Synthetic Data Generation for Enterprise DBMS

Anupam Sanghi
IBM Research
Bengaluru 560045, India
anupam.sanghi1@ibm.com

Jayant R. Haritsa
Indian Institute of Science
Bengaluru 560012, India
haritsa@iisc.ac.in

***Abstract*—A critical need for enterprise DBMS vendors is to generate synthetic databases for testing their engines and applications in a range of environments. These synthetic databases are targeted toward capturing the desired schematic properties, and the statistical profiles of the data hosted on these schemas.**

Several data generation frameworks have been proposed for OLAP over the past three decades. The early efforts focused on *ab initio* generation based on standard mathematical distributions. Subsequently, there was a shift to *database-dependent* regeneration, which aims to create a database with similar statistical properties to a specific client database. This client-specific perspective has been taken further in recent times through *workload-dependent* database regeneration, where the databases generated ensure similar query executions to those observed at the client site.

In this tutorial, we present a holistic coverage of synthetic data generation, highlighting the strengths and limitations of the above-mentioned framework classes. At the end, a suite of open technical problems and future research directions are enumerated.

***Index Terms*—Synthetic Data Generation, DBMS Testing.**

I. Introduction

In industrial practice, a critical requirement for database vendors is to adequately test their database engines with representative data and workloads that accurately mimic the data processing environments at customer deployments. These simulations aid in a variety of ways, including evaluation of new engine designs, root-cause analysis of operational problems encountered by clients, proactive assessment of the performance impacts of planned engine upgrades, data masking for business analytics, query execution tuning, and system benchmarking.

The enormous inherent variety in database usage environments makes it challenging to create test protocols that adequately cover this diversity. While, in principle, clients could transfer their original data to the vendor for the intended evaluation purposes, this is often infeasible because of: (a) Client privacy concerns; (b) Regulatory protocols such as GDPR [2]; and (c) Data transfer overheads from client to vendor, especially in the current Big Data era. Therefore, it is essential to continue our efforts to develop synthetic databases that exercise application functionality, efficiency and robustness. The contemporary need for such database generation tools was highlighted in a recent Dagstuhl seminar [1].

Generation of synthetic databases is targeted toward capturing the desired schematic properties (e.g. keys, referential constraints, functional dependencies, domain constraints), and the statistical profiles (e.g. value distributions, column correlations, data skew, output volumes) of the data hosted on these schemas. A rich body of literature exists on synthetic data generation, covering a wide variety of frameworks, especially in the context of modeling OLAP-driven data warehouses. Further, the recent advent of machine learning techniques has triggered the development of several new approaches that leverage their potent modeling power.

Data generation frameworks can be broadly classified into the following three categories: (a) **Ab Initio Generation**, (b) **Database Dependent Regeneration**, and (c) **Query-Workload Dependent Regeneration**. In this tutorial, we cover, as described in Sections II through V, representative database generation techniques from each of these taxonomy dimensions, and highlight their strengths and limitations. At the end, a suite of open technical problems and future research directions are enumerated.

II. Ab Initio Generation

The frameworks in this category deal with the *ab initio* generation of new databases using standard mathematical distributions. Descriptive languages for the definitions of data dependencies and column distributions were proposed in [14], [23], [34]. For example, [14] proposed Data Generation Language (DGL) for generating synthetic data distributions through iterators that sequentially created the tables in schema graph dependency order. Popular benchmarks such as TPC-D [4], and its successors, TPC-H [5] and TPC-DS [6], which model OLAP environments, have been developed over the past three decades using similar approaches. Very recently, the **DSB** benchmark [19] proposes to extend TPC-DS with complex data distributions, and feature evaluation mechanisms for both workload-driven and traditional database systems.

In this context, a variety of techniques have been proposed to improve the speed of database production. For instance, MUDD [42] and PSDG [23] generate all related data at the same time. These frameworks decouple data generation from data description, facilitating customization to suit user needs. In the distributed setting, a fast way of generating references is through recomputing since it eliminates the I/O costs incurred to satisfy referential constraints across relations that are present across different nodes. PDGF [34] was designed with this goal of achieving scalability and decoupling. To regenerate the same sequences, it exploits determinism in pseudo-random number generators (PRNG).

A recent high-performance technique, **Myriad** [7], also leverages parallelism for achieving scalability. It implements an efficient parallel execution strategy with extensive use of PRNGs having random access support. With these PRNGs, Myriad distributes the generation process across the compute nodes and ensures that they run independently, without imposing any restrictions on the data modeling language.

In the tutorial, we cover Myriad and DSB as exemplars of ab initio generators. Their limitations, such as the difficulty of matching standard distributions to real-world data, especially over multivariate spaces, are also highlighted.

III. Database-Dependent Regeneration

We now turn our attention to database-dependent regeneration techniques. DBSynth [33], an extension to PDGF, builds data models from an existing database by extracting schema information, and using sampling to construct histograms and dictionaries of text-valued data. For textual data, Markov chain generators are employed to analyze the word combination frequencies and probabilities. After the model construction, PDGF is invoked to generate the corresponding data. Similar to DBSynth, RSGen [41] takes a metadata dump, including 1-D histograms, as the input. It uses a bucketization technique that is capable of generating trillions of records within a small memory footprint.

UpSizeR [43] is a graph-based tool that uses attribute correlations extracted from an existing database to generate an equivalent synthetic database. A derivative work, Rex [15] produces a uniformly scaled version of the original database, while maintaining referential constraints and the distributions between the consecutive linked tables. In contrast, the more recent **Dscaler** [51] addresses the problem of generating a non-uniformly scaled version of a database, where individual tables are scaled by different factors, using fine-grained, per-tuple correlations for key attributes.

Learning-based Approaches

A different trajectory of synthetic data regeneration has been followed by the ML community, captured in the comprehensive survey by Fan et al [20]. In this corpus, the frameworks fall into the following classes:

a) Statistical Models: Here the input data is modeled with a multivariate distribution, which is then used to sample the output synthetic data. The dependence between variates is captured using techniques such as copulas [24], [32], Bayesian networks [50], Gibbs sampling [30], and Fourier decompositions [12]. Alternatively, synopses-based approaches such as wavelets and multi-dimensional sketches, build compact data summaries that are amenable to the estimation of joint distributions [16], [47].

b) Neural Models: The frameworks here use deep generative models to approximate the input data. The underlying techniques include autoencoders [22], variational autoencoders (VAE) [44], and more recently, generative adversarial networks (GANs) [11], [17], [18], [29], [31], [48]. For instance, **CTGAN** [48] uses a conditional generator to model complex real-world data featuring multimodal distributions of continuous columns, and highly imbalanced categorical columns.

In the tutorial, we cover Dscaler and CTGAN in detail. Their limitations, such as the propensity for empty results over complex queries due to not fully capturing subtle correlations across attributes and tables, are also highlighted.

IV. Parameterized Workload-based Regeneration

The query-workload-based regeneration techniques focus on ensuring *volumetric similarity*. That is, assuming a common choice of query execution plans at the client and vendor sites, the output row cardinalities of individual operators in these plans are very similar in the original and synthetic databases. This similarity helps to preserve the multi-dimensional layout and flow of the data, a prerequisite for achieving similar performance on the client's workload.

With parameterized workload inputs, the queries are parameterized on the constants in the query, and the generators provide some parameter settings for which the intermediate cardinalities are matched. Supporting parameterized inputs allows for generating volumetrically similar databases for input query workloads while preserving client data security.

The concept of using cardinalities from a query plan tree was first introduced in QAGen [13], [26]. Given a query plan, it constructs a *symbolic database*, where attribute values are generalized to symbols (variables), and models the volumetric constraints using the symbols in the database. Subsequently, a constraint satisfaction program (CSP) is invoked to identify values for these symbols such that all the constraints are satisfied. While QAGen supported only one query plan in the input, a follow-up tool called MyBenchmark [27], [28], created a symbolic database on a per-query basis and then heuristically merged these individual databases to the extent possible.

In practice, constructing a database on a per-query basis can lead to impractical overheads. This scalability limitation was recently addressed by **Touchstone** [25], [46]. Touchstone builds a group of random column generators, which determine the data distributions of all non-key columns. It chooses the query parameter values by adjusting the related column generators and decomposes the query trees in a manner that decouples the dependencies among the columns. The data is finally generated over a distributed platform for independent and parallel tuple generation.

We cover Touchstone as an exemplar of parameterized workload-based regeneration in the tutorial. Limitations such as the inherent difficulty in satisfactorily modeling new queries that may be evaluated at the vendor site are also highlighted.

V. Instantiated Workload-based Regeneration

In this class of workload-dependent generation, the predicate constants in the queries are *pre-specified* in the input. This restriction renders the generation problem more difficult since we now need to ensure that the intermediate cardinalities are matched for precisely these constants. The benefit, however, of

ensuring volumetric similarity in this "strict" environment is that the resultant databases are closer to the original, thereby providing better modeling of new queries that are evaluated at the vendor site as part of the testing process.

A unified and declarative mechanism for representing volumetric constraints from the query plans, called *cardinality constraints* (CCs), was proposed in DataSynth [8], [9]. In particular, a CC dictates that the output of a given relational expression over the generated database should feature a specified number of rows. Given a set of CCs, DataSynth proposed algorithms based on the LP solver and graphical models to instantiate tables that satisfy the constraints. Specifically, the data space is partitioned into a set of regions and a variable is created wrt each region encoding its volume. Using these variables, the CCs are expressed as a linear feasibility problem. The solution is used to generate the output database with a sampling-based approach.

HYDRA [3], [40], extends DataSynth by adding functionality, dynamism, scale, and robustness. It provides extended workload coverage by additionally handling queries with (duplicate eliminating) Projection-based SQL constructs such as `Distinct, Group By, Union` [36], [37]. Workload scalability is provided by optimizing the data-space partitioning strategy leading to a significant reduction in the complexity of the LP without compromising on the solution quality [39].

Inspired by the "dataless database" philosophy of the CODD metadata processor [10], [45], a unique feature of Hydra is that it delivers a minuscule database summary as the output rather than the materialized data itself. This summary can be used for dynamically generating data during query execution. Also, the summary production pipeline is inherently data-scale-free, and does not depend on the size of the database. Further, to improve the accuracy for new queries, Hydra additionally exploits metadata statistics maintained by the database engine and adds an objective function to the LP to create a more representative database [38].

In an interesting twist, [21] focused specifically on generating key columns that comply with referential integrity constraints. To model these requirements, a novel approach that leverages *denial constraints* was proposed. These denial constraints forbid tuples from having the same foreign-key value under specified conditions. The foreign-key column is constructed using graph-theoretic concepts of conflict hypergraphs and hypergraph coloring.

Learning-based Approaches

A learning-based framework, called **SAM** (Supervised Deep Autoregressive Model) [49], was recently proposed in this generation category. SAM trains an autoregressive model to capture the joint data distribution of the database tables. Further, inverse probability weighting and scaling algorithms are used to efficiently sample from the model to produce the base relations. Finally, the join keys are assigned using a Group-and-Merge algorithm.

In the tutorial, we discuss the Hydra and SAM approaches in detail. Open challenges such as ensuring workload scalability in LP-based techniques, and enhancing the coverage of SQL operators in learning-based techniques, are also highlighted.

VI. Future Research Directions

In the final stage of the tutorial, we outline a set of open technical problems and future research directions, including:

a) Dynamic Data Distribution: All the generators discussed in this proposal are passive in the sense that the database is produced once and subsequently consumed. However, we could also consider the creation of active databases that adapt their contents based on the testing environment. As a case in point, the DSB benchmark could be made truly dynamic by incorporating active databases as a complementary addition to its adaptive query workload.

b) GAN-based Synthesis for Multi-Relational Databases: The GAN-based approaches have shown impressive results in producing realistic tabular data. However, for viability, it is essential to extend them to multi-relational environments. This would require dealing with challenges such as: (a) modelling join cross correlations, (b) ensuring referential integrity, (c) ensuring volumetric similarity, which typically gets harder as the number of joins increase [38].

c) Going Beyond Volumetric Similarity: While volumetric similarity captures the critical characteristic of data flow, it lacks fine-grained information such as data skew and data ordering. These characteristics can be especially important for mimicking performance of (a) hash operations, due to the potential for data spilling to disk in presence of duplicates, and (b) sort operations, due to the impact on data movement and number of comparisons. Therefore, there is a need to expand the generator scope to include data characteristics such as duplication distribution and the extent of presortedness [35].

d) Expand SQL Coverage: As described previously, the contemporary workload-dependent frameworks do support the core SQL operators. However, for comprehensive testing it is necessary to expand the scope to include other common SQL constructs such as group filters, nested queries, conditional statements and set operators.

e) Improve Workload Scalability: While the workload-dependent techniques achieve volumetric similarity, the size of the input workloads that they can handle does not scale well. For instance, the LP constraint solver used in most of these techniques is crippled by the exponential increase in the number of variables with workload scale. A promising recourse is to introduce *approximation*, whereby volumetric accuracy is marginally compromised to achieve solution tractability.

f) Improve Robustness to Unseen Queries: Due to the under-determined nature of the constraint space, there are many feasible solutions to the LP constructed by workload-dependent techniques. The solvers produce a random outcome from the convex polytope of feasible solutions. However, from the perspective of robustness to future unseen queries, it is preferable to identify a geometrically-centric solution as compared to a sparse corner solution that assigns zeros to many variables.

VII. Presenter Details

Anupam Sanghi is a Research Scientist at IBM Research India. He pursued his PhD at the Indian Institute of Science, Bangalore, where he was an IBM PhD Fellow. His research interests include the design and testing of database systems, and data mining.

Jayant Haritsa is on the computer science faculty of the Indian Institute of Science, Bangalore, since 1993. He received the PhD degree from the Univ. of Wisconsin (Madison). He is a Fellow of ACM and IEEE for his contributions to the design and implementation of database engines.

References

[1] Dagstuhl Seminar 21442. Ensuring the Reliability and Robustness of Database Management Systems. dagstuhl.de/en/program/calendar/semhp/?semnr=21442

[2] General Data Protection Regulation. en.wikipedia.org/wiki/General_Data_Protection_Regulation

[3] HYDRA Database Regenerator. https://dsl.cds.iisc.ac.in/projects/HYDRA/index.html

[4] TPC-D. tpc.org/tpcd/default5.asp

[5] TPC-H. tpc.org/tpch/

[6] TPC-DS. tpc.org/tpcds/

[7] A. Alexandrov, K. Tzoumas, and V. Markl. Myriad: Scalable and Expressive Data Generation. *PVLDB*, 5(12):1890–1893, 2012.

[8] A. Arasu, R. Kaushik, and J. Li. Data Generation using Declarative Constraints. *Proc. of ACM SIGMOD Conf.*, 2011, pgs. 685–696.

[9] A. Arasu, R. Kaushik, and J. Li. DataSynth: Generating Synthetic Data using Declarative Constraints. *PVLDB*, 4(12):1418–1421, 2011.

[10] Ashoke S., and J. R. Haritsa. CODD: A Dataless Approach to Big Data Testing. *PVLDB*, 8(12):2008–2011, 2015.

[11] M. K. Baowaly, C. Lin, C. Liu, and K. Chen. Synthesizing electronic health records using improved generative adversarial networks *JAMIA*, 26(3):228–241, 2019.

[12] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar Privacy, Accuracy, and Consistency Too: A Holistic Solution to Contingency Table Release *Proc. of 26th PODS Conf.*, 2007, pgs. 273–282.

[13] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: Generating Query-Aware Test Databases. *Proc. of ACM SIGMOD Conf.*, 2007, pgs. 341–352.

[14] N. Bruno and S. Chaudhuri. Flexible Database Generators. *Proc. of 31st VLDB Conf.*, 2005, pgs. 1097–1107.

[15] T. S. Buda, T. Cerqueus, J. Murphy and M. Kristiansen. ReX: Extrapolating Relational Data in a Representative Way. *Proc. of BICOD Conf.*, 2015, pgs. 95–107.

[16] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends Databases*, 4(1-3):1–294, 2012.

[17] H. Chen, S. Jajodia, J. Liu, N. Park, V. Sokolov, and V. S. Subrahmanian. FakeTables: Using GANs to Generate Functional Dependency Preserving Tables with Bounded Real Data. *Proc. of 28th IJCAI Conf.*, 2019, pgs. 2074–2080.

[18] E. Choi, S. Biswal, B. A. Malin, J. Duke, W. F. Stewart, and J. Sun. Generating Multi-label Discrete Patient Records using Generative Adversarial Networks. *PMLR*, 68:286–305, 2017.

[19] B. Ding, S. Chaudhuri,J. Gehrke, and V. Narasayya. DSB: A Decision Support Benchmark for Workload-Driven and Traditional Database Systems. *PVLDB*, 14(13):3376–3388, 2021.

[20] J. Fan, T. Liu, G. Li, J. Chen, Y. Shen, X. Du. Relational Data Synthesis using Generative Adversarial Networks: A Design Space Exploration. *PVLDB*, 13(11):1962–1975, 2020.

[21] A. Gilad, S. Patwa, and A. Machanavajjhala. Synthesizing Linked Data Under Cardinality and Integrity Constraints. *Proc. of ACM SIGMOD Conf.*, 2021, pgs. 619–631.

[22] L. Gondara, and K. Wang MIDA: Multiple Imputation Using Denoising Autoencoders. *Proc. of PAKDD Conf.*, 2018, pgs. 260–272.

[23] J. E. Hoag, and C. W. Thompson. A Parallel General-Purpose Synthetic Data Generator. *Proc. of ACM SIGMOD Conf.*, 2007, pgs. 19–24.

[24] H. Li, L. Xiong, L. Zhang, and X. Jiang. DPSynthesizer: Differentially Private Data Synthesizer for Privacy Preserving Data Sharing. *PVLDB*, 7(13):1677–1680, 2014.

[25] Y. Li, R. Zhang, X. Yang, Z. Zhang, and A. Zhou. Touchstone: Generating Enormous Query-Aware Test Databases. *USENIX ATC*, 2018, pgs. 575–586.

[26] E. Lo, C. Binnig, D. Kossmann. A framework for testing DBMS features. *The VLDB Journal*, 19(2):203–230, 2010.

[27] E. Lo, N. Cheng, W.-K. Hon. Generating Databases for Query Workloads. *PVLDB*, 3(1):848–859, 2010.

[28] E. Lo, N. Cheng, W. W. Lin, W.-K. Hon, and B. Choi. MyBenchmark: generating databases for query workloads. *The VLDB Journal*, 23(6):895–913, 2014.

[29] P. Lu, P. Wang, and C. Yu. Empirical Evaluation on Synthetic Data Generation with Generative Adversarial Network. *Proc. of 9th WIMS Conf.*, 2019, pgs. 16:1–6.

[30] Y. Park, and J. Ghosh. PeGS: Perturbed Gibbs Samplers that Generate Privacy-Compliant Synthetic Data. *Trans. Data Priv.*, 7(3):253–282, 2014.

[31] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim. Data Synthesis based on Generative Adversarial Networks *PVLDB*, 11(10): 1071–1083, 2018.

[32] N. Patki, R. Wedge, and K. Veeramachaneni. In DSAA, pages 399–410, 2016. T. Rabl, M. Danisch, M. Frank, S. Schindler, and H. Jacobsen. The Synthetic Data Vault. *Proc. of IEEE DSAA Conf.*, 2016, pgs. 399–410.

[33] T. Rabl, M. Danisch, M. Frank, S. Schindler, and H. Jacobsen. Just can't get enough - Synthesizing Big Data. *Proc. of ACM SIGMOD Conf.*, 2015, pgs. 1457–1462.

[34] T. Rabl, M. Frank, H. M. Sergieh and H. Kosch. A Data Generator for Cloud-Scale Benchmarking. *Proc. of 2nd TPCTC Conf.*, 2010, pgs. 41–56.

[35] A. Sanghi. HYDRA: A Dynamic Approach to Database Regeneration *Ph.D. Thesis*, IISc, 2022. dsl.cds.iisc.ac.in/publications/thesis/anupam_sanghi.pdf

[36] A. Sanghi, S. Ahmed, and J. R. Haritsa. Projection-Compliant Database Generation. *PVLDB*, 15(5):998–1010, 2022.

[37] A. Sanghi, S. Ahmed, P. Rawale, and J. R. Haritsa. Data Generation using Join Constraints. *Tech. Report TR-2022-01*, DSL/CDS,IISc, 2022. dsl.cds.iisc.ac.in/publications/report/TR/TR-2022-01.pdf.

[38] A. Sanghi, Rajkumar S., and J. R. Haritsa. Towards Generating HiFi Databases. *Proc. of 26th DASFAA Conf.*, 2021, pgs. 105-112.

[39] A. Sanghi, R. Sood, J. R. Haritsa, and S. Tirthapura. Scalable and Dynamic Regeneration of Big Data Volumes. *Proc. of 21st EDBT Conf.*, 2018, pgs. 301-312.

[40] A. Sanghi, R. Sood, D. Singh, J. R. Haritsa, and S. Tirthapura. HYDRA: A Dynamic Big Data Regenerator. *PVLDB*, 11(12):1974-1977, 2018.

[41] E. Shen, and L. Antova. Reversing statistics for scalable test databases generation. *Proc. of DBTest Workshop*, 2013, pgs. 1–6.

[42] J. M. Stephens and M. Poess. MUDD: A Multi-dimensional Data Generator. *Proc. of 4th WOSP*, 2004, pgs. 104–109.

[43] Y. C. Tay, B. T. Dai, D. T. Wang, E. Y. Sun, Yong Lin and Yuting Lin. UpSizeR: Synthetically scaling an empirical relational database. *Inf. Syst.*, 38(8):1168–1183, 2013.

[44] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate Query Processing using Deep Generative Models. *arXiv:1903.10000*, 2019.

[45] R. S. Trivedi, I. Nilavalagan, and J. R. Haritsa. CODD: COnstructing Dataless Databases. *Proc. of DBTest Workshop*, 2012, pgs. 1–6.

[46] Q. Wang, Y. Li, R. Zhang, K. Shu, Z. Zhang, and A. Zhou. A Scalable Query-Aware Enormous Database Generator for Database Evaluation. *TKDE*, 10.1109/TKDE.2022.3153651, 2022.

[47] X. Xiao, G. Wang, and J. Gehrke. Differential Privacy via Wavelet Transforms. *TKDE*, 23(8):1200–1214, 2011.

[48] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling Tabular Data using Conditional GAN. *Proc. of 33rd NeurIPS Conf.*, 2019.

[49] J. Yang, P. Wu, G. Cong, T. Zhang, X. He. SAM: Database Generation from Query Workloads with Supervised Autoregressive Models. *Proc. of ACM SIGMOD Conf.*, 2022, pgs. 1542–1555.

[50] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private Data Release via Bayesian Networks. *TODS*, 42(4):1–41, 2017.

[51] J. W. Zhang and Y. C. Tay. Dscaler: Synthetically Scaling A Given Relational Database. *PVLDB*, 9(14):1671–1682, 2016.