PopulAid: In-Memory Test Data Generation

Ralf Teusner[1], Michael Perscheid[2], Malte Appeltauer[2], Jonas Enderlein[1], Thomas Klingbeil[2], and Michael Kusber[2]

[1] Hasso Plattner Institute, University of Potsdam, Germany
ralf.teusner@hpi.uni-potsdam.de,
jonas.enderlein@student.hpi.uni-potsdam.de
[2] SAP Innovation Center Potsdam, Germany
{firstname.lastname}@sap.com

Abstract. During software development, it is often necessary to access real customer data in order to validate requirements and performance thoroughly. However, company and legal policies often restrict access to such sensitive information. Without real data, developers have to either create their own customized test data manually or rely on standardized benchmarks. While the first tends to lack scalability and edge cases, the latter solves these issues but cannot reflect the productive data distributions of a company.
In this paper, we propose *PopulAid* as a tool that allows developers to create customized benchmarks. We offer a convenient data generator that incorporates specific characteristics of real-world applications to generate synthetic data. So, companies have no need to reveal sensible data but yet developers have access to important development artifacts. We demonstrate our approach by generating a customized test set with medical information for developing SAP's healthcare solution.
Keywords: In-Memory Database, Data Generation, Application Testing

1 Introduction

Development and maintenance of enterprise applications highly depends on productive data to take the risk of slow, wrong, or inconsistent information into account. These issues can be handled best with productive data because it embodies the optimal basis for adding new features, debugging, and profiling performance bottlenecks. So, developers can ensure that their enterprise applications are able to handle expected data and increased usage in the future.

However, in most cases it is impossible to develop with productive data. Legal reasons, competitive advantages, clues to business secrets, and data privacy usually prohibit the usage of such data. Unfortunately, developers need test data, therefore they have to spend parts of their time on the creation of data that suits their needs. For two reasons, this data is likely to be biased in some way:

- Developers cannot know every single detail of productive data so that they tend to miss important requirements and edge cases. In consequence, the generated data is based on false assumptions and does not reflect productive data in all circumstances.

– Time is limited. Therefore, the creation is done as simple as possible and the amount of generated data is limited to the amount absolutely required– scalability is not tested under these conditions.

To solve these issues, standardized benchmarks are a means to help developers during development of large applications. They offer not only realistic data volumes but also cover possible edge cases and faulty inputs. However, based on the fact that they are standardized, they can neither reflect the productive data distributions of a certain company nor include individual queries sufficiently [1].

To circumvent these shortcomings, an ideal solution would be customized benchmarks with *generated test data* that shares the same characteristics as in real applications. While the standard benchmark includes appropriate and common edge cases, the customization reflects specific tables, productive data volumes, and distributions present in a company. This way, developers would get a scalable dataset to test specific queries without requiring customers to reveal their sensitive information.

In this paper, we present PopulAid[3] as a tool for generating customized test data. With the help of our web interface, developers can easily adjust their schemas, assign generators to columns, and get immediate previews of potential results. In doing so, generators configure not only specific value properties for one column such as data type, range and data pools, distribution, or number of distinct and undefined values; but also dependencies for column combinations such as foreign keys, pattern evaluation, and functional relations. PopulAid allows developers to create data in a scalable and efficient manner by applying these generators to SAP HANA. This columnar in-memory database can leverage the performance potentials also for write-intensive tasks such as data generation [2]. We present our approach with the help of a real-world example that generates a test set representing medical data from SAP's healthcare solution.

The remainder of this paper is structured as follows: Section 2 presents PopulAid and its core features, Section 3 evaluates our approach, Section 4 discusses related work, and Section 5 concludes.

2 PopulAid

PopulAid is designed to be a convenient solution for realistic application testing. We seamlessly integrate our data generation into development processes and offer different ways of interacting with our approach in order to support good usability. To reach this goal, we focus on three core concepts:

– No setup is required. When available, PopulAid is shipped together with the target database SAP HANA.
– Immediate feedback of the input via a preview of the values to be generated. As depicted in Fig. 1, for every column, a representative selection of ten values is shown directly in the web front-end.

[3] More information (including a screencast) can be found at: `https://epic.hpi.uni-potsdam.de/Home/PopulAid`

– Assistive guessing of suitable generators. Especially for wide tables with more than 100 columns, assigning generators manually is tedious. Guessing of suitable generators for missing columns is done on the basis of the present datatype, the name of the column, and past generation tasks. Columns that have a not null constraint and did not get assigned a generator manually, are chosen automatically.

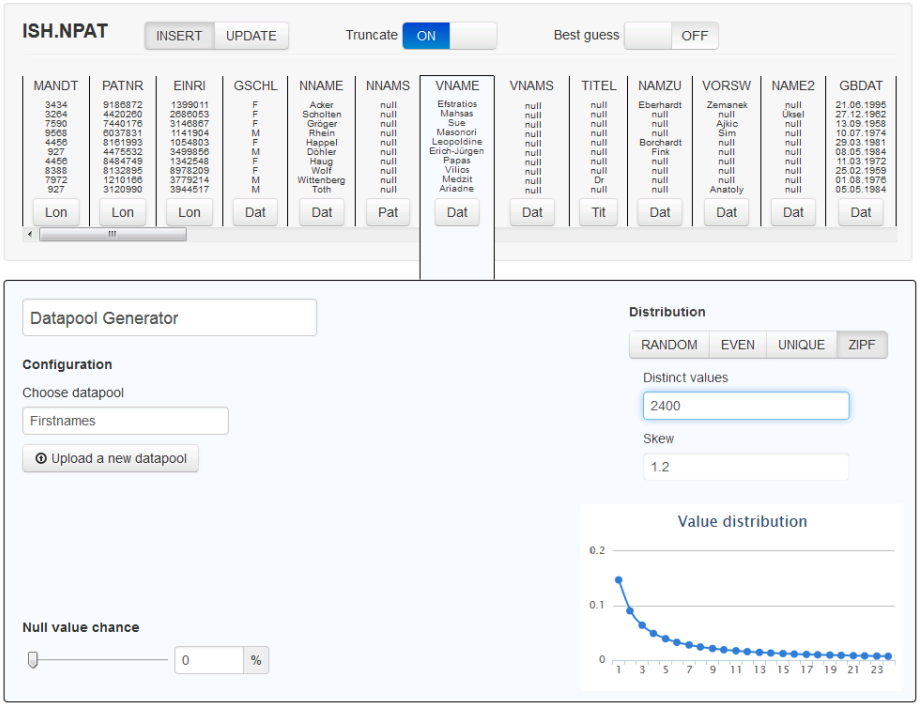

Fig. 1: Webfrontend with Value Previews for Table `NPAT`

The main interface is a web front-end as shown in Fig. 1. Additionally, data generation can also be configured via an Eclipse plugin or directly via a Java API, which, for example, allows unit tests to automatically generate data.

Even if PopulAid was initially aimed for SAP HANA, its API supports arbitrary databases reachable via JDBC. Concerning the user interface, a generic approach to retrieve existing distributions and present immediate feedback is currently under development.

To demonstrate the process and PopulAid's main features, we generate medical data for SAP's healthcare solution (IS-H). Its primary tables comprise patient

master data (NPAT), cases (NFAL) and services performed (NLEI)[4]. These tables hold sensitive data that is liable to strict privacy protection and therefore cannot be distributed to developers. For the patient master data table (NPAT), we focus on the generation of the first name (VNAME) and the combined primary key (fields MANDT and PATNR). On the cases table (NFAL), we explain the generation of the foreign key to NPAT. Finally, we use the services performed table (NLEI) to measure the achieved performance of our data generation.

We begin with the first name column in table NPAT. In our front-end, the user selects the VNAME column and chooses the data pool generator for Firstnames with a Zipf distribution [3] (see Fig. 1). In order to execute the generator, the selected configuration is sent to the backend via JSON (see Fig. 2). The backend instantiates a data pool generator, loads the first name dataset and assigns the generator to the VNAME column. The Zipf distribution is enforced by the container that delivers the values for the generator. As the configuration specifies a Zipf distribution with 2400 distinct values and a skew of 1.2, the container picks the distinct values randomly from the dataset and assigns them the appropriate probabilities. Whenever the generator is called, it returns its pick from the containers' weighted value set. After all generators of a table were called, the picks are set as values for a prepared insert statement, which is then executed in batches.

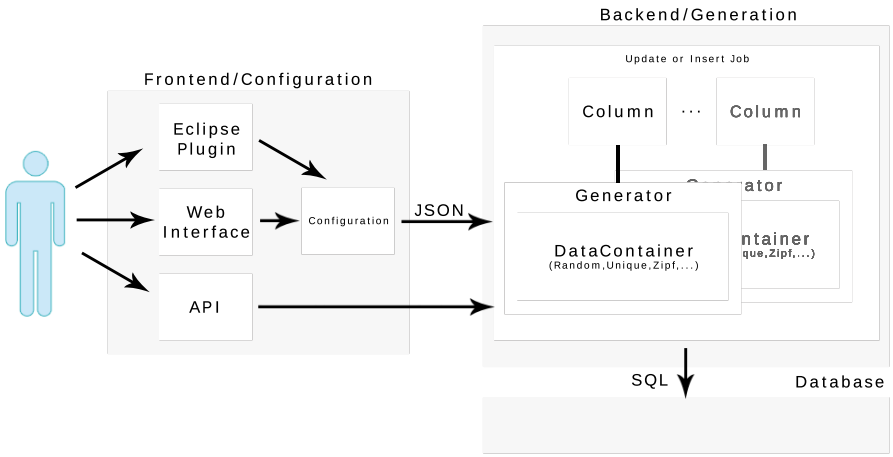

Fig. 2: Schematic Architecture Overview

For the combined primary key in NPAT, we create long generators for both columns (MANDT and PATNR). To enforce uniqueness of the combinations, the gen-

[4] More information concerning the tables and the included attributes is accessible under: http://help.sap.com/saphelp_crm60/helpdata/de/09/a4d2f5270f4e58b2358fc5519283be/content.htm

erators are ordered and handed over to an instance of `MultiColumnGenerator` with distribution set to unique. Afterwards, the `MultiColumnGenerator` is assigned to both columns. On each call, the generator returns the next entry from an unique array of values for the columns it is assigned to.

In table `NFAL`, the fields `MANDT` and `PATNR` realize the connection of the case to the patient master data. Therefore, we use a `FunctionalDependencyGenerator` here. This generator retrieves all distinct values from the source columns `PATNR` and uses them as its value pool. Apart from that, the `ForeignKeyGenerator` works like the data pool generator.

In addition to the aforementioned generators, PopulAid features various other generators to create dates and timestamps, texts that follow a specified pattern, e.g, for phone numbers or email addresses, or use custom SQL queries to extract their value pools. With regard to data types, all standard types are supported. To reflect realistic data, it is also possible to pollute the generated values with a predefined share of undefined or null values. PopulAid also allows to update values in existing tables, for example to add missing columns, apply expected characteristics to already existing datasets, complete sparse data or to provoke certain edge cases.

3 Evaluation

After creating data with PopulAid, we have a more detailed look into its performance characteristics. For that reason, we generate data for our previous introduced tables (`NPAT`: 116 columns, `NFAL`: 76 columns, `NLEI`: 112 columns). These tables also have specific requirements that have to be fulfilled: `NPAT` has a two multi-column uniqueness constraint over two columns; `NFAL` has a random foreign key dependency over two combined columns to `NPAT`; and `NLEI` has a random foreign key dependency over two combined columns to `NFAL`.

We measure the performance for creating our customized test data under the following conditions. We consider both the number of entries per second and the raw data throughput (in MB per second). When profiling the generator, we focus on the pure execution time and exclude the setup time that has to be done only once. We choose a batch size of 20 entries to be inserted together, which turned out to be optimal in most cases. The inserting transaction is always committed just at the very end of the generation. To rule out statistical outliers, we perform each measurement 10 times and choose the median value. We execute all benchmarks on an Intel Core i7 4x2.6 GHz CPU with 4 GB RAM assigned to the Java Virtual Machine, connected via Ethernet to a virtual machine-based SAP HANA running on a Xeon X5670 with 4 cores at 2.93 GHz with 32 GB RAM.

Table 1 presents our performance measurements for SAP's healthcare solution. As can be seen in the table, the run-time of PopulAid's generation process increases nearly linearly when incrementing the data size. For example, generating data for `NFAL` requires 2,5 seconds for 100,000 entries and 25 seconds for 1,000,000 entries. Fig. 3 also illustrates the linear correlation between generated

	Run-time in ms			Entries/s			MB/s		
Data Size	NPAT	NFAL	NLEI	NPAT	NFAL	NLEI	NPAT	NFAL	NLEI
1,000	120	51	60	8,333	19,608	16,667	9.63	13.84	17.01
10,000	387	296	360	25,839	33,783	27,777	29.83	23.85	28.33
100,000	3,844	2,505	3,305	26,014	39,920	30,257	30.11	28.21	30.86
1,000,000	45,611	25,550	34,112	21,925	39,139	29,315	25.37	27.66	29.90

Table 1. Performance Measurements for `NPAT`, `NFAL`, and `NLEI` Tables

entries and the runtime. The throughput seems to be constantly high with generated entries between 20,000 and 40,000 per second. Only for small amounts of data size, the throughput is considerably lower than the throughputs achieved for larger amounts. This can be explained with the higher influence of opening transactions and accompanying database connections on the entire performance. For greater datasets than shown in Table 1, the throughput tested on `NLEI` remained constant at about 30,000 entries per second for 5 million entries. The differences in throughput between tables is based on the individual constraints for data generation. A probable reason for the decrease in `NPAT` is the uniqueness requirement that has to be fulfilled. Considering `NLEI`, the greater size in terms of columns may be the reason for lower entries per second in comparison to `NFAL`.

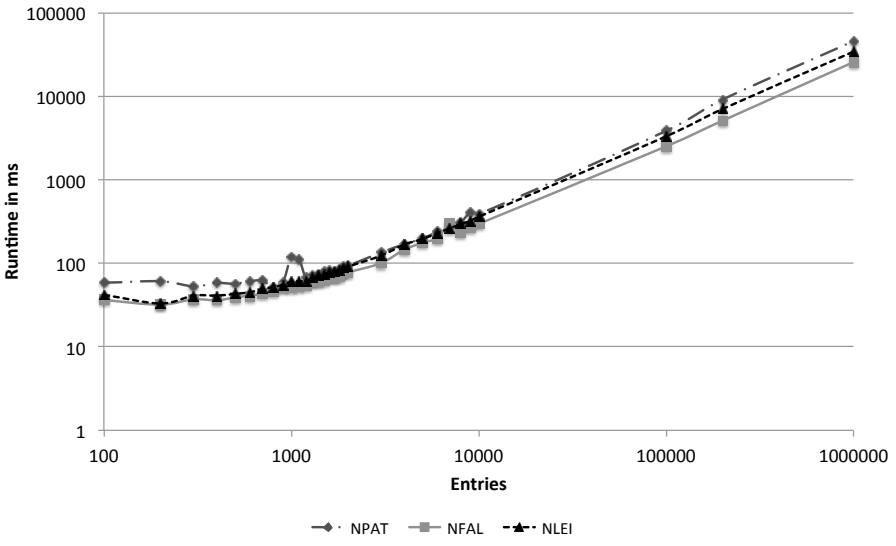

Fig. 3: Generation Runtime on Logarithmic Scale

While conducting our performance tests, we spotted the JDBC connection as the limiting factor of throughput. For that reason, we still wanted to measure

the theoretical throughput without network limitations. When writing the data to disk into a comma-separated values (CSV) file, for each table the achieved throughput ranged between 29,9 MB/s and 33,4 MB/s. This speed is close to the optimal performance when writing to the database because the CPU utilizations while inserting was fluctuating in the 90 % range. Nevertheless, we can still achieve higher writing throughput if we implement full parallelization in the near future.

4 Related Work

In general, data generation is a widely researched area in computer science, which would go beyond the scope of this paper. In the context, we therefore focus on related work concerning data generation in the context of customized benchmarks for application testing. The need for customizing standard benchmarks such as defined by the Transaction Processing Council (TPC) has already been identified [1]. The authors figured out that the domain-specific benchmarks are increasingly irrelevant due the diversity of data-centric applications. For that reason, they call for techniques and tools to synthetically scaling data and create application-specific benchmarks. The multi-dimensional data generator (MUDD) [4] allows for generating huge data volumes with appropriate distributions and hierarchies. However, it is designed specifically to support the tables used in the TPC-DS benchmark. To generate data for customized tables, their configuration has to be implemented as ANSI-C classes. The parallel data generation framework (PDGF) [5] supports the processing, storage, and loading aspects of big data analytics and so allows for end-to-end benchmarks. This generic data generator is currently applied in TPC benchmarks because it generates large amount of relational data very fast by parallelizing with seeded random number generators. The big data generator suite (BDGS) [6] reuses PDGF and enhances the approach with the 4V requirements (volume, variety, velocity, and veracity). For example, it derives characteristics from real data and so preserves data veracity or supports several data sources and types in order to achieve variety. Finally, Myriad [7] is an expressive data generation toolkit that makes extensive use of parallelization.

Compared to PopulAid, the flexibility of the presented approaches for customized application test data is limited. While the aforementioned projects cover various aspects of the 4V requirements, they lack capabilities to configure the generation in a fast and intuitive way. In order to play out their technical features, generators have to be easily usable and fit into the development process seamlessly. PopulAid satisfies this requirement with an intuitive web front-end, immediate feedback about expected results, and semi-automatic configuration of generators.

5 Conclusion

In this paper, we presented a data generation tool named PopulAid. It allows developers to easily create customized test data when productive data is not available. For this purpose, PopulAid offers a convenient web interface in order to adjust schemas, get assisted with generators, and obtain immediate previews. Our approach offers a broad spectrum to generate data from adapting specific values to defining complex dependencies between multiple columns. We integrated PopulAid into the SAP HANA in-memory database and showed how to generate a customized test data set for medical data from SAP's healthcare solution.

Future work deals with two topics. First, we will further improve the performance of PopulAid. Currently, we are working on a fully parallelized solution that is directly integrated into the SAP HANA in-memory database. Second, we are experimenting with different approaches concerning the anonymization of customer data. Instead of the "traditional" way, generating data with similar characteristics, we experiment with means to manipulate real-world data until a back reference is not possible anymore. This method allows developers to create customized benchmarks which are still closer to customer data.

Acknowledgments We thank Janusch Jacoby, Benjamin Reissaus, Kai-Adrian Rollmann, and Hendrik Folkerts for their valuable contributions during the development of PopulAid.

References

1. Tay, Y.C.: Data Generation for Application-specific Benchmarking. VLDB, Challenges and Visions (2011) 1470–1473
2. Plattner, H.: A Course in In-Memory Data Management. Springer (2013)
3. Newman, M.E.: Power laws, pareto distributions and zipf's law. Contemporary physics **46**(5) (2005) 323–351
4. Stephens, J.M., Poess, M.: MUDD: A Multi-dimensional Data Generator. In: Proceedings of the 4th International Workshop on Software and Performance. WOSP '04, ACM (2004) 104–109
5. Rabl, T., Jacobsen, H.A.: Big Data Generation. In: Specifying Big Data Benchmarks, Springer (2014) 20–27
6. Ming, Z., Luo, C., Gao, W., Han, R., Yang, Q., Wang, L., Zhan, J.: BDGS: A Scalable Big Data Generator Suite in Big Data Benchmarking. arXiv preprint arXiv:1401.5465 (2014) 1–16
7. Alexandrov, A., Tzoumas, K., Markl, V.: Myriad: Scalable and Expressive Data Generation. Proceedings of the VLDB Endowment **5**(12) (2012) 1890–1893