RESEARCH-ARTICLE

The LDBC Financial Benchmark: Transaction Workload

SHIPENG QI, Ant group, Hangzhou, Zhejiang, China

BING TONG

JIATAO HU, Tsinghua University, Beijing, China

HENG LIN, Ant group, Hangzhou, Zhejiang, China

YUE PANG, Peking University, Beijing, China

WEI YUAN

View all

Published: 01 May 2025

The LDBC Financial Benchmark: Transaction Workload

Shipeng Qi[1,9], Bing Tong[2,3], Jiatao Hu[4], Heng Lin[1], Yue Pang[5], Wei Yuan[6], Songlin Lyu[1], Zhihui Guo[1], Ke Huang[1], Xujin Ba[7], Qiang Yin[7], Youren Shen[8], Yan Zhou[2], Tao Lv[6], Jia Li[3], Lei Zou[5], Yongwei Wu[4], Gábor Szárnyas[9], Xiaowei Zhu[1], Wenguang Chen[4], Chuntao Hong[1]
[1]Ant Group [2]CreateLink Technology [3]HKUST(GZ) [4]Tsinghua University [5]Peking University
[6]China Software Testing Center [7]Shanghai Jiao Tong University [8]Beijing Haizhixingtu Technology [9]LDBC
[1]{qishipeng.qsp, linheng.lin, lyusonglin.lsl, guozhihui.gzh, hk286513, robert.zxw, chuntao.hct}@antgroup.com [2]{tongbing, zhouyan}@createlink.com [3]jialee@ust.hk [4]hjt22@mails.tsinghua.edu.cn,
{wuyw, cwg}@tsinghua.edu.cn [5]{michelle.py, zoulei}@pku.edu.cn [6]{yuanwei, lvtao}@cstc.org.cn
[7]{baxujin2023, q.yin}@sjtu.edu.cn [8]shenyouren@stargraph.cn [9]gabor.szarnyas@ldbcouncil.org

ABSTRACT

Graph databases play a pivotal role in the FinTech industry. However, existing graph benchmarks fail to capture the unique characteristics of financial datasets and workloads, rendering them inadequate for evaluating graph databases in financial scenarios. This paper presents the LDBC Financial Benchmark (FinBench) Transaction Workload, a novel benchmark that adopts a choke point-driven design methodology, emphasizing performance bottlenecks, and incorporates distinct features such as dataset skewness, edge multiplicity, temporal window filtering, recursive path filtering, read-write query patterns, and truncation on hub vertices. Key contributions include a scalable data generator that synthesizes datasets with financial-specific features, a parameter generator that leverages bucketed data statistics for runtime consistency across queries, and a scalable benchmark driver that biases query execution by time windows. Experimental evaluations on graph databases demonstrate the benchmark's capability to reveal novel choke points and provide insights into system performance in financial scenarios.

PVLDB Reference Format:
Shipeng Qi, Bing Tong, Jiatao Hu, Heng Lin, Yue Pang, Wei Yuan, Songlin Lyu, Zhihui Guo, Ke Huang, Xujin Ba, Qiang Yin, Youren Shen, Yan Zhou, Tao Lv, Jia Li, Lei Zou, Yongwei Wu, Gábor Szárnyas, Xiaowei Zhu, Wenguang Chen, Chuntao Hong. The LDBC Financial Benchmark: Transaction Workload. PVLDB, 18(9): 3007 - 3020, 2025.
doi:10.14778/3746405.3746424

PVLDB Artifact Availability:
The source code, data, and other artifacts have been made available at https://github.com/ldbc/ldbc_finbench_transaction_impls.

1 INTRODUCTION

With the rapid growth of graph scenarios, many graph technologies have emerged, drawing more and more attention from both academia and industry [49]. Graph technologies have become critical for modern enterprises across nearly every sector [19], including e-commerce, social networks, and financial applications. In the financial industry, graph technologies enhance various applications such as fraud detection, anti-money laundering (AML), and know-your-customer (KYC). Top banks, rating agencies, and financial companies worldwide adopt graph technology to offer financial services [37], such as J.P. Morgan [50], PayPal [48], and Alipay [23]. Explicitly, the labeled property graph model is a natural way to reflect activities in financial systems, with vertices representing financial entities such as persons, companies, accounts, and assets, while edges representing the activities among them. Graph data allows queries to effectively retrieve system activities by matching patterns and paths, typically consisting of cycles and chains, which may involve multiple hops across the graph to uncover hidden connections. The global payments industry handled over 3 trillion transactions in 2023 [35], posing challenges in storing and processing financial data in graph databases. These challenges highlight the importance of robust performance evaluation to ensure the reliability and efficiency of graph database solutions.

Limitation of Existing Benchmarks. Benchmarks are the best way to make different database systems comparable. A good benchmark should be relevant, portable, scalable, reproducible, and fair [20, 25]. The growth in the prevalence and scale of graphs in the financial industry necessitates benchmarks tailored to data and workload characteristics and industry performance requirements. Most existing graph database benchmarks are primarily designed with social network scenarios, focusing on the dataset and query patterns prevalent in social graphs. This overlooks the unique characteristics and requirements of financial data and workloads, limiting their applicability and effectiveness in evaluating graph databases for financial scenarios. We found that in the financial dataset (see Section 2), there are multiple transfer records between the account pairs, resulting in multiple transfer edges existing between the same source and destination account vertex, which requires special design in the storage engine and aggregation operations in the query engine. We also found that in the financial workload (see Section 3), queries are always combined with time-window filtering, as well as queries matching subgraph patterns and recursive path filtering, posing unique challenges to the graph databases. Therefore, we propose the Financial Benchmark (FinBench) to fairly evaluate graph databases targeting financial scenarios through Linked Data Benchmark Council [3, 45], which is dedicated to providing standardized benchmarks for measuring graph processing performance.

Proceedings of the VLDB Endowment, Vol. 18, No. 9 ISSN 2150-8097.
doi:10.14778/3746405.3746424

Table 1: Comparison of SNB Interactive workload, SNB Business Intelligence workload, FinBench Transaction workload, and Graphalytics. Notation: ⊗: yes, ○: no

LDBC benchmarks	FinBench Transaction	SNB Interactive v1*	LinkBench	SNB BI	Graphalytics
labelled property graph	⊗	⊗	⊗	⊗	○
temporal property graph	⊗	⊗	○	⊗	○
edge multiplicity	⊗	○	○	○	○
insert operations	⊗	⊗	⊗	⊗	○
delete operations	⊗	○	⊗	⊗	○
query footprint	small	small	small	large	all data
inter-query parallelism	required	required	required	optional	not applicable
path finding	⊗	○	○	⊗	⊗
time-window queries	⊗	○	○	○	○
recursive path filtering	⊗	○	○	○	○
truncated traversal	⊗	○	○	○	○
read-write queries	⊗	○	○	○	○
workload type	OLTP	OLTP	OLTP	OLAP	graph algorithms
query mix	⊗	⊗	⊗	⊗	not applicable
time-biased query mix	⊗	○	○	○	not applicable

* Note: SNB Interactive v2 has not been officially released yet, so we compare it with v1.

Comparison of Benchmarks. There are already some benchmarks for graph databases and systems, including SNB Interactive [11], LinkBench [4], SNB BI [44], and Graphalytics [27]. In Table 1, we compare the similarities and differences of these benchmarks in identified characteristics in financial scenarios, respectively in the dataset, the choke points embedded in the queries, and the workload. SNB and LinkBench, inspired by social network scenarios, do not emphasize temporal features within datasets and workloads or the execution of path finding queries. In contrast, FinBench is distinguished by focusing on path finding queries, read-write operations, temporal features such as temporal graphs in data schema, time-based filters, and time-based query mix. More comparisons with other benchmarks can be found in Section 6.

Figure 1 provides an overview of FinBench's software components and the workflow for generating necessary data artifacts inherited from the LDBC benchmark framework [2]. The scalable data generator is a core component that ensures the generation of large-scale datasets while maintaining high fidelity to real-world financial data. It creates three data products: (1) the dataset for bulk-loading, which serves as the initial state of the system under test; (2) the update dataset, which simulates dynamic changes during benchmarking by providing a stream of write operations; and (3) the factor tables, which are specialized data structures designed to capture and summarize key statistical properties of the generated dataset. These factor tables play a crucial role in ensuring that query parameters are derived from realistic distributions, thereby enhancing the benchmark's realism and relevance. The parameter generator is designed to ensure stable runtime across different query variants [44]. It generates the query parameters according to the query cost estimation based on statistical information stored in the factor tables to avoids scenarios where queries become either trivially simple or excessively complex. These parameters are then passed to the benchmark driver for read queries generation.

The benchmark design is derived from a comprehensive analysis of over 20 business clusters spanning diverse financial domains, including fund transaction graphs, anti-fraud graphs, insurance claim graphs, device relationship networks, and composite analytical graphs. Following the choke point-driven methodology [6], which emphasizes the performance bottleneck embedded in applications to ensure the benchmark's coverage of challenging data management features, we designed FinBench based on features and identified choke points profiled in real financial datasets and queries across multiple business clusters. In this paper, we describe our main contributions in the following sections. Section 2 presents the data characteristics, such as *power-law degree distribution* and *edge multiplicity*, profiled in real financial datasets. It also discusses the scalable event-based data generator used to create datasets with these characteristics and the parameter generator to ensure stable runtime across different query variants. Section 3 illustrates the typical query patterns we found, such as *temporal filtering*, *recursive path*, and *read-write queries*, as well as the new identified choke points. Section 4 describes the query categories, their interrelationship, and the method to orchestrate the workload. Section 5 shows the experiments results on choke points and end-to-end evaluation. Section 6 provides the review of related benchmarks. We present the conclusions in Section 7 and future research directions in Section 8.

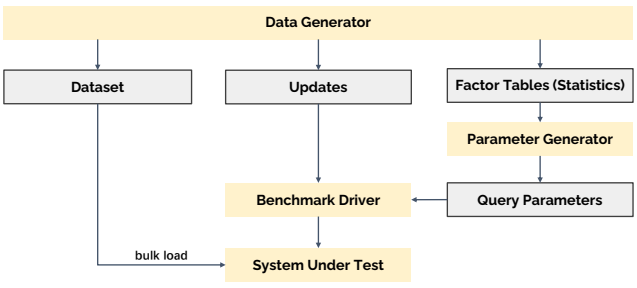


Figure 1: A high-level overview of LDBC benchmarks' framework. Yellow boxes are software components, and gray boxes are data artifacts. [2]

2 FINANCIAL DATASET

Through systematic abstraction across multiple business clusters, we identified common entity types (e.g., accounts and media devices) and relationship patterns that form the foundation of benchmark schema presented in Section 2.1. The statistical characteristics presented in Section 2.2 reflect key distribution patterns observed in these real-world datasets. In Section 2.3, we present the data generator implemented to generate the benchmark dataset of the data schema and approximate the dataset characteristics.

2.1 Financial Data Schema

Modeling financial data as graphs enables the representation of activities like money transfers among accounts and investments among companies, facilitating analysis. Financial data is often modeled as a directed graph, where vertices represent entities (e.g., persons, companies, accounts) and edges represent interactions (e.g., owning, transferring, guaranteeing). By investigating the data schema across different financial business production environments from Ant Group [23] and clients served by the task force members [8], we found that each vertex and edge type is associated with rich properties, such as creation timestamp, transfer amounts, and various annotations. We selected a typical fund exchange scenario in financial business operations to design the schema. Online schema often include various subtypes of the same entity to reflect real-world distinctions. For example, corporate accounts and individual accounts, both derived from the broader concept of accounts, are differentiated based on specific applications. To avoid unnecessary complexity, we abstract the schema by merging the similar vertex and edge subtypes (see Figure 2) derived from real financial graphs.

Online schemas often include various subtypes of the same entity to reflect real-world distinctions. For example, Corporate Accounts and Individual Accounts, both derived from the broader concept of "accounts," are differentiated based on their specific applications.

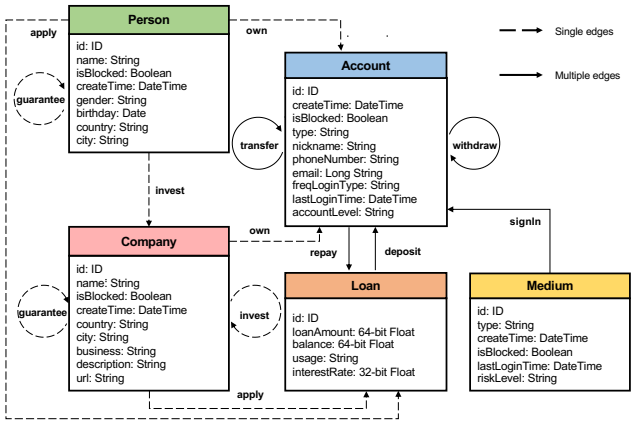


Figure 2: The schema of FinBench dataset. Edge properties are omitted.

The data schema (Figure 2) comprises five vertex types and nine edge types. It consists of accounts owned by persons and companies, signed in via media, which repay loans after receiving the deposited amounts. Persons can guarantee one another and invest in companies; companies can guarantee and invest in one another; and accounts can transfer funds to and withdraw funds from each other. Edge multiplicity, where multiple edges can exist between the same pair of source and destination vertices, was observed during our analysis of real-world graphs. This observation, absent in existing benchmarks, motivated the design of our schema to address the resulting new requirements for systems (e.g., in the storage model). In Figure 2, solid arrows represent edge types that allow edge multiplicity, such as "signIn", "transfer", "withdraw", "deposit", and "repay" while other edge types represented by dashed arrows do not allow edge multiplicity.

2.2 Financial Dataset Characteristics

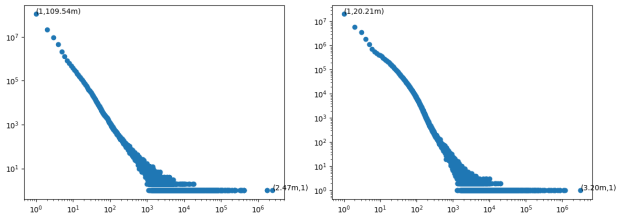
In typical financial applications, skewness and temporality are two key features frequently observed. Skewness refers to the asymmetry in the distributions of financial activities, such as the uneven number of transfers across the accounts influenced by the market structure. Temporality refers to the periodical fluctuations of financial activities over time, like transfer peaks during busy hours. In financial graphs, skewness is reflected in skewed degree distributions, while temporality manifests as the periodical patterns in the timestamps of vertices or edges. To illustrate these features in a real-world context, we conducted empirical profiling experiments on a financial graph from Ant Group. Specifically, we randomly sampled three subgraphs from the online production environment by uniformly selecting vertices to ensure representative profiling.

Degree distribution. Figure 3 illustrates the in- and out-degree distributions of transfer edges within the three sampled subgraphs. Although degree values scale with the subgraph size, all distributions follow the power law, which means the probability of a vertex's in- or out-degree being x is inversely proportional to x raised to a negative exponent: $\Pr(x) = c \cdot x^{\alpha}$, where α is the power law exponent and c is a normalization constant. The power law degree distributions demonstrate the skewness phenomenon, with most accounts having fewer than ten transfers while a few accounts have millions of transfers. Table 2 lists the power law exponents and normalization constants derived from the distribution regression analysis. Power law distribution is also observed in social networks and applied in existing benchmarks, such as SNB [11, 44] and LinkBench [4]. SNB dataset model is derived from Facebook's social graph [55], which enforces a strict degree cap of 5,000 due to platform constraints (user friendship limits) [12]. This results in bounded skewness and a power-law distribution with a shallower tail. LinkBench models a broader Facebook graph, including objects like posts and albums, but its maximum degree, 1 million, still reflects social interactions rather than financial transactions. The power-law parameters differ from the financial graph we profiled in Table 2 and Figure 3. We observed unbounded degree scaling (e.g., up to 150 million) and scale-sensitive degree growth (degrees increase organically with graph size).

Edge multiplicity distribution. Edge multiplicity is a common feature in real-world financial graphs, where multiple edges may exist in the same vertex pair, representing repeated transactions over time. For instance, a company account might transfer salaries to the same employees periodically, resulting in multiple edges. We

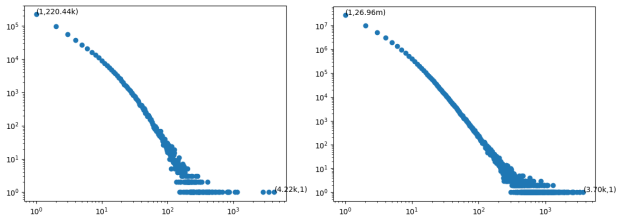
Table 2: Power law exponents & normalization constants of degree distributions.

	Subgraph 1	Subgraph 2	Subgraph 3
In-degree, α	-2.319	-2.319	-2.085
In-degree, c	109,539,041.821	78,379,700.038	133,908,623.887
Out-degree, α	-1.720	-1.719	-1.720
Out-degree, c	20,186,572.914	14,153,912.686	20,194,472.855



(a) Subgraph 1, in-degree **(b) Subgraph 1, out-degree**

Figure 3: Degree distribution of the real financial graph. The X-axis is the degree, and the Y-axis is the number of vertices.



(a) In-edge multiplicity **(b) Out-edge multiplicity**

Figure 4: Edge multiplicity distribution of the top-1 hub vertex. The X-axis is the edge multiplicity, and the Y-axis is the number of vertices.

concisely refer to the number of edges between a vertex pair as *multiplicity*. Skewness is evident in the edge multiplicity distributions of hub vertices with high in or out degrees. Most neighbors of a hub vertex have low multiplicity, while a few exhibit very high multiplicity. Based on transfer edges, the edge multiplicity distributions of the top hub vertices (ranked by degree) are observed to follow a power-law distribution. Figure 4 presents in- and out-edge multiplicity distributions of the top-1 hub vertex.

Edge timestamp distribution. The distribution of edge timestamps in a financial graph demonstrates pronounced temporal patterns, as the creation of new vertices and edges fluctuates periodically. Figure 5 displays the distribution of transfer edges' timestamps throughout the day, where x on the x-axis represents each one-hour interval starting from x o'clock. The distribution is notably uneven. The most prominent peak occurs from 8:00-9:00, aligning with the opening time of banks and financial markets. Smaller peaks occur around mealtimes, while few transfers occur from midnight to early morning.

Financial graphs exhibit skewness and temporal features in their vertex degree, edge multiplicity, and timestamp distributions, as

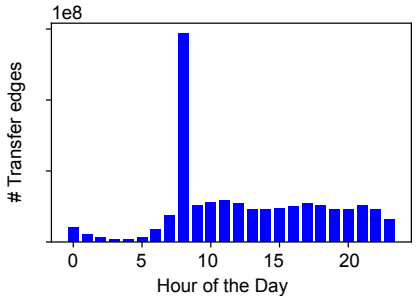


Figure 5: Edge timestamp distribution.

shown by real financial graph datasets' profiling results. These features set financial graphs apart from other types of graphs. For instance, social networks are typically less skewed than financial graphs due to little or no edge multiplicity, and their timestamp distribution' peaks do not significantly correlate with busy hours. Inherent skewness causes biased performance in the data dimension, while temporality leads to biased performance in the time dimension. Both aspects challenge resource allocation and load-balancing in systems design targeting financial scenarios.

2.3 Benchmark Dataset Generation

The data generator is designed to generate data of schema illustrated in Section 2.1, inspired by SNB generator [43]. It integrates the degree and timestamp distribution profiled in Section 2.2. The generator operates in two phases: simulation and segmentation. The simulation generates raw data over a configurable period (default: 3 years). The segmentation splits the raw data into snapshot data and incremental data. Snapshot data is loaded as the initial graph state, while the incremental data is used by the benchmark driver to simulate the write queries, as described in Section 4.1. Scale factors (SF) determine the size of the generated datasets. For example, SF1 generates a graph of 1 GiB on disk. The data statistics of the datasets with different SF are shown in Table 7 in Appendix B.

Simulation process. The data generator uses an event-based framework to simulate financial activities. According to the data schema, the data generator first creates persons and companies, then simulates the account registration, media sign-ins, investments, loans applications, and fund transfers. In each event, vertices and edges are recorded and will be written to the disk upon completion. Entity counts are scaled proportionally to the target dataset size. The event-based framework is implemented with Spark to ensure scalability. The degree distribution, edge multiplicity, and edge timestamp distributions presented in Section 2.2 are set as parameters to approximate the real financial graphs. For example, the data generator parses the specified degree distribution and assigns the target degree to each account vertex before the fund transfer event. Then, the transfer edges are created, targeting the assigned degree. When the target degree exceeds a predefined threshold, the account is labeled as a hub vertex whose edge multiplicity with neighbors is determined by the edge multiplicity distribution. The timestamps of edges are generated based on the edge timestamp distribution. With the event-by-event simulation, the data generator creates the raw data aligning with the specified distributions and

scale. All parameters, including simulation duration, distributions (degree, edge multiplicity, timestamp), and timeline breakpoints, are configurable to users, which supports tailored dataset generation for diverse benchmarking scenarios.

Segmentation process. The raw data generated by the simulation is partitioned into snapshot data and incremental data at a configurable breakpoint on the simulation timeline. The default breakpoint is the 97th percentile of the total period in the simulation process. This allows the snapshot to represent a stable initial state while incremental data drives dynamic write workloads. The incremental data will be transformed to write queries (TW, see Section 4.3) in terms of adding a vertex or adding an edge.

With the two step dataset simulation process and segmentation process, the data generator outputs the snapshot dataset as the initial state of system under test and the write queries applied as the data updates during benchmarking.

3 FINANCIAL QUERIES

We adopt the choke point-driven methodology [6], a strategy integral to developing the LDBC benchmark, particularly in investigating and formulating queries. Choke points are technological challenges embedded in the key aspects of query execution, optimization, and storage when designing and building systems. Focusing on these choke points identified in real applications challenges systems and provides potential optimization opportunities driving system advancement [10]. This approach highlights nuanced performance differences across systems, providing insight into their capabilities and limitations in handling complex scenarios.

This work rigorously evaluates graph database systems by introducing choke points derived from real-world financial scenarios. In storage optimization, the temporal window filtering queries described in Section 3.1 necessitate optimizations in the storage layer design to address temporal locality in graph traversal operations. The recursive path filtering patterns in Section 3.2 and native truncation mechanisms in Section 3.4 challenge the graph query language to express complex relationships concisely while maintaining computational efficiency by developing intelligent pruning strategies to eliminate redundant intermediate results. The read-write queries in Section 3.3 pose challenges in efficiently managing high-concurrency read-write operations. Due to space constraints, we highlight the most crucial choke points here and refer readers to Appendix A for details.

3.1 Queries with Time-window Filtering

Time is integral to financial systems, influencing various business operations ranging from transaction processing to strategic decision-making. Recent data is typically accessed with significantly higher frequency, which poses challenges for systems in improving data access locality in storage and optimizing temporal performance (e.g., temporal indexing) in query optimization. It aligns with the Five-Minute Rule [21], which is a cost-benefit principle suggesting that data accessed within short time intervals (e.g., five minutes) should reside in faster storage mediums. Potential optimization opportunities are separating the hot/cold data and storing hot data in caches. For example, GeaBase [18] incorporates a Time-to-Live property to mark the data lifetime for regular compaction to prune the cold data, leading to query performance improvement. Moreover, temporal access patterns are often represented as time-window filters within queries.

We design queries with temporal filters to address these challenges. These queries test a system's efficiency in processing time-critical data across financial activities, focusing on optimizing data access based on temporal parameters. As shown in Figure 6, complex read query 4 (TCR4) identifies transfer cycles, a pattern indicative of potential risks, using a time window filter defined by parameters *startTime* and *endTime*.

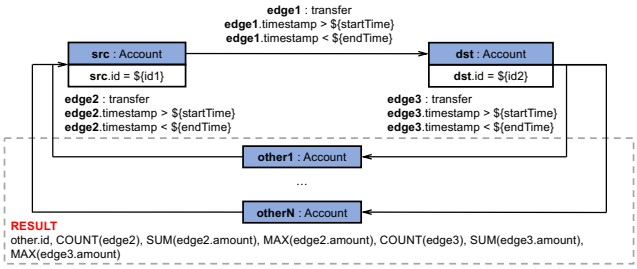


Figure 6: The pattern of complex read query 4 (TCR 4) in transaction workload.

Query description: Given two accounts *src* and *dst*, and a specified time window between *startTime* and *endTime*, (1) check whether *src* transferred money to *dst* in the given time window (*edge1*). If *edge1* does not exist, return with empty results (the result size is 0). (2) find all other accounts (*other1, . . .*) which received money from *dst* (*edge3*) and transferred money to src (*edge2*) in a specific time. For each of these other accounts, return the account ID, the count of transfer edges, the sum, and the max of the transfer amount (*edge2* and *edge3*).

3.2 Queries with Recursive Path Filtering

A critical task in financial risk control involves detecting suspicious accounts by analyzing their transactional linkages to blocked accounts, particularly through preplanned fund flows. As illustrated in Figure 7, complex read query 1 finds the paths (fund flow) to blocked accounts filtered by monotonically ascending timestamp order (see timestamps in Figure 7). The fund flow analysis becomes computationally intensive when traversing multiple interconnected accounts, highlighting the challenge of efficient multi-hop recursive filtering. These recursive path filtering queries assess a system's ability to navigate and filter intricate fund transfer paths, where each hop is filtered based on the previous hop's match. Pruning the traversal with such filter conditions is challenging, as it aims to minimize unnecessary computation and intermediate results, which involves managing complex path constraints from query language expression ability to query optimization. Optimization opportunity exists in designing appropriate path-level filter predicates to reduce the search space and efficient traversal pruning strategy to minimize computational overhead during query execution. These queries involving path filtering focus on the system's ability to perform complex graph traversals efficiently, and also pose challenges to concise query language expression, which both are key requirements for tracking and analyzing financial graphs.

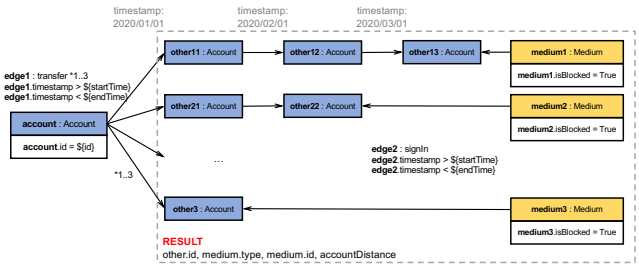


Figure 7: The pattern of complex read query 1 (TCR 1) in transaction workload.

Query description: Given an Account and a specified time window between *startTime* and *endTime*, find all the *Account* that are signed in by a blocked *Medium* and have funds transferred via *edge1* by at most 3 steps. Note that all timestamps in the transfer trace must be in ascending order (only greater than). Return the account ID, the distance from the account to the given one, the ID, and the type of the related medium.

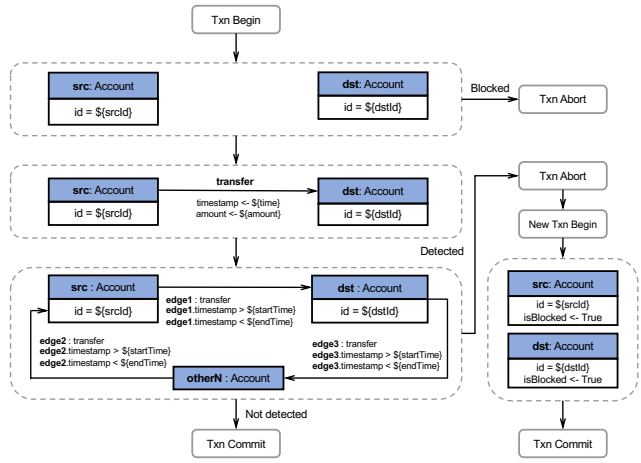


Figure 8: The pattern of read-write query 1 (TRW 1) in transaction workload.

Query description: The workflow of this read-write query contains at least one transaction. It works as:

(1) Read the blocked status of related accounts with given IDs of the *src* and *dst* accounts. The transaction aborts if one of them is blocked. Go to the next step if none is blocked.
(2) Add a transfer edge from *src* to *dst* inside a transaction. Given a specified time window between *startTime* and *endTime*, find the other accounts that received money from *dst* and transferred money to *src*. The transaction aborts and go to the next step if a new transfer cycle is formed. Otherwise, the transaction is committed.
(3) Mark the *src* and *dst* accounts as blocked in a new transaction.

3.3 Read-Write Queries

In risk control applications, a critical process involves recording users or transactions flagged by risk control mechanisms, leading to a significant volume of operations involving reading and writing data. The challenge for systems in such scenarios lies in efficiently managing high-concurrency read-write operations. This requires optimization for concurrent processing and careful management of potential read-write conflicts in high-throughput environments. FinBench's read-write queries rigorously test the system's capability to manage dynamic data flows, where rapid and precise read-write operations are crucial. This capability is crucial for maintaining the integrity and security of financial transactions, thereby making system performance on these queries essential. TRW 1 (Figure 8) exemplifies the pattern of read-write queries, where the Write Query execution depends on the result of the Complex Read Query. It encompasses multiple read and write queries in an intricate transactional procedure. Efficient concurrent transaction processing is vital because the read result, which influences the execution path, may depend on another query's write. For example, an account whose information is written in the third step of a TRW1 query may be read during the first step of another TRW1 query. Therefore, a blockage in the former query can delay the latter's execution.

3.4 Truncation on Hub Vertices

Hub vertices, often with more than hundreds of thousands or even more adjacent edges, represent hotspots in business activities, such as an account handling salary transfers for numerous employees. Traversing all adjacent edges of hub vertices can lead to exponential growth in the computational load. In real time financial query processing, introducing constraints to prune the traversal on hub vertices is a common solution. FinBench addresses this issue by introducing a truncation mechanism, which ensures result consistency across benchmark runs. It specifies the order of the edge sorting and the maximum number of edges to traverse from each vertex, acting as a sampling mechanism to balance load. Each query involving truncation specifies truncation order and limit as parameters. For example, in complex read query 1 (TCR1) shown in Figure 7), if the truncation order is set to descending by transfer timestamps and the limit is 1000, only the latest 1000 edges will be traversed, even if there are more than 1000 outgoing transfer edges from the start account vertex. It is noteworthy that to ensure result consistency, truncation needs to be explicitly specified at a particular hop in the graph traversal process, and this requirement will be prioritized in future benchmark optimizations.

The truncation mechanism introduces challenges in several areas. From the language perspective, current widely used declarative graph query languages, such as ISO GQL [16], Cypher [22], and SPARQL [40] lack native support for truncation. In terms of query execution, truncation impacts intra-query parallelization (e.g., load balance) on hub vertices. From the storage perspective, organizing hub vertices' adjacent edges in truncation order can provide better data access locality and expedite query processing.

4 TRANSACTION WORKLOAD

The business clusters exhibit significant variations in user scale and business complexity, which makes unified modeling of workload and performance metrics impractical. To balance between the coverage of all kinds of queries and realism of test scenario, we distilled the most challenging query types and the workload organizations by abstracting business-specific queries and analyzing them purely from graph query patterns. In this section, we describe the organization of the workload. Section 4.1 describes the query classes in

transaction workload including *Complex Read Queries*, *Simple Read Queries*, *Write Queries*, and *Read-Write Queries*. Section 4.2 describes the parameter curation for *Complex Read Queries* and Section 4.3 describes the method to mix the queries.

4.1 Queries Classes in Transaction Workload

Aligning with actual business workflows, complex multi-hop queries presented in Section 3 mirror investigative patterns where analysts trace transaction chains, followed by single entity queries (simple reads) for detailed entity inspection. Write operations represent the new data inserted or updated. This abstraction decouples business semantics from computational patterns while preserving essential workload challenges embedded in applications observed across clusters. The workload consists of four query classes: Complex Read Queries (TCR), Simple Read Queries (TSR), Write Queries (TW), and Read-Write Queries (TRW).

Complex Read Queries (TCR). This category includes 12 complex read-only queries designed to address various financial activities over a specific period. Most queries involve at least three hops and include complex aggregate queries, such as identifying the shortest transfer paths or detecting money laundering activities involving loans. In real financial scenarios, near real-time computation is expected, even though these queries involve a certain level of complexity. Query on-time compliance is required in the benchmark to meet the real-time processing demands. The query-on-time timeout for each query is set to 1 second, an empirical and practical value. The benchmark driver calculates the on-time query rate and fails the test when it is less than 95%.

Simple Read Queries (TSR). This category comprises 6 simple read-only queries involving no more than 2 hops, such as inquiries about account details or transaction messages related to transfers to or from an account (e.g., TSR 4 and TSR 5), which are straightforward and require immediate results.

Write Queries (TW). This category includes 17 insert/delete queries required by the benchmark driver to insert data focusing on inserting new data and deleting existing data and another 2 in-place update queries for entity state marking purpose (e.g., marking an account as blocked). Insert queries create a vertex/edge with properties, while delete queries remove a vertex and all its adjacent edges from the graph. Update queries locate a vertex by its ID and modify the value of one of its properties.

Read-Write Queries (TRW). This category introduces a unique design called the read-write query, which incorporates both read and write operations within transactions. There are 3 read-write queries, where the success of the entire transaction depends on each operation. Failure in any part results in the transaction being unsuccessful. This approach is commonly used in financial scenarios to assess the potential risks of a write operation, such as a transfer, before committing it. It allows for matching a newly formed risky pattern without actually writing them to storage.

Real Workload in Financial Systems. The workload is designed closely to represent real-world scenarios by reflecting the ratio of different query classes and the pattern in which they are orchestrated. We profiled queries in a real financial graph production environment, focusing on the read-to-write query ratio (20:1). We adjusted the frequency parameters for each query in the configuration to reproduce this ratio in the generated workload. Profiling the query mix is challenging because it requires uncovering unknown correlations in a long sequence of queries, which has not been extensively studied in previous research. Therefore, we design the query mix mechanism based on empirical observations tailored to the financial scenario, which will be introduced in Section 4.3.

4.2 Parameters Generation for Complex Read Queries

Parameter curation ensures benchmarking reliability in financial graphs characterized by power-law distributions and temporal dependencies. Random parameter selection in such skewed distributions causes significant variations in intermediate result sizes and query execution times, leading to unreliable benchmarks [11, 44]. Performance-sensitive parameters like *truncationOrder* and *truncationLimit* require specialized handling beyond conventional automation. Traditional methods relying on entity-level statistics may still yield high runtime variance due to hub vertices with numerous edges. To mitigate this, we combine factor tables with iterative processing that incorporates edge-level statistics.

Factor Table Generation. Factor tables capture statistical relationships between entities. For example, in TCR 12 (see Figure 9), the tables include data such as each person's account list (Table 3), transfer out-edges per account, and bucketed statistics on edge timestamps and transaction amounts. We record the number of transfer edges per account across exponentially growing transaction amount ranges, ensuring even distribution of high and low-amount transactions. These bucketed statistics improve the accuracy of intermediate result size estimation during parameter curation.

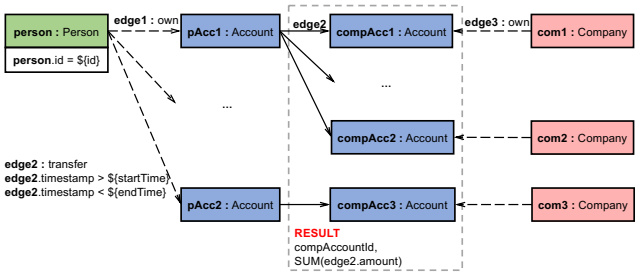


Figure 9: The pattern of complex read query 12 (TCR 12) in transaction workload.

Query description: Given a *Person* and a specified time window between *startTime* and *endTime*, find all the *companies' accounts* that s/he has transferred to. Return the IDs of the *companies' accounts* and the sum of their *transfer amount*.

Hybrid parameter curation. We use a multi-hop iteration mechanism to apply truncation filters at each traversal step using factor tables. As shown in Figure 9, iterating over each person's account list and linked accounts generates second-hop account entities as intermediate results. Truncation is applied at each hop to narrow down edges iteratively. For example, when retaining the top 100 edges by transaction amount for a hub vertex, we select edges

Table 3: Person ID and Account List Table

person_id	account_list
1048	[4748482857108769042, 4896538694858574097, 4671358713490049299]
1049	[4819977501193275668]
1052	[4865294972443691290, 4915116043321477401, 4675862313117419803]
1057	[4766778730594961703, 4670514288559917350]

from buckets with the highest amounts until the cutoff is reached. A greedy algorithm minimizes variance by selecting parameters that yield balanced results [24]. This iterative approach ensures consistent query execution times and enhances benchmark reproducibility and reliability.

4.3 Mixing the Queries

In real-world applications, workloads include a broad spectrum of scenarios, including simple read operations, complex read operations, insert operations, delete operations, and read-write operations. Incorporating mixed queries into benchmarking processes enables a more precise evaluation of system performance and efficiency under realistic conditions. We accurately simulate real-world mixed queries based on the observations from real-world applications and scenarios.

Complex Read Queries. An observed fact is that users frequently rely on complex queries to extract valuable insights from large datasets in business operations. These queries consume significant resources, substantially impacting system performance and often revealing system bottlenecks. Complex read queries are central to system processing and form a key focus of our evaluation. To construct a representative benchmark, we derive complex queries from multiple business scenarios, including fund transaction graphs, media asset graphs, anti-fraud graphs, reimbursement graphs, insurance graphs, and media device graphs. Through cross-scenario normalization, we calibrate query frequencies to avoid workload dominance by any single query pattern, which is a common pitfall in benchmarks. This ensures balanced testing across query categories while maintaining reproducibility. Notably, the benchmark supports configurable query frequencies and selective activation to enable domain-specific customization that prioritizes business-critical patterns based on their specific characteristics.

Simple Read Queries. Another observed fact is that simple read queries serve as low-cost instant data inspections and business investigations alongside complex read queries. In most cases, complex queries involve broader datasets or longer time windows, producing more results. The results can subsequently trigger simple queries, using them as input parameters for further analysis. For instance, when analyzing total funds from accounts applying for loans, longer complex queries can lead to simpler read queries to investigate specific accounts related to the applications and the transactions.

To address the dynamic interaction between simple and complex read-only queries, we designed a time window-biased simple read query generator that connects the two query types. This generator ensures that the complex query outputs are inputs for subsequent simple read-only queries. This interconnected approach enhances data retrieval efficiency and relevance, ensuring that the comprehensive analysis informs each simple read query of preceding complex read queries. As the number of simple read queries is influenced by the time window of complex read queries but remains variable, we aimed to simulate this distribution while preserving real-world uncertainty. We achieved this with a biased random walk strategy [39], employing an exponential distribution to adjust the randomness based on the time window's length. Longer durations increase the likelihood of generating simpler queries. The probability formula for the time-based exponential distribution is as follows:

$$Prob(x) = 1 - U^{k \cdot t}$$

where:

- U is a random variable uniformly distributed in the interval $[0, 1)$.
- t is the time window length for the complex read query.
- k is a coefficient that adjusts the time window.

This approach enables computation of the cumulative probability value associated with the exponentially distributed random variable U, which effectively maps U to a value within the interval $[0, 1)$. This transformation aligns with the need to generate a ratio of simple read queries based on the scaled time window of complex read queries. The probability value is used to determine the count of simple read queries.

Write and Read-Write Queries. The generation of parameters for write and read-write queries ensures that insert operations always involve new data. There is a necessary sequence among different types of write and read-write queries. For example, account creation must precede actions such as setting up transfer relationships. This order reflects natural dependencies, ensuring that each query executes only after completing the prerequisite steps. The data generator (See Section 2.3) assigns a creation timestamp to each vertex and edge during the simulation, mapping these to the relative timestamp for each write query. During benchmarking, the driver reads the load size parameter, defined as a time compression coefficient relative to the simulation, and dynamically calculates actual request times based on these relative timestamps.

5 EVALUATION

While domain-specific metrics like fraud detection accuracy are critical for financial applications, they depend on higher-level business logistics beyond the benchmark scope. By distilling financial scenarios into workload and dataset characteristics, we focus on infrastructure-level performance metrics of the foundational queries, including throughput and query latency, reflecting the system's capacity to accelerate downstream financial tasks. Since the testing of ACID properties in modern databases has been well-established, FinBench introduces a standardized tool [14], which evaluates ACID compliance and isolation levels by simulating diverse concurrency scenarios and injecting faults.

FinBench has been used to evaluate graph databases such as TuGraph [46], Galaxybase [51], and gStore [30], and the implementations[1] have been open-sourced [15]. In Section 5.1, we present the

[1]Note that the experiment results presented here are not official LDBC Financial Benchmark results as they were not audited.

analysis of experiment results on TuGraph-DB [33, 47] to demonstrate the identified choke points. In Section 5.2, we conduct benchmarking experiments on Galaxybase [51] and gStore [30] to demonstrate benchmark capability and portability [20, 25].

5.1 Choke Points Analysis

As illustrated in Section 2 and Section 3, the new choke points identified in financial scenarios present challenges to graph systems, ranging from the language expression ability to the query execution and storage management.

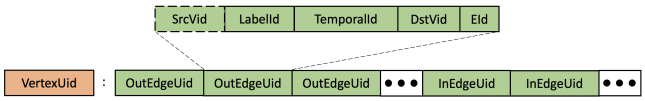


Figure 10: TuGraph-DB's graph topology packing [33].

Optimization in Storage for Queries with Temporal Filters. A common feature of financial datasets and queries is the frequent usage of timestamps and time windows-based filters, which challenge systems to improve temporal access locality when filtering the edges. Designing systems with optimization strategies, such as sorting edges by timestamp in storage and employing effective search methods, can enhance performance in such scenarios.

TuGraph-DB [47], an open-source graph database of TuGraph, adopts an innovative storage design that sorts edges by timestamp within a graph topology built on key-value storage [33]. As shown in Figure 10, the vertex's unique identifier serves as the key, while the inedges and outedges are packed as the value in the key-value store. A unique identifier combination of SrcVid (source vertex id), LabelId (edge type id), TemporalId (timestamp id), DstVid (destination vertex id), and Eid (edge id) is assigned to each edge, where Eid is designed to support edge multiplicity, and TemporalId allows sorting edges in timestamp order. TuGraph-DB provides a temporal sorting feature that sorts the edges by timestamp. Without this feature, edges are randomly ordered, requiring iterating over all edges and then selecting those whose timestamp is in the given time window during graph traversal. Enabling this feature improves data access locality to time-window-based queries. Sorting edges in timestamp order allows the binary search to efficiently find the first edge after the start time, iterating through subsequent edges until the first edge beyond the end time. Binary search leverages storage locality, minimizing unnecessary memory page loads during iteration, particularly with high-degree vertices, thereby enhancing time window query performance.

To test the effectiveness of the temporal sorting feature, we conducted experiments on Complex Read Query 1 (see Figure 7), with an example Cypher query provided in Listing 1, using the SF100 dataset on TuGraph-DB. After cold starts, TuGraph-DB was tested in two modes: with and without the enabled temporal sorting feature. The elapsed times for query execution and edge iteration were recorded. Table 4 shows the query execution time and edge iteration time of the two modes. To illustrate the effectiveness of the optimization strategy, enabling temporal sorting reduces edge iteration time by 69.5% and query execution time by 40.9%.

```
MATCH p = (acc:Account {id: $accId})-[e1:transfer *1..3]->(
    other:Account)<-[e2:signIn]-(medium)
WHERE isAsc(relationships(e1, 'timestamp'))=true
AND head(relationships(e1, 'timestamp')) > $startTime
AND last(relationships(e1, 'timestamp')) < $endTime
AND e2.timestamp > $startTime
AND e2.timestamp < $endTime
AND medium.isBlocked = true
RETURN DISTINCT other.id as otherId, length(p)-1 AS
    accountDistance,
medium.id AS mediumId, medium.type AS mediumType
ORDER BY accountDistance, otherId, mediumId
```

Listing 1: An example Cypher query of TCR 1

Table 4: The average edge iteration time (ms) and query execution time (ms) of TCR 1 on TuGraph-DB on SF100 when the temporal sorting feature is disabled and enabled.

Avg. time (ms)	Disabled	Enabled	Improvement
Edge Iteration Time	102,511	31,265	69.5%
Query Execution Time	196,065	115,817	40.9%

Optimization in Query Execution for Recursive Path Filtering Queries. Pruning traversals using recursive path filters is challenging because these filters cannot be applied in advance. Currently, FinBench includes six recursive path filters: isAsc, isDesc, head, last, minInList, and maxInList. For example, the isAsc filter requires that the timestamps along a path be in ascending order. Let p be a recursive path filter. A baseline approach to implement p is obtaining a path set without applying p and then scanning each path to verify if it satisfies p. This approach generates numerous invalid intermediate results, indicating significant potential for optimization in both memory usage and query efficiency.

We implemented an optimized version of recursive path filtering in TuGraph-DB. Specifically, for a recursive path filter p, we introduce a traversal state, denoted as $state$, and a new filter p'. The $state$ is maintained during path traversal, and each time the path is expanded, we check if the new filter p' applies to the newly expanded vertex or edge and the current $state$. For example, for the recursive path filter isAsc, the $state$ tracks the *maximum* timestamp encountered so far. The new filter p' returns true if the new timestamp exceeds the current $state$; otherwise, it returns false. Thus, instead of post-processing paths with the filter p, we can push down another filter p' to prune invalid traversals in advance, thereby improving query performance. We experimented on both the baseline and optimized versions for recursive path filtering and compared their performance. We selected three query types from FinBench (TCR 1, TCR 2, and TCR 5) that require recursive path filtering. Table 5 shows the average execution time. Compared to the baseline version, our optimized version reduced the execution time by 70.3%, 87.4%, and 83.3%, for the respective queries, verifying the effectiveness of our approach.

Optimization in Query Language to Express Truncation and Recursive Path Filtering. Graph Query Languages, such as ISO GQL [16] and SQL/PGQ [17], cannot express Truncation and Recursive Path Filtering concisely, limiting their applicability in financial

Table 5: The average execution time (ms) of queries involving recursive path filter on two methods.

Avg. time (ms)	Baseline	Optimized	Improvement
TCR1	1,339,644	397,078	70.3%
TCR2	1,772,853	222,250	87.4%
TCR5	1,715,553	285,430	83.3%

scenarios. As an example solution, Crowe et al.'s proposal [9] addresses the Truncation language choke point by extending the GQL syntax with a `TRUNCATING` construct. It specifies both the truncation limit and order, which is compiled into a constraint applied to graph traversals. Further proposals and discussions are expected to advance the standard graph query language.

5.2 End-to-end evaluation

In this section, we present the performance results of Galaxybase [51] using Cypher and gStore [30] using stored procedures to validate FinBench's capability, portability, and completeness. The experiments were conducted on a machine with 32 cores, powered by a HiSilicon Kunpeng 920 CPU and 256GB of DDR4 memory, running the Ubuntu 22.04 LTS.

Galaxybase is a distributed graph database that uses a native graph storage format specifically designed to handle HTAP (Hybrid Transactional & Analytical Processing) query workloads. In particular, for the queries that require the retrieval of paths with sequentially increasing property values, Galaxybase implements a Cypher extension that specifies the property that needs to increase in the respective MATCH clause, effectively addressing the recursive path filtering language choke point (Choke Point 8 in Appendix A). Galaxybase ensures data consistency by implementing read-write queries as transactions, which are rolled back if any condition is not met during the query process. gStore is an open-source, centralized graph database system. Though gStore was originally designed for RDF graph data, its extended version now supports RDF and property graphs simultaneously with the same underlying native graph storage backend. gStore implements the FinBench workload queries as stored procedures.

We ran the benchmark on gStore at SF10 and Galaxybase at SF100 over a continuous 2-hour period. The average execution times for the workload queries are shown in Table 6. The latencies of recursive path filtering queries (see Section 3.2), such as TCR 1 and 2, are relatively high compared to the other queries. This indicates room for optimization at this performance bottleneck.

6 RELATED WORK

Benchmarks for graph data. Benchmarks for graph technologies have been proposed to match the ever-increasing graph scenarios, and most of them are designed for social network data. LinkBench [4] is inspired by social network data and web service applications at Meta. BigDataBench [5, 56] is a benchmark suite that provides a graph analytics workload on the Meta social network dataset for different big data systems, such as Hadoop, Spark GraphX, and GraphLab. BG [1] measures a graph database's capability of processing interactive social network workloads based on the simulation of individual stateful users. Graph Benchmark [34] provides a micro-benchmarking framework with a broader spectrum of test queries and datasets of LDBC, DBpedia, and air routes [29]. TAOBench [7] captures the social graph workloads by accurately simulating the production request patterns at Meta. LDBC plays an important role in benchmarking graph data. The LDBC Semantic Publishing Benchmark (SPB) [28] is an LDBC benchmark for RDF database engines inspired by the Media/Publishing industry, assuming that an RDF database is used to store both the reference knowledge (mostly static) and the metadata (that grows constantly, to stay in sync with the inflow of streaming content). For labeled properties graph systems, the widely-used LDBC SNB [2] targets systems with graph-processing capabilities from the perspective of social networks, and it contains two workloads: Interactive Workload [11] and Business Intelligence Workload [44].

Regarding the workloads, only LDBC SNB [2] and TAO contain transactions, and TAO provides read-only and write-only transactions. In particular, FinBench and SNB both involve multi-hop traversal and property filters on the data graph. They are compared in Table 1. FinBench introduces several key features unique to the financial scenarios but not shown in social networks, including edge multiplicity, rich properties, and temporal skewness of the data (see Section 2.2), query choke points such as hub vertex truncation and transactional read-write queries (See Section 4.1), and a query mix mechanism that considering the time window of complex read queries (See Section 4.3).

Benchmarks for relational databases. Some of the most widely used relational database benchmarks are proposed by the Transaction Processing Performance Council (TPC), including TPC-C [52], TPC-H [54], TPC-DS [53], etc. TPC-C is an online transaction processing (OLTP) benchmark. It tests the OLTP system using a commodity sales model involving five concurrent transactions of different types and complexity. Both TPC-H and TPC-DS are decision support benchmarks for online analytical processing (OLAP)[36, 41, 42]. TPC-H consists of business-oriented ad hoc queries and concurrent data modifications built with a third Normal Form (3NF) schema. Compared with TPC-H, TPC-DS has more difficult queries and implements multiple Snowflake schemas with shared dimensions to support the evaluation of new emerging databases. Despite their intricate design and wide adoption, the TPC benchmarks reflect neither the characteristics of graph data and queries nor the specific demands of financial scenarios, which are the focuses of FinBench.

7 CONCLUSION

This paper introduced the LDBC Financial Benchmark (FinBench) Transaction Workload, which fills a critical gap in the current landscape of graph benchmarks and sets a new standard for performance evaluation in the FinTech industry. By emphasizing critical novel performance bottlenecks identified in financial workloads, FinBench includes a scalable data generator capable of synthesizing datasets with financial-specific characteristics, a parameter generator ensuring runtime consistency across queries through the use of bucketed data statistics, and a scalable benchmark driver that biases query execution based on time windows. We conduct two experimental evaluations of FinBench on two graph database

Table 6: Detailed benchmark results for gStore@SF10 and Galaxybase@SF100. Query latencies are reported in milliseconds.

Query	TCR1	TCR2	TCR3	TCR4	TCR5	TCR6	TCR7	TCR8	TCR9	TCR10	TCR11	TCR12	TSR1	TSR2	TSR3	TSR4	TSR5	TSR6
gStore, SF10	1.40	1.56	11.42	0.77	0.57	1.44	2.16	0.80	1.47	1.23	1.39	1.74	0.73	0.68	0.73	0.64	0.63	0.80
Galaxybase, SF100	2.52	2.87	0.82	1.01	2.87	0.89	0.91	2.34	1.63	1.23	0.80	0.99	0.50	0.85	1.01	0.65	0.69	3.89

Query	TW1	TW2	TW3	TW4	TW5	TW6	TW7	TW8	TW9	TW10	TW11	TW12	TW13	TW14	TW15	TW16	TW17	TRW1	TRW2	TRW3
gStore, SF10	3.22	3.83	2.77	10.39	3.81	24.37	3.11	1.06	1.01	4.36	0.51	0.97	1.84	0.64	0.68	1.43	1.07	2.81	1.25	0.62
Galaxybase, SF100	0.54	0.55	0.53	0.67	0.64	0.64	0.64	0.62	0.64	0.63	0.65	0.60	0.60	0.61	0.61	0.60	2.73	1.38	4.81	0.97

management systems to demonstrate its effectiveness in uncovering novel choke points and offering valuable insights into system performance. The results indicate that FinBench can serve as a critical tool for researchers and practitioners aiming to enhance the efficiency and reliability of graph databases in the FinTech industry.

8 FUTURE WORK

The future development of this work encompasses system optimization and benchmark enhancement.

System optimization opportunities. The choke points highlighted in Section 3 are introduced to the benchmark to evaluate graph databases, which indicate optimization opportunities, such as TRUNCATION support in existing graph languages. With larger datasets and more complex queries emerging from the continuous iteration of this work, a broader range of tuning features and optimization techniques will need to be addressed, including distributed system configuration, advanced indexing techniques, and materialized views. These optimizations are critical for maintaining system performance and scalability as the workload grows.

Benchmark Enhancement. Truncation discussed in Section 3.4 may lead to inconsistencies in query results due to the arbitrary traversal direction during optimized query execution (e.g., when finding a path between vertices A and B, the traversal may expand from A to B or from B to A potentially yielding different results). To address this issue, a possible solution is to restrict truncation to a specific traversal direction and hop limit. Beyond this transaction workload, FinBench also aims to define an OLAP workload. As an OLAP workload example in financial scenarios, Geaflow [38] solves complex graph analysis and computation problems in high-throughput scenarios, such as subgraph matching, stream graph construction, and full graph analysis.

Automated Benchmarking. In this work, throughput measures high-frequency query volume. However, real-world applications require both high throughput and low latency. Maximizing throughput while adhering to a fixed latency constraint is crucial to accurately assessing system performance. Manually determining this optimal throughput is complex and time-consuming due to lengthy stability tests and variations in hardware performance. To address this, an automated benchmarking system that utilizing pre-execution parameter estimation and automatic tuning can efficiently determine the optimal throughput under varying conditions.

HTAP Systems Benchmarking. To eliminate the ETL process and facilitate real-time data analysis on transactional data, various database systems supporting Hybrid Transactional/Analytical Processing (HTAP) have emerged [31, 58]. This trend is also observed in the context of graph databases. Recent studies have begun to benchmark HTAP systems [57]. However, there currently exists no benchmark specifically tailored to graph databases with HTAP metrics. We aim to develop and support benchmarks for HTAP systems, incorporating features pertinent to graph databases.

ACKNOWLEDGMENTS

This work was supported by the Linked Data Benchmark Council (LDBC) [3, 45], which is an independent non-profit organization that aims to define standard benchmarks and foster a community around graph processing technologies, which consists of members from both industry and academia. The FinBench Transaction Workload was accomplished through the collaborative efforts of the LDBC members with their expertise and pushed forward by a task force [13] to make this work credible. We appreciate the support provided by the LDBC community in the process of defining this benchmark. The LDBC FinBench task force is formed by relevant actors, mainly from industry. All the participants contributed with their experience and expertise to make this benchmark credible. The task force members are listed as follows: AntGroup, CreateLink, Ultipa, StarGraph, Vesoft, Pometry, Katana, Intel, TigerGraph, Memgraph, and Koji Annoura (individual). In the development, we also received support and suggestions from fellows in the industry and academia outside the task force. We also thank the following people for their contributing: Peter Boncz, Malcolm Crowe, Parviz Peiravi, Zheng Wang, Bin Yang, Jiansong Zhang, Changyuan Wang, Yiping Xiong, Mingxi Wu.

A CHOKE POINTS

Here are some novel choke points found in financial scenarios.

CP 1: Intra-query Parallelization on Hub Vertex

When traversing on the hub vertex, the number of edges is unevenly distributed beyond estimation based on the degree distribution of the graph. This choke point tests the query optimizer to automate the intra-query parallelization when traversing on the hub vertex to speed up.

CP 2: Write Operation Contention and Conflicts

The read-write query is expected to execute inside a transaction. The transaction is a potential write after a long read. It means a long-time write transaction that holds write locks longer than expected. This may results in contention and conflicts between write operations to the same datum.

CP 3: Intermediate Result Propagation

When calculating some final share or final ratio values, a common pattern is to calculate the value in the current hop based on the value in the last hop, which is similar to propagation. The intermediate results can be cached for the next hop to ensure query efficiency.

CP 4: Temporal Access Locality and Performance

When filtering edges in a temporal graph, the query performance with temporal window filters can be improved when the edges are sorted by timestamp in storage, which optimizes the data access locality for timestamps.

CP 5: Hub Vertex Storage Balance

Especially in distributed systems, hub vertices mean bigger data units, e.g., sharding, which may need to split to balance the storage, load, and inter-shard communication.

CP 6: Multiplicity Support in Graph Model

Edge multiplicity requires that systems support multiple edges between the same vertex pair. Another dimension is required to annotate the edge id.

CP 7: Concise Temporal Window Expression

Temporal window filtering is a common expression when filtering edges in the navigational pattern to bound the query for expected data. It is verbose to express a temporal window by adding timestamp filtering clauses on each vertex and edge. A more concise expression is desired. A possible solution is adding keywords like *RANGE_SLICE*, *LEFT_SLICE* and *RIGHT_SLICE* referring to an extension of *Cypher* [26].

CP 8: Recursive Path Filtering Pattern

When tracing a fund flow, it is expected to find a path with recursive filters. For example, filters are expected to assume a path A-[e_1]-> B -[e_2]-> ... -> X.

- The timestamp order: $e_1 < e_2 < \ldots < e_i$
- The amount order: $e_1 > e_2 > \ldots > e_i$
- The time window: $e_{i-1} < e_i < e_{i-1} + \vec{\Delta}$, $\vec{\Delta}$ is a given constant.

Such queries that require *all timestamps in the transfer trace are in ascending order* or the *upstream* edge are difficult to explain in plain Cypher (or GQL [16] or SQL/PGQ [17]) because they require support for the category of queries *regular expression with memory* as described in [32]. Another possible solution is adding keywords like *SEQUENTIAL* and *DELTA* referring to an extension of *Cypher*.

CP 9: Traversal Limit Pattern

When traversing the hub vertex, the intermediate result may experience exponential growth. When the performance is not enough to satisfy the queries on the hub vertex, a language feature is needed so that the number of edges traversed out from the hub vertex can be limited.

B DATA STATISTICS

This table presents the vertices and edges count of each type in the FinBench datasets on each Scale Factor.

Table 7: The number of entities per SF in FinBench datasets. To derive these numbers, 100% of the network was generated as an initial bulk data set with no updates. Notation – C: entity category, V: vertex, E: edge.

C	File	SF0.1	SF0.3	SF1	SF3	SF10	SF30	SF100
V	person	1 000	3 000	10 000	30 000	100 000	300 000	1 000 000
E	personOwnAccount	2 201	6 551	21 880	65 900	219 164	658 171	2 195 916
E	personApplyLoan	1 330	3 940	13 250	39 714	132 080	396 244	1 322 867
E	personGuarantee	799	2 349	7 871	23 884	79 839	239 847	800 162
E	personInvest	2 941	8 927	29 998	89 958	300 114	900 439	3 001 011
V	company	1 000	3 000	10 000	30 000	100 000	300 000	1 000 000
E	companyOwnAccount	2 207	6 596	22 048	66 025	220 007	660 329	2 203 087
E	companyApplyLoan	1 336	4 022	13 369	40 039	132 554	396 394	1 321 058
E	companyGuarantee	752	2 376	7 906	23 976	79 839	239 714	799 164
E	companyInvest	3 034	8 963	30 193	90 041	299 749	900 262	3 001 730
V	account	4 408	13 247	43 928	131 925	439 171	1 318 500	4 399 003
E	transfer	16 105	49 512	164 433	483 262	1 823 676	5 496 824	18 919 028
E	withdraw	33 317	101 144	332 828	996 004	3 331 896	10 005 719	33 360 249
V	loan	2 666	7 962	26 619	79 753	264 634	792 638	2 643 925
E	loantransfer	8 148	24 144	80 942	242 262	805 802	2 414 134	8 057 740
E	deposit	8 516	25 505	85 415	257 081	852 754	2 551 090	8 507 512
E	repay	6 018	17 964	60 086	179 995	595 683	1 785 011	5 962 923
V	medium	2 000	6 000	20 000	60 000	200 000	600 000	2 000 000
E	signIn	9 240	27 222	90 807	361 742	1 491 643	4 562 637	15 455 625

C RELATED REPOSITORIES

The source code of this specification and the benchmark suite are available on Github:

- LDBC FinBench specification: https://github.com/ldbc/ldbc_finbench_docs
- LDBC FinBench data generator and parameters generator: https://github.com/ldbc/ldbc_finbench_datagen
- LDBC FinBench driver: https://github.com/ldbc/ldbc_finbench_driver
- Transaction workload reference implementations: https://github.com/ldbc/ldbc_finbench_transaction_impls

REFERENCES

[1] Yazeed Alabdulkarim, Sumita Barahmand, and Shahram Ghandeharizadeh. 2018. BG: A Scalable Benchmark for Interactive Social Networking Actions. *Future Generation Computer Systems* 85 (2018), 29–38.

[2] Renzo Angles, János Benjamin Antal, Alex Averbuch, Altan Birler, Peter Boncz, Márton Búr, Orri Erling, Andrey Gubichev, Vlad Haprian, Moritz Kaufmann, Josep Lluís Larriba Pey, Norbert Martínez, József Marton, Marcus Paradies, Minh-Duc Pham, Arnau Prat-Pérez, David Püroja, Mirko Spasić, Benjamin A. Steer, Dávid Szakállas, Gábor Szárnyas, Jack Waudby, Mingxi Wu, and Yuchen Zhang. 2024. The LDBC Social Network Benchmark. arXiv:2001.02299 [cs.DB] Retrieved June 17, 2025 from https://arxiv.org/abs/2001.02299

[3] Renzo Angles, Peter Boncz, Josep Larriba-Pey, Irini Fundulaki, Thomas Neumann, Orri Erling, Peter Neubauer, Norbert Martinez-Bazan, Venelin Kotsev, and Ioan Toma. 2014. The linked data benchmark council: a graph and RDF industry benchmarking effort. *SIGMOD Rec.* 43, 1 (May 2014), 27–31. https://doi.org/10.1145/2627692.2627697

[4] Timothy G Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. 2013. Linkbench: A Database Benchmark Based on the Facebook Social Graph. In *Proceedings of the 2013 ACM International Conference on Management of Data (SIGMOD)*. 1185–1196.

[5] BenchCouncil. 2014. *BigDataBench: A Big Data Benchmark Suite*. Retrieved June 17, 2025 from https://www.benchcouncil.org/BigDataBench

[6] Peter Boncz, Thomas Neumann, and Orri Erling. 2013. TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In *Revised Selected Papers of the 5th TPC Technology Conference on Performance Characterization and Benchmarking - Volume 8391*. Springer-Verlag, Berlin, Heidelberg, 61–76. https://doi.org/10.1007/978-3-319-04936-6_5

[7] Audrey Cheng, Xiao Shi, Aaron Kabcenell, Shilpa Lawande, Hamza Qadeer, Jason Chan, Harrison Tin, Ryan Zhao, Peter Bailis, Mahesh Balakrishnan, et al. 2022. TAOBench: An End-to-end Benchmark for Social Network Workloads. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1965–1977.

[8] Linked Data Benchmark Council. 2022. LDBC FinBench. Retrieved June 17, 2025 from https://ldbcouncil.org/benchmarks/finbench/

[9] Malcolm Crowe and Fritz Laux. 2024. Implementing the draft Graph Query Language Standard. arXiv:2407.09566 [cs.DB] Retrieved June 17, 2025 from https://arxiv.org/abs/2407.09566

[10] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. 2020. Quantifying TPC-H choke points and their optimizations. *Proc. VLDB Endow.* 13, 8 (April 2020), 1206–1220. https://doi.org/10.14778/3389133.3389138

[11] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *Proceedings of the 2015 ACM International Conference on Management of Data (SIGMOD)*. 619–630.

[12] Facebook. 2025. Unable to send or accept a friend request on Facebook. Retrieved June 17, 2025 from https://www.facebook.com/help/211926158839933

[13] finbench. 2022. finbench task force. Retrieved June 17, 2025 from https://ldbcouncil.org/benchmarks/finbench/ldbc-finbench-work-charter.pdf

[14] finbench. 2023. finbench acid test tool. Retrieved June 17, 2025 from https://github.com/ldbc/ldbc_finbench_acid

[15] finbench. 2023. finbench reference implementation. Retrieved June 17, 2025 from https://github.com/ldbc/ldbc_finbench_transaction_impls

[16] International Organization for Standardization. 2016. ISO/IEC 7816-6:2016. Retrieved June 17, 2025 from https://www.iso.org/standard/76120.html

[17] International Organization for Standardization. 2016. ISO/IEC 9075-1:2023. Retrieved June 17, 2025 from https://www.iso.org/standard/76583.html

[18] Zhisong Fu, Zhengwei Wu, Houyi Li, Yize Li, Min Wu, Xiaojie Chen, Xiaomeng Ye, Benquan Yu, and Xi Hu. 2017. GeaBase: A High-Performance Distributed Graph Database for Industry-Scale Applications. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*. 170–175. https://doi.org/10.1109/CBD.2017.37

[19] Gartner. 2021. Gartner Identifies Top 10 Data and Analytics Technology Trends for 2021. Retrieved June 17, 2025 from https://www.gartner.com/en/newsroom/press-releases/2021-03-16-gartner-identifies-top-10-data-and-analytics-technologies-trends-for-2021

[20] Jim Gray. 1993. Database and Transaction Processing Performance Handbook. In *The Benchmark Handbook*. Retrieved June 17, 2025 from https://api.semanticscholar.org/CorpusID:5341081

[21] Jim Gray and Franco Putzolu. 1987. The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. *SIGMOD Rec.* 16, 3 (Dec. 1987), 395–398. https://doi.org/10.1145/38714.38755

[22] Alastair Green, Martin Junghanns, Max Kießling, Tobias Lindaaker, Stefan Plantikow, and Petra Selmer. 2018. openCypher: New Directions in Property Graph Querying.. In *EDBT*. 520–523.

[23] Ant Group. [n.d.]. Ant Group Official Website. Retrieved June 17, 2025 from https://www.antgroup.com

[24] Andrey Gubichev and Peter Boncz. 2014. Parameter Curation for Benchmark Queries. In *Performance Characterization and Benchmarking: Traditional to Big Data - 6th TPC Technology Conference, TPCTC 2014, Revised Selected Papers*, Vol. 8904. Springer/Verlag, 113–129. https://doi.org/10.1007/978-3-319-15350-6_8 6th TPC Technology Conference on Performance Evaluation and Benchmarking, TPCTC 2014 held in conjunction with 40th International Conference on Very Large Data Bases, VLDB 2014 ; Conference date: 01-09-2014 Through 05-09-2014.

[25] Karl Huppler. 2009. *The Art of Building a Good Benchmark*. Springer-Verlag, Berlin, Heidelberg, 18–30. https://doi.org/10.1007/978-3-642-10424-4_3

[26] INRIA. 2023. T-CYPHER: A Temporal Query Platform for Evolving Graphs. https://project.inria.fr/tcypher/demo/. Accessed: 2023-10-20.

[27] Alexandru Iosup, Tim Hegeman, Wing Lung Ngai, Stijn Heldens, Arnau Prat-Pérez, Thomas Manhardto, Hassan Chafio, Mihai Capotă, Narayanan Sundaram, Michael Anderson, et al. 2016. LDBC Graphalytics: A Benchmark for Large-scale Graph Analysis on Parallel and Distributed Platforms. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1317–1328.

[28] Venelin Kotsev, Nikos Minadakis, Vassilis Papakonstantinou, Orri Erling, Irini Fundulaki, and Atanas Kiryakov. 2016. Benchmarking RDF Query Engines: The LDBC Semantic Publishing Benchmark.. In *BLINK@ ISWC*. 1–16.

[29] Kuzeko. 2023. Graph Databases Testsuite, Version 2. Retrieved June 17, 2025 from https://github.com/kuzeko/graph-databases-testsuite/tree/V2 Accessed: 2023-10-17.

[30] PKU Data Management Lab. 2023. gStore: A Graph Database for RDF Data. Retrieved June 17, 2025 from https://github.com/pkumod/gStore Accessed: 2023-10-17.

[31] Guoliang Li and Chao Zhang. 2022. HTAP Databases: What is New and What is Next. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2483–2488. https://doi.org/10.1145/3514221.3522565

[32] Leonid Libkin and Domagoj Vrgoč. 2012. Regular path queries on graphs with data. In *Proceedings of the 15th International Conference on Database Theory* (Berlin, Germany) *(ICDT '12)*. Association for Computing Machinery, New York, NY, USA, 74–85. https://doi.org/10.1145/2274576.2274585

[33] Heng Lin, Zhiyong Wang, Shipeng Qi, Xiaowei Zhu, Chuntao Hong, Wenguang Chen, and Yingwei Luo. 2023. Building a High-Performance Graph Storage on Top of Tree-Structured Key-Value Stores. *Big Data Mining and Analytics* 7, 1 (2023), 156–170.

[34] Matteo Lissandrini, Martin Brugnara, and Yannis Velegrakis. 2018. Beyond macrobenchmarks: microbenchmark-based graph database evaluation. *Proc. VLDB Endow.* 12, 4 (Dec. 2018), 390–403. https://doi.org/10.14778/3297753.3297759

[35] McKinsey & Company. 2024. Global Payments in 2024: Simpler Interfaces, Complex Reality. (2024). Retrieved June 17, 2025 from https://www.mckinsey.com/industries/financial-services/our-insights/global-payments-in-2024-simpler-interfaces-complex-reality Accessed: 2024-10-28.

[36] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS.. In *VLDB*, Vol. 6. 1049–1058.

[37] Neo4j Inc. 2021. Graph Technology for Financial Services: How Top Financial Firms Harness Connected Data to Increase Their Bottom Line. Retrieved June 17, 2025 from https://neo4j.com/whitepapers/financial-services-neo4j

[38] Zhenxuan Pan, Tao Wu, Qingwen Zhao, Qiang Zhou, Zhiwei Peng, Jiefeng Li, Qi Zhang, Guanyu Feng, and Xiaowei Zhu. 2023. GeaFlow: A Graph Extended and Accelerated Dataflow System. *Proc. ACM Manag. Data* 1, 2, Article 191 (June 2023), 27 pages. https://doi.org/10.1145/3589771

[39] Peyton Z Peebles Jr. 2001. *Probability, random variables, and random signal principles*. McGraw-Hill.

[40] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)* 34, 3 (2009), 1–45.

[41] Meikel Poess and Chris Floyd. 2000. New TPC Benchmarks for Decision Support and Web Commerce. *ACM Sigmod Record* 29, 4 (2000), 64–71.

[42] Meikel Poess, Bryan Smith, Lubor Kollar, and Paul Larson. 2002. TPC-DS: Taking Decision Support Benchmarking to the Next Level. In *Proceedings of the 2002 ACM International Conference on Management of Data (SIGMOD)*. 582–587.

[43] snb. 2025. snb datagen hadoop. Retrieved June 17, 2025 from https://github.com/ldbc/ldbc_snb_datagen_hadoop

[44] Gábor Szárnyas, Jack Waudby, Benjamin A Steer, Dávid Szakállas, Altan Birler, Mingxi Wu, Yuchen Zhang, and Peter Boncz. 2022. The LDBC Social Network Benchmark: Business Intelligence Workload. *Proceedings of the VLDB Endowment* 16, 4 (2022), 877–890.

[45] Gábor Szárnyas, Brad Bebee, Altan Birler, Alin Deutsch, George Fletcher, Henry A. Gabb, Denise Gosnell, Alastair Green, Zhihui Guo, Keith W. Hare, Jan Hidders, Alexandru Iosup, Atanas Kiryakov, Tomas Kovatchev, Xinsheng Li, Leonid Libkin, Heng Lin, Xiaojian Luo, Arnau Prat-Pérez, David Püroja, Shipeng Qi, Oskar van Rest, Benjamin A. Steer, Dávid Szakállas, Bing Tong, Jack Waudby, Mingxi Wu, Bin Yang, Wenyuan Yu, Chen Zhang, Jason Zhang, Yan Zhou, and Peter Boncz. 2024. The Linked Data Benchmark Council (LDBC): Driving competition and collaboration in the graph data management space. arXiv:2307.04350 [cs.DB] Retrieved June 17, 2025 from https://arxiv.org/abs/2307.04350

[46] Tugraph Team. 2021. Tugraph Technology Case Study. Retrieved June 17, 2025 from https://tugraph.tech/case/?lang=en-US
[47] TuGraph team. 2022. TuGraph-DB Source Code. Retrieved June 17, 2025 from https://github.com/TuGraph-family/tugraph-db
[48] PayPal Tech. 2021. How PayPal Uses Real-Time Graph Database and Graph Analysis to Fight Fraud. *Medium* (2021). Retrieved June 17, 2025 from https://medium.com/paypal-tech/how-paypal-uses-real-time-graph-database-and-graph-analysis-to-fight-fraud-96a2b918619a
[49] Yuanyuan Tian. 2023. The World of Graph Databases from An Industry Perspective. *ACM SIGMOD Record* 51, 4 (2023), 60–67.
[50] TigerGraph. 2021. JPMorgan Chase Inducted into the Hall of Innovation. Retrieved June 17, 2025 from https://www.tigergraph.com/press-article/jpmorgan-chase-hall-of-innovation/
[51] Bing Tong, Yan Zhou, Chen Zhang, Jianheng Tang, Jing Tang, Leihong Yang, Qiye Li, Manwu Lin, Zhongxin Bao, Jia Li, et al. 2024. Galaxybase: A High Performance Native Distributed Graph Database for HTAP. *Proc. VLDB Endow* 17, 12 (2024), 3893–3905.
[52] Transaction Processing Performance Council (TPC). 2010. *TPC Benchmark C Standard Specification Revision 5.11.* Retrieved June 17, 2025 from https://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf
[53] Transaction Processing Performance Council (TPC). 2021. *TPC Benchmark DS Standard Specification Version 3.2.0.* Retrieved June 17, 2025 from https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf
[54] Transaction Processing Performance Council (TPC). 2022. *TPC Benchmark H (Decision Support) Standard Specification Revision 3.0.1.* Retrieved June 17, 2025 from https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf
[55] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. 2011. The Anatomy of the Facebook Social Graph. arXiv:1111.4503 [cs.SI] Retrieved June 17, 2025 from https://arxiv.org/abs/1111.4503
[56] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. 2014. Bigdatabench: A Big Data Benchmark Suite from Internet Services. In *Proceedings of IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 488–499.
[57] Chao Zhang, Guoliang Li, and Tao Lv. 2024. HyBench: A New Benchmark for HTAP Databases. *Proc. VLDB Endow.* 17, 5 (May 2024), 939–951. https://doi.org/10.14778/3641204.3641206
[58] Chao Zhang, Guoliang Li, Jintao Zhang, Xinning Zhang, and Jianhua Feng. 2024. HTAP Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (Nov. 2024), 6410–6429. https://doi.org/10.1109/tkde.2024.3389693