 2016), 1593 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016.
[14] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, M.-E. Vidal, SDM-RDFizer: an RML interpreter for the efficient creation of RDF knowledge graphs, in: ACM CIKM, 2020, pp. 3039–3046.
[15] U. Şimşek, E. Kärle, D. Fensel, RocketRML - A NodeJS implementation of a use-case specific RML mapper, 2019. Accessed: 24-06-2022. arXiv:1903.04969.
[16] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus, Ó. Corcho, GTFS-Madrid-Bench: a benchmark for virtual knowledge graph access in the transport domain, J. Web Semant. 65 (2020) 100596. https://doi.org/10.1016/j.websem.2020.100596
[17] M. Vidal, K.M. Endris, S. Jazashoori, A. Sakor, A. Rivas, Transforming heterogeneous data into knowledge for personalized treatments - a use case, Datenbank-Spektrum 19 (2) (2019) 95–106.
[18] M.T. Özsu, P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition, Upper Saddle River, NJ, USA, 1999.
[19] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M.S. Pérez, O. Corcho, Morph-KGC: scalable knowledge graph materialization with mapping partitions, Semant. Web (2022). https://doi.org/10.3233/SW-223135
[20] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M.S. Pérez, Ó. Corcho, Morph-KGC: scalable knowledge graph materialization with mapping partitions, Semant. Web 15 (1) (2024) 1–20. https://doi.org/10.3233/SW-223135
[21] E. Iglesias, M.-E. Vidal, D. Collarana, D. Chaves-Fraga, Empowering the SDM-RDFizer tool for scaling up to complex knowledge graph creation pipelines1, Semant. Web 16 (2) (2025) SW–243580. https://doi.org/10.3233/SW-243580
[22] E. Iglesias, S. Jozashoori, M. Vidal, Scaling up knowledge graph creation to large and heterogeneous data sources, J. Web Semant. 75 (2023). https://doi.org/10.1016/j.websem.2022.100755
[23] S. Jozashoori, M. Vidal, MapSDI: a scaled-up semantic data integration framework for knowledge graph creation, in: ODBASE, 2019.
[24] J.D. Ullman, Principles of Database and Knowledge-Base Systems, Volume I, 14 of *Principles of computer science series*, Computer Science Press, 1988. https://www.worldcat.org/oclc/310956623.
[25] E. Iglesias, M. Vidal, KGSaw: one size does not fit all-planning methods for data fragmentation for efficiently creating knowledge graphs, in: J. Hong, J.W. Park (Eds.), Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC 2024, Avila, Spain, April 8–12, 2024, ACM, 2024, pp. 1668–1670. https://doi.org/10.1145/3605098.3636186.
[26] E.I. Samaneh Jozashoori, M.-E. Vidal, SDM-Genomic-Dataset, 2022. https://figshare.com/articles/dataset/SDM-Genomic-Datasets/14838342?file=38623277. https://doi.org/10.57702/4c9ivpgs
[27] D. Van Assche, D. Chaves-Fraga, A. Dimou, KROWN: A Benchmark for RDF Graph Materialization, https://w3id.org/kg-construct/KROWN, (2024). https://doi.org/10.5281/zenodo.10979322
[28] C. Gutiérrez, J.F. Sequeda, Knowledge graphs, Commun. ACM 64 (3) (2021).
[29] T.M. Connolly, C. Begg, Database Systems: A Practical Approach to Design, Implementation, and Management, Addison-Wesley Longman Publishing Co., Inc., USA, 3rd edition, USA, 2001.

[30] M.T. Özsu, P. Valduriez, Principles of Distributed Database Systems, Second Edition, Prentice-Hall, 1999.
[31] A. Dimou, T. De Nies, R. Verborgh, E. Mannens, R. Van de Walle, Automated metadata generation for linked data generation and publishing workflows, in: Workshop on Linked Data on the Web, 2016.
[32] M. Freund, S. Schmid, R. Dorsch, A. Harth, FlexRML: a flexible and memory efficient knowledge graph materializer, in: A. Meroño Peñuela, A. Dimou, R. Troncy, O. Hartig, M. Acosta, M. Alam, H. Paulheim, P. Lisena (Eds.), The Semantic Web, Springer Nature Switzerland, Cham, 2024, pp. 40–56.
[33] J.F. Sequeda, On the semantics of R2RML and its relationship with the direct mapping, in: Proceedings of the 12th International Semantic Web Conference (Posters & Demonstrations Track) - Volume 1035, ISWC-PD '13, CEUR-WS.org, Aachen, DEU, 2013, p. 193–196.
[34] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, The RML ontology: a community-driven modular redesign after a decade of experience in mapping heterogeneous data to RDF, in: International Semantic Web Conference, Springer, 2023, pp. 152–175.
[35] J. Arenas-Guerrero, A. Iglesias-Molina, D. Chaves-Fraga, D. Garijo, O. Corcho, A. Dimou, Declarative generation of RDF-star graphs from heterogeneous data, Semant. Web (2024) 1–19. http://dx.doi.org/10.3233/SW-243602. https://doi.org/10.3233/sw-243602
[36] C.A. Knoblock, P. Szekely, J.L. Ambite, S. Gupta, A. Goel, M. Muslea, K. Lerman, M. Taheriyan, P. Mallick, Semi-Automatically mapping structured sources into the semantic web, in: Proceedings of the Extended Semantic Web Conference, 2012.
[37] A. Schultz, A. Matteini, R. Isele, C. Bizer, C. Becker, LDIF - linked data integration framework, in: O. Hartig, A. Harth, J.F. Sequeda (Eds.), Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23, 2011, 782 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2011. https://ceur-ws.org/Vol-782/SchultzEtAl_COLD2011.pdf.
[38] A. Jentzsch, R. Isele, C. Bizer, Silk - generating RDF links while publishing or consuming linked data, in: Proceedings of the 2010 International Conference on Posters & Demonstrations Track - Volume 658, CEUR-WS.org, Aachen, DEU, 2010.
[39] P.N. Mendes, H. Mühleisen, C. Bizer, Sieve: linked data quality assessment and fusion, in: Proceedings of the 2012 Joint EDBT/ICDT Workshops, EDBT-ICDT '12, Association for Computing Machinery, New York, NY, USA, 2012. https://doi.org/10.1145/2320765.2320803
[40] F. Castanedo, A review of data fusion techniques, Sci. World J. 2013 (1) (2013) 704504.
[41] M. Ye, X. Yan, X. Hua, D. Jiang, L. Xiang, N. Chen, MRCFN: A multi-sensor residual convolutional fusion network for intelligent fault diagnosis of bearings in noisy and small sample scenarios, Expert Syst. Appl. 259 (2025) 125214. https://www.sciencedirect.com/science/article/pii/S0957417424020815. https://doi.org/10.1016/j.eswa.2024.125214
[42] X. Yan, D. Jiang, L. Xiang, Y. Xu, Y. Wang, CDTFAFN: a novel coarse-to-fine dual-scale time-frequency attention fusion network for machinery vibro-acoustic fault diagnosis, Inf. Fusion 112 (2024) 102554. https://www.sciencedirect.com/science/article/pii/S1566253524003324. https://doi.org/10.1016/j.inffus.2024.102554
[43] M. Ye, X. Yan, D. Jiang, L. Xiang, N. Chen, MIFDELN: a multi-sensor information fusion deep ensemble learning network for diagnosing bearing faults in noisy scenarios, Knowl. Based Syst. 284 (2024) 111294. https://www.sciencedirect.com/science/article/pii/S0950705123010420. https://doi.org/10.1016/j.knosys.2023.111294
[44] L.J. Chen, P.A. Bernstein, P. Carlin, D. Filipovic, M. Rys, N. Shamgunov, J.F. Terwilliger, M. Todic, S. Tomasevic, D. Tomic, Mapping XML to a wide sparse table, IEEE Trans. Knowl. Data Eng. 26 (6) (2014) 1400–1414. https://doi.org/10.1109/TKDE.2012.221
[45] Z.H. Liu, B. Hammerschmidt, D. McMahon, JSON data management: supporting schema-less development in RDBMS, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 1247–1258. https://doi.org/10.1145/2588555.2595628
[46] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison Wesley, 1995.