On Complex Value Relations in Hive

Matthieu Pilven, Stefanie Scherzinger, Laurent d'Orazio

On Complex Value Relations in Hive

Matthieu Pilven[1,2], Stefanie Scherzinger[1], and Laurent d'Orazio[2]

[1] OTH Regensburg, Regensburg, Germany
stefanie.scherzinger@oth-regensburg.de
[2] Univ Rennes, CNRS, IRISA, France
laurent.dorazio@irisa.fr

Abstract. In this paper, we raise the question how data architects model their data for processing in Apache Hive. This well-known SQL-on-Hadoop engine supports complex value relations, where attribute types need not be atomic. In fact, this feature seems to be one of the prominent selling points, e.g., in Hive reference books. In an empirical study, we analyze Hive schemas in open source repositories. We examine to which extent practitioners make use of complex value relations and accordingly, whether they write queries over complex types. Understanding which features are actively used will help make the right decisions in setting up benchmarks for SQL-on-Hadoop engines, as well as in choosing which query operators to optimize for.

Keywords: Hive · Complex value relations · Empirical study.

1 Introduction

Originally a Facebook-internal data warehouse [17], the SQL-on-Hadoop engine Hive is now an official Apache project. After only a decade of development, Apache Hive is listed among the top-10 relational database systems on the DB-Engines Ranking website[3], alongside IBM DB2 and Oracle Database. Yet the latter are commercial products with development histories up to four times longer. Part of this success is owed to the fact that working with Hive feels very familiar:

- Hive users conveniently interact with a relational data model, yet the raw data is typically stored in the Hadoop Distributed File System (HDFS).
- While the query language HiveQL closely resembles the SQL dialect of MySQL [18], queries are executed as scalable MapReduce workflows [17].

Moreover, Hive relations need not be in first normal form, where all attribute values are atomic. Rather, the data model allows for (a restricted form of) complex value relations: A tuple constructor `struct` declares tuples. The complex types `map` and `array` declare maps and arrays, concepts familiar from programming languages. These constructors can be applied recursively, to an arbitrary level of nesting. This allows ingestion of data that already arrives in nested form (e.g., in JSON format). Further, denormalization accelerates query processing in

[3] `https://db-engines.com/de/ranking/relational+dbms`, as of September 2019.

```
1  CREATE TABLE employees (
2    name STRING,
3    salary FLOAT,
4    subordinates ARRAY<STRING>,
5    deductions MAP<STRING, FLOAT>,
6    address STRUCT<street:STRING, CITY:STRING, state:STRING, zip:INT>
7  );
8
9  SELECT name, subordinates[0] FROM employees;
10 SELECT explode(subordinates) AS sub FROM employees;
11
12 SELECT name, sub FROM employees
13 LATERAL VIEW explode(subordinates) subView AS sub;
```

Fig. 1. A Hive table declaration and queries, taken from [5].

data warehouse settings, where data is updated rarely (if at all). Complex value relations are also supported in related systems, such as Impala [12] and Presto[4].

Example 1 (From [5]). The table declaration shown in Figure 1 captures employees with their name, salary, and their nested subordinates. It further captures employee-specific tax deductions, as well as their home address. The HiveQL syntax for accessing a field in an array is straightforward, as seen in line 9. HiveQL offers table generating functions: accordingly, the query in line 10 produces one tuple for each element of array `subordinates`. To list pairs of a manager and a subordinate, we need to declare a `LATERAL VIEW`, as seen in lines 12 and 13. □

Contributions. The support for complex value relations is one of the major selling points for using Hive, c.f. [5]. Yet it is an open question whether practitioners make use of this feature. We therefore explore complex value relations in Hive by analyzing open source repositories on GitHub. There is a tradition of empirical studies on database schemas in open source projects, typically in the context of schema evolution, e.g. [7,15]. However, we are not aware of any studies on the dissemination of complex value relations in SQL-on-Hadoop processing. In particular,

- we formalize complex value relations in Hive and point out connections to the theory of V-relations [1], dating back to the 80s.
- We identify 133 unique and relevant GitHub repositories with a total of over 900 table declarations. We then identify complex value relations in Hive schemas, as well as the occurrence of matching operators in HiveQL queries.
- We discuss our findings w.r.t. existing benchmarks targeted at Hive.

Structure. In Section 2, we provide formal preliminaries. In Section 3, we lay down our methodology. In Section 4, we present the detailed study results, which we discuss in Section 5. We list threats to the validity of our study in Section 6. Section 7 gives an overview over related work. We then conclude with Section 8.

[4] `https://prestosql.io/`

2 Preliminaries

We recap the definition of complex value relations and formalize complex types in Hive. We then point out a connection between these concepts.

Complex value relations [1]. In complex value relations, attribute values need not be atomic. Intuitively, the data structure makes use of two constructors, which can be applied recursively, (1) a *tuple constructor* to make tuples, and (2) a *set constructor* to make sets of tuples, and thus relations.

Underlying the notion of a schema, there is the notion of *complex types* (or sorts). The abstract syntax for complex types is shown next:

$$\tau \;=\; \mathbf{dom} \mid \langle B_1 : \tau, \ldots, B_k : \tau \rangle \mid \{\tau\},$$

where $k \geq 0$, and $B_1, \ldots, B_k$ are distinct attribute names. Intuitively, an element of **dom** is a constant, an element of $\langle B_1 : \tau, \ldots, B_k : \tau \rangle$ is a k-tuple with an element of type τ_i in entry B_i for each i. We refer to [1] for the formal definition of $[\![\tau]\!]$, the set of values of type τ, and merely operate on the level of examples. We define a *complex value relation of type* τ to be a finite set of values of type τ.

Example 2. For Figure 1 (up to line 3), the type (abstracting from **string** and **float**) is $\{\langle \text{name} : \mathbf{dom},\ \text{salary} : \mathbf{dom} \rangle\}$. We state a value of this type: $\{\langle \text{name} : \texttt{"John Doe"}, \text{salary} : 100000.0 \rangle,\ \langle \text{name} : \texttt{"Mary Smith"}, \text{salary} : 80000.0 \rangle\}$. □

Example 3. Below, we consider a complex type for declaring a non-flat complex value relation (equation 1), and a value of this type (equation 2):

$$\{\langle \text{name} : \mathbf{dom},\ \text{salary} : \mathbf{dom},\ \mathbf{subordinates} : \{\langle \text{key} : \mathbf{dom}, \text{value} : \mathbf{dom} \rangle\} \rangle\}. \quad (1)$$

$$\begin{aligned} &\{\langle \text{name} : \texttt{"John Doe"}, \text{salary} : 80000.0, \\ &\quad \text{subordinates} : \{\langle \text{key} : 0, \text{value} : \texttt{"Mary Smith"} \rangle,\ \langle \text{key} : 1, \text{value} : \texttt{"Tod Jones"} \rangle\} \rangle, \\ &\langle \text{name} : \texttt{"Mary Smith"}, \text{salary} : 80000.0, \text{subordinates} : \{\} \rangle, \\ &\langle \text{name} : \texttt{"Todd Jones"},\ \text{salary} : 70000.0, \text{subordinates} : \{\} \rangle\} \end{aligned} \quad (2)$$

□

Figure 2 shows a visualization of the complex type from the above example as a finite tree. The tuple constructor is denoted by a node labeled $\times$ and the set constructor is denoted by a node labeled $*$. Outgoing edges from tuple nodes are labeled, while set nodes have a single child. Based on this tree visualization, it is

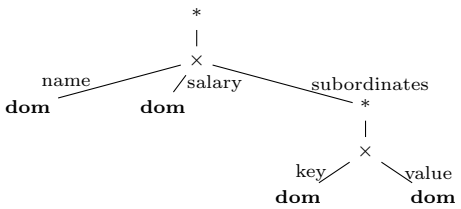

Fig. 2. The complex type from Example 3 as a tree, with a set height of 2.

intuitive to define the *set height* [1] of a complex type as the maximum number of set constructors in any branch. In the tree shown, the set height is 2.

Hive types. The Hive data model offers a second tuple constructor via `struct`. While it does not allow to nest with a proper set constructor, we can nest `map`s and `array`s. We declare the abstract syntax for *Hive type* τ_H as

$$\begin{aligned}\tau_H &= \{\langle B_1 : \tau, \ldots, B_k : \tau\rangle\}\\ \tau &= \mathbf{dom} \mid \texttt{map} <\tau, \tau> \mid \texttt{array} <\tau> \mid \texttt{struct} <B_1 : \tau, \ldots, B_k : \tau>.\end{aligned}$$

Thus, the top-level type is always a set of tuples. Underneath, maps, arrays, and structs may be nested arbitrarily. Given a flat Hive relation over type τ_H, we say the *Hive nesting level* is 1. For any (recursive) declaration of a map, an array, or a struct, the Hive nesting level increases by one.

The set of values of a Hive type τ_H is denoted by $[\![\tau_H]\!]$ and declared next. The values for the tuple and set constructor are defined the same as for complex types. Below, we equate structs with the tuple constructor. Arrays and maps are encoded as sets of key-value pairs:

$$\begin{aligned}[\![\texttt{struct} <B_1 : \tau_1, \ldots, B_k : \tau_k>]\!] &= [\![\langle B_1 : \tau_1, \ldots, B_k : \tau_k\rangle]\!]\\ [\![\texttt{map} <\tau_k, \tau_v>]\!] &= \{\{\langle \text{key} : k_1, \text{value} : v_1\rangle, \ldots, \langle \text{key} : k_j, \text{value} : v_j\rangle\}\\ &\qquad \mid\ j \geq 0, k_i \in [\![\tau_k]\!], v_i \in [\![\tau_v]\!], i \in [1, j]\}\\ [\![\texttt{array} <\tau>]\!] &= \{\{\langle \text{key} : 1, \text{value} : v_1\rangle, \ldots, \langle \text{key} : j, \text{value} : v_j\rangle\}\\ &\qquad \mid\ j \geq 0, v_i \in [\![\tau]\!], i \in [1, j]\}\end{aligned}$$

Then, a *Hive relation of type* τ_H is a value of type τ_H.

Hive relations as Verso-relations. The Hive data model does not allow nesting with a proper set constructor, a limitation has also been observed in [16]. Nevertheless, a Hive type may be generalized to a complex value type. For instance, we may generalize a map or an array to a set of key-value tuples:

$$\begin{aligned}[\![\texttt{map} <\tau_k, \tau_v>]\!] &\subset [\![\{\langle \text{key} : \tau_k, \text{value} : \tau_v\rangle\}]\!]\\ [\![\texttt{array} <\tau>]\!] &\subset [\![\{\langle \text{key} : \mathbf{dom}, \text{value} : \tau\rangle\}]\!].\end{aligned}$$

Note that the generalized type allows non-consecutive and even repetitive array indexes, so it really defines a superset of values.

Given this generalization, we can relate Hive relations to a data model explored in earlier research: We may safely assume that all recursive nestings of structs with structs have been flattened to a single struct. This can be done without loss of information. Then, the above generalization of Hive types to complex value types actually produces Verso-relations [1]. These data structures have the appealing property that the information contained can be equivalently represented using (several) flat relations. This imposes a polynomial bound on the cardinality of a set in a Verso-relation, which is a nice property for the practical evaluation of tuple constructors (such as `explode`).

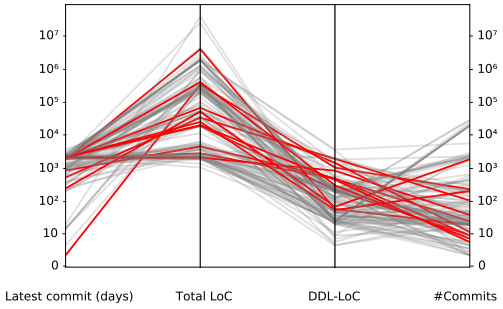

Fig. 3. Overview of the analyzed repositories (axes with log scales).

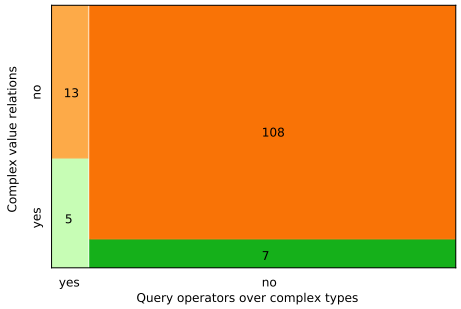

Fig. 4. Repositories with complex value relations vs. queries over complex types.

3 Methodology

3.1 Context Description

We used Google BigQuery[5] to identify relevant open source repositories on GitHub, as of July 16th, 2019. This cloud service allows for querying the GitHub open data collection, mostly non-forked projects with an open source license. We consider a repository relevant if it contains at least one file with ending `.hql`, which commonly denotes a file with HiveQL statements. This revealed 417 repositories. We discarded any projects that were registered as forks of other repositories, which leaves 158 (the GitHub open data collection is not free of forks). We further eliminated 25 repositories that had no table declarations in `hql`-files.

Figure 3 characterizes the 133 analyzed repositories (for now, we disregard the colors): We report the days passed since the latest commit, the total lines of code (LoC), and *DDL-LoC*, the lines of code for table declarations (c.f. Section 3.3). We further report the number of commits. The lines of code and the number of commits are indicators of project maturity. The lines of code of schema declarations are generally considered a *proxy metric* for schema complexity [10].

3.2 Research Questions

RQ1: How common are complex value relations in Hive schemas?
RQ2: What is the usage of query operators over complex types?

3.3 Analysis Process

Parsing Hive schemas. We wrote a Python-based parser for processing `hql`-files. One `hql`-file was missing a semicolon as a delimiter. We fixed this manually

[5] `https://cloud.google.com/bigquery/`

Table 1. The subset of repositories with (non-flat) complex value relations.

	Repository	#CVRs	HNL	SH	#Cont
1	`sixeyed/hive-succinctly`	3 out of 16	3	2	1
2	`jbrambleDC/predict_restaurant_success`	2 out of 3	3	2	1
3	`flaminem/flamy`	21 out of 51	2	2	1
4	`yhemanth/hive-samples`	1 out of 22	2	2	1
5	`Benjguin/UnlockLuxury`	1 out of 15	2	2	2
6	`DXFrance/data-hackathon`	1 out of 15	2	2	1
7	`mellowonpsx/ESCA`	2 out of 7	2	1	1
8	`Sicmatr1x/Sicmatr1x.github.io`	1 out of 7	2	2	1
9	`PolymathicCoder/Avempace`	4 out of 5	2	2	1
10	`airbnb/aerosolve`	1 out of 4	2	2	21
11	`gliptak/hadoop-course`	1 out of 4	2	2	1
12	`EXEM-OSS/Flamingo2`	1 out of 3	2	2	1

to include this file in our analysis. We extract all `CREATE TABLE` statements and pretty-print them such that one attribute is declared per line, like in Figure 1. This is the pre-processing step in measuring DDL-LoC. Note that we ignore all declarations of the form `CREATE TABLE LIKE <T>` and `CREATE TABLE AS <Q>`, of which there are 17 and 117 respectively. We justify this as follows: With the first, we merely create copies of tables that we already analyze, so there is little added value. The second construct is commonly used for storing intermediate results, like one would declare a materialized view. We argue that these are not the base tables holding the original data, and therefore choose to ignore them in our analysis. Along the same line of arguments, we do not analyze `CREATE VIEW` statements, where complex types may be introduced over flat base relations.

Analyzing HiveQL queries. The Hive query language offers several constructs to deal with complex types[6], such as **(1)** collection functions (e.g., the operation `array_contains(A,v)` checks whether array `A` contains value `v`), **(2)** complex type constructors (e.g. for creating a map), and **(3)** built-in table-generating functions (e.g., `json_tuple` for generating a relational tuple from a JSON string). **(4)** Further, there are built-in functions for XPath-style queries.[7]

We grep for `SELECT`-statements with these constructs in `hql`-files. Thus, we cover a wide range of operators and only exclude access to single fields, such as `A[i]` to access the ith field of array `A`, or `S.c` to access component `c` of struct `S`.[8]

4 Detailed Study Results

4.1 RQ: How common are complex value relations in Hive schemas?

In the following, whenever we mention complex value relations, we assume *non-flat* relations, and explicitly refer to *flat* relations otherwise. Only 9 % of all

[6] `https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF`

[7] `https://cwiki.apache.org/confluence/display/Hive/LanguageManual+XPathUDF`

[8] These operators are difficult to match reliably by keyword search alone.

analyzed repositories contain complex value relations, and are listed in Table 1: next to the repository, we state how many of the relations analyzed are non-flat (column #CVRs). For the first repository, this means 3 out of 16 relations. We state the maximum Hive nesting level (HNL), set height (SH), and the number of contributors (#Cont). The entries are sorted hierarchically, by HNL and the total number of tables analyzed. While Hive does not restrict the nesting level, the maximum HNL observed is only 3. The set height is at most 2, so there is not a single recursive nesting of maps or arrays. Repository 2 is one of the few repositories that is actually a data analytics project, it predicts restaurant success. Further, Repository 8 analyzes server logs. Repository 10 is a machine learning library where complex value relations hold training data. The other repositories seem experimental, as suggested by names like `hadoop-course` and `hive-samples`, some even contain the tutorial table declared in Figure 1. Most are for personal development, `aerosolve` stands out with 21 contributors.

In Figure 3, we show these 12 repositories in context (we recommend that this figure is viewed in color). The red lines denote the repositories from Table 1. What is common to these repositories is that (1) the latest commit dates back half a year or more. Thus, projects undergoing active development have flat relations only. (2) Further, all repositories from Table 1 have schema declarations spanning at least 57 DDL-LoC, which is more than in 40 % of all repositories.

Results. We find little evidence of complex value relations being used. Mostly, the developers merely experiment with complex types. Moreover, even though Hive does not restrict the nesting level, the maximum observed is 3. Of course, complex types may also be introduced in views, as discussed next.

4.2 RQ: What is the usage of query operators over complex types?

In total, we have analyzed 2,771 HiveQL queries in `hql`-files. The mosaic plot in Figure 4 reads as follows: Along the vertical, we distinguish the repositories that use complex value relations from those that declare only flat relations. Along the horizontal, we distinguish repositories with queries over complex types. The largest area with 108 repositories represents the repositories that have neither complex value relations, nor queries over complex types. Among the repositories with complex value relations, only about half contain matching queries (repositories 5–9). Interestingly, 13 repositories have no complex value relations, but queries over complex types: some evaluate XPath over string-valued attributes, or views introduce complex types and the queries operate over these views.

Next, we list all observed query operators over complex types, ordered by the number of occurrence: `explode`: 27, `lateral_view`: 26, `xpath_string`: 6, `json_tuple`: 5, `size`: 5, `xpath_int`: 4, `stack`:1. Some words on operators that we have not introduced yet. `stack` breaks down tuples into several, smaller-sized tuples. `size` returns the number of elements in a `map` or an `array`. While we have searched for 20 different syntactic constructs, we have only found evidence of 7.

Results. We found query operators over complex types to be rare. There are even repositories with complex value relations but no matching queries. More-

Table 2. Database benchmarks used to also benchmark Hive.

Benchmark	(Non-flat) CVRs	Queries over complex types
TPC-H[11], TPC-DS[11] , Hive-600 [14]	◯	◯
HiBench [9], SmartBench[9]	◯	◯
Pavlo et al [13]	◯	◯
Hive-testbench[10]	◯	◯
BigBench [8], TPC-xbb[11]	✓	✓
UniBench [19]	✓	✓

over, despite the richness of the HiveQL query language, authors of queries seem to restrict themselves to a chosen few operators over complex types.

5 Discussion

In the repositories analyzed, the majority of schemas is actually in first normal form. This matches our impression from discussions with practitioners. We have several conjectures: **(1)** One reason may be that when data is ingested into Hive from a relational data warehouse, the data is "flat" to start with. **(2)** A further conjecture is that different tools might share access to the data in HDFS, but not all tools can handle complex types. Thus, data architects may be less inclined to declare complex value relations. **(3)** Queries over complex types tend to become complex. In fact, repository 2 in Table 1 contains a "flat" view over a complex value relation, probably to facilitate query formulation.

In Table 2, we list benchmarks commonly used for Hive. Interestingly, few include (non-flat) complex value relations and matching queries. BigBench specifies a log processing scenario, with an `explode` statement in a query over semi-structured data. This matches the results of our study, as we found `explode` to occur most frequently (c.f. Section 4.2). UniBench, in contrast, contains multi-model data and is not restricted to complex value relations, e.g., it also includes queries over graph and key-value data. However, there are several benchmarks targeted at Hive that do not include complex value relations or queries over complex types. This mismatch between our findings and the schemas in these benchmarks motivates future and larger-scale studies.

6 Threats to Validity

Construct validity. (1) In identifying relevant repositories, we rely on the convention that `CREATE TABLE` statements are contained in `hql`-files. However, **(1a)** `CREATE TABLE` statements can be embedded in the application code. While this is certainly a limitation of our methodology, ignoring SQL statements in application code is common in virtually all earlier empirical studies on relational

[9] `https://github.com/bomeng/smartbench`

[10] `https://github.com/hortonworks/hive-testbench`

[11] `http://www.tpc.org/`

schemas, e.g. [7,15]. **(1b)** Some `CREATE TABLE` statements for Hive are contained in `sql`-files, rather than `hql`-files: if we also were to analyze `sql`-files, then repository 7 in Table 1 would have maximum HNL 3. We carefully assessed the risk of ignoring `sql`-files: Across all analyzed repositories, we count 1,006 `hql`-files and even 5,870 `sql`-files. At first, this seems promising: In Table 1, we could add two further, personal-development projects with HNL 2 and SH 2. Yet while the table would grow by two entries, the maximum Hive nesting level and set height would remain unchanged. Thus, there is little information gain regarding research question 1. Regarding research question 2, analyzing the query constructs in `sql`-files increases the absolute numbers, but not the relative ranking of occurrences (e.g., `explode` being the most frequent). In particular, no new constructs are found. Again, there is little information gain.

At the same time, there is considerable risk in including `sql`-files: For instance, `cloudera/hue` is a SQL workbench that supports several database systems. Analyzing the contents of `sql`-files files would introduce considerable bias into our analysis. Considering this risk, we restrict ourselves to `hql`-files.

(2) Similarly, if an Impala-based repository contains `hql`-files, we falsely include this in our analysis. However, we consider this a minor threat, as we have carefully inspected the repositories from Table 1 for signs that they might not be Hive projects. **(3)** We currently do not analyze ALTER TABLE statements. We have verified that while ALTER TABLE statements occur 489 times, none of them introduce complex types. Thus, this threat can be safely ignored.

External validity. It is a fundamental question how representative studies on open source projects are [4, 11]. Actually, we have encountered Stack Overflow questions about declaring tables with deep nesting levels, so some developers go beyond HNL 3, yet we do not see evidence of this in our data.

7 Related Work

There is a long tradition of research on normal forms and also complex value databases, dating back to the 80s [1]. Recent experiments have shown (for Spark) that denormalization into complex value relations can indeed speed up query processing in Big Data scenarios [2].

In software engineering research, analyzing open source applications is an established practice [4]. It is only natural that the availability of public code repositories has enabled empirical studies on database schemas. For instance, there is a line of studies on schema evolution, e.g. [7,15]. There is also a history of empirical studies on real-world data in nested data models, such as DTDs [6] and XML Schema [3]. These studies are very similar to ours in their methodology.

8 Conclusion and Future Work

By analyzing real-world Hive schemas, we are able to show that complex value relations occur to only a small extent and that nesting is shallow.

In future work, we plan to conduct a larger-scale study that involves more data, and further SQL-on-Hadoop engines, to obtain a wider perspective.

Acknowledgements: This work was supported by the *Franco-German Youth Office* (FGYO), which funded Matthieu Pilven's internship at OTH Regensburg. This project was further supported by the *Deutsche Forschungsgemeinschaft* (DFG, German Research Foundation), grant #385808805, as well as a Google Cloud Platform Research Credit award. We thank Uta Störl for her feedback on an earlier version of this paper.

References

1. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases: The Logical Level. Addison-Wesley Longman Publishing Co., Inc., 1st edn. (1995)
2. Arrascue Ayala, V.A., Koleva, P., Alzogbi, A., Cossu, M., et al.: Relational Schemata for Distributed SPARQL Query Processing. In: Proc. SBD'19 (2019)
3. Bex, G.J., Neven, F., Van den Bussche, J.: DTDs Versus XML Schema: A Practical Study. In: Proc. WebDB'04 (2004)
4. Bird, C., Menzies, T., Zimmermann, T.: The Art and Science of Analyzing Software Data. Morgan Kaufmann Publishers Inc., 1st edn. (2015)
5. Capriolo, E., Wampler, D., Rutherglen, J.: Programming Hive. O'Reilly Media, Inc., 1st edn. (2012)
6. Choi, B.: What are real DTDs like? In: Proc. WebDB'02 (2002)
7. Curino, C.A., Tanca, L., Moon, H.J., Zaniolo, C.: Schema evolution in Wikipedia: Toward a Web Information System Benchmark. In: Proc. ICEIS'08 (2008)
8. Ghazal, A., Rabl, T., Hu, M., Raab, F., et al.: BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In: Proc. SIGMOD'13 (2013)
9. Huang, S., Huang, J., Dai, J., Xie, T., et al.: The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In: Proc. ICDEW'10 (2010)
10. Jain, S., Moritz, D., Howe, B.: High variety cloud databases. In: Proc. ICDE Workshops'16 (2016)
11. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., et al.: The Promises and Perils of Mining GitHub. In: Proc. MSR'14 (2014)
12. Kornacker, M., Behm, A., Bittorf, V., Bobrovytsky, T., et al.: Impala: A Modern, Open-Source SQL Engine for Hadoop. In: Proc. CIDR'15 (2015)
13. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., et al.: A comparison of approaches to large-scale data analysis. In: Proc. SIGMOD'09 (2009)
14. Poess, M., Rabl, T., Jacobsen, H.: Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems. In: Proc. SoCC'17 (2017)
15. Qiu, D., Li, B., Su, Z.: An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications. In: Proc. ESEC/FSE'13 (2013)
16. Sauer, C., Härder, T.: Compilation of Query Languages into MapReduce. Datenbank-Spektrum **13**(1), 5–15 (2013)
17. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., et al.: Hive: A Warehousing Solution over a Map-Reduce Framework. Proc. VLDB Endow. **2**(2), 1626–1629 (Aug 2009)
18. White, T.: Hadoop: The Definitive Guide. O'Reilly Media, Inc., 3rd edn. (2009)
19. Zhang, C., Lu, J., Xu, P., Chen, Y.: UniBench: A Benchmark for Multi-model Database Management Systems. In: Proc. TPCTC'18 (2018)