KROWN: A Benchmark for RDF Graph Materialisation

Dylan Van Assche[1(✉)], David Chaves-Fraga[2,3], and Anastasia Dimou[3]

[1] IDLab, Dept Electronics and Information Systems, Ghent University – imec, Ghent, Belgium
dylan.vanassche@ugent.be

[2] Grupo de Sistemas Intelixentes, Universidade de Santiago de Compostela, Santiago de Compostela, Spain
david.chaves@usc.es

[3] Department of Computer Science, KU Leuven, Sint-Katelijne-Waver, Belgium
anastasia.dimou@kuleuven.be

Abstract. RDF graphs are commonly derived from (semi-)structured data by applying a set of mappings. RDF graph construction is performed by either materialising or virtualising an RDF graph and several benchmarks were purposed to measure their performance. However, even though significantly more materialisation systems exist, most benchmarks focus on virtualisation systems. Materialisation benchmarks are currently derived from virtualisation benchmarks, overlooking parameters that affect materialisation systems. In this paper, we introduce KROWN, a new benchmark to investigate the impact of datasets and mappings on RDF materialisation. We establish several benchmark scenarios with various scaling parameters to measure the execution time and computing resources, e.g., CPU time, and memory consumption. Through this work, we have now a benchmark suitable for materialisation systems which allows to execute each system in a reproducible pipeline through our execution framework. Thanks to our benchmark, we identified parameters which heavily influence the execution of materialisation systems and no optimizations were explored for them so far.

GitHub: https://github.com/kg-construct/KROWN
Results: https://zenodo.org/doi/10.5281/zenodo.10973891

Keywords: RML · Benchmark · Framework

1 Introduction

RDF graphs are often derived from heterogeneous (semi-)structured data, e.g., databases, JSON, CSV, or XML files. The last decade has seen a shift towards generating RDF graphs through a set of declarative mappings described through a mapping language [42] e.g., R2RML [15], RML [17,25], SPARQL-Generate [30], SPARQL-Anything [14]. RDF graphs are generated by either virtualisation or materialisation. *Virtualisation* uses a query and mappings to translate the query

G. Demartini et al. (Eds.): ISWC 2024, LNCS 15233, pp. 20–39, 2025.
https://doi.org/10.1007/978-3-031-77847-6_2

into an equivalent query over the data to generate a RDF graph, only answering that specific query. *Materialisation* uses mappings and some data to generate a RDF graph which is (re-)used then to answer multiple queries.

Many virtualisation benchmarks were proposed, but no materialisation benchmarks, despite the plethora of materialisation systems [42]. *Virtualisation benchmarks*, e.g., BSBM [9], NPD [28], GTFS-Madrid-Bench [13], evaluate the performance of virtualisation systems with respect to increasing data volume and complexity of queries. To the contrary, no *materialisation benchmarks* were proposed so far. Instead, most materialisation systems were evaluated by re-purposing virtualisation benchmarks [2], or by creating datasets to evaluate the performance of specific optimizations, such as joins [12], empty value removal [12,21], or RDF-Star [3]. Nevertheless, re-purposed virtualisation benchmarks do not evaluate all parameters that influence the performance of materialisation systems [12,16], while datasets evaluating specific optimizations do not provide a universal basis for comparison across all systems. Therefore, a materialisation benchmark is needed to cover both scaling data and mappings which constitute the input for materialisation systems with multiple scaling parameters.

In this paper, we introduce KROWN, a new benchmark for materialisation systems consisting of two parts: (i) a *data generator* to provide different materialisation scenarios, and, (ii) an *execution framework* to automatically execute reproducible benchmark pipelines. As a large majority of materialisation systems adopted the RDF Mappping Language (RML) [4,21,22,32,34,43], we consider this mapping language for our benchmark. Recently, a new version of the RML specification [25] was published by the W3C Community Group on Knowledge Graph Construction, as well as a new set of test cases [25] to assess the systems' compliance. This benchmark complements these resources, supporting the adoption of the RML mapping language and the development of systems with better performance and use of resources. We summarize our contributions:

1: Data Generator which allows scaling different materialisation scenarios based upon studied factors that influence materialisation systems [12,16].

2: Execution Framework which automatically executes reproducible benchmark pipelines and metrics measurement e.g., execution time, CPU time, and memory usage. Our data generator generates these pipelines for every scenario and system, offering a reproducible benchmark pipeline.

3: Evaluation of 5 popular materialisation systems: RMLMapper, RMLStreamer, Morph-KGC, SDM-RDFizer, and Ontop to show how our benchmark affects these systems in terms of performance and usage of computing resources.

KROWN's data generator and execution framework were already adopted as they were used in the Knowledge Graph Construction Challenge in 2023 [40] and 2024 [41]. KROWN has also been used with different pipelines in benchmarking incremental mapping execution [44], demonstrating its re-usability.

Thanks to KROWN, we observed limitations in materialisation systems that had not been previously uncovered by virtualisation benchmarks. Many materialization systems employ already optimizations to speed up their execution e.g., parallelization materialisation among multiple CPU cores, physical operators e.g., indexes to optimize duplicate or empty value removal, and planning through heuristics or partitioning of Triples Maps to group Logical Sources or

Table 1. Benchmarks and scenarios comparison. KROWN has more scaling parameters than other benchmarks and a stand-alone execution framework with flexible pipelines.

	Dataset	Parameters			Execution		Generator	
	Type	Size	Mapping	Join	Type	Pipeline	Own	VIG
BSBM	Real	✓			None		✓	✓
GTFS-Madrid	Real	✓			None			✓
ForBackBench	Re-use	✓			Built-in	Fixed	Re-use	
LUBM(4OBDA)	Real	✓			Built-in	Fixed	✓	
KROWN	Synthetic	✓	✓	✓	Separate	Flexible	✓	
KG Parameters	Synthetic	✓	✓	✓	None		✓	
GENOMICS	Real	None			None		None	
Reification	Real	✓			None		✓	

joins. However, not all aspects of materialisation are considered, e.g., the construction of RDF Quads involves Named Graphs which prevent parallelization optimisations of materialisation systems.

The paper is structured as followed: Sect. 2 outlines related work and Sect. 3 our KROWN benchmark. Section 5 presents the results and Sect. 6 discusses them. Section 7 concludes our work and future plans.

2 Related Work

In this section, we discuss existing (i) benchmarks, data generators, and scenarios (Sect. 2.1), and (ii) RDF materialization systems (Sect. 2.2).

2.1 Benchmarks, Data Generators, and Scenarios

In this Section, we discuss existing (i) benchmarks to evaluate RDF construction systems, (ii) data generators for generating benchmarks' data, and (iii) scenarios focusing on specific parameters. We consider a *benchmark* [27] as a standardised set of scenarios to compare systems in terms of performance or resource consumption. A *scenario* is a standardised setup for evaluating a system against multiple parameters with datasets generated through data generators [13]. Data generators are either entangled with the benchmark or re-usable among multiple benchmarks. The output is a new dataset scaled according to the parameter(s).

Benchmarks. Several benchmarks and scenarios (Table 1) were proposed over the past decade to evaluate materialisation systems, but most of them are (i) repurposed from existing virtualisation benchmarks, e.g., BSBM [9], GTFS Madrid Benchmark [13], NPD [28], LUBM(4OBDA) [5,19], or ForBackBench [1], (ii) or focus on specific parameters of materialisation e.g., duplicate and empty value removal (GENOMICS [26]). Some benchmarks feature an execution framework to run the benchmark in a reproducible environment, but these frameworks are either limited to a specific pipeline e.g., OBDA-Mixer [28] or require

re-implementing the system inside the framework e.g., ForBackBench. No re-usable execution framework exists with flexible pipelines and multiple scalable parameters, independently of the scenarios to benchmark.

The *Berlin SPARQL BenchMark (BSBM)* [9] assesses virtualisation systems and triplestores. It provides 3 use cases with different query mixes for query performance analysis. It has its own data generator which outputs dumps as RDF or SQL and allows scaling BSBM's data dumps using a scaling parameter, but its data generator is entagled in BSBM and is not re-usable for other benchmarks. A set of R2RML mappings are available to map the SQL data dumps into RDF which allows repurposing BSBM for materialisation benchmarking. However, only the dataset size (number of rows) can be scaled, the mappings remain the same. Therefore, the impact of the mappings cannot be evaluated.

The Norwegian Petroleum Directorate (NPD) benchmark. [28] provides a set of OBDA and R2RML mappings with 31 queries to analyze the performance and reasoning capabilities of virtualisation systems. It uses the VIG data generator [29] to scale the number of rows from the Norwegian Petroleum Directorate.

The *GTFS Madrid Benchmark (GTFS-Madrid-Bench)* [13] is a benchmark for virtualisation systems, but it was often repurposed to materialisation benchmark [4,22]. GTFS-Madrid-Bench uses the same VIG data generator as NPD and a set of SPARQL queries to analyse the performance of a system. It also allows generating data dumps in SQL, JSON, XML, or CSV formats by converting the output of the VIG data generator. R2RML and RML mappings are available for transforming the data dumps into RDF. Scaling individual parameters is not possible, only the number of rows is scaled and it does not provide an execution framework for a consistent evaluation environment among all systems.

The *ForBackBench* [1] analyzes performance of virtualisation systems and triplestores, with a focus on reasoning. It does not introduce new scenarios, but combines scenarios from NPD [28], OWL2Bench [35], Deep from chaseBench [7].

The *Lehigh University Benchmark (LUBM)* [19] is a benchmark for triplestores with 14 queries about university data. LUBM4OBDA [5] extends LUBM to analyze virtualisation systems with RDF-Star support through a set of additional queries for standard reification, singleton property or RDF-Star.

Data Generators. Several data generators were proposed for benchmarks, some are stand-alone, e.g., VIG while others are specific for the benchmark, e.g., UBA. The *VIG generator* [29] provides a stand-alone re-usable data generator for benchmarks. However, VIG only scales with a single parameter for the data size; the RML mappings are not scaled. The *UBA data generator* [5,19] generates real-life data about universities for the LUBM and LUBM4OBDA benchmarks. Only the number of universities scales, the mappings are static. LUBM4OBA extended the original UBA data generator for RDF reification and RDF-Star.

Scenarios. *KG Parameters scenario* [12] consists of various scenarios to analyze the impact of dataset size (row/columns), mappings, joins, empty values, and duplicates on materialisation systems. A set of RML mappings is provided to

map CSV files into RDF. The data is generated through scripts as CSV files with fixed scaling parameters for each scenario. However, it does not consider Named Graphs (NG) or cell size as scaling parameters. KROWN extends KG Parameters' scenarios with named graphs and cell size parameters, offering a complete benchmark with a scalable data generator and an execution framework and measuring resource consumption e.g., CPU/RAM, besides the execution time.

The *GENOMICS scenario* [26] investigates the impact of empty values and duplicates on materialisation systems. It does not have a data generator, thus it cannot be scaled, instead a fixed CSV dump is provided together with RML mappings to generate RDF. The *Reification scenario* [24] provides a set of scenarios to investigate RDF-Star's impact on RDF materialisation. The data size can be scaled and multiple versions of the mappings are provided to investigate RDF-Star compared to standard reification, named graphs, and N-Ary Relationships. Although this scenario focuses on materialisation, it only covers RDF-Star and standard reification.

Execution Frameworks. Some benchmarks provide execution frameworks to have a reproducible environment and pipeline for the benchmark. *UBT* is used in LUBM and executes a fixed virtualisation pipeline to perform data loading and querying with configurable test plans. *OBDA-Mixer* [28] also executes a fixed virtualisation pipeline to load the relational database and it was used in the NPD benchmark for automatized execution. *ForBackBench execution framework* executes multiple scenarios with a fixed pipeline and it has been used in ForBackBench to analyze chasing vs query rewriting. However, they cannot be re-used for constructing alternative pipelines, since they are limited to virtualisation pipelines consisting of a relational database, the system, and query execution.

2.2 Materialization Systems

A recent survey [42] showed that different systems for RDF materialisation exist to construct RDF graphs from heterogeneous data. These systems leverage different mapping languages, such as *dedicated mapping languages*, e.g., R2RML [15], RML [17,25], *query-based mapping languages* e.g., SANSA/RPT [36], SPARQL-Generate [30], SPARQL-Anything [6], or *constraints-based mapping languages*, e.g., ShExML [18]. We focus on materialisation systems with RML since multiple systems exist compared to systems for other mapping languages. In RML, a Triples Map (TM) creates RDF Triples, a Subject Map (SM) creates the subject, a Predicate Object Map (POM) the predicate and the object and a Graph Map (GM) creates the named graph of an RDF Quad.

RMLMapper [17] executes mappings by iterating over Triples Maps. It performs no optimizations: it does not push any task to the relational database, and performs the joins in memory as a double nested for-loop.

Ontop [10] is a virtualisation system which also supports materialisation. It implements well-known SQL [33] and join [11,46] optimisations, and push down

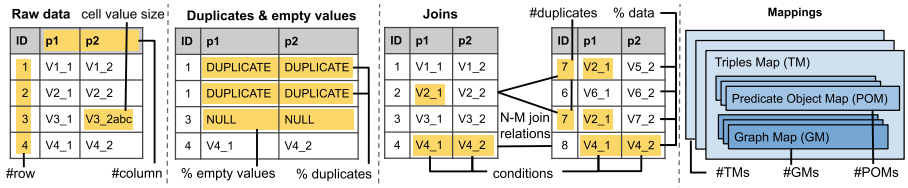

Fig. 1. KROWN's benchmark scenarios: raw data, duplicates & empty values, joins, and mappings. Each scaling parameter per scenario is highlighted.

of SPARQL functions to the relational database [10]. As a materialization system (abbreviated as 'OntopM'), it transforms each Triples Map into a SQL query which is executed against the relational database, and normalises it with 1 Logical Table, 1 Subject Map, and 1 Predicate Object Map.

SDM-RDFizer [21] employs physical data structures to optimize the execution of joins through Predicate Join Tuple Maps which perform all joins as index based joins [21]. The removal of duplicates and empty values is pushed to RDBMS by leveraging the `DISTINCT` SQL operator and a dedicated data structure (Predicate Join Tuple Table) is used to avoid duplicates during joins.

RMLStreamer [20,32] executes RML mappings in streaming fashion using Apache Flink. It groups Triples Maps based on their Logical Source and provides 2 modes: *cluster mode* to scale horizontally among machines and vertically among CPU cores, and *stand-alone mode* to scale vertically among CPU cores on the same machine.

Morph-KGC [2] relies on mapping planning to process large amounts of data at scale through parallelization and ensure that no duplicate RDF triples are generated during parallel execution of mapping partitions. It leverages Python's Pandas to perform operations on data, e.g., removing duplicates before executing the mappings and during joins, and uses the `IS NOT NULL` SQL operator to delegate the removal of empty values to RDBMS, but not the joins' execution.

CARML [31] supports RML, but it does not allow to use relational databases as data source yet. It also employs a streaming based process, similar to RML-Streamer, but without scaling horizontally among machines.

3 KROWN: Knowledge Graph Construction Benchmark

In this Section, we introduce (i) the benchmark scenarios we generate, (ii) the data generator for generating synthetic data, and (iii) the execution framework for a reproducible execution of the benchmark scenarios against any system.

3.1 Benchmark Scenarios

In this work, we consider several benchmark scenarios (Fig. 1): (i) raw data, (ii) duplicates and empty values, (iii) mappings, and (iv) joins based on the

parameters introduced by Chaves et al. [12] and Dimou et al. [16], and new parameters e.g., cell size for raw data and Graph Maps (GM) for mappings. Our scenarios can be scaled indefinitely with a set of parameters through our data generator and are executed with our execution framework. Samples of each scenario are available in KROWN's repository [38].

The ***Raw Data scenario*** contains 3 parameters: number of rows, number of columns, and cell size to investigate the data size impact on RDF construction. KROWN currently uses references in the generated mappings to refer to the data. IRI templates, different data types, and language tags are planned for the future. The ***number of rows*** scales the data *vertically* by increasing the number of rows. The higher the number of rows, the more data needs to be processed. The ***number of columns*** scales the data *horizontally* by increasing the number of columns per row. The higher the number of columns, the more data per row needs to be processed. The ***cell size*** scales the actual cell value that is present in the data. The higher the cell size, the bigger a cell is to process.

The ***Duplicates and Empty Values scenario*** benchmarks the impact of duplicated and empty values in a dataset on RDF construction as more duplicated and empty values require RDF triplestores to remove them during the loading phase. The percentage of ***duplicate values*** scales the amount of duplicates in the data. KROWN's data generator use the same duplicated value across the data, but different variants are planned where multiple duplicated values are used across the data. Similarly, the percentage of ***empty values*** scales the amount of empty values in the data. The higher the percentage, the more empty or duplicate values are present which need to be removed.

The ***Mappings scenario*** assesses the impact of different types of RML mappings with the following parameters: number of Triples Maps (TM), Predicate Object Maps (POM), and Graph Maps (GM). The ***number of Triples Maps*** (TMs) specifies how many TMs exist in the RML mapping. The higher the number of TMs, the more different RDF subject terms will appear in the RDF graph. The ***number of Predicate Object Maps*** (POMs) specifies how many POMs are specified in the RML mapping. The higher the number of POMs, the more predicates and objects are generated for a subject. The ***number of Graph Maps*** (GMs) specifies how many GMs are specified for TMs and POMs. If multiple GMs are specified for the same RDF Subject/Predicate/Object, multiple RDF Quads are generated as each RDF Quad has its own Named Graph (NG). For example: if 2 NGs are specified for the same RDF terms, the RDF terms will be generated twice, each one in a different NG. NGs can be static (`rr:constant`) or dynamic from the dataset (`rr:template`) and may appear in a a Subject Map (SM) for the whole TM or in each POM.

The ***Joins scenario*** consists of several parameters: amount of data involved in joins, join relations, multiple join conditions, and duplicates involved in joins. The ***amount of data*** involved in joins scales the percentage of the data that results in a join to investigate the join selectivity. The higher the percentage, the more data which will result in a join. The ***join relations*** parameter scales the N-M join relations to investigate the impact of different types of join relations. The

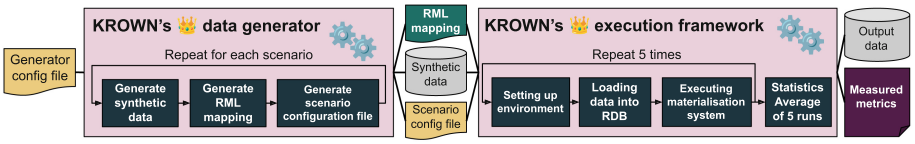

Fig. 2. KROWN's benchmark pipeline consists of a data generator and execution framework which both can be used stand-alone.

higher the N or M relation values, the more data results in a join. For example: N=1 and M=5 (1–5) results in every join with 1 entry from the dataset and 5 entries from the other dataset. The ***join conditions*** parameter scales the number of join conditions that must be valid to result into a join. More join conditions means more comparisons to perform for the data to check if it should result into a join or not. The ***join duplicates*** parameter scales the number of duplicates that are generated by a join. The more duplicates are generated by a join, the more duplicates that must be removed.

3.2 Data Generator

KROWN's data generator (Fig. 2) uses synthetic data with fully configurable parameters for each scenario. It generates the datasets, mappings, and pipeline of any scenario with any RDF construction system, e.g., raw data (rows, columns, cell size), RML mappings (TMs, POMs, GMs), duplicated and empty values, and joins (1–1, 1-N, N-1, N-M relations, duplicates, conditions, selectivity).

Our data generator generates synthetic data to scale the different materialisation parameters for our scenarios. It generates R2RML mappings to remain compatible with all systems as each R2RML mapping can be transformed into a RML mapping by replacing the R2RML's Logical Table with a RML's Logical Source. Some systems use R2RML's Logical Table description to generate optimised SQL queries for the RDBMS. Therefore, KROWN is configured in this work to generate tabular data, but it can be extended to support other data formats as well e.g., JSON or XML. KROWN's data generator can scale one or multiple parameters by a set of configuration files and new scenarios can be added, re-using the existing synthetic data generation. It extends the proposed parameters further by adding cell size parameter for raw data scaling and GMs for RML mappings. We add cell size scaling because most RDF construction systems optimise for row and column based parallelisation without taking into account the size of the cell in a row/column. GMs are added to investigate the impact of RDF Quads generation upon system since existing optimisations focus on TMs and POMs without considering NGs. Through our data generator, any parameter of each scenario can be scaled indefinitely which allows us to benchmark systems in the future against bigger parameter scalings while using the same data generator. We focus on synthetic data to analyse the impact of each scaling parameter on materialisation systems, but KROWN can also handle real-world data such as GTFS-Madrid-Bench [13].

3.3 Execution Framework

KROWN has also a stand-alone execution framework (Fig. 2) to execute pipelines based on Docker containers in a reproducible way. During execution, the framework automatically measures metrics e.g., execution time, CPU time, and memory consumption. The execution framework initialises all resources e.g., relational databases, systems, etc. Afterwards, the configurable pipeline is executed multiple times. After all pipelines are executed, statistics are calculated e.g., min, max, median on every measured metric. Since our pipelines can be configured, our execution framework is not limited to pipelines in this work.

4 Evaluation

In this Section, we describe materialisation systems we benchmark, our benchmark configuration, and our experimental setup.
Materialisation Systems. We consider the most popular [R2]RML systems in our benchmark which are actively maintained, open source, and support SQL databases: RMLMapper v6.0.0 [17] as a baseline without any optimisations, SDM-RDFizer v4.6.3.4 [21] for its joins and duplicate/empty value optimisations, Morph-KGC v2.1.1 [2] due to its parallelisation optimisation, RMLStreamer v2.5.0 [20,32] in stand-alone mode because of its streaming capabilities. We do not use cluster mode as it requires more machines which would be unfair for the other systems. Support for cluster mode is planned for the future. To compare the RMLStreamer for its joining capabilities, we execute it with CSV data (*RMLStreamer-CSV*) because its SQL support cannot handle joins properly with SQL databases yet, timing out on all cases related to joins. For all other scenarios, we use RMLStreamer with a SQL database. Join selectivity scenario did not provide interesting results due to optimisations (see Zenodo [39]). OntopM v4.2.0 [10] as it is a well established virtualisation system but we assess its capacity as materialisation system too. SANSA/RPT [36] and CARML [31] were excluded because they do not support SQL databases yet.

Experimental Setup. We select certain values for KROWN's scenario parameters to benchmark RDF materialisation systems, but KROWN is not limited to these values. Table 2 lists the values we use in our evaluation for each scenario. We use the measured metrics (execution time, CPU time, and peak memory usage) by KROWN's execution framework in our evaluation. Measured metrics are the total resource consumption of both the RDBMS and materialisation system.

We execute our experiments on Ubuntu 22.04 LTS machines (Linux 5.15.0, x86_64) with each Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz, 48 GB RAM memory, and 2 GB swap memory. All experiments are executed 5 times with a time-out of 6 h from which we report the median value of the metrics. Output of each RDF construction system is set to N-Triples or N-Quads in case of Named Graphs because it is supported by all systems we evaluate. We use PostgreSQL 14.5 as RDBMS without configuring any additional RDBMS indexes. Generated RDF is checked by isomorphic comparison for each benchmark scenario using

Table 2. Parameter values for each scenario in our evaluation.

Scenario	Parameter values
Raw data: rows	10K, 100K, 1M, 10M
Raw data: columns	1, 10, 20, 30
Raw data: cell size	500, 1K, 5K, 10K
Duplicates: percentage	0%, 25%, 50%, 75%, 100%
Empty values: percentage	0%, 25%, 50%, 75%, 100%
Mappings: TMs + 5POMs	1, 10, 20, 30 TMs
Mappings: 20TMs + POMs	1, 3, 5, 10 POMs
Mappings: NG in SM	1, 5, 10, 15 NGs
Mappings: NG in POM	1, 5, 10, 15 NGs
Mappings: NG in SM/POM	1/1, 5/5, 10/10, 15/15 NGs
Joins: 1-N relations	1–1, 1–5, 1–10, 1–15
Joins: N-1 relations	1–1, 5–1, 10–1, 15–1
Joins: N-M relations	3–3, 3–5, 5–3, 10–5, 5–10
Joins: join conditions	1, 5, 10, 15
Joins: join duplicates	0, 5, 10, 15

a small scaling parameter and a triple count on larger scaling parameters due to memory constraints. All systems involved in the benchmarks are executed as Docker containers and configured according to their documentation. KROWN's resources are available on DockerHub [37], GitHub [38], and Zenodo [39].

5 Results

In this section, we discuss the results (Table 3) obtained from executing KnowBench on the materialisation systems we selected in Sect. 4.

5.1 Raw Data Scenario

We present the raw data scenario results (Table 3) for scaling the number of rows, columns, and cell size in terms of execution time, CPU time, and memory.

Execution Time. Morph-KGC is the fastest materialisation system because it parallelises its execution through partitioning of the Triples Maps. SDM-RDFizer executes bigger value sizes (10K), but it is slower than Morph-KGC. OntopM can also execute this scale and other scalings, but slower than SDM-RDFizer. Both OntopM and SDM-RDFizer finish all the scalings while others run out of memory (Morph-KGC, RMLMapper) or time out (RMLStreamer). RMLStreamer is also slower than Morph-KGC, but does not run out of memory due to its streaming capabilities as Morph-KGC, instead it times out for the bigger scalings.

CPU Time. RMLMapper consumes the least amount of CPU resources because it does not optimize its execution through parallelisation or planning compared

Table 3. Results for the raw data, duplicates & empty values, and TM+POM scenarios. RMLMapper, Morph-KGC, and SDM-RDFizer run out of of memory (OOM) and RMLStreamer times out with increasing value sizes.

		MorphKGC			OntopM			RMLMapper			SDM-RDFizer			RMLStreamer		
		Time (s)	CPU (s)	RAM (GB)	Time (s)	CPU (s)	RAM (GB)	Time (s)	CPU (s)	RAM (GB)	Time (s)	CPU (s)	RAM (GB)	Time (s)	CPU (s)	RAM (GB)
Columns	1	18.59	23.07	2.02	47.07	64.43	4.91	23.64	43.22	9.01	494.27	536.46	2.25	25	278.65	6.78
	10	23.85	196.43	7.90	466.89	508.92	16.83	83.72	163.42	12.71	520.64	553.53	4.22	48.61	808.61	18.69
	20	34.62	511.19	12.92	943.35	995.21	16.86	155.65	310.00	15.89	707.9	747.33	7.41	69.22	1329.52	19.75
	30	47.01	1095.95	18.14	1417.65	1490.91	16.89	236.86	456.53	19.32	971.74	1020.43	10.86	90.15	1844.62	20.75
Rows	10K	4.94	16.43	2.07	15.33	31.36	3.14	5.28	13.70	2.58	9.72	13.71	1.45	19.17	130.70	3.77
	100K	7.67	61.42	3.27	106.87	132.49	15.48	21.43	33.58	7.97	76.96	86.21	1.82	24.55	263.02	7.88
	1M	34.62	511.19	12.92	943.35	995.21	16.86	155.65	310.00	15.89	707.9	747.33	7.41	69.22	1329.52	19.75
	10M	OOM			9613.90	10009.57	16.9	OOM			OOM			TIME OUT		
Value size	500	21.28	51.94	5.00	72.51	89.91	7.71	42.93	78.42	11.67	69.52	71.25	3.12	39.99	373.44	20.01
	1K	34.93	71.11	8.93	95.37	109.67	14.54	67.73	213.57	18.9	86	84.26	4.03	56.53	638.43	23.23
	5K	93.06	230.91	32.31	150.99	166.32	18.07	OOM			176.08	176.67	11.61	TIME OUT		
	10K	OOM			377.43	406.89	22.40	OOM			273.57	266.79	21.32	TIME OUT		
Duplicates	0%	13.38	67.44	3.53	99.04	119.77	8.28	25.3	34.84	8.00	85.22	91.05	1.80	26.73	254.22	11.62
	25%	11.34	62.43	3.35	78.33	96.17	6.74	23.9	32.03	7.94	76.63	82.15	1.69	26.39	252.75	8.89
	50%	10.48	57.56	3.21	53.25	70.18	4.90	22.7	30.45	7.92	71.87	77.59	1.57	26.52	258.50	8.81
	75%	9.82	51.3	3.07	32.45	46.98	3.70	21.55	29.84	7.90	69.45	73.75	1.44	25.72	254.85	7.87
	100%	8.92	46.13	2.91	8.65	17.33	1.94	17.09	24.67	7.87	64.87	68.31	1.33	26.46	250.41	8.87
Empty values	0%	13.53	70.09	3.54	101.17	119.81	8.49	25.50	33.39	8.03	80.51	85.83	1.82	26.54	254.92	11.05
	25%	12.42	56.94	3.31	77.22	97.46	6.37	23.01	30.24	7.95	68.84	73.2	1.73	25.55	235.14	8.05
	50%	11.1	38.41	3.13	53.23	70.17	5.21	19.29	27.14	7.99	58.83	63.01	1.68	24.98	219.49	8.31
	75%	9.93	27.04	2.77	31.5	46.69	3.79	16.9	24.38	7.91	63.81	67.91	1.56	24.35	214.78	7.24
	100%	8.11	13.92	1.96	7.68	16.29	2.05	13.84	19.97	7.76	82.79	85.52	1.47	22.71	179.48	7.48
TM+POM	1+5	10.00	17.36	1.75	30.29	51.19	3.45	14.32	21.16	6.50	72.43	80.93	1.32	23.54	171.66	4.07
	10+5	29.39	97.32	2.62	242.19	267.55	16.73	55.95	83.34	9.75	752.96	774.02	1.79	35.55	416.36	17.71
	20+5	54.16	190.26	3.46	498.69	537.29	16.8	99.48	152.76	11.63	1519.78	1560.95	2.44	43.06	680.06	17.72
	30+5	74.71	288.94	4.45	762.16	809.1	16.81	141.17	219.22	11.99	2256.31	2309.72	3.43	53.26	942.34	17.74
	20+1	40.88	45.67	1.66	100.61	121.3	8.20	42.81	70.04	9.45	862.25	917.91	1.58	26.90	241.74	6.58
	20+3	43.43	110.62	2.59	290.31	317.82	16.80	74.1	119.63	10.62	1312.42	1364.94	2.01	36.46	462.14	17.70
	20+5	54.16	190.26	3.46	498.69	537.29	16.80	99.48	152.76	11.63	1519.78	1560.95	2.44	43.06	680.06	17.72
	20+10	75.58	420.83	5.75	1059.15	1115.57	16.81	162.67	246.29	12.32	1472.72	1509.70	3.67	61.86	1202.76	17.70

to the other systems. Morph-KGC uses the most CPU time during its short execution time (burst), OntopM uses more but spread over a longer period of time. RMLStreamer consumes the most CPU time because of its parallelisation features. Flink – on which RMLStreamer is based upon – parallelises its tasks among multiple CPU cores resulting in higher CPU time usage. While Morph-KGC also uses parallelism for its execution, Morph-KGC is more efficient in CPU time usage compared to RMLStreamer.

Memory. SDM-RDFizer uses the least amount of memory in exchange for a longer execution time. Morph-KGC is faster, but runs out of memory in bigger scales as it loads all data for each parallel process separately, consuming more memory, while OntopM does not increase its memory usage heavily as the scale increases. RMLStreamer's memory usage increases with the scaling because SQL databases are not optimized in the same way as CSV files [45]. RMLMapper runs out of memory faster because it loads everything in memory.

5.2 Duplicate and Empty Values Scenario

We present the results of the duplicate and empty value scenario (Table 3) for each system in terms of execution time, CPU time, and memory consumption.

Execution Time. Morph-KGC is the fastest system when scaling the amount of duplicates or empty values in the dataset because Python's Pandas removes

duplicates and empty values efficiently. RMLStreamer uses the same amount of time for any scaling as it does not remove duplicates, while the RMLMapper becomes faster with a higher scaling for both duplicates and empty values, because it processes less data. SDM-RDFizer also becomes faster when more duplicates are involved because it optimises for duplicate removal. In the case of empty values, it reaches its optimum with 50% empty values. Despite SDM-RDFizer's optimisations, Morph-KGC is still faster. OntopM reduces its execution time when more empty values or duplicates are involved, ultimately surpassing Morph-KGC on scaling 100%, by pushing down the removal to the relational database.

CPU Time. Almost all systems reduce their CPU time with more duplicates or empty values involved, except for SDM-RDFizer. SDM-RDFizer performs extensive execution planning for empty values and duplicates removal which affects its CPU time usage.

Memory. SDM-RDFizer consumes the least amount of memory, increasing the scaling results into even lower memory consumption, thanks to its memory optimisations. RMLMapper's memory consumption is rather similar no matter the scaling because all data is loaded into memory. OntopM lowers its memory consumption significantly with more duplicates or empty values, removing duplicates and empty values from its memory. Morph-KGC and RMLStreamer slightly lower their consumption, but not significant.

5.3 Mapping Scenario

We present the mapping scenario results for each system for (i) number of Triples Maps, (ii) number of Predicate Object Maps, and (iii) Graph Maps.

Triples Maps and Predicate Object Maps. scenarios scales either the number of TMs or POMs with a fixed data size (Table 3).

Execution Time. All systems increase their execution time when more Triples Maps or Predicate Object Maps are defined in the RML mappings. RMLStreamer scales the best among Triples Maps and Predicate Object Maps because Flink optimises the execution in parallel better than Morph-KGC. While Morph-KGC is faster with smaller mappings, RMLStreamer becomes faster from 20 Triples Maps and higher. Morph-KGC and RMLStreamer both double their execution time when the number of Predicate Object Maps become a magnitude bigger. OntopM's execution time increases equally with the number of TMs or POMs since OntopM normalises the mappings to have multiple TMs with each 1 POM. RMLMapper and SDM-RDFizer increase their execution time faster when the number of TMs increases compared to the number of POMs.

CPU Time. Systems parallelizing the execution of TMs and POMs (RMLStreamer, Morph-KGC) increase their CPU time usage more compared to other systems in exchange for a lower execution time. OntopM's CPU time increase linearly with the number of POMs, but not with the number of TMs. RMLMapper and SDM-RDFizer have a steady increase in CPU time for both scalings.

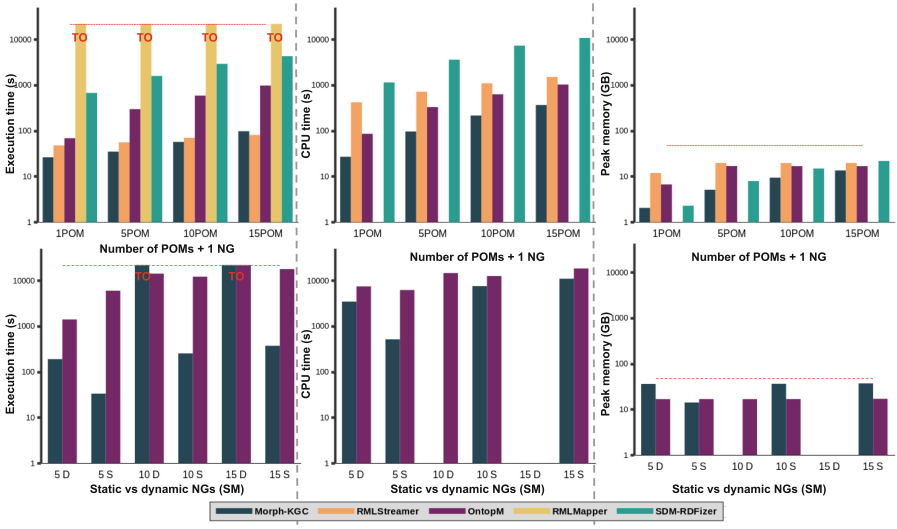

Fig. 3. Results for the Graph Maps subscenarios: scaling the number of POMs with 1 Named Graph (NG) (top) and scaling the number of NGs from 5 to 15 Statically (S) and Dynamically (D) in a Subject Map (bottom). RMLMapper always times out, RMLStreamer does not support multiple GMs. SDM-RDFizer fails the multiple GMs with an error. All systems fail or time out (TO) after 6 h the 15NG dynamic case.

Memory. SDM-RDFizer remains the most efficient system regarding memory due to its memory optimisations in exchange for a slower execution time. RML-Streamer has a steady memory consumption once it scales past the smallest scaling because of its streaming capabilities. OntopM also has a fixed memory usage. Morph-KGC increases its memory consumption by a factor of 3–4 times when scaling up because it loads everything in memory for each parallel process.

Graph Maps. (GM) scenario scales the number of NGs in SMs and POMs. We also scale the number of POMs with a single Graph Map (Fig. 3).

Execution Time. All systems[1] increase their execution time with more GMs generating NGs. When the NG is a constant value (`rr:constant`), the execution time is lower than when the NG's IRI is generated dynamically (`rr:template`). Morph-KGC times out when a SM consist of 10 or 15 dynamical GMs because

[1] RMLStreamer only supports 1 GM/SM or POM, thus we cannot investigate its performance when the number of GMs increase.

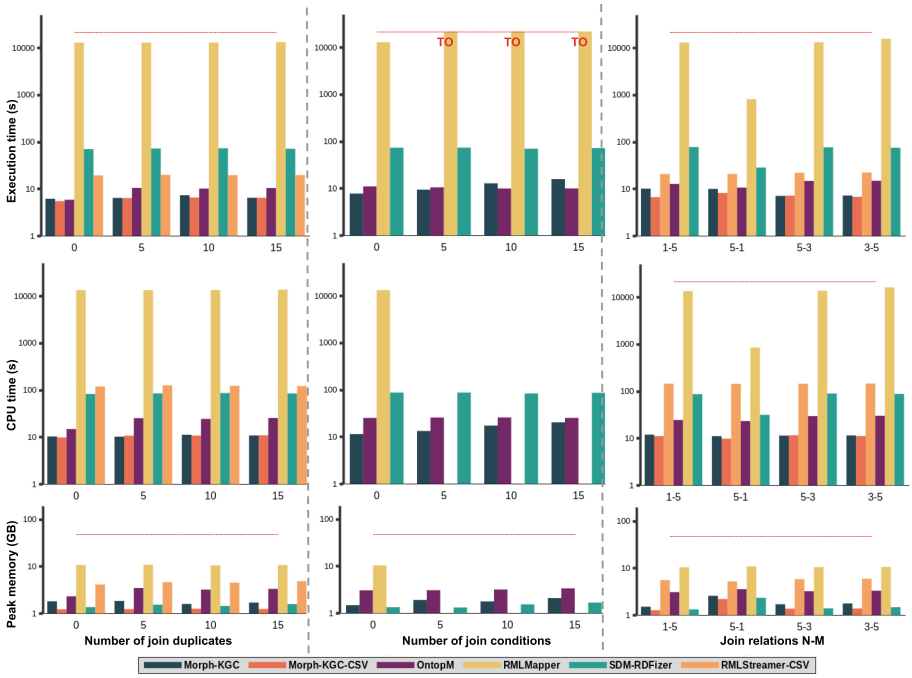

Fig. 4. Results for join scenarios: number of join duplicates (left), number of join conditions (middle), and join relations N-M (right). RMLStreamer-CSV is excluded from number of join conditions because it does not support multiple join conditions. RMLMapper times out (TO) after 6 h for 5,10, 15 join conditions.

it cannot apply its parallelism optimisation. OntopM also times out with 15 dynamic GMs in a SM. RMLMapper times out in all scalings.

CPU Time. All systems increases their CPU time usage with more GMs in SM or POMs. Morph-KGC and OntopM are the most affected in CPU time usage.

Memory. SDM-RDFizer's memory optimisations do not consider GMs, its memory usage increases by a magnitude when increasing the number of POMs from 1 to 15 with 1NG. SDM-RDFizer fails to execute mappings with multiple dynamic GMs with an error when assigning the GM to a TM in its code. Morph-KGC increases its memory usage heavily when more GMs are involved while RMLStreamer and OntopM remain consistent no matter the number of GMs.

5.4 Joins Scenario

We present (Fig. 4) for each system how it performs regarding (i) N-M join relations, (ii) join conditions, and (iii) duplicates generated during a join.

Execution Time. Morph-KGC is the fastest system, OntopM is almost as fast as Morph-KGC, followed by RMLStreamer and SDM-RDFizer, RMLMapper is the slowest. Morph-KGC leverages Python's Pandas to execute joins which are optimized compared to RMLMapper. RMLStreamer does the same, but uses Apache Flink instead. OntopM has also join optimisations from the relational database. Execution time is affected by the N-M join relation, a N-1 join is faster than a 1-N or N-M relation. If a join has multiple conditions, Morph-KGC's execution time increases because it uses a different join algorithm of Pandas with multiple join conditions. OntopM is unaffected as the join is pushed to the relational database. SDM-RDFizer is unaffected because of its physical operators[2]. RMLMapper times out with multiple join conditions.

CPU Time. RMLMapper consumes the most CPU time with joins because it has no optimisations for executing joins. All other systems have optimisations (push to relational database, physical operators) which reduces the CPU time.

Memory. All systems' memory usage is affected by the N-M relation. A N-1 relation has a lower execution time, but a higher memory consumption since join results are kept in a Predicate Join Tuple Table by SDM-RDFizer, in a cache by RMLMapper, etc. All systems have a similar memory consumption for duplicates during joins or multiple join conditions.

6 Discussion

As we observe from the results (Sect. 4) we got after benchmarking different materialisation systems with KROWN, materialisation systems employ parallelisation e.g., Morph-KGC [2], RMLStreamer [20], to reduce execution time in favor of a higher CPU time and memory consumption. Some systems e.g. SDM-RDFizer [21] reduce their memory consumption in favor of execution time by not keeping all data in memory. However, access then to the data is slower because it must be loaded first into memory.

Physical operators e.g., hashtable indexes for duplicate and empty value removal, and planning e.g., partitioning of Triples Maps are employed by systems, e.g., SDM-RDFizer, Morph-KGC, to reduce their execution time and memory consumption. By planning the execution properly, the data has to be loaded only once, even when joins are involved. This way, the slow access time from disk only occurs once while memory consumption is reduced as the data can be removed from memory once it is not needed anymore according to the planning.

However, optimisations (Morph-KGC, RMLStreamer, SDM-RDFizer, OntopM) cause additional overhead when executing mappings such as increasing CPU time Systems not applying any optimisations (RMLMapper [17]) have a lower CPU time for smaller datasets without any joins.

While many optimisations have improved materialisation systems over the past decade, they do not consider all aspects of datasets and mappings for materialisation systems, e.g., parallelisation of mapping execution is not possible when

[2] RMLStreamer does not support multiple join conditions.

Named Graphs occur in the mapping. Thanks to KROWN, we observe that existing parallelisation approaches, e.g., Morph-KGC, cannot handle Named Graphs in mappings, since they only consider RDF Triples instead of RDF Quads when applying parallelisation. Furthermore, physical operators and planning optimisations for memory consumption, e.g., SDM-RDFizer, only consider the row and column of datasets without the size of the actual cell. If the cell size increases, the memory consumption increases accordingly.

Beyond the evaluation's results of our KROWN benchmark, KROWN was also applied during the Knowledge Graph Construction Challenge 2023 [40] & 2024 [41]. In this Challenge, participants use scenarios generated by KROWN's data generator to benchmark their RDF construction systems with KROWN's execution framework. KROWN's execution framework captures the metrics e.g., execution time, CPU time, and memory usage for participants. Through KROWN, participants identified bottlenecks in their systems and improved them for the next edition of the Challenge. Moreover, KROWN allows to customize the materialisation pipeline and select the right system for a task. Based on the feedback of the ESWC 2023 edition, we improved KROWN and noticed that participating systems [8,23,31,45] improved e.g., RMLStreamer gained SQL support while CARML works on it, SDM-RDFizer received stability improvements and reported issues with KROWN's execution framework, and SANSA/RPT gained initial RML support. KROWN's execution framework is also re-used in IncRML [44] measuring the performance of incremental RML mappings on data changes. KROWN's sustainability plan [38] relies on the community to further expand KROWN for newer editions of the Challenge for gaining scenarios to benchmark FnO functions, RDF-Star, RDFS Collections and Containers which are introduced by the new RML specification [25], more benchmark scenarios e.g., language tags, data types, duplicate value variants, or IRI templates, and support for benchmarking across multiple machines. While we focus in this work on materialisation systems, KROWN's could also be applied on virtualisation systems. KROWN already supports RDF triplestores and SPARQL queries.

7 Conclusion

In this paper, we introduce KROWN, a benchmark for RDF materialisation systems to evaluate their performance and usage of computing resources e.g. CPU time and memory. Our benchmark contains a data generator to scale any scenario unlimited through a set of scaling parameters e.g. data size, mappings and an execution framework for executing the scenarios in a reproducible way.

Through KROWN, we discover 2 limitations: (i) Named Graphs, and (ii) cell size. Named Graphs are not considered in existing RDF materialisation optimisations, causing them to fail, resulting in increased execution time and consumption of computing resources. Since optimisations slice data by row and column, the cell size is not considered, resulting in out-of-memory situations.

Further research includes applying KROWN on more systems and even bigger scalings such as data formats (e.g. XML, JSON), sizes, mappings, and joins to

push the most performant systems to their limits. This way, we open the path towards more optimisations for both computing resources and execution time.

Resource Availability Statement. KROWN's source code and documentation are available on GitHub [38]. KROWN benchmark's results are available on Zenodo [39]. Docker images for the experiments are available on DockerHub [37].

Acknowledgements. Dylan Van Assche is funded by the Special Research Fund of Ghent University under grant BOF20/DOC/132. David Chaves-Fraga is funded by the Galician Ministry of Education, University and Professional Training and the European Regional Development Fund (ERDF/FEDER program) through grants ED431C2018/29 and ED431G2019/04. Anastasia Dimou is partially supported by Flanders Make (REXPEK project), the strategic research centre for the manufacturing industry and the Flanders innovation and entrepreneurship (VLAIO – KG3D project). The researchers' collaboration is stimulated by the KG4DI FWO scientific research network (W001222N).

References

1. Alhazmi, A., Blount, T., Konstantinidis, G.: ForBackBench: a benchmark for chasing vs. query-rewriting. Proc. VLDB Endow. **15**(8), 1519–1532 (2022). https://doi.org/10.14778/3529337.3529338
2. Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., Pérez, M.S., Corcho, O.: Morph-KGC: scalable knowledge graph materialization with mapping partitions. Semant. Web **15**(1), 1–20 (2024). https://doi.org/10.3233/SW-223135
3. Arenas-Guerrero, J., Iglesias-Molina, A., Chaves-Fraga, D., Garijo, D., Corcho, O., Dimou, A.: Declarative generation of RDF-star graphs from heterogeneous data. Semant. Web **Pre-press**, 1–19 (2024)
4. Arenas-Guerrero, J., et al.: Knowledge graph construction with R2RML and RML: an ETL system-based overview. In: CEUR Workshop Proceedings, vol. 2873. CEUR Workshop Proceedings (2021)
5. Arenas-Guerrero, J., Pérez, M.S., Corcho, O.: LUBM4OBDA: benchmarking OBDA systems with inference and meta knowledge. J. Web Eng. **22**(8), 1163–1186 (2024). https://doi.org/10.13052/jwe1540-9589.2284
6. Asprino, L., Daga, E., Gangemi, A., Mulholland, P.: Knowledge graph construction with a Façade: a unified method to access heterogeneous data sources on the web. Trans. Internet Technol. (2022). https://doi.org/10.1145/3555312
7. Benedikt, M., et al.: Benchmarking the chase. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pp. 37–52. PODS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3034786.3034796
8. Bin, S., Stadler, C.: KGCW2023 challenge report RDFProcessingToolkit/Sansa (2023)
9. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Semant. Web Inf. Syst. **5**(2), 1–24 (2009). https://doi.org/10.4018/jswis.2009040101
10. Calvanese, D., et al.: Ontop: answering SPARQL queries over relational databases. Semant. Web **8**(3), 471–487 (2017). https://doi.org/10.3233/SW-160217
11. Chakravarthy, U.S., Grant, J., Minker, J.: Logic-based approach to semantic query optimization. ACM Trans. Database Syst. **15**(2), 162–207 (1990). https://doi.org/10.1145/78922.78924

12. Chaves-Fraga, D., Endris, K.M., Iglesias, E., Corcho, O., Vidal, M.E.: What are the parameters that affect the construction of a knowledge graph? In: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (eds.) OTM 2019. LNCS, vol. 11877, pp. 695–713. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33246-4_43
13. Chaves-Fraga, D., Priyatna, F., Cimmino, A., Toledo, J., Ruckhaus, E., Corcho, O.: GTFS-Madrid-Bench: a benchmark for virtual knowledge graph access in the transport domain. J. Web Semant. **65**, 100596 (2020). https://doi.org/10.1016/j.websem.2020.100596
14. Daga, E., Asprino, L., Mulholland, P., Gangemi, A.: Facade-X: an opinionated approach to SPARQL anything. Stud. Semant. Web **53**, 58–73 (2021). https://doi.org/10.3233/SSW210035
15. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. In: Working Group Recommendation, World Wide Web Consortium (W3C) (2012). http://www.w3.org/TR/r2rml/
16. Dimou, A., Heyvaert, P., De Meester, B., Verborgh, R.: What factors influence the design of a linked data generation algorithm? In: Berners-Lee, T., Capadisli, S., Dietze, S., Hogan, A., Janowicz, K., Lehmann, J. (eds.) Proceedings of the 11th Workshop on Linked Data on the Web (2018). http://events.linkeddata.org/ldow2018/papers/LDOW2018_paper_12.pdf
17. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: a generic language for integrated RDF mappings of heterogeneous data. In: Bizer, C., Heath, T., Auer, S., Berners-Lee, T. (eds.) Proceedings of the 7th Workshop on Linked Data on the Web. CEUR Workshop Proceedings, vol. 1184 (2014). http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf
18. García-González, H., Boneva, I., Staworko, S., Labra-Gayo, J.E., Lovelle, J.M.C.: ShExML: improving the usability of heterogeneous data mapping languages for first-time users. PeerJ Comput. Sci. **6**, e318 (2020). https://doi.org/10.7717/peerj-cs.318
19. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. J. Web Semant. **3**(2), 158–182 (2005). https://doi.org/10.1016/j.websem.2005.06.005
20. Haesendonck, G., Maroy, W., Heyvaert, P., Verborgh, R., Dimou, A.: Parallel RDF generation from heterogeneous big data. In: Groppe, S., Gruenwald, L. (eds.) Proceedings of the International Workshop on Semantic Big Data - SBD '19. No. 1 in SBD '19, ACM Press, Amsterdam, Netherlands (2019). https://doi.org/10.1145/3323878.3325802
21. Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., Vidal, M.E.: SDM-RDFizer: an RML interpreter for the efficient creation of RDF knowledge graphs. In: Proceedings of the 29th ACM International Conference on Information and Knowledge Management, pp. 3039–3046 (2020). https://doi.org/10.1145/3340531.3412881
22. Iglesias, E., Jozashoori, S., Maria-Esther, V.: Scaling up knowledge graph creation to large and heterogeneous data sources. J. Web Semant. **75**, 100755 (2023). https://doi.org/10.1016/j.websem.2022.100755
23. Iglesias, E., Vidal, M.E.: Knowledge graph creation challenge: results for SDM-RDFizer. In: CEUR Workshop Proceedings; 3471, vol. 3471, p. 13 (2023)
24. Iglesias-Molina, A., Toledo, J., Corcho, O., Chaves-Fraga, D.: Re-construction impact on metadata representation models. In: Proceedings of the 12th Knowledge Capture Conference 2023, pp. 197–205. K-CAP '23, Association for Com-

puting Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3587259.3627554
25. Iglesias-Molina, A., et al.: The RML ontology: a community-driven modular redesign after a decade of experience in mapping heterogeneous data to RDF. In: Payne, T.R., et al. (eds.) ISWC 2023, pp. 152–175. Springer, Cham (2023)
26. Jozashoori, S., Iglesias, E., Vidal, M.E.: SDM-genomic-dataset. Technical report (2023). https://doi.org/10.57702/4c9ivpgs
27. v. Kistowski, J., Arnold, J.A., Huppler, K., Lange, K.D., Henning, J.L., Cao, P.: How to build a benchmark. In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, pp. 333–336. ICPE '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2668930.2688819
28. Lanti, D., Rezk, M., Xiao, G., Calvanese, D.: The NPD benchmark: reality check for OBDA systems. In: Proceedings of the 18th International Conference on Extending Database Technology, EDBT, pp. 617–628 (2015). https://doi.org/10.5441/002/edbt.2015.62
29. Lanti, D., Xiao, G., Calvanese, D.: VIG: data scaling for OBDA benchmarks. Semant. Web **10**(2), 413–433 (2019). https://doi.org/10.3233/SW-180336
30. Lefrançois, M., Zimmermann, A., Bakerally, N.: a SPARQL extension for generating RDF from heterogeneous formats. In: The Semantic Web 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, pp. 35–50. Springer, Portoroz, Slovenia (2017). https://doi.org/10.1007/978-3-319-58068-5_3
31. Maria, P.: Carml: a pretty sweet RML engine, for RDF. Technical report (2018). https://github.com/carml/carml
32. Oo, S.M., Haesendonck, G., De Meester, B., Dimou, A.: RMLStreamer-SISO: an RDF Stream generator from streaming heterogeneous data. In: Sattler, U., et al. (eds.) The Semantic Web – ISWC 2022, pp. 697–713. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19433-7_40
33. Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings. J. Web Semant. **33**, 141–169 (2015). https://doi.org/10.1016/j.websem.2015.03.001
34. Scrocca, M., Comerio, M., Carenini, A., Celino, I.: Turning transport data to comply with EU standards while enabling a multimodal transport knowledge graph. In: International Semantic Web Conference, pp. 411–429. Springer (2020). https://doi.org/10.1007/978-3-030-62466-8_26
35. Singh, G., Bhatia, S., Mutharaju, R.: Owl2Bench: a benchmark for OWL 2 reasoners. In: Pan, J.Z., et al. (eds.) The Semantic Web - ISWC 2020, pp. 81–96. Springer, Cham (2020)
36. Stadler, C., Buhmann, L., Meyer, L.P., Martin, M.: Scaling RML and SPARQL-based knowledge graph construction with apache spark (2023)
37. Van Assche, D., Chaves-Fraga, D., Dimou, A.: KROWN: a benchmark for RDF graph materialization docker images. Technical report (2024). https://hub.docker.com/u/kgconstruct
38. Van Assche, D., Chaves-Fraga, D., Dimou, A.: KROWN: a benchmark for rdf graph materialization GitHub repository. Technical report (2024). http://github.com/kg-construct/KROWN
39. Van Assche, D., Chaves-Fraga, D., Dimou, A.: KROWN: a benchmark for RDF graph materialization results. Technical report (2024). https://doi.org/10.5281/zenodo.10973891

40. Van Assche, D., Chaves-Fraga, D., Dimou, A., Iglesias-Molina, A., Serles, U.: Knowledge graph construction workshop 2023 (KGCW) challenge. Technical report (2023). http://w3id.org/kg-construct/workshop/2023/challenge.html
41. Van Assche, D., Chaves-Fraga, D., Dimou, A., Iglesias-Molina, A., Serles, U.: Knowledge graph construction workshop 2024 (KGCW) challenge. Technical report (2024). http://w3id.org/kg-construct/workshop/2024/challenge.html
42. Van Assche, D., et al.: Declarative RDF graph generation from heterogeneous (semi-)structured data: a systematic literature review. J. Web Semant. **75**, 100753 (2023). https://doi.org/10.1016/j.websem.2022.100753
43. Van Assche, D., et al.: Leveraging web of things W3C recommendations for knowledge graphs generation. In: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (eds.) ICWE 2021. LNCS, vol. 12706, pp. 337–352. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-74296-6_26
44. Van Assche, D., Rojas, J., De Meester, B., Colpaert, P.: IncRML: incremental knowledge graph construction from heterogeneous data sources. https://www.semantic-web-journal.net/content/incrml-incremental-knowledge-graph-construction-heterogeneous-data-sources
45. de Vleeschauwer, E., Haesendonck, G., Van Assche, D., De Meester, B.: RML-Streamer with reference conditions in the KGCW challenge 2023. In: KGCW2023, the 4th International Workshop on Knowledge Graph Construction (2023)
46. Xiao, G., Kontchakov, R., Cogrel, B., Calvanese, D., Botoeva, E.: Efficient handling of SPARQL OPTIONAL for OBDA. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11136, pp. 354–373. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00671-6_21