Towards Generating HiFi Databases

Anupam Sanghi(✉)*, Rajkumar S., and Jayant R. Haritsa

Indian Institute of Science, Bengaluru, India
{anupamsanghi,srajkumar,haritsa}@iisc.ac.in

Abstract. Generating synthetic databases that capture essential data characteristics of client databases is a common requirement for database vendors. We recently proposed Hydra, a workload-aware and scale-free data regenerator that provides statistical fidelity on the volumetric similarity metric. A limitation, however, is that it suffers poor accuracy on unseen queries. In this paper, we present HF-Hydra (HiFi-Hydra), which extends Hydra to provide better support to unseen queries through (a) careful choices among the candidate synthetic databases and (b) incorporation of metadata constraints. Our experimental study validates the improved fidelity and efficiency of HF-Hydra.

Keywords: Big Data Management, Data Summarization, Data Warehouse, OLAP Workload, DBMS Testing

1 Introduction

Database vendors often need to generate synthetic databases for a variety of use-cases, including: (a) testing engine components, (b) testing of database applications with embedded SQL, and (c) performance benchmarking. Several approaches to synthetic data generation have been proposed in the literature (reviewed in [7]) – in particular, a declarative approach of workload-aware data regeneration has been advocated over the last decade ([2], [1], [4], [5]).

Workload-Aware Data Regeneration. Consider the database schema with three relations shown in Fig. 1(a), and a sample SQL query (Fig. 1(b)) on it. In the corresponding query execution plan (Fig. 1(c)), each edge is annotated with the associated cardinality of tuples flowing from one operator to the other. This is called an annotated query plan (AQP). From an AQP, a set of cardinality constraints (CCs) are derived, as enumerated in Fig. 1(d). The goal here is to achieve *volumetric similarity* – that is, on a given query workload, when these queries are executed on the synthetic database, the result should produce similar AQPs. In other words, the database should satisfy all the CCs.

* This work was supported by IBM PhD Fellowship Award.

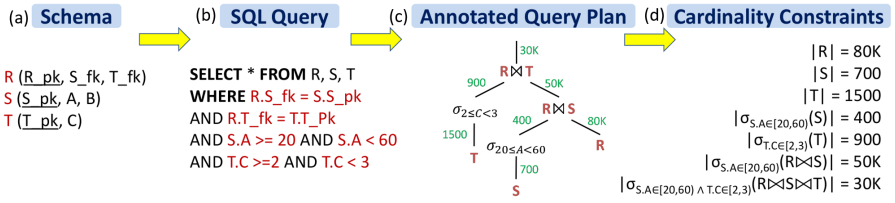

Fig. 1. Example Annotated Query Plan (AQP) and Cardinality Constraints (CC)

1.1 Hydra

Hydra ([5], [6]) is a workload-aware generator recently developed by our group. For each relation in the database, Hydra first constructs a corresponding denormalized relation (without key columns), called a *view*. To generate a view R, the domain space of R is partitioned into a set of disjoint *regions* determined by the filter predicates in the CCs. Further, a variable is created for each region, representing its row cardinality in the synthetic database. Next, an SMT Problem is constructed, where each CC is expressed as a linear equation in these variables. After solving the problem, the *data generator* picks a unique tuple within the region-boundaries and replicates it as per the region-cardinality obtained from the solution.

To make the above concrete, consider the constraint for relation S (from Fig. 1) shown by the red box (A in [20, 40) and B in [15000, 50000)) in Fig. 2, and having an associated row-count 150 (say). Likewise, a green constraint is also shown – say with row-count 250. Accordingly, the SMT problem constructed is:

$$x_1 + x_2 = 250, \quad x_2 + x_3 = 150, \quad x_1, x_2, x_3, x_4 \geq 0$$

The process of extracting relations from the views, while ensuring referential integrity (RI), forces the addition of some (spurious) tuples in the dimension tables. At the end, the output consists of concise constructors, together called as the *database summary*. An example summary is shown in Fig. 3 – the entries of the type a - b in the PK column (e.g. 101-250 for S_pk), represent rows with values (a, $a+1$,...,b) for that column, keeping others unchanged. The summary

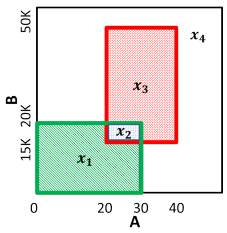

Fig. 2. Domain Partitioning

R			S			T	
R_pk	S_fk	T_fk	S_pk	A	B	T_pk	C
1-30K	101	601	1-100	0	15K	1-600	0
30001-50K	251	1	101-250	20	15K	601-1500	2
50001-60K	1	601	251-500	30	5K		
60001-80K	501	1	501-700	0	20K		

Fig. 3. Hydra Example Database Summary

makes Hydra amenable to handle Big Data volumes because the data can now be generated *dynamically*, i.e., "on-demand" during query execution, thereby obviating the need for materialization. Also, the summary construction time is *data-scale-free*, i.e., independent of the database size.

Limitations. As discussed above, Hydra is capable of efficiently delivering volumetric similarity on *seen* queries. However, the ability to generalize to new queries can be a useful feature for the vendor as part of the ongoing evaluation exercise. This is rendered difficult for Hydra due to the following design choices:

No Preference among Feasible Solutions: There can be several feasible solutions to the SMT problem. However, Hydra does not prefer any particular solution over the others. Moreover, due to the usage of Simplex algorithm internally, the SMT solver returns a sparse solution, i.e., it assigns non-zero cardinality to very few regions. This leads to very different *inter-region distribution* of tuples in the original and synthetic databases.

Artificial Skewed Data: Within a region that gets a non-zero cardinality assignment, Hydra generates a single unique tuple. As a result, a highly skewed data distribution is generated, which leads to an inconsistent *intra-region distribution* of tuples. Furthermore, the artificial skew can cause hindrance in efficient testing of queries, and gives an *unrealistic look* to the data.

Non-compliance with the Metadata: The metadata statistics present at the client site are transferred to the vendor and used to ensure matching plans at both sites. However, these statistics are not used in the data generation process, leading to data that is out of sync with the client meta-data.

1.2 HF-Hydra

In this work, we present **HF-Hydra** (High-Fidelity Hydra), which materially extends Hydra to address the above robustness-related limitations while retaining its desirable data-scale-free and dynamic generation properties.

The end-to-end pipeline of HF-Hydra's data generation is shown in Fig. 4. The client AQPs and metadata stats are given as input to *LP Formulator*. Using the inputs, the module constructs a refined partition, i.e. it gives finer regions. Further, a linear program (LP) is constructed by adding an objective function to pick a *desirable* feasible solution. From the LP solution, which is computed using the popular Z3 solver [8], the *Summary Generator* produces a richer database summary.

A sample summary produced by HF-Hydra on our running example is shown in Fig. 5. We see that the number of regions, characterized by the number of rows in the summary tables, are more in comparison to Hydra. Also, intervals are stored instead of points which support generation of a spread of tuples within each region. Tuples are generated uniformly within the intervals using the *Tuple Generator* module. These stages are discussed in detail in Sections 2 and 3.

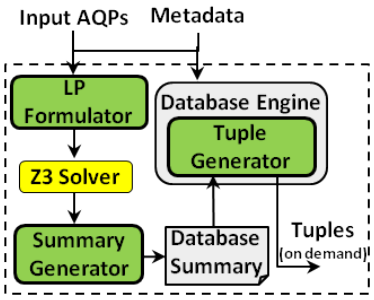

Fig. 4. HF-Hydra Pipeline

R		
R_pk	S_fk	T_fk
1-20k	101-180	601-900
20001-30k	181-250	901-1500
30001-45k	251-300	1-400
45001-50k	301-500	401-600
50001-55k	1-60	601-900
55001-60k	61-100	901-1500
60001-68k	501-550	1-400
68001-80k	551-700	401-600

T	
T_pk	C
1-400	[0,1) U [3,7)
401-600	[2,3)
601-900	[1,2) U [7,10)
901-1500	[2,3)

S		
S_pk	A	B
1-60	[0-10)	[15k-20k)
61-100	[10-20)	[15k-20k)
101-180	[20-30)	[15k-18k)
181-250	[20-30)	[18k-20k)
251-300	[30-60)	[0,10k) U [50k,60k)
301-500	[30-60)	[10k,15k) U [60k,80k)
501-550	[0-15) U [60,75)	[20k-50k)
551-700	[15-20) U [75,80)	[20k-50k)

Fig. 5. HF-Hydra Database Summary

In a nutshell, the addition of an objective function in the LP improves the inter-region tuple distribution. Further, having refined regions, plus uniform tuple distribution within these finer regions, improves the intra-region tuple distribution. Finally, addition of constraints from the metadata statistics ensures metadata-compliance.

We evaluate the efficacy of HF-Hydra by comparing its volumetric similarity with Hydra on unseen queries. Our results, elaborated in Section 4, indicate a substantive improvement – specifically, the volumetric similarity on filter constraints of unseen queries was better by more than **30 percent**, as measured by the UMBRAE model-comparison metric [3]. Further, we also show that HF-Hydra ensures metadata compliance. A sample table illustrating HF-Hydra delivers more realistic databases in comparison to Hydra is shown in [7].

2 LP Formulation

We now show how the LP is constructed from the AQPs and metadata. The summary steps are the following (complete details in [7]):

1. **Creating Metadata CCs.** Constraints are derived from the metadata statistics. Typically, the statistics include histograms and most common values (MCVs) with the corresponding frequencies. These are encoded as *metadata CCs*:
   i. $|\sigma_{A=a}(R)| = c_a$, for a value a stored in MCVs with frequency c_a (for column A).
   ii. $|\sigma_{A\in[l,h)}(R)| = B$, for a histogram bucket (for column A) with boundary $[l, h)$, having total row-count equal to B.

   Let there be a total of m such metadata CCs.
2. **Region Partitioning.** *Refined* regions are constructed using region-based domain partitioning, leveraging the CCs derived from both AQPs and metadata. Let the total number of resultant regions be n, where the row-cardinality of region i is captured in variable x_i.

minimize $\sum_{j=1}^{m} \epsilon_j$, subject to:
1. $-\epsilon_j \leq (\sum_{i:I_{ij}=1} x_i) - k_j \leq \epsilon_j, \forall j \in [m]$,
2. $C_1, C_2, ..., C_q$,
3. $x_i \geq 0 \quad \forall i \in [n], \quad \epsilon_j \geq 0 \quad \forall j \in [m]$

Fig. 6. MDC LP Formulation

minimize $\sum_{i=1}^{n} \epsilon_i$, subject to:
1. $-\epsilon_i \leq x_i - \tilde{x}_i \leq \epsilon_i, \forall i \in [n]$,
2. $C_1, C_2, ..., C_q$,
3. $x_i, \epsilon_i \geq 0, \quad \forall i \in [n]$

Fig. 7. OE LP Formulation

3. **Formulating LP Constraints.** The CCs from AQPs are added as explicit LP constraints, as in the original Hydra. Let there be q such CCs denoted by $C_1, C_2, ..., C_q$.
4. **Constructing Objective.** An *optimization function* is added to find a feasible solution that is close to the *estimated solution*. We use two notions of estimated solution:
   i. **Metadata Constraints Satisfaction (MDC):** Here the distance between the output cardinalities from metadata CCs and the sum of variables that represent the CCs is minimized. The LP thus obtained is shown in Figure 6, with I_{ij} being an indicator variable, which takes value 1 if region i satisfies the filter predicate in the jth metadata CC, 0 otherwise.
   ii. **Optimizer Estimates Satisfaction (OE):** Here, instead of directly enforcing metadata CCs, the estimated cardinality for each region is obtained from the database engine using the optimizer's selectivity estimation logic. The objective function minimizes the distance between the solution and these estimates. The estimated cardinality $\tilde{x}_i$ for a region i is computed by constructing an SQL query equivalent for the region and using the query's estimated selectivity obtained from its compile-time plan. The LP produced using OE strategy is shown in Figure 7.

   Our choice of minimizing L1 distance is because query execution performance is linearly dependent on the row count, especially when all joins are PK-FK joins. In picking between the MDC and OE strategies, the following considerations apply: MDC has better metadata compliance due to explicit enforcement of the associated constraints. Further, its solution has higher sparsity because no explicit constraint is applied at a per-region level. However, while sparsity does make summary production more efficient, it adversely affects volumetric accuracy for higher levels of joins, as compared to OE.

3 Data Generation

Post LP-solving, the data generation pipeline proceeds in the following stages (complete details in [7]):

1. **Ensuring Referential Integrity.** Since each view is processed independently, these solutions may have inconsistencies. Specifically, when F, the fact table view, has a tuple whose value combination for the attributes that

it borrows from D, the dimension table view, does not have a matching tuple in D, then it causes a reference violation. To avoid it, for each region f of F, we maintain the populated regions in D that have an interval intersection with f for the borrowed columns. If no such region in D is found, then a new region with the intersection portion is added and assigned a cardinality of 1. This fixes the reference violation but leads to an additive error of 1 in the relation cardinality for the dimension table.

2. **Generating Relation Summary.** Here the borrowed attribute-set in a view is replaced with appropriate FK attributes. In contrast to Hydra's strategy of picking a single value in the FK column for a region, here we indicate a *range* to achieve a good span. To compute the FK column values for a region f, the corresponding matching regions from dimension table are fetched and the union of PK column ranges of these regions is returned.
3. **Tuple Generation.** The aim here is generate tuples uniformly within each region. Based on interval lengths that are contained for an attribute in the region, the ratio of tuples to be generated from each interval is computed. Now, if n values have to be generated within an interval I, then I is split into n equal sub-intervals and the center point within each interval is picked. If the range does not allow splitting into n sub-intervals, then it is split into the maximum possible sub-intervals, followed by a round-robin instantiation. The PK column values are generated consecutively, similar to row-numbers. This deterministic approach is well-suited for dynamic generation. If a materialized output is desired, then random values can be picked within intervals.

4 Experimental Evaluation

We now move on to empirically evaluating the performance of HF-Hydra against Hydra. For our experiments, we used a 1 GB version of the TPC-DS benchmark, hosted on a PostgreSQL v9.6 engine operating on a vanilla workstation. The SMT/LP problems were solved using Z3 [8].

We constructed a workload of 110 representative queries, which was then split randomly into *training* and *testing* sets of 90 and 20 queries. The associated AQPs led to formulation of 225 and 51 CCs, respectively. These CCs were a mix of pure filters on base relations, and CCs that involve filters along with 1 to 3 joins. Further, 2622 metadata CCs were derived from histograms and MCVs data.

4.1 Volumetric Similarity

For evaluating volumetric accuracy, we used the **UMBRAE** (Unscaled Mean Bounded Relative Absolute Error) model-comparison metric [3], with Hydra serving as the reference model. An UMBRAE value U ranges over positive numbers, where $U < 1$ implies $(1 - U) * 100\%$ better performance wrt baseline model, $U > 1$ implies $(U - 1) * 100\%$ worse performance and $U = 1$ shows no improvement.

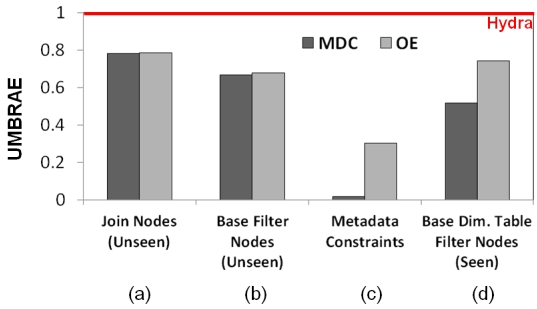

Fig. 8. Accuracy

The UMBRAE values obtained by the two flavors of HF-Hydra over the 20 test queries are shown in Fig. 8 (a)-(b). For clear understanding, the results for base filters and join nodes are shown separately. We see that HF-Hydra delivers more than **30%** better performance on filters, and an improvement of over **20%** with regard to joins. The higher improvement on filters is expected because the accuracy of metadata statistics, being on a per-column basis, is best at the lower levels of the plan tree.

Metadata compliance is evaluated in Fig. 8(c). A substantial improvement over Hydra is seen here – **98%** and **70%** for MDC and OE, respectively.

Interestingly, HF-Hydra outperforms Hydra even on the volumetric accuracy for *seen queries*, as captured in Fig. 8(d). Specifically, an improvement of **48%** and **26%** for MDC and OE, respectively, with regard to the base filter nodes on dimension tables. This benefit is an outcome of better distribution of tuples over regions, reducing the likelihood of mismatch between populated regions in the fact-table and empty regions in the dimension-table.

On the metrics considered thus far, MDC outperformed OE. However, for higher level joins, OE did better than MDC. Specifically, **33%** better for two-join cases and **13%** better for three-join cases. This is primarily because OE adheres to constraints at a per region level while MDC generates a sparse solution.

4.2 Database Summary Overheads

The database summaries generated by HF-Hydra and Hydra differ significantly in their structures. The former has many more regions, and stores intervals instead of points within a region. Due to these changes, a legitimate concern could be the impact on the size of the summary and the time taken to generate

	Hydra	MDC	OE
Summary Size	40 KB	6 MB	985 MB
Tuple Instantiation Time	6 s	37 s	51 s

Table 1. Space and Time Analysis

data from it at run-time. To quantitatively evaluate this concern, the space and time overheads are enumerated in Table 1. We see here that there is certainly a large increase in summary size, going from kilobytes to megabytes – however, in *absolute* terms, the summary size is still small enough to be easily viable on contemporary computing platforms. When we consider the time aspect, again there is an expected increase in the generation time from a few seconds to several tens of seconds, but here too the absolute values are small enough (sub-minute) to make HF-Hydra usable in practice. Further, it is important to recall that these summary sizes and their construction time are independent of the client database size (experiments validating this claim are described in [7]).

5 Conclusions

Testing database engines efficiently is a critical issue in the industry, and the ability to accurately mimic client databases forms a key challenge in this effort. In contrast to the prior literature which focused solely on capturing database fidelity with respect to a known query workload, in this paper we have looked into the problem of generating databases that are robust to unseen queries. In particular, we presented HF-Hydra, which materially extends the state-of-the-art Hydra generator by bringing the potent power of metadata statistics and optimizer estimates to bear on the generation exercise. The resulting fidelity improvement was quantified through experimentation on benchmark databases, and the UMBRAE outcomes indicate that HF-Hydra successfully delivers high-fidelity databases.

Acknowledgements. We thank Tarun Kumar Patel and Shadab Ahmed for their valuable inputs in the implementation of this work.

References

1. A. Arasu, R. Kaushik, and J. Li. Data Generation using Declarative Constraints. In *ACM SIGMOD Conf.*, 2011, pgs. 685-696.
2. C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: Generating Query-Aware Test Databases. In *ACM SIGMOD Conf.*, 2007, pgs. 341-352.
3. C. Chen, J. Twycross, and J. M. Garibaldi. A new accuracy measure based on bounded relative error for time series forecasting. *PLoS ONE*, 12(3): e0174202, 2017.
4. Y. Li, R. Zhang, X. Yang, Z. Zhang, and A. Zhou. Touchstone: Generating Enormous Query-Aware Test Databases. In *USENIX ATC*, 2018, pgs. 575–586.
5. A. Sanghi, R. Sood, J. R. Haritsa, and S. Tirthapura. Scalable and Dynamic Regeneration of Big Data Volumes. In *21st EDBT Conf.*, 2018, pgs. 301-312.
6. A. Sanghi, R. Sood, D. Singh, J. R. Haritsa, and S. Tirthapura. HYDRA: A Dynamic Big Data Regenerator. *PVLDB*, 11(12):1974-1977, 2018.
7. A. Sanghi, Rajkumar S., and J. R. Haritsa. High Fidelity Database Generators. *Tech. Report TR-2021-01*, DSL/CDS, IISc, 2021, `dsl.cds.iisc.ac.in/publications/report/TR/TR-2021-01.pdf`
8. Z3. `https://github.com/Z3Prover/z3`