# Hollywood: Towards A Large Movie Dataset for Database Benchmarking

Ivan Iachnyk
University of Technology Nuremberg
Nuremberg, Germany
ivan.iachnyk@utn.de

Mihail Stoian
University of Technology Nuremberg
Nuremberg, Germany
mihail.stoian@utn.de

Andreas Kipf
University of Technology Nuremberg
Nuremberg, Germany
andreas.kipf@utn.de

## ABSTRACT

The IMDb real-world dataset of the JOB benchmark has been extensively used in the last decade as part of the research line on cardinality estimation, given its ability to stress test both traditional and learned estimators. However, unlike the synthetic TPC family, it does not come with a scale factor, being a simple dump.

We introduce Hollywood, a synthetic IMDb-compatible benchmark generator that combines LLM-generated semantic dictionaries with deterministic temporal-graph-based relational data generation. We analyze a preliminary Hollywood-200K, which contains 200,000 primary movies, generated series and episode title rows, 19.7M IMDb-style rows, and 213 nonzero JOB-Light, JOB, and JOB-Complex queries. Experiments with two open systems demonstrate that Hollywood induces cardinality estimation errors comparable to or exceeding those observed on the original IMDb dataset. The release includes generation settings and prompt/LLM-output provenance together with adapted SQL and labels, enabling tests of whether cardinality estimators generalize beyond a fixed movie snapshot and distribution.

**VLDB Workshop Reference Format:**
Ivan Iachnyk, Mihail Stoian, and Andreas Kipf. Hollywood: Towards A Large Movie Dataset for Database Benchmarking. VLDB 2026 Workshop: Applied AI for Database Systems and Applications (AIDB 2026).

**VLDB Workshop Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/utndatasystems/hollywood.

## 1 INTRODUCTION

**Motivation.** Query optimizers are commonly evaluated on benchmarks that make different compromises. TPC-H, TPC-DS, and SSB provide controlled scale factors and reproducible data generation [23, 31, 32], following a long line of synthetic database generators [5, 6, 8, 20]. By contrast, the JOB benchmark family [15, 17, 33] uses a 2013 snapshot of IMDb, exposing optimizers to string-heavy predicates and correlated multi-join queries [17]. Despite its small size, it has been *the* dataset for learned-cardinality, learned-cost, and learned-optimizer work [10, 11, 15, 21, 22].


Proceedings of the VLDB Endowment. ISSN 2150-8097.

**The Gap.** The cost of JOB's realism is that the data instance is fixed, making it difficult to ask whether an estimator generalizes to a structurally similar but previously unseen data distribution. Recent work revisits JOB [18], extends JOB-style query families [33] or synthesizes trace-shaped cloud workloads over existing datasets, including the IMDb dump [16, 30, 34]. Relational-data generation also has complementary lines based on query-aware or constraint-driven regeneration [1, 3, 26–29] and graph-conditional diffusion over relational databases [13]. Yet the IMDb/JOB ecosystem still lacks a generator-style path that retains JOB's correlated predicates.

**Hollywood.** In this work, we address this missing path. Given a generation profile specifying primary-movie count, years, entity-pool sizes, and a seed, Hollywood produces a fresh IMDb-style database with jointly generated movie, person, company, and literal values. The LLM component runs through Google's Gemini API [7]; in the current prompting profile, movie-plot generation costs roughly US$100 per 100,000 primary movies, excluding optional series/episode plot text. For Hollywood-200K, we additionally adapt JOB-family templates to existing literals, execute them to obtain labels, and release the files needed to reproduce the learned-estimator measurements on the released database.

We focus on Hollywood-200K, a 200,000-primary-movie export with 351,455 IMDb title rows, and evaluate adapted JOB-Light, JOB, and JOB-Complex query workloads on it. We compare it with two IMDb references: the original benchmark workload on the full IMDb dataset and a matched sample with the same title-row count.

**Contributions.** Thus, our contributions are:

(1) an IMDb-compatible generator combining LLM semantic priors, deterministic temporal-graph materialization, and strict relational export,
(2) an evaluation of PostgreSQL, DuckDB, MSCN, and ZeroShot on Hollywood, with full-query, single-table selection, and selected-plan cost references against IMDb references,
(3) and a release package containing the generator, the Hollywood-200K export, the nonzero adapted JOB-Light, JOB, and JOB-Complex workloads used here, and the evaluation artifacts needed to reproduce the reported measurements.

**Outlook.** Building on the framework presented here, we plan to release Hollywood at scale factors 10 and 100, treating IMDb as sf=1, and replay all existing evaluations of learned cardinality estimators beyond the widely used, yet relatively small, IMDb dataset. We next outline Hollywood's data generation process.

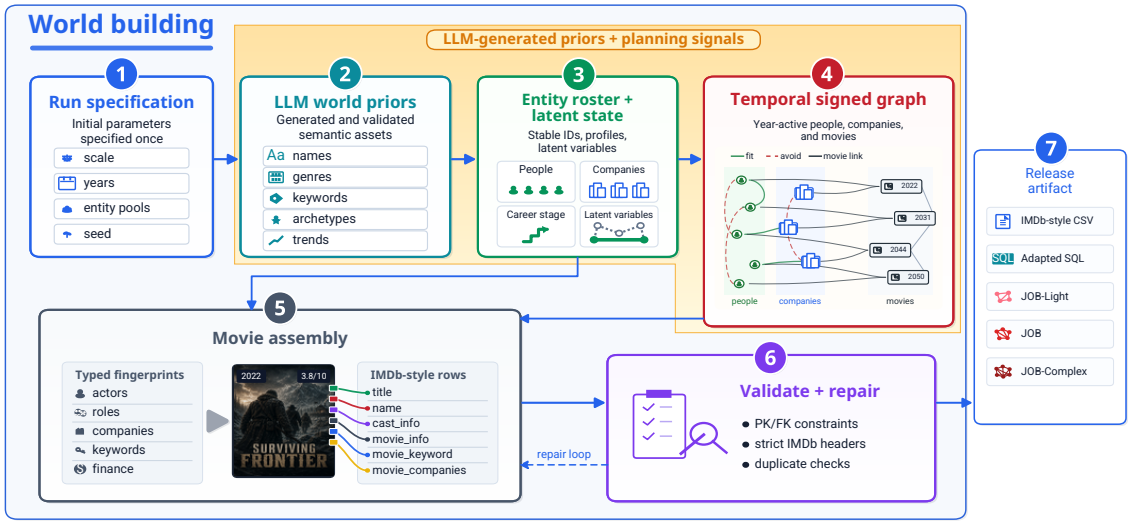


**Figure 1: Hollywood generation flow from LLM-authored priors to temporal graph state, movie assembly, validation/repair, and the released IMDb-style CSV plus adapted SQL artifact.**

## 2 GENERATION METHOD

**Construction Flow.** Fig. 1 sketches the construction path. A run specification fixes scale, years, entity pools, and seed, after which LLM priors, entity rosters, a temporal signed graph, and movie assembly produce a validated IMDb-style export. The evaluated artifact packages the export with adapted SQL workloads, labels, plans, runtime records, and validation and repair reports, making the benchmark instance inspectable and replayable.

**Export Scale.** Our current IMDb/JOB export contains 200,000 primary movies plus generated TV-series and episode title rows, yielding 351,455 title rows, 480,000 name rows, 4.98M cast rows, 2.10M movie-keyword rows, 5.39M movie-info rows, and 19.7M rows overall. The matched IMDb sample has the same title-row count and a similar largest JOB-facing relationship table: 5.25M `cast_info` rows versus Hollywood's 4.98M. It is therefore a title/cast-scale reference, not a distributional twin.

**Rows, SQL, and Labels.** The generator holds typed profiles for people and movies. For a movie $m$ generated in year $y$, its profile $z_m$ and the temporal graph snapshot $G_y$ active in that year jointly condition the selection of cast slots, companies, keywords, and metadata. The exporter then emits coordinated rows in `name`, `title`, `cast_info`, `movie_info`, `movie_companies`, and `movie_keyword`. Thus predicates over names, genres, companies, keywords, and cast roles originate from one generated movie rather than independent per-table draws. The release also includes the final adapted SQL workloads. Adaptation performs a bounded search over replacements for hard-coded literals using values found in the Hollywood-200K database, while preserving aliases, joins, operators, grouping, and count wrappers. It retains candidates that parse, execute, and have positive labels, favoring those that stress both PostgreSQL and DuckDB; the database remains fixed throughout this search.

### 2.1 LLM Priors

**Semantic Priors.** Hollywood uses LLMs as semantic prior generators, not tuple emitters. In ② of Fig. 1, LLM calls use Gemini 3.1 Flash-Lite Preview [7]; the exact API model identifier is recorded in artifact provenance. Asking an LLM to write millions of rows would be expensive, hard to audit, and difficult to repair, especially because long-context studies show that larger windows do not guarantee reliable use of every relevant fact [12, 19]. Instead, bounded prompts produce reusable lexical assets, entity priors, and temporal priors that are parsed as JSON, normalized, deduplicated, checked for coverage, versioned, and consumed by seeded code.

**World-Building Bias.** LLM-generated latents in ③ provide one input to later deterministic selectors. A purely random latent table would not automatically couple names, regions, genres, career stages, company strategies, and risk profiles. We therefore use the model as a nonuniform world-building prior: its co-occurrence patterns couple otherwise independent labels, but training-data biases can also enter the generated priors [2, 4]. The artifact stores these versioned priors separately from tuple materialization. Seeded selectors combine them with graph state and validation rules when materializing rows.

### 2.2 Temporal Graph

**Graph State.** The temporal graph in ④ stores cross-entity dependencies that are reused when movies are assembled. Its nodes represent people and companies; movies are assembled using the graph but are not themselves graph nodes. Each edge has a type, sign, strength, provenance record, and active-year interval. Positive edges encode collaboration, mentorship, genre fit, and company–talent affinity, while negative edges encode rivalry, avoid lists, brand mismatch, or market competition. Multiple selectors reuse these signed edges during movie assembly, so a casting choice can also

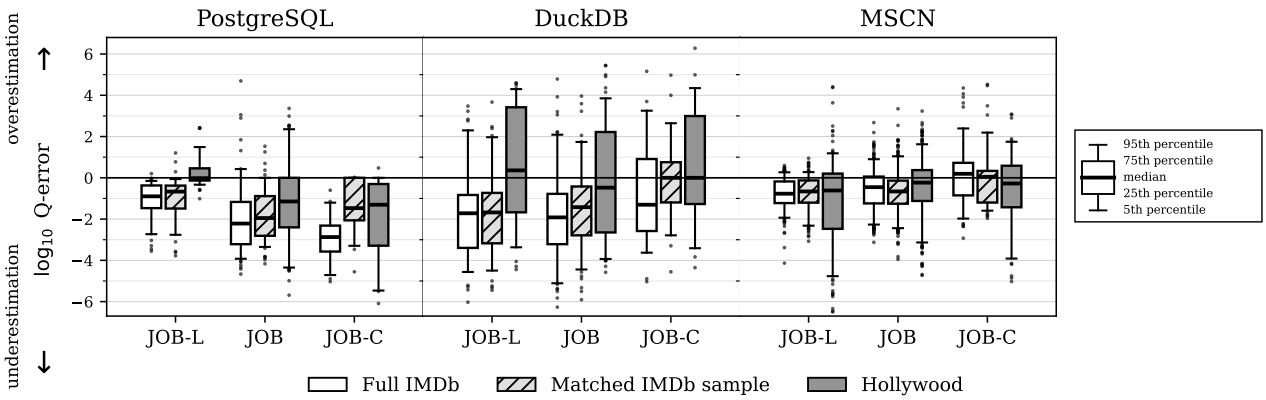


**Figure 2: Signed full-query cardinality error. Boxes compare full IMDb, the matched IMDb sample, and Hollywood; MSCN uses separately trained models for each dataset.**

affect company links, keywords, and metadata rather than leaving each relation group to be sampled separately.

**Graph Sampling.** Initial edges are stochastic but feature-driven. Candidate pairs come from genre, market, community, agency, career-stage, and company pools, then weighted draws combine creative-style similarity, genre overlap, risk compatibility, prior workload, degree caps, and seeded noise. Company-person and company-company edges use analogous genre, budget, market, risk, and strategy features. The generator consumes the graph through an active-at-year view: assembly asks for friendship, rivalry, director-preference, avoid, and company-affinity edges active in the target year. Movie assembly and year-boundary evolution can add, update, or expire edges before later years are generated. The history borrows slowly changing dimension type-2 semantics [14]: updates close the old active interval and insert a replacement edge version, while expirations only close the interval, preserving which edge version was active in each synthetic year.

## 2.3 Chronological Assembly

**Yearly Assembly.** Hollywood first samples a yearly slate of movie slots. It then materializes each movie in chronological order by selecting concept, title, people, companies, keywords, and metadata. For a movie profile $z_m$ in year $y$, let $x$ be a candidate person, company, or keyword and let $G_y$ be the graph snapshot active in $y$. Its selection weight is schematically

$$\Pr(x \mid z_m, G_y, y) \propto w_{\text{pop}}(x, y) \cdot w_{\text{fit}}(x, z_m) \cdot w_{\text{cap}}(x, y) \cdot w_{\text{latent}}(x, z_m) \cdot w_{\text{graph}}(x, G_y) \cdot w_{\text{policy}}(x, y).$$

The factors represent current popularity or activity, movie-role compatibility, remaining yearly capacity, latent-profile similarity, affinity in the active graph, and policy constraints such as diversity or degree caps. Each selector uses the applicable factors and caps for its candidate type. After each year, coappearances, company activity, genre movement, and performance signals are summarized into graph operations, latent deltas, stage/tier updates, retirements, dissolutions, and genre shifts. These changes condition the next year, making the graph dynamic and probabilistic while preserving chronological consistency.

# 3 EVALUATION

The evaluation separates base-selection difficulty from full-query cardinality and selected-plan runtime prediction. The comparison scope depends on the task: full-query cardinality compares Hollywood with both full IMDb and a title-count-matched IMDb sample, single-table selections isolate local predicate difficulty on the matched setting, and selected-plan runtime prediction compares Hollywood with the matched IMDb sample on PostgreSQL plans.

## 3.1 Setup

**Workloads and References.** All measurements use the frozen Hollywood-200K database and the released adapted SQL and label files. The released instance includes plot rows for movies but not for TV series or episodes. The Hollywood workload contains 70 JOB-Light, 113 JOB, and 30 JOB-Complex queries selected from adapted candidates with positive labels [17, 33]. As references, full IMDb uses the canonical benchmark workload, while the matched IMDb sample uses adapted literals at the same title-row scale.[1] Single-table selection labels are exact filtered base-relation counts obtained with the canonical PostgreSQL IMDb schema.

**Estimators.** We use PostgreSQL 16.14 [24] and DuckDB 1.5 [25] as traditional cardinality estimators. For learned models, MSCN [15] is evaluated for cardinality and selected-plan cost, while ZeroShot [10] is evaluated for selected-plan cost; PostgreSQL cost provides a conventional selected-plan runtime proxy. PostgreSQL and DuckDB run on explicit IMDb-compatible schemas; full-query cardinalities use PostgreSQL planner rows and the first DuckDB cardinality marker below count-style aggregates. The MSCN cardinality model follows SetConv with table, predicate, join, and sample-bitmap inputs, plus our deterministic string-hash encoding for string literals. For MSCN, cardinality models train separately per plotted dataset (90K full-query examples, 100 epochs, repeated seeds); selected-plan cost models do the same for Fig. 3 datasets with 10K plan examples.

**Runtime Prediction.** For each query, PostgreSQL selects a physical plan whose measured execution time is the prediction target. We compare PostgreSQL cost, ZeroShot, and an MSCN cost model on

[1] Note that 28 of the 30 literal-adapted JOB-Complex queries on the matched IMDb sample have nonzero cardinality; all full-query summaries use this fixed subset.

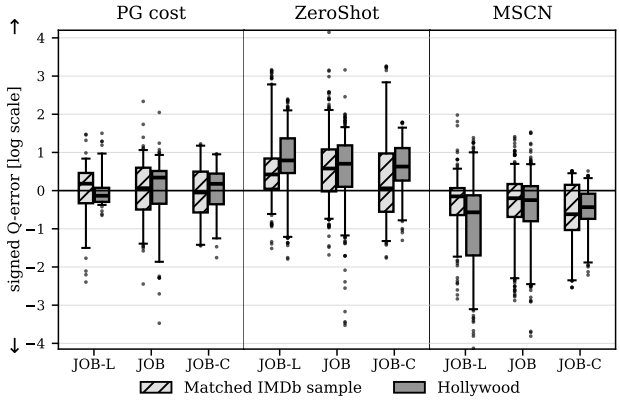


**Figure 3: Signed selected-plan runtime-prediction error comparing matched IMDb and Hollywood. PG cost denotes the PostgreSQL optimizer cost. Positive values indicate overprediction and negative values indicate underprediction.**

that plan, with intra-query parallelism disabled because gather and parallel-aware operators are outside the learned-plan vocabulary. We map PostgreSQL cost to milliseconds with a workload-specific geometric-mean scale. ZeroShot uses the pretrained cross-database cost model of Hilprecht et al. [10] under the evaluation setup of Heinrich et al. [9].

## 3.2 Full-Query Cardinality

**PostgreSQL.** For full-query cardinality, Fig. 2 compares Hollywood with two IMDb references: the canonical JOB-family workload on full IMDb and a literal-rebound workload on a same-title-count IMDb sample. PostgreSQL has smaller Hollywood medians on JOB and JOB-Complex than full IMDb (28.0 vs. 261, and 20.5 vs. 758), but the tail is larger: p95 Q-error reaches $2.37 \times 10^4$ on JOB and $3.07 \times 10^5$ on JOB-Complex, above the full-IMDb p95 values of $1.15 \times 10^4$ and $6.64 \times 10^4$. The matched IMDb sample shows the same contrast, with lower Hollywood medians but p95 values about 9.9 times larger on JOB and 118 times larger on JOB-Complex. In the signed boxes, these large PostgreSQL errors mostly lie below zero; together with the much smaller base-selection errors in Table 1, this pattern is consistent with cardinality underestimation emerging after relationship joins.

**DuckDB.** DuckDB gives the contrasting case. Hollywood shifts its signed boxes upward compared with the IMDb references: on JOB-Complex, median Q-error is 190, compared with 117 on full IMDb and 11.7 on the matched IMDb sample; on JOB-Light, median signed log error moves from about $-1.7$ on both IMDb references to 0.36 on Hollywood. The same export therefore produces predominantly negative post-join errors for PostgreSQL and frequent overestimation for DuckDB.

**Learned Estimator.** For JOB-Light, JOB, and JOB-Complex, MSCN medians remain low: full IMDb (5.89, 4.10, 6.62), matched IMDb (5.02, 5.81, 5.50), and Hollywood (7.47, 4.23, 5.32). Tails differ more: Hollywood p95s ($7.12 \times 10^4$, 1,480, $1.14 \times 10^4$) exceed full IMDb (90.1, 194, 1,874) and matched IMDb (210, 298, 155), so typical learned errors are small but Hollywood contributes large tail cases.

## 3.3 Base Predicates

Tab. 1 separates local predicate difficulty from join difficulty: for PostgreSQL, Hollywood single-table selection p95 is only 2.30, 22.0, and 48.0 on JOB-Light, JOB, and JOB-Complex, while full-query p95 reaches $2.37 \times 10^4$ on JOB and $3.07 \times 10^5$ on JOB-Complex. These results are consistent with the largest PostgreSQL errors emerging after relationship joins rather than in individual filtered base-relation estimates, while DuckDB's largest single-table deviations are a few JOB-Light `movie_info_idx` rating outliers.

**Table 1: Single-table selection Q-error. Cells report median / p95 for filtered base-relation predicates; Hollywood columns are shaded.**

| Workload | Estimator | Matched IMDb sample | Hollywood |
|---|---|---|---|
| JOB-L | PG | 1.01 / 1.85 | 1.01 / 2.30 |
| | DuckDB | 2.28 / 91.6 | $4.86 / 3.52 \times 10^4$ |
| JOB | PG | 1.00 / 14.2 | 1.00 / 22.0 |
| | DuckDB | 2.40 / 4,830 | 2.23 / 3,600 |
| JOB-C | PG | 1.00 / 9.98 | 1.00 / 48.0 |
| | DuckDB | 2.01 / 2,328 | 2.28 / 1,248 |

## 3.4 Selected-Plan Runtime Prediction

**Selected PostgreSQL Plans.** Following learned-cost-model evaluations on JOB-derived tasks [9], Fig. 3 evaluates runtime prediction on fixed selected PostgreSQL plans. With the workload-specific scale, PostgreSQL cost has similar p95 error on Hollywood and matched IMDb JOB-Complex (24.2 vs. 26.8), lower p95 error on Hollywood JOB-Light (12.7 vs. 46.7), and higher p95 error on Hollywood JOB (137 vs. 28.5).

**Learned Runtime Models.** Relative to the matched IMDb, ZeroShot has higher Hollywood medians on JOB-Light and JOB, but lower p95 on JOB and JOB-Complex (66.9 vs. 130, and 51.4 vs. 1,080). The signed boxes show that ZeroShot usually overpredicts Hollywood runtimes, although Hollywood JOB contains a severe underprediction. MSCN cost is mixed, with a larger Hollywood JOB-Light tail (1,287 vs. 68.8) but a smaller Hollywood JOB-Complex tail (83.2 vs. 228).

# 4 CONCLUSION & FUTURE WORK

With Hollywood, we combine LLM semantic priors with deterministic temporal graph generation to produce IMDb/JOB-style benchmark instances with strict relational exports, adapted SQL, and executable labels. The generator exposes entity counts, company and keyword pools, role mixes, and yearly slate sizes as parameters for future scale factors and controlled workload variants.

In future work, we plan to introduce TPC-style scale factors and replay all existing learned cardinality estimators across these scaled IMDb-compatible datasets. We also aim to extend Hollywood's generation process to other schemas, such as those of TPC-H and TPC-DS, whose data generators rely on predefined statistical distributions rather than semantic generation.

# REFERENCES


[1] Arvind Arasu, Raghav Kaushik, and Jian Li. 2011. Data Generation Using Declarative Constraints. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 685–696. https://doi.org/10.1145/1989323.1989395

[2] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. Association for Computing Machinery, New York, NY, USA, 610–623. https://doi.org/10.1145/3442188.3445922

[3] Carsten Binnig, Donald Kossmann, Eric Lo, and M. Tamer Özsu. 2007. QAGen: Generating Query-Aware Test Databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 341–352. https://doi.org/10.1145/1247480.1247520

[4] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the Opportunities and Risks of Foundation Models. https://doi.org/10.48550/arXiv.2108.07258 arXiv:2108.07258 [cs.LG]

[5] Nicolas Bruno and Surajit Chaudhuri. 2005. Flexible Database Generators. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi (Eds.). ACM, 1097–1107. http://www.vldb.org/archives/website/2005/program/paper/wed/p1097-bruno.pdf

[6] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek R. Narasayya. 2021. DSB: A Decision Support Benchmark for Workload-Driven and Traditional Database Systems. *Proceedings of the VLDB Endowment* 14, 13 (2021), 3376–3388. https://doi.org/10.14778/3484224.3484234

[7] Google DeepMind. 2026. Gemini 3.1 Flash-Lite Model Evaluation: Approach, Methodology & Results. https://storage.googleapis.com/deepmind-media/gemini/gemini_3-1_flash-lite_model_evaluation.pdf Published March 2026; accessed 2026-05-31.

[8] Jim Gray, Prakash Sundaresan, Susanne Englert, Ken Baclawski, and Peter J. Weinberger. 1994. Quickly Generating Billion-Record Synthetic Databases. *ACM SIGMOD Record* 23, 2 (1994), 243–252. https://doi.org/10.1145/191843.191886

[9] Roman Heinrich, Manisha Luthra, Johannes Wehrstein, Harald Kornmayer, and Carsten Binnig. 2025. How Good are Learned Cost Models, Really? Insights from Query Optimization Tasks. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–27. https://doi.org/10.1145/3725309

[10] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2361–2374. https://doi.org/10.14778/3551793.3551799

[11] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *Proceedings of the VLDB Endowment* 13, 7 (2020), 992–1005. https://doi.org/10.14778/3384345.3384349

[12] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? https://doi.org/10.48550/arXiv.2404.06654 arXiv:2404.06654 [cs.CL]

[13] Mohamed Amine Ketata, David Lüdke, Leo Schwinn, and Stephan Günnemann. 2025. Joint Relational Database Generation via Graph-Conditional Diffusion Models. https://openreview.net/forum?id=Z3OtNSwuXX OpenReview.

[14] Ralph Kimball and Margy Ross. 2013. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling* (3rd ed.). Wiley, Hoboken, NJ, USA. https://www.oreilly.com/library/view/the-data-warehouse/9781118530801/

[15] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR 2019)*. CIDR, Asilomar, CA, USA, Article 101, 8 pages. https://www.cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf

[16] Skander Krid, Mihail Stoian, and Andreas Kipf. 2025. Redbench: A Benchmark Reflecting Real Workloads. *arXiv preprint arXiv:2506.12488* (2025).

[17] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215. https://doi.org/10.14778/2850583.2850594

[18] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2025. Still Asking: How Good Are Query Optimizers, Really? *Proceedings of the VLDB Endowment* 18, 12 (2025), 5531–5536. https://doi.org/10.14778/3750601.3760521

[19] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. https://doi.org/10.1162/tacl_a_00638

[20] Eric Lo, Nick Cheng, Wilfred W. K. Lin, Wing-Kai Hon, and Byron Choi. 2014. MyBenchmark: Generating Databases for Query Workloads. *The VLDB Journal* 23, 6 (2014), 895–913. https://doi.org/10.1007/s00778-014-0354-1

[21] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 1275–1288. https://doi.org/10.1145/3448016.3452838

[22] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1705–1718. https://doi.org/10.14778/3342263.3342644

[23] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking*. Springer Berlin Heidelberg, Berlin, Heidelberg, 237–252. https://doi.org/10.1007/978-3-642-10424-4_17

[24] PostgreSQL Global Development Group. 2026. PostgreSQL 16 Documentation: Using EXPLAIN. https://www.postgresql.org/docs/16/using-explain.html Accessed 2026-05-31.

[25] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 1981–1984. https://doi.org/10.1145/3299869.3320212

[26] Anupam Sanghi, Shadab Ahmed, and Jayant R. Haritsa. 2022. Projection-Compliant Database Generation. *Proceedings of the VLDB Endowment* 15, 5 (2022), 998–1010. https://doi.org/10.14778/3510397.3510398

[27] Anupam Sanghi, Shadab Ahmed, Prashik Rawale, and Jayant R. Haritsa. 2022. *Data Generation Using Join Constraints*. Technical Report TR-2022-01. Database Systems Lab, Indian Institute of Science. https://dsl.cds.iisc.ac.in/publications/report/TR/TR-2022-01.pdf

[28] Anupam Sanghi, Rajkumar Santhanam, and Jayant R. Haritsa. 2021. Towards Generating HiFi Databases. In *Database Systems for Advanced Applications*. Springer, Cham, Switzerland, 105–112. https://doi.org/10.1007/978-3-030-73194-6_8

[29] Anupam Sanghi, Raghav Sood, Jayant R. Haritsa, and Srikanta Tirthapura. 2018. Scalable and Dynamic Regeneration of Big Data Volumes. In *Proceedings of the 21st International Conference on Extending Database Technology*. OpenProceedings.org, Vienna, Austria, 301–312. https://doi.org/10.5441/002/edbt.2018.27

[30] Tobias Schmidt, Viktor Leis, Peter Boncz, and Thomas Neumann. 2025. SQLStorm: Taking Database Benchmarking into the LLM Era. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4144–4157. https://doi.org/10.14778/3749646.3749683

[31] Transaction Processing Performance Council. 2022. TPC Benchmark H (Decision Support) Standard Specification, Revision 3.0.1. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf Accessed 2026-05-25.

[32] Transaction Processing Performance Council. 2024. TPC Benchmark DS – Standard Specification, Version 4.0.0. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v4.0.0.pdf Accessed 2026-05-25.

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

[34] Johannes Wehrstein, Roman Heinrich, Mihail Stoian, Skander Krid, Martin Stemmer, Andreas Kipf, Carsten Binnig, and Muhammad El-Hindi. 2025. Redbench: Workload Synthesis From Cloud Traces. https://doi.org/10.48550/arXiv.2511.13059 arXiv:2511.13059 [cs.DB] Accepted to VLDB 2026 Experiment, Analysis, and Benchmark track.