Survivorship Bias in Industrial Database Workloads

Ryan Marcus[*]
rcmarcus@seas.upenn.edu
University of Pennsylvania

Jeffrey Tao[*]
jefftao@seas.upenn.edu
University of Pennsylvania

Peizhi Wu[*]
pagewu@seas.upenn.edu
University of Pennsylvania

Zijie Zhao[*]
bytez@seas.upenn.edu
University of Pennsylvania

Abstract

Several industrial data platforms have recently released characterizations of their real-world workloads, prompting database researchers to consider new research directions and benchmarks that go beyond synthetics. While these workload characterizations are undoubtedly useful, researchers (including the authors of this paper) and practitioners often assume that existing workloads are representative of user needs. Through two case studies over real workload data, we show that industrial workloads represent a "negotiation" between users and the data platform, in which users shape their workloads to take advantage of the parts of the data platform that work well, while avoiding the parts of the data platform that are not optimized. This shaping effect is an example of *survivorship bias*: workload characterizations are built over the queries that "survive" (or thrive) on a particular data platform. Based on these case studies, we make several suggestions for how both researchers and practitioners can better contextualize workload traces and characterizations.

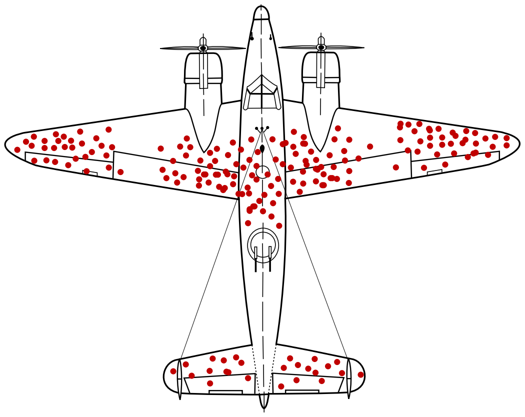

Figure 1: A diagram of bullet holes on returning WWII fighter planes. Since these planes all returned, the sample is biased towards "survivors." One might think that additional armor should be added to the shot areas, but a more effective strategy is to add armor to the areas that appear unscathed.

1 Introduction

Database management research has always been concerned with workloads: the queries one needs to execute determine which design tradeoffs are efficient, and which problems are important to address. Thus, getting the "right" workload is extremely important, as optimizing a system for the "wrong" workload could lead one in a direction that is not relevant to the needs of real users. While data management researchers have been using the synthetic TPC benchmark [19] suite for a long time, recent work from database vendors has shown that these synthetic benchmarks are drastically different from the real workloads they observe [28]. The deluge of "learned" database components [31] has added to the demand for real workloads, as benchmarking a learned technique on synthetic data is not particularly enlightening.

Fortunately, several database vendors have answered the research community's needs, providing detailed statistics about real workloads [1, 7, 13, 28]. These workload statistics have influenced a number of efforts in the research community, from workload synthesis [14] to learned data systems [23, 38]. Combined with existing synthetic benchmarks, these workload descriptions represent a significant step towards aligning academic experimental evaluations with real-world user needs.

In this paper, we discuss a hidden danger in performing research over industrial workloads. Since industrial workloads inherently only capture queries that users execute on that vendor's platform, those workloads are biased towards query workloads that are compatible (both in terms of capability and efficiency) with that vendor's platform. This is a type of *survivorship bias*, where one only gets to measure a filtered subset of a population. The classic example of survivorship bias is described in Figure 1: during World War II, fighter planes engaged in combat were either destroyed by gunfire or returned with damage. The group tasked with determining how to better armor the fighter planes collected data from the returned planes, plotting where gunfire damage appeared. The group determined that their sample was biased, because their sample only contained data from the surviving aircraft. Thus, the group armored the areas *around* the plotted gunfire damage, inferring that planes damaged in those areas did not return [30]. Here, we argue that industrial database workloads have a similar survivorship bias problem, since industrial workloads represent the queries that the user did run, rather than the queries users *wanted* to run.

Put another way, a common misconception among young database researchers is that users write queries for their application, point them at the database, and accept whatever performance they get. In this misconception, it is the DBMS engineer's job to improve performance. Of course, this misconception does not survive contact with real users: in reality, users constantly change their queries, schema, and physical design to match the assumptions of the DBMS.

[*]Authors listed alphabetically.

For example, users might respond to slow join performance by denormalizing a table. If a user cannot get the DBMS to process a query efficiently enough, the user either gets that information in some other way (e.g., by building a bespoke tool) or does without. Thus, what we observe as a user workload is influenced by the DBMS – if the user had picked a different DBMS, the user might have optimized their workload differently. **The queries that we see, and that industry publishes about, are the end result of this "negotiation" between the user and the DBMS. We only see the queries that "survived" the process.**

In this paper, we present two case studies of survivorship bias in workload data. The first case study (Section 2.1) considers the high rate of query repetition seen by several different OLAP data platform vendors (e.g. [1, 13, 28, 29]). While repetition is expected in reporting and dashboarding queries, different vendors report significantly different repetition rates, ranging from 58% to 75% in our data. We show that the query repetition rate is correlated with how well the DBMS is able to handle ad-hoc queries: that is, systems with higher "startup costs" for queries see more repetition, indicating a two-way street between workloads and DBMSes. Researchers (including our group) and engineers using these repetition rates to justify design decisions may be creating self-fulfilling prophecies.

The second case study (Section 2.2) examines the popular "hotspot" optimization technique, in which engineers consider which query patterns to optimize based on a query pattern's frequency and importance in the data. While this data-driven technique is powerful, it rests on a key assumption that optimizing a particular query pattern does not change the query pattern's frequency or importance. We show that at least one counterexample to this assumption exists, and make an argument that more users are likely to use a particular query pattern if that query pattern is optimized. When engineers optimize specific query patterns in a DBMS, they may be implicitly changing the system's user workload as well.

Finally, in Section 3, we summarize key observations from the case studies and discuss their implications for data management research. First, we argue that industry workloads continue to be useful, but that we should understand them in context. Second, we observe that it is difficult to understand what queries your users *wish* they could execute, but could not execute. We argue (with a certain degree of intentional provocativeness) that, at least historically, performance improvements have been the main driver of database capabilities. Third, we flag the inherent difficulty in understanding what the next generation of data research should target, while making several suggestions.

2 Case studies

In this section, we present two case studies of survivorship bias in workloads for real OLAP data platforms. The first case study (Section 2.1) looks at how query repetition rates are related to query startup costs. The second case study (Section 2.2) looks at how optimizations can impact the usage of a query pattern.

2.1 Case study 1: query repetition rates

One of the commonalities between recent analyses of industrial workloads (e.g., from Amazon [28, 33], Meta [1], Microsoft [13], and Snowflake [29]) is the presence of *repetitive queries*. For example, a study on Microsoft SCOPE found a repetition rate of nearly 95% [13]. Repetition in OLAP workloads makes intuitive sense: users building dashboards, running reports, and using BI tools are all reasonable explanations for repetitive workloads. Several subsequent works, both in academia (including from our group [26, 34]) and in industry, have built systems around an assumption of repetition: for example, the Redshift query compilation cache [2] effectively eliminates compilation times for repetitive queries, while ad-hoc queries may suffer from some compilation overhead.

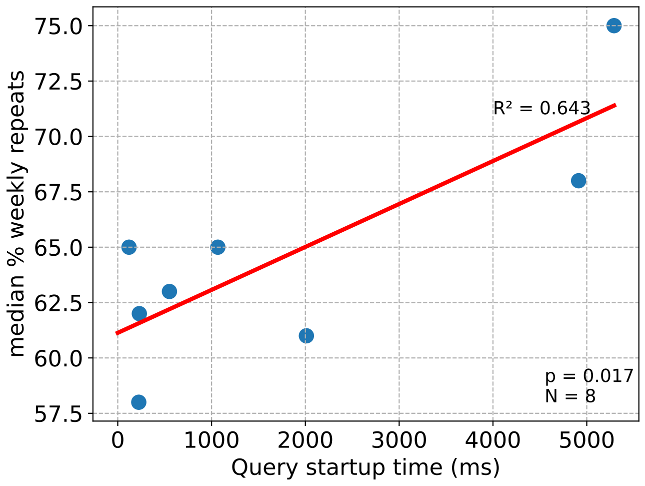

Figure 2: Repeated queries vs. cold cache query startup cost across different DBMSes. Users of DBMSes with high query startup costs have workloads that are more repetitive.

But is repetition a fundamental element of OLAP workloads, or an artifact of existing systems? In other words, is the high repetition we observe an artifact of the queries that survive the user/DBMS negotiation? An alternative explanation of the high repetition rate seen by so many vendors is that, since vendors keep optimizing for repetitive queries, ad-hoc queries are (comparatively) slow, and thus avoided by users. An ideal experiment to test this question would be to take an existing system and significantly slow down ad-hoc or repetitive queries to see if users shift to using the other; perhaps reasonably, no database vendor seemed willing to run this experiment.

Since the ideal experiment is out of reach, we instead settle for a *natural experiment* (i.e., an observational study). If users of a particular data platform have repetitive workloads because ad-hoc queries are slow, then we would expect to see lower query repetition rates on data platforms with a lower penalty for ad-hoc queries. We test this hypothesis by using query compilation time for an ad-hoc (cold cache) query, which we call "startup cost," as a proxy for the "ad-hoc query penalty." Note that we do not measure query latency, only the startup cost – it is entirely possible that a DBMS with a higher startup cost processes a particular query faster end-to-end than a DBMS with a lower startup cost. Intuitively, the query startup cost can be thought of as a lower bound on latency.

Figure 2 plots query startup cost against the weekly query repeat rate for eight different database vendors, which we cannot identify [22]. Data was combined from public sources, private correspondence with vendors, and benchmarks run by our group. Interestingly, **systems with higher query startup costs have more repetitive workloads**, suggesting that users are more likely to execute repetitive workloads on systems that are optimized for them. This correlation does not prove causation; but it is consistent with survivorship bias, because ad-hoc queries that under-perform are simply never issued. Of course, even the data systems with near-zero startup cost still have reasonably high repetition rates, but the difference between the highest (75%) and lowest (58%) repetition rates is partially explained by startup cost. Thus, while it is overall safe to assume that OLAP workloads contain a significant number of repetitive queries, the interaction between the data platform and users determines if repeated queries are a thin majority or a large majority – a potentially impactful difference, as we discuss next.

Discussion. This case study provides evidence for the claim that **workloads are shaped by the performance properties of the DBMS**. This can occur through at least two processes: (1) users could modify their workloads to take advantage of the fastest features provided by the DBMS, or (2) users could actively select a different DBMS based on their workload. Regardless of the process, imagine being an optimizer engineer at the two most extreme points in Figure 2: at the lower-left vendor, an engineer might think that ad-hoc queries are nearly half the workload, and therefore deserve attention, whereas at the upper-right vendor, an engineer might think that repetitive queries are dominant. Since tradeoffs are inevitable in engineering, the engineer at the upper-right vendor might make decisions that improve the performance of repetitive queries at the expense of ad-hoc queries, *which could result in the workload becoming even more repetitive!* The problem is precisely survivorship bias: the engineer's data is based on the bulk of the platform's workload, which represents queries that customers continue to issue (i.e., queries that "survive"). This phenomenon also resembles the Matthew effect [18] ("the rich get richer").

One concrete implication for data researchers is in the area of reinforcement learning powered query optimizers. A learned query optimizer trained on repetition-heavy data will develop a low estimate of aleatoric uncertainty, increasing the value of exploration over exploitation (because any exploration could bring a benefit to many queries). Such a learned optimizer would further worsen the performance of ad-hoc queries, potentially causing users to issue even fewer of them, which in turn further lowers the aleatoric uncertainty, creating a cycle which is vicious or virtuous depending on your perspective.

2.2 Case study 2: if you build it, they will come?

Both industry and academia depend on data from real workloads to decide what kinds of problems to target. If a particular query pattern appears frequently, or uses a disproportionate amount of resources, that pattern might be a good target for optimization. We call this the "hotspot" approach. While this approach seems obviously correct, it can have surprising blind spots. For example, consider a query pattern that is unusably slow; by definition, customers will not use this pattern, and thus vendor data will not show this pattern. But, if that pattern was faster, perhaps users would use it.

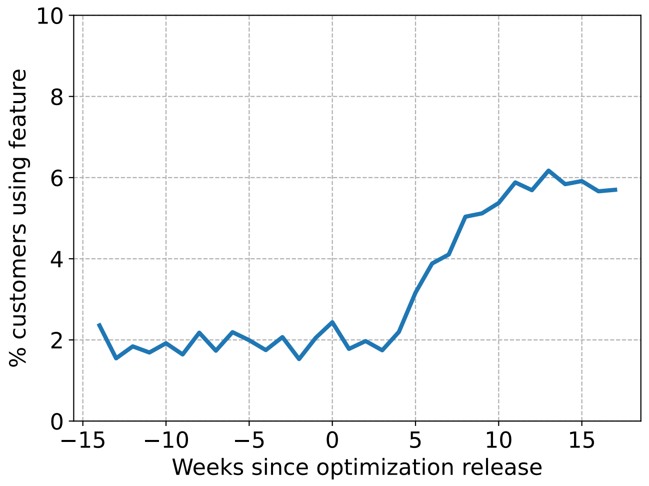

Figure 3: Prior to week zero, only ≈ 2% of users issued rollup-style analytic queries. At week zero, the vendor releases an optimization for rollup queries. After week zero, ≈ 6% of users issued rollup queries.

As in the first case study, this is a difficult hypothesis to test. Ideally, a vendor could (1) make a query pattern intentionally unusable, or (2) could invest significant effort in optimizing a query pattern that does not currently appear in their data. Then, one could observe whether or not usage of the modified feature changed over time. While the first idea is unreasonable, the second idea happens naturally almost every year when data vendors hire summer interns, who are generally assigned self-contained projects with low risk (i.e., if the project fails, the consequences are not severe).

In the 2010s, a summer intern at a data vendor successfully completed a project to accelerate rollup analytic queries (i.e., analytic queries that, at the same time, partition by year, by year/month, and by year/month/day) by a large factor using the technique from Gray et al. [9]. Rollup queries were not common on the vendor's platform (only about 2% of users issued a rollup query), making this an ideal experiment: because pre-optimization rollup queries rarely "survived" into the logs, a hotspot strategy would never choose to optimize them. If user workloads are independent of DBMS performance, then the proportion of users issuing rollup queries will remain constant; if user workloads depend on DBMS performance, then the proportion of users issuing rollup queries will increase.

Figure 3 shows the weekly proportion of users issuing rollup queries before and after the optimization. In the 16 weeks following the release of the optimization, the proportion of users issuing rollup queries tripled, albeit from 2% to 6%. These results show that **a particular query feature became more popular after it was optimized**, possibly indicating that the feature was unusably slow beforehand. The traditional "hotspot" approach would have undervalued this optimization, assuming only 2% of users would benefit. Another potential explanation is that, when users saw the release notes for the data platform, they investigated rollup queries and decided to issue some. It is thus possible that the increase in users is confounded by users discovering rollup queries via the release notes.

Discussion. The traditional technique of identifying "hotspots" or frequent query patterns in a workload and targeting those for optimization is practiced by virtually every data platform vendor known by the authors. While this data-driven approach has excellent precision (i.e., every factor identified is likely worthy of optimization), it has poor recall (i.e., important optimization opportunities may be overlooked). Fundamentally, users want value out of their DBMS, and if the DBMS cannot process a particular query reasonably (either in terms of time, cost, or ease), then that query, *which would be of value to the customer*, will not be "noticed" by the "hotspot" approach. Or, more succinctly: **you cannot measure the query your user never issues.** Again, the problem is survivorship bias. By only looking at "successful" queries, the "hotspot" approach essentially ignores "potential" queries that users might have issued if a particular capability existed.

2.3 Threats to validity

While we have done our best to ensure that the data collected here is accurate and representative, there are several noteworthy considerations:

(1) *Non-uniform data availability*: case study 1 uses data from eight different vendors, but these vendors were not selected randomly. Instead, vendors were selected based on if data was attainable. Thus, there is a risk that these vendors represent a biased sample. This is difficult to mitigate, as many vendors are unwilling to release the required metrics. Case study 2 examines only a single feature of a single data platform, again selected because the data was available to the authors.
(2) *Mixture of data sources*: case study 1's data mixes data from public research, vendor reports, private correspondences, and our own experiments. While we did our best to ensure the results are comparable, some risk remains.
(3) *Possible confounders*: the availability of particular features (such as low query startup cost or rollup analytics) might lead data vendors to pursue different customers, therefore changing how workloads appear. As noted in case study 2, a major data vendor releasing performance improvements to a particular feature may create "buzz" that encourages new users to try that feature, independent of the strength of the optimization. Finally, we note that most database vendors have a financial incentive to represent the performance of their product in a positive light, which may skew available data on query performance.

3 Observations and discussion

Are we doing it wrong? Given the case studies presented in Section 2, we ask: **are we putting armor on the wrong part of the plane?** Or, in context, if the queries that users issue are a poor proxy for the queries that the user cares about, what is the value of analyzing real workloads?

First, while there are issues with assuming that a user's desires are perfectly aligned with their workload, there is certainly still a strong correlation between them. We do not believe that any of these results indicate that we should stop analyzing real workloads,

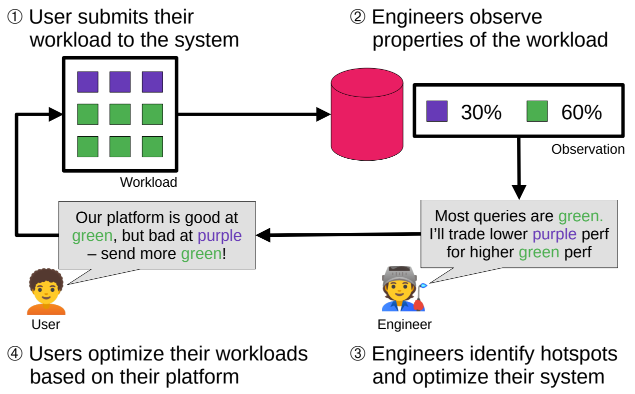
① User submits their workload to the system
② Engineers observe properties of the workload
30%
60%
Observation
Workload
Our platform is good at green, but bad at purple – send more green!
Most queries are green. I'll trade lower purple perf for higher green perf
User
Engineer
④ Users optimize their workloads based on their platform
③ Engineers identify hotspots and optimize their system

Figure 4: The hotspot optimization feedback loop. As engineers optimize for the majority of queries, users adapt their workloads to match the majority and take advantage of those optimizations.

merely that we should *contextualize our understanding of real workloads through the lens of survivorship bias*: when we see a query in a workload, we should view that as the result of an ongoing process that depends on both the user and the DBMS.

Second, we should stop viewing workloads as static, or as independent of the underlying DBMS. Users frequently change their queries and practices to accommodate the DBMS, for example, by avoiding specific query patterns or using specific features. Thus, when the DBMS changes, we should expect workloads to change as well. Figure 4 depicts what we believe is a good mental model of the dynamic nature of workloads. Users submit queries, which engineers profile. Engineers notice hotspots and modify the platform to optimize for those hotspots. Users, seeking good performance, modify their workloads in order to take advantage of those optimizations. This "negotiation" between the user and the data platform engineers is ongoing, so workload traces or statistics can be seen as a snapshot of the state of this process.

The case for performance. Imagine Bob, a hypothetical engineer working at a data platform company prior to the invention of the columnar database.[1] How many dashboarding or reporting queries do you expect Bob would see in his internal metrics? Given the orders-of-magnitude speed improvement columnar databases brought to these workloads, it is likely that Bob would see far fewer such queries in his logs than a modern data platform vendor would see today. Looking at his company's data, Bob might conclude that there is little usage of analytic features. But it would obviously not be reasonable to conclude that users are not interested in dashboarding or reporting queries; these types of queries simply could not be processed efficiently, so they were not used. The invention of the columnar database *enabled* new types of queries to be processed efficiently. Put another way, before columnar databases (for example), certain types of workloads were infeasible to execute on a DBMS, so such workloads would never appear in DBMS logs.

[1]Data platform companies were rare prior to cloud computing, which coincided closely in time with columnar databases, but we ask the reader to suspend their disbelief for the sake of an illustrative example.

Fast forward to today. Are we so different from Bob? **As we look at our workload logs now, how can we possibly predict the query the user *wants* to run, but cannot run?** A wide variety of computations are expressible in SQL but would be executed inefficiently by any production-ready DBMS today (e.g., computing the forward pass of a transformer [25]). For the sake of the reader's entertainment, we will state our interpretation in the most provocative way possible: *performance is the only thing that matters.* Or, perhaps more precisely, previous improvements to DBMS performance, like columnar databases, greatly expanded the scope of data management. The way to enable new workloads and widen the applicability of DBMSes is to make the database the fastest place to do computation. Of course, ease of use is also an important consideration, but DBMSes are generally easier to use than crafting bespoke systems. Other dimensions, like expressivity, should also be studied. But performance is a prerequisite: no amount of ease-of-use or expressivity can make up for an unusably slow query.

Can't we just talk to users? A tempting way out of this conundrum is to simply ask users what they wish their DBMS could do that it cannot do today. This is undoubtedly an important step, as every data platform vendor combines workload metrics with customer testimonials and interviews. Customer needs were a major motivator for the development of materialized views (e.g., [7]), and today are a major motivator for LLM-powered "semantic" operators (e.g., [16, 21]). Unfortunately, DBMS users often simply say that they want "faster horses, not cars:" the same workload, but cheaper and faster. This is something our community will doubtlessly deliver, but, in a scientific sense, accelerating existing workloads can feel like "polishing a shiny ball." We can think of no easy ways out of this issue. It appears to require imagination and courage to pursue something entirely new, that, in the end, users may or may not want.

4 Related work

Survivorship bias has been noted in a wide range of domains, especially in finance research [4], where securities that "survive" are often those that perform well. The most well-known example of survivorship bias is likely the example given in Figure 1 from Wald [30]. In data management research, survivorship bias has been noted in information retrieval datasets [10], where annotators are more likely to annotate "easy" questions compared to hard ones, making benchmarks artificially simpler for automated systems. Most related to the present topic is recent work analyzing survivorship bias in log analysis [27]: for example, when a database is misconfigured, queries that fail early may never make it to the central log, despite representing an important anomaly to detect. Other work has directly addressed correcting for survivorship bias in network logging, where packets dropped early do not make it to logging infrastructure [36]. Survivorship bias has also been examined in the context of webtables, where Wikipedia articles that are older are shown to have higher quality, since those articles "survived" more rounds of edits [3]. At a higher level, the "hardware lottery" [12] in machine learning – where ideas that best take advantage of existing hardware win out over those that do not – is also a form of survivorship bias. Survivorship bias also impacts computer science research itself, both in terms of papers ("surviving" peer review) and people (publishing or perishing) [8]. A related concept in biology is niche construction [20], in which organisms are both impacted by their environment and actively modify their environment over time.

There are several works in statistics about correcting for, or at least measuring, survivorship bias. The most straightforward technique, used in economics, is to "include the dead" [5] by finding non-surviving data in historical records. More advanced techniques, such as simulations based on prior beliefs [17], have been proposed in the health care domain. At first glance, neither technique is immediately applicable to the problems discussed here, as there is no historical data to sift through, and any simulated prior belief would need to be informed by exactly the data that we are missing. Finding ways to correct for survivorship bias in workload traces – or measuring it – is an interesting direction for future work.

The database research community has been recently blessed with several different workload characterizations from industry, including from Amazon [28], Snowflake [29], Meta [1, 7], Microsoft [13, 37], and Google [6]. These workload characterizations have directly led to a wide range of work too long to enumerate here, but some examples include: predicate caching [23], latency prediction [33], end-to-end learned systems [38], cost models [11], string indexing [24], workload synthesis [14, 15, 32], and query optimization [26, 35]. We hope that this work is not read as a criticism of industrial workload characterizations – in fact, this work was only possible precisely because of those characterizations. We strongly believe that vendors releasing workload information will continue to have a large impact on data management research, and we hope this work can provide some guidelines on how to best interpret those data. We emphasize that, while survivorship bias is important to keep in mind, basing database research on industrial workloads is a major "step up" from making purely synthetic or arbitrary assumptions.

5 Conclusions

This work discussed the concept of survivorship bias in the industrial workloads through two case studies. These case studies demonstrated that the capabilities, efficiencies, and inefficiencies of a data platform impact the workloads that users execute on those data platforms. While we still believe that real-world workload traces are critical for data management research, we discussed how making decisions based entirely on these workload traces may lead to incorrect conclusions.

Acknowledgments

This work was supported by the Google ML and Systems Junior Faculty Award and a gift from Merly AI.

References

[1] Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithvi Pandian, Nikolay Leptev, and Ryan Marcus. 2023. AutoSteer: Learned Query Optimization for Any SQL Database. *PVLDB* 14, 1 (Aug. 2023). doi:10.14778/3611540.3611544

[2] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J. Green, Monish Gupta, Sebastian Hillig, Eric Hotinger, Yan Leshinksy, Jintian Liang, Michael McCreedy, Fabian Nagel, Ippokratis Pandis, Panos Parchas, Rahul Pathak, Orestis Polychroniou, Foyzur Rahman, Gaurav Saxena, Gokul Soundararajan, Sriram Subramanian, and Doug Terry. 2022. Amazon Redshift Re-invented. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD*

'22). Association for Computing Machinery, New York, NY, USA, 2205–2217. doi:10.1145/3514221.3526045

[3] Tobias Bleifuß, Leon Bornemann, Felix Naumann, and Divesh Srivastava. 2025. Schema Change Recommendation for User-Curated Webtables Using Temporal Data. *ACM Transactions on the Web* (June 2025), 3742920. doi:10.1145/3742920

[4] Stephen J. Brown, William Goetzmann, Roger G. Ibbotson, and Stephen A. Ross. 1992. Survivorship Bias in Performance Studies. *The Review of Financial Studies* 5, 4 (Oct. 1992), 553–580. doi:10.1093/rfs/5.4.553

[5] Jennifer N. Carpenter and Anthony W. Lynch. 1999. Survivorship bias and attrition effects in measures of performance persistence. *Journal of Financial Economics* 54, 3 (Dec. 1999), 337–374. doi:10.1016/S0304-405X(99)00040-9

[6] Biswapesh Chattopadhyay, Priyam Dutta, Weiran Liu, Ott Tinn, Andrew Mccormick, Aniket Mokashi, Paul Harvey, Hector Gonzalez, David Lomax, Sagar Mittal, Roee Ebenstein, Nikita Mikhaylin, Hung-ching Lee, Xiaoyan Zhao, Tony Xu, Luis Perez, Farhad Shahmohammadi, Tran Bui, Neil McKay, Selcuk Aya, Vera Lychagina, and Brett Elliott. 2019. Procella: unifying serving and analytical data at YouTube. *Proceedings of the VLDB Endowment* 12, 12 (Aug. 2019), 2022–2034. doi:10.14778/3352063.3352121 Publisher: Association for Computing Machinery (ACM).

[7] Sun et al. 2023. Presto: A Decade of SQL Analytics at Meta. *Proceedings of the ACM on Management of Data* 1, 2 (June 2023), 189:1–189:25. doi:10.1145/3589769

[8] Eitan Frachtenberg and Noah Koster. 2020. A survey of accepted authors in computer systems conferences. *PeerJ Computer Science* 6 (Sept. 2020), e299. doi:10.7717/peerj-cs.299 Publisher: PeerJ Inc..

[9] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data Cube: A Relational Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In *Transactions on Data Mining and Knowledge Discovery (DMKD '97, Vol. 1)*. 29–53.

[10] Prashansa Gupta and Sean MacAvaney. 2022. On Survivorship Bias in MS MARCO. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2214–2219. doi:10.1145/3477495.3531832

[11] Roman Heinrich, Manisha Luthra, Johannes Wehrstein, Harald Kornmayer, and Carsten Binnig. 2025. How Good are Learned Cost Models, Really? Insights from Query Optimization Tasks. *Proc. ACM Manag. Data* 3, 3 (June 2025), 172:1–172:27. doi:10.1145/3725309

[12] Sara Hooker. 2020. The Hardware Lottery. doi:10.48550/arXiv.2009.06489 arXiv:2009.06489 [cs].

[13] Hanxian Huang, Tarique Siddiqui, Rana Alotaibi, Carlo Curino, Jyoti Leeka, Alekh Jindal, Jishen Zhao, Jesús Camacho-Rodríguez, and Yuanyuan Tian. 2024. Sibyl: Forecasting Time-Evolving Query Workloads. *Proc. ACM Manag. Data* 2, 1 (March 2024), 53:1–53:27. doi:10.1145/3639308

[14] Skander Krid, Mihail Stoian, and Andreas Kipf. 2025. Redbench: A Benchmark Reflecting Real Workloads. doi:10.1145/3735403.3735998 arXiv:2506.12488 [cs].

[15] Jiale Lao and Immanuel Trummer. 2025. Demonstrating SQLBarber: Leveraging Large Language Models to Generate Customized and Realistic SQL Workloads. In *Companion of the 2025 International Conference on Management of Data (SIGMOD '25)*. Association for Computing Machinery, New York, NY, USA, 151–154. doi:10.1145/3722212.3725101

[16] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing *(CIDR '25)*. Amsterdam, Netherlands. https://www.vldb.org/cidrdb/papers/2025/p12-liu.pdf

[17] Myra B. McGuinness, Jessica Kasza, Amalia Karahalios, Robyn H. Guymer, Robert P. Finger, and Julie A. Simpson. 2019. A comparison of methods to estimate the survivor average causal effect in the presence of missing data: a simulation study. *BMC Medical Research Methodology* 19 (Dec. 2019), 223. doi:10.1186/s12874-019-0874-x

[18] Robert K. Merton. 1988. The Matthew Effect in Science, II: Cumulative Advantage and the Symbolism of Intellectual Property. *Isis* 79, 4 (Dec. 1988), 606–623. doi:10.1086/354848

[19] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS. In *VLDB (VLDB '06)*. VLDB Endowment, Seoul, Korea, 1049–1058. http://dl.acm.org/citation.cfm?id=1182635.1164217

[20] F. John Odling-Smee, Kevin N. Laland, and Marcus W. Feldman. 2003. *Niche Construction: The Neglected Process in Evolution (MPB-37)*. Princeton University Press. https://www.jstor.org/stable/j.ctt24hqpd

[21] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators: A Declarative Model for Rich, AI-based Data Processing. doi:10.48550/arXiv.2407.11418 arXiv:2407.11418 [cs] version: 3.

[22] Anthony G Read. 2006. DeWitt clauses: can we protect purchasers without hurting Microsoft. *Rev. Litig.* 25 (2006), 387.

[23] Tobias Schmidt, Andreas Kipf, Dominik Horn, Gaurav Saxena, and Tim Kraska. 2024. Predicate Caching: Query-Driven Secondary Indexing for Cloud Data Warehouses. In *Companion of the 2024 International Conference on Management of Data (SIGMOD '24)*. Association for Computing Machinery, New York, NY, USA, 347–359. doi:10.1145/3626246.3653395

[24] Mihail Stoian, Johannes Thürauf, Andreas Zimmerer, Alexander van Renen, and Andreas Kipf. 2025. Instance-Optimized String Fingerprints. doi:10.48550/arXiv.2507.10391 arXiv:2507.10391 [cs].

[25] Wenbo Sun, Ziyu Li, and Rihan Hai. 2025. Database as Runtime: Compiling LLMs to SQL for In-database Model Serving. In *Companion of the 2025 International Conference on Management of Data (SIGMOD '25)*. Association for Computing Machinery, New York, NY, USA, 231–234. doi:10.1145/3722212.3725093

[26] Jeffrey Tao, Natalie Maus, Haydn Jones, Yimeng Zeng, Jacob R. Gardner, and Ryan Marcus. 2025. Learned Offline Query Planning via Bayesian Optimization. In *Proceedings of Proceedings of the 2025 International Conference on Management of Data (SIGMOD '25)*. ACM, Berlin, Germany.

[27] Charanraj Thimmisetty, Praveen Tiwari, Didac Gil de la Iglesia, Nandini Ramanan, Marjorie Sayer, Viswesh Ananthakrishnan, and Claudionor Nunes Coelho Jr. 2021. Log2NS: Enhancing Deep Learning Based Analysis of Logs With Formal to Prevent Survivorship Bias. doi:10.48550/arXiv.2105.14149 arXiv:2105.14149 [cs].

[28] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Eknath Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. 2024. Why TPC is not enough: An analysis of the Amazon Redshift fleet. *Proceedings of the VLDB Endowment* (2024). https://www.amazon.science/publications/why-tpc-is-not-enough-an-analysis-of-the-amazon-redshift-fleet

[29] Alexander Van Renen and Viktor Leis. 2023. Cloud Analytics Benchmark. *Proceedings of the VLDB Endowment* 16, 6 (Feb. 2023), 1413–1425. doi:10.14778/3583140.3583156

[30] Abraham Wald. 1980. *A Reprint of "A Method of Estimating Plane Vulnerability Based on Damage of Survivors"*. Technical Report AD A091083. Center for Naval Analyses. 101 pages.

[31] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *SIGMOD Rec.* 45, 2 (Sept. 2016), 17–22. doi:10.1145/3003665.3003669

[32] Johannes Wehrstein, Timo Eckmann, Roman Heinrich, and Carsten Binnig. 2025. JOB-Complex: A Challenging Benchmark for Traditional & Learned Query Optimization. doi:10.48550/arXiv.2507.07471 arXiv:2507.07471 [cs].

[33] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query Execution Time Prediction in Amazon Redshift. In *Proceedings of the 2024 International Conference on Management of Data (SIGMOD '24) (SIGMOD '24)*. Santiago, Chile. doi:10.48550/arXiv.2403.02286

[34] Zixuan Yi, Yao Tian, Zachary G. Ives, and Ryan Marcus. 2024. Low Rank Approximation for Learned Query Optimization. In *International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM @ SIGMOD '24)*. ACM, Santiago, Chile. doi:10.1145/3663742.3663974

[35] Zixuan Yi, Yao Tian, Zachary G. Ives, and Ryan Marcus. 2025. Low Rank Learning for Offline Query Optimization. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 183. doi:10.1145/3725412

[36] Yufei Zheng, Xiaoqi Chen, Mark Braverman, and Jennifer Rexford. 2022. Unbiased Delay Measurement in the Data Plane *(APOCS '22)*. Society for Industrial and Applied Mathematics, Philadelphia, PA. doi:10.1137/1.9781611977059

[37] Wangda Zhang, Matteo Interlandi, Paul Mineiro, Shi Qiao, Nasim Ghazanfari, Karlen Lie, Marc Friedman, Rafah Hosn, Hiren Patel, and Alekh Jindal. 2022. Deploying a Steered Query Optimizer in Production at Microsoft. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)*. ACM, Philadelphia PA USA, 2299–2311. doi:10.1145/3514221.3526052

[38] William Zhang, Wan Shen Lim, Matthew Butrovich, and Andrew Pavlo. 2024. The Holon Approach for Simultaneously Tuning Multiple Components in a Self-Driving Database Management System with Machine Learning via Synthesized Proto-Actions. *Proceedings of the VLDB Endowment* 17, 11 (July 2024), 3373–3387. doi:10.14778/3681954.3682007