The LDBC Social Network Benchmark Interactive Workload v2: A Transactional Graph Query Benchmark with Deep Delete Operations

David Püroja[1], Jack Waudby[2], Peter Boncz[1], and Gábor Szárnyas[1]

[1] CWI, the Netherlands, [2] Newcastle University, School of Computing
david.puroja@ldbcouncil.org, j.waudby2@newcastle.ac.uk, boncz@cwi.nl,
gabor.szarnyas@ldbcouncil.org

Abstract. The LDBC Social Network Benchmark's Interactive workload captures an OLTP scenario operating on a correlated social network graph. It consists of complex graph queries executed concurrently with a stream of updates operation. Since its initial release in 2015, the Interactive workload has become the de facto industry standard for benchmarking transactional graph data management systems. As graph systems have matured and the community's understanding of graph processing features has evolved, we initiated the renewal of this benchmark. This paper describes the draft Interactive v2 workload with several new features: delete operations, a cheapest path-finding query, support for larger data sets, and a novel temporal parameter curation algorithm that ensures stable runtimes for path queries.

1 Introduction

LDBC. The Linked Data Benchmark Council (LDBC)[1] is a non-profit organization dedicated to designing benchmarks for graph data management [20,21]. LDBC has strong industrial participation in the form of 21 companies, including database, hardware, and cloud vendors. Its membership also includes 3 non-commercial institutions and 60+ individual members. LDBC acts as an independent authority for benchmarks and oversees the use of its benchmarks through a stringent auditing process. Thanks to this, audited LDBC benchmark results allow quantitative and objective comparison of different technological solutions, which is expected to stimulate progress through competition. Next, we describe the two main workloads of the LDBC SNB suite.

SNB Interactive v1 workload. The LDBC Social Network Benchmark (SNB) Interactive v1 workload was published in 2015 [7]. It is a transactional benchmark that targets OLTP data management systems with graph features (e.g. path-finding). SNB Interactive has been influential in the graph data management space: as of August 2023, 24 audited results were published using this workload.[2]

---

[1] https://ldbcouncil.org

[2] https://ldbcouncil.org/benchmarks/snb-interactive/

SNB Business Intelligence workload. The LDBC SNB Business Intelligence (BI) workload was released in 2022 [24]. This workload uses an improved data generator, which introduces support for delete operations [28] and scale factors up to SF30,000. The workload captures an OLAP scenario with heavy-hitting analytical queries that touch on large portions of the graph (e.g. Messages created within a 100-day period or Persons living in China) and applies daily batches of updates. It targets both DBMSs and data analytical systems such as Spark.

Motivation. As of 2023, more than 8 years passed since the SNB Interactive v1 workload's release. Therefore, we decided to renew it to ensure its continued relevance. The new version's key novel features are improved scalability, coverage of cheapest path-finding, and inclusion of delete operations. The first two new features were part of the natural evolution of the benchmark. The decision to support delete operations was motivated by a number of factors. On the technical side, running the workload's complex queries efficiently while applying delete operations assumes mature transaction support that is now expected by users of graph(-capable) DBMSs. Deletes also make certain graph algorithms, such as cheapest paths, more difficult to compute incrementally [19], thus limiting the effects of caching and incremental view maintenance. On the business side, supporting deletes is necessitated by law in several jurisdictions, exemplified by the EU's General Data Protection Regulation (GDPR) [22].
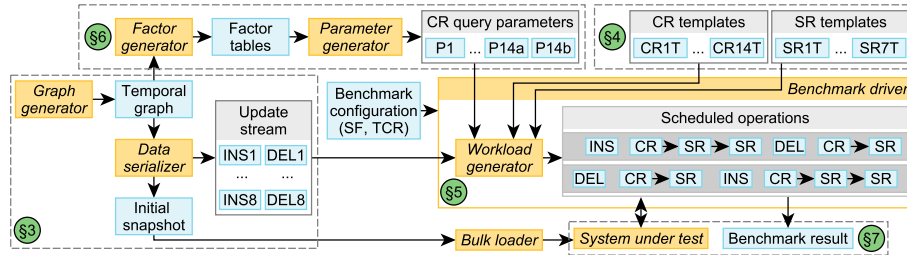

Fig. 1: Components and workflow of the Interactive v2 workload. The corresponding sections are shown in green circles §. Legend: Software component, Data artifact

Contributions and paper structure. This paper presents the updated SNB Interactive v2 workload following the workflow shown in Figure 1. Interestingly, while all three new features (scalability, cheapest path-finding, and delete operations) were already supported in the BI workload, adopting them into the highly transactional, concurrent SNB Interactive v2 workload presented several complex technical challenges. We document the challenges and our key design principles in Section 2. We present the SNB data set in Section 3, the benchmark's operations in Section 4, and the workload in Section 5. In Section 6, we introduce the parameter generator's novel extensions that were necessitated by delete operations. In Section 7, we discuss how the benchmark is used in practice. We discuss LDBC's other transactional benchmark, the FinBench, in Section 8. Section 9 summarizes our contributions and outlines future directions.

2 Design principles

During LDBC's benchmark design process, we follow the Benchmark Handbook [9], which prescribes four criteria for domain-specific benchmarks: (1) *relevance*, (2) *portability*, (3) *scalability*, (4) *simplicity*. In the following, we discuss the SNB Interactive v2 workload's approach for complying with these criteria.

2.1 Relevance: Choke point-based design process and domain

> The benchmark must measure the peak performance of systems when performing typical operations within the target problem domain.

Choke points. To ensure *relevance*, LDBC's benchmark design process uses *choke points* [4], i.e. technical difficulties that are known to be challenging for the present generation of DBMSs. Choke points are identified by expert data systems architects and are also influenced by the input from users of graph data management systems who contribute their use cases at LDBC's Technical User Community meetings[3]. The initial choke points of SNB Interactive were based on the influential TPC-H benchmark [25] benchmark and were later extended with choke points that target graph-specific features such as cardinality estimation for paths and the execution of path-finding queries. LDBC workloads are designed using an iterative process to ensure full coverage of the choke points required for a given workload category.

Social network domain. The LDBC SNB uses the *social network* domain because its concepts (Person, Forum, Message, etc.) are well-understood. Moreover, the social network domain makes it easy to reason about some of the interesting phenomena captured in the choke points. For example, the power law distribution and correlations (Section 3.2) observable in real-life (social) networks trigger the challenges for cardinality estimation.

2.2 Portability: Implementation rules

> The benchmark should be implementable on different systems/architectures.

The SNB Interactive v2 workload guarantees *portability* by taking an agnostic stance on implementation details.

Data model. Implementations are allowed to use any data model, including the property graph, RDF, and relational models. They are also free to choose their input format for bulk loading (e.g. CSV, N-Triples).

Implementation language. Implementations may use declarative query languages (SQL, Cypher [8], GQL, SQL/PGQ [6], etc.) or general-purpose imperative programming languages (C++, Java, etc.). However, results in these two categories are ranked on separate leaderboards as the latter systems have a significant advantage due to their use of hand-coded highly-optimized query plans.

Setup. There are no restrictions on the operating system, hardware architecture, or number of machines used (both single-node and distributed setups are allowed).

---

[3] https://ldbcouncil.org/tags/tuc-meeting/

2.3 Scalability: Scalable data generator and driver

> The benchmark should apply to small and large computer systems.

Improving *scalability* was a key goal during the design of SNB Interactive v2. While we could leverage the improved data generator of the BI workload [24], scaling the *workload execution* posed additional challenges. By its nature, simulating a transactional database workload requires highly concurrent execution of the operations.[4] This requires the operations in the workload to be partitioned, which is a major challenge as most of the Persons in the social network belong to a single connected component that does not lend itself to any naïve partitioning strategy. Moreover, update operations often have long dependency chains that need to be tracked, e.g. a friendship can only be deleted if it already exists, the creation of a friendship requires both Persons to exist, etc. Therefore, simulating a transactional graph processing scenario is not possible using on-the-fly workload generation techniques commonly employed in database benchmarks. Instead, SNB Interactive v2 requires extensive offline data, update stream (Section 3.3), and parameter generation steps (Section 6) prior to the benchmark.

2.4 Simplicity: Stable query runtimes, single output metric

> The benchmark must be understandable and its results must be easy to interpret, otherwise, it lacks credibility.

Stable runtimes. To make the benchmark results easy to interpret, it is desirable that instances of a given query type have similar expected runtimes (referred to as *stable runtimes*). Ensuring this for graph workloads is non-trivial due to the highly skewed distribution exhibited in real-world networks [16]. For example, in a social network, a few Person nodes have a very large number of edges while others only have a few connections. This has a significant impact on runtimes: if query parameters are selected using uniform random sampling, query runtimes will be unstable, often exhibiting a multimodal distribution that spreads across many orders of magnitude and has several outliers [12,24]. SNB Interactive v2 employs a sophisticated *parameter curation* process to select input parameters that ensure stable runtimes (Section 6).

Guaranteed executability. Stable runtimes also necessitate that operations are *executable* at their scheduled start time. For example, if an operation targets entities that do not yet exist or were already deleted, the operation becomes trivial or results in a runtime exception, compromising stable runtimes. Therefore, our workload generator ensures that its operations are always executable.

Single metric. The result of an SNB Interactive benchmark run is characterized by a single metric, *throughput* (operations/second), which captures the system-under-test's end-to-end performance on a transactional graph workload.

---

[4] Most audited Interactive v1 implementations use 48 read and 32/64 write threads.

3 Data sets

The LDBC SNB workloads include a scalable distributed data generator based on Spark.[5] Here, we give an overview of the data sets used in the benchmark.
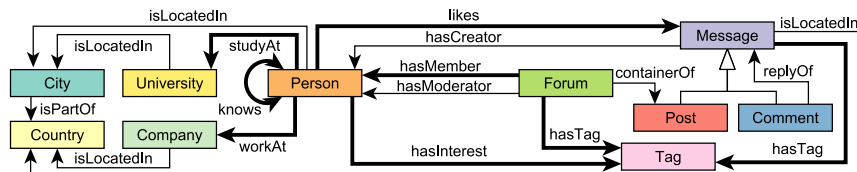
3.1 Graph schema

Fig. 2: A subset of the LDBC SNB graph schema visualized using a UML-like notation. Thick lines denote many-to-many relationships.

The graph schema of LDBC SNB has 14 node types connected by 20 edge types. The data set consists of a Person–knows–Person (friendship) graph and a number of Message threads within Forums. The root of a Message thread is a Post and the rest of the thread consists of Comments. All Messages are connected to Persons by creatorship and likes edges. A simplified schema is shown in Figure 2.

3.2 Distribution and correlations

The data set contains two types of graph-shaped data structures. First, the Message threads form trees and constitute the majority of the data. Second, the Person–knows–Person subgraph is a network with many-to-many relationships whose distribution is modelled after Facebook [26] with the social graph exhibiting the small-world phenomenon [27] characterized by a small diameter.

A unique feature not observed in other data generators is that the attribute distributions are skewed and correlate both within an entity (e.g. people living in France have predominantly French names). Moreover, the graph has structural correlations: following the *homophily principle* [15], people are more likely to be friends if they studied at the same University at the same time, live in close proximity, and/or have the same interests. These correlations are exploited by the workload to stress choke points for querying correlated data (see Section 6.4).

3.3 Graph generation stages

Temporal graph. The data generator first produces a *temporal graph*, which contains all entities that exist at some point in the simulated social network's 3-year time period, i.e. between Jan 1, 2010 and Dec 31, 2012. During this time, entities are inserted and deleted in the network, and the timestamps of

---

[5] https://github.com/ldbc/ldbc_snb_datagen_spark

these events are captured according to their time of occurrence in the *simulation time*. The insertion and deletion of entities follow realistic time intervals and conform to the semantics of the social network. Namely: (1) The deletion dates of Persons are based on the statistics collected from the collapse of a real-world social network [13]. When a Person is deleted, the content they created is also deleted [29]. (2) The network contains infrequent flashmob events such as spikes in insertions of Messages for a given Tag [7]. Deep delete operations and flashmob events are unique to the LDBC SNB data generator: according to a recent survey [5], these features are not supported by any other (graph) data generator. **Initial snapshot and update stream.** As the second step in the data generation, the *data serializer* splits the temporal graph into two parts by setting a *cutoff date* at 97% of the simulation time (Nov 29, 2012). The entities created before the cutoff date form the *initial snapshot*, while the entity creations and deletions occurring after the cutoff date form the *update stream*.

3.4 Scale factors

Table 1: SNB Interactive v2 data sets. *k:* thousand, *M:* million, *B:* billion.

Scale Factor (SF)	10	30	100	300	1,000	3,000	10,000	30,000
#nodes	27M	78M	255M	738M	2.4B	7.2B	23B	82.76B
#edges	170M	506M	1.7B	5.1B	17B	51.9B	173B	340.5B
#Person nodes	68k	170k	473k	1.2M	3.4M	9M	26M	77M
#knows edges	1.8M	5.5M	19M	55.7M	187M	559M	1.9B	6.8B
#insert operations	44.6M	127M	399M	1.1B	3.3B	8.9B	27B	76.7B
#delete operations	353k	1M	3.3M	9.3M	28.9M	79.7M	245M	721.8M

The data generator produces dynamic social network graphs in different sizes, characterized by *scale factors* (SF) which correspond to the data set's disk usage when serialized in CSV (comma-separated values) format, measured in GiB. The data generator used for Interactive v1 only supports data sets up to SF1,000. To improve scalability, Interactive v2 uses the new Spark-based generator, which was optimized extensively,[6] allowing it to scale up to SF30,000. Table 1 shows the main statistics of the data sets.

4 Operations

The LDBC SNB Interactive v2 workload uses four types of operations. There are 14 complex (`CR`) and 7 short read queries (`SR`). Update operations include 8 inserts (`INS`) and, newly introduced in the Interactive v2 workload, 8 deletes (`DEL`). The workload mix consists of approximately 8% `CR`, 72% `SR`, 20% `INS`, and 0.2% `DEL` operations. In this section, we describe the four operation types using

---

[6] For details on the optimization steps, see `https://ldbcouncil.org/tags/datagen/`.

examples. We also give ranges on how long operations are expected to take in state-of-the-art systems (Section 7.2).

4.1 Complex read queries (CR)

Complex read queries CR1–CR12 discover a given Person's social environment (one-to three-hop neighbourhoods) and retrieve related content (Forums, Messages, etc.). Queries CR13 and CR14 perform path-finding between pairs of Persons. The runtimes of complex read queries are typically between 1 and 500 ms, making them feasible to compute interactively, in line with the workload's name.

CR3. For a given Person, find their *friends* and *friends of friends*, who created Messages in both Country $countryX and $countryY within a given time period. Only consider Persons that are foreign to both of those Countries. Return the number of their Messages per Country, xCount and yCount (Figure 3a).

CR13. Return the length of the (unweighted) shortest path between two Persons.

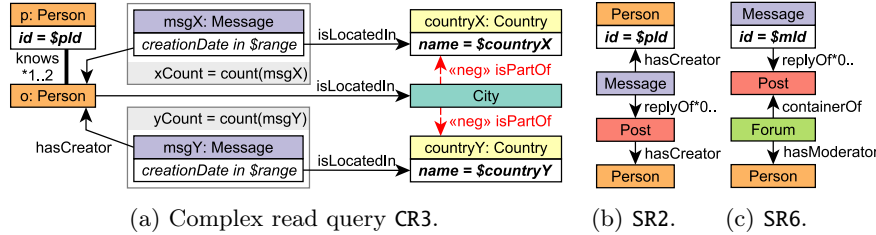(a) Complex read query CR3.   (b) SR2.   (c) SR6.

Fig. 3: Graph patterns of complex and short read queries.

Cheapest path-finding. While we strived to keep the changes to the queries minimal, we replaced CR14 due to two reasons. First, we found the original query in Interactive v1 to be ill-suited to the workload as it required the enumeration of *all shortest paths* between two Persons, which can be prohibitively expensive on large scale factors. Second, we introduced a new choke point, CP-7.6 *Cheapest path-finding,*[7] a key computational kernel and a language opportunity for GQL [6]. Therefore, we changed CR14 to use *cheapest paths* instead of *all shortest paths*.

CR14 (new). Given two Persons, find *any cheapest path* in the interaction sub-graph. This graph contains edges from the Person–knows–Person graph where the endpoint Persons have exchanged at least one Message (i.e. one Person created a direct Comment to a Message of the other Person). The weights of knows edges are integers defined as $\max(\text{round}(40 - \sqrt{numInteractions}), 1)$.

4.2 Short read queries (SR)

Short read queries perform local neighbourhood lookups on Persons and Messages. Most short read queries can be evaluated in 0.1 to 75 ms.

---

[7] The term *shortest paths* refers to the problem of finding *unweighted shortest paths*, which can be solved with the BFS algorithm. We use *cheapest paths* to refer to the *weighted shortest paths* problem which can be solved using e.g. Dijkstra's algorithm.

SR2. Given a start Person, retrieve their last 10 Messages. For each Message, return it with the root Post in its thread, and the author of that Post (Figure 3b).
SR6. Given a Message, retrieve its container Forum (directly for Posts, via the root Post for Comments) and the Person that moderates that Forum (Figure 3c).

4.3 Insert operations (`INS`)

Insert operations add new entities from the update stream to the graph. A typical insert operation takes between 0.1 and 100 ms.
INS5. Insert a hasMember edge between a Person and a Forum. The executability of this operation depends on the existence of its two endpoint nodes.
INS6. Insert a Post node. This operation's executability depends on two nodes: both the Person creating the Post and the Forum containing it must exist. When the Post is inserted, they are linked to it via hasCreator and containerOf edges.

4.4 Delete operations (`DEL`)

The Interactive v2 workload uses *deep cascading delete operations*. Cascading deletes capture the behaviour of real social networks where users expect their content to be removed once they delete their accounts. The technical reasons for requiring cascading delete operations are two-fold: (1) **Preventing dangling edges.** To maintain the integrity of the graph, it is required there are no dangling edges thus nodes must be always deleted with all their edges. To prevent dangling edges, most graph DBMSs support the automatic deletion of edges attached to a given node, e.g. through Cypher's `DETACH DELETE` clause [11]. To achieve the same effect, RDBMSs can make use of `FOREIGN KEY` constraints with the `ON DELETE CASCADE` clause. (2) **Testing triggered deletions.** Node deletions can trigger the deletion of other nodes (Figure 4). For example, according to the SNB schema's constraints, the deletion of a Post implies the deletion of all its descendant Comments along with their edges. Such deletions may be implemented using triggers, constraints (RDBMSs may again harness `FOREIGN KEY`s), or by formulating (potentially recursive) subqueries that determine which other nodes need to be deleted with `DELETE ... USING` clause.
**Choke points.** The coverage of delete features is ensured by three new choke points: CP-9.3 *Delete node* (stressed by 4 operations), CP-9.4 *Delete edge* (8 operations), and CP-9.5 *Delete recursively* (4 operations). In the following, we present two delete operations, both of which cover all new choke points.
DEL6. Remove a Post with all its edges and child Comments via `DEL7` (Figure 4a).
DEL7. Remove a Comment with all its edges and child Comments, which are deleted recursively by invoking `DEL7` (Figure 4b).

5 Workload scheduling and benchmark driver

In this section, we explain how operations are scheduled in the SNB Interactive workload, how the driver operates, and how the final *throughput* metric is deter-
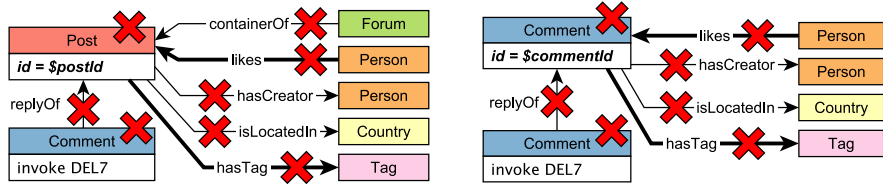
(a) Delete operation DEL6: Remove Post. (b) Delete operation DEL7: Remove Comment.

Fig. 4: Cascading delete operations in Interactive v2. Symbol ✖ denotes deletion.

mined. In all cases, we assume that the system-under-test has been populated with the *initial snapshot* using a *bulk loader* before the driver runs the operations.

5.1 Scheduling operations

TCR (total compression ratio). The scheduling follows the *simulation time* of the temporal social network graph. The user-provided *total compression ratio* (TCR) value controls the speed at which the simulation is replayed. For example, a TCR value of 0.02 means that the simulation is replayed 50× faster, i.e. for every 20 milliseconds in wall clock time, 1 second passes in the simulation time.
Update operations. The driver replays the update operations starting from the cutoff date (Section 3.3), Nov 29, 2012. The operations are scheduled according to the distance of their start time from this date, adjusted by the TCR. They are then used to set the cadence of the schedule for the complex reads and, in turn, the short read queries, as we will explain momentarily.
Complex read queries. The *complex read queries* differ significantly in their expected runtimes as they touch on different amounts of data. As each query instance contributes equally to the output metric,[8] we balance them such that each query type is expected to take the same amount of time to execute. For example, CR14 (new) is expected to be more difficult than CR13, therefore it is scheduled less frequently. Frequencies vary based on the SF as the relative difficulties of queries change with the data size (e.g. three-hop neighbourhood queries grow faster on larger SFs than one-hop ones).
Short read queries. Short read queries are triggered by complex read queries and other short read queries, and use their output as their input. For example, both CR3 and CR14 trigger SR2, which also triggers itself. This mimics the real-life scenario of a user retrieving more information about Person profiles based on the result of the earlier queries. The mapping between complex and short read queries is given in the specification [2, Chapter 5].

5.2 Driver

Driver modes. The SNB driver has two key modes of operation. In *cross-validation mode*, it tests an implementation against the output of another implementation. To ensure deterministic results, operations in this mode are executed

---

[8] Unlike in TPC-H [25] and SNB BI [24], which use *geometric mean* in their metrics.

sequentially with no overlap between queries and updates. In *benchmark mode* the driver performs a benchmark run where queries and updates are issued concurrently from multiple threads. The run starts with a 30-minute warm-up period, followed by a 2-hour *measurement window*. This mode does not perform validation as query results may differ (slightly) due to concurrent updates.

Dependency tracking. To ensure that updates are executable, concurrent threads must be synchronized so that an operation is only executed when its dependencies exist in the network (e.g. two Persons can only become friends if both of them already exist). This is achieved via maintaining a global clock in the driver and performing *dependency tracking* for the updates [7]: each update operation has a timestamp denoting the creation time of the last operation it depends on. The data generator calculates these timestamp during generation and ensures that there is a minimum time separation, T_{safe}, between dependent entities to reduce synchronization overhead in the driver when executing operations. The driver then only needs to check every T_{safe} time whether a given update operation can be executed. By default, T_{safe} is set to 10 seconds in the simulation time.

Latency requirements. The workload simulates a highly transactional scenario where operations are subject to (soft) latency requirements. To incorporate this in the workload, it prescribes the *95% on-time requirement*: for a benchmark run to be successful, 95% of the operations must start *on-time*, i.e. within 1 second of their scheduled start time. Benchmark runs where the system-under-test falls behind too much from the schedule are considered invalid.

Throughput. The throughput of a run is the total number of operations (CR, SR, INS, DEL) executed per second. A lower TCR value implies a higher throughput.

Individual execution times. To facilitate deeper analyis, the benchmark driver also collects all individual query execution times. Based on these, the benchmark reports must include statics for each operation type (min, max, mean, P_{50}, P_{90}, P_{95}, and P_{99} of the execution times).

Driver implementation in v2. The Interactive v2 is implemented in Java 17. It consists of 26,500 lines of code for the core project and an additional 18,000 lines of test code. The new version contains several patches including bug fixes, usability improvements, and performance optimizations.[9]

6   Parameter curation

To prevent caching query results, the SNB Interactive v2 driver instantiates the parameterized complex read (CR) query templates with different *substitution parameters* (a.k.a. parameter bindings). However, as explained in Section 2.4, the naïve approach (using a uniform random sampling of parameters and ignoring updates) leads to unstable runtimes, which compromise both the benchmark's understandability and reproducibility. To ensure stable runtimes, LDBC invented *parameter curation* techniques, which select parameters that produce query runtimes with a unimodal (preferably Gaussian) distribution [12,24].

---

[9] `github.com/ldbc/ldbc_snb_interactive_driver/releases/tag/v2.0.0-RC2`

6.1 Building blocks for parameter curation

Temporal bucketing. To ensure that operations are always executable, i.e. they avoid targeting nodes that are yet to be inserted or ones that are already deleted, the parameter curation process in Interactive v2 employs *temporal bucketing*. Namely, we create a parameter bucket for *each day in the simulation time of the update streams*, i.e. each day in the simulation time has its own distinct set of parameters. This is a novel feature in Interactive v2 – previous SNB benchmarks lacked this feature and only selected parameters from the *initial snapshot*.

Factor tables. As shown in Figure 1, the parameter generation is a two-step process. The *factor generator* produces *factor tables*, which contain data cube-like summary statistics [10] of the temporal graph such as the number of Messages for friends. The factor generator is executed in a distributed setup using Spark as this computation includes expensive joins over large tables, e.g. knows(person, friend) ⋈ hasCreator(person, comment).

6.2 Parameter curation for relational queries

For relational queries (without path-finding), we based our parameter generation on two techniques.

(1) Selecting windows. To select the parameters that are expected to yield similar runtimes, we look for windows with the smallest variance for a given value using SQL window functions. The parameters are first sorted and grouped together based on their difference in frequency. Groups that are smaller than a given minimum threshold are discarded to select a group of parameters large enough to generate a sufficient amount of parameters. From the latter, we select the group with the smallest standard deviation.

(2) Selecting distributions. For queries where we want to select parameters that are correlated or anti-correlated, we use factor tables encoding possible combinations (e.g. countryPairsNumFriends for CR3): we select values near a high percentile for the correlated and a low percentile for the anti-correlated case.

Generating the parameters. The parameter candidates discovered by the previous approaches are stored in temporary tables. The parameter generation step uses these tables to select parameters for each day in the update stream.

6.3 Parameter curation for path-finding queries

The effect of deletes. A key distinguishing feature of graph data management systems is their first-class support for path queries [1]. We demonstrate why ensuring stable query runtimes for path queries is particularly challenging through the example of Figure 5a, where we query for the (unweighted) shortest path between *Ada* and *Bob* over a dynamic graph. Initially, at $t = 1$, the length of the shortest path is 4 hops. Then, the edge between *Carl* and *Dan* is deleted, making *Ada* and *Bob* unreachable from each other at $t = 2$. Finally, a new edge is inserted between *Carl* and *Bob*, yielding a shortest path of length 3 at $t = 3$. This illustrates how a given input parameter (a pair of Persons) can oscillate

between being reachable and being in disjoint connected components over a short period. To ensure stable query runtimes for path queries in the presence of inserts and deletes, Interactive v2 introduces a novel *path curation* algorithm, which produces pairs of Person nodes whose shortest path length from each other is guaranteed to be exactly k hops at any point during a given day.

Graph construction. The parameter curation algorithm builds two variants of the Person–knows–Person subgraph for each day based on the *temporal graph*: graph G_1 has the inserts applied until the beginning of the day and the deletes applied until the end of the day, while G_2 has the deletes applied until the beginning of the day and the inserts applied until the end of the day. For a given pair of Person nodes, their shortest path length in G_1 is an upper bound k_{upper} on their shortest path length at any point in the day – when the inserts during the day are gradually applied, the shortest path length can only become shorter. Conversely, G_2 gives a lower bound k_{lower} for the shortest path – the deletes can only make the shortest path length become longer.

Parameter selection. The bounds provided by G_1 and G_2 guarantee for the shortest path length k that $k_{\text{lower}} \leq k \leq k_{\text{upper}}$ will hold at any point during the day. We can ensure that k will stay constant during the day by selecting Person pairs where $k_{\text{lower}} = k_{\text{upper}}$ holds. To this end, we select pairs who are exactly 4 hops apart in both G_1 and G_2, hence they will be always 4 hops apart during the given day. Unreachable pairs of nodes can be generated by calculating the connected components of G_2 and selecting nodes from disjoint components. The path curation for both the reachable and the unreachable cases is implemented using the NetworKit graph algorithm library [23].
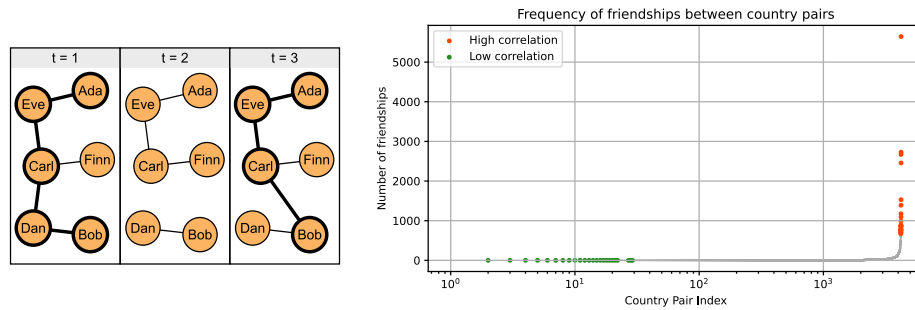
(a) Shortest path (denoted with thick lines) between *Ada* and *Bob* in the presence of updates.

(b) Pairs of Countries in the `countryPairsNumFriends` factor table representing the number of friendships between both Countries.

Fig. 5: Example graph and distribution for path curation.

6.4 Query variants

The new workload introduces variants for three queries: CR3, CR13, and CR14.

CR3: Correlated vs. anti-correlated Countries. We introduce variants for CR3 (Figure 3a): variant CR3(a) starts from Countries that have a high correlation

in the friendship network, while variant `CR3(b)` starts from `Countries` that have a low correlation of friendships between. To generate these inputs, we use the `countryPairsNumFriends` factor table visualized in Figure 5b and select values at percentile 1.00 for variant `(a)` and percentile 0.01 for variant `(b)`.

CR13 and CR14: Reachable vs. unreachable Persons. Path queries are expected to have different runtimes if there is a path vs. when there is no path. While the performance characteristics vary highly between systems, in principle, the "no path" case should be simpler in the SNB graph, where one of the nodes is always in a small connected component. To distinguish between these cases, we have two variants for the two path queries `CR13` and `CR14`. For variants `(a)` we select `Person` pairs which *do not have a path*, and for variants `(b)` we select pairs which *have* a path of length 4.

6.5 Parameter generator implementation

The parameter generator is implemented in Python using NetworKit [23] and SQL queries executed by DuckDB [18]. Based on our experiments in [17, Figure 4.3], the new parameter generator is scalable. Even with the significant extra work performed for temporal bucketing, it outperforms the old parameter generator by more than 100× on SF1,000, and finishes in less than 1.5 hours on SF10,000.

7 Using the SNB Interactive v2 workload

In Sections 3 to 6, we presented the components that make up the SNB Interactive v2 benchmark (Figure 1): its data sets, operations, driver, and parameter generator. We continue by describing how the benchmark is used, including its current implementations and considerations for auditing implementations.

7.1 Implementations

The portability of Interactive v2 is demonstrated by having four complete initial implementations[10] based on the Neo4j graph DBMS; and the Microsoft SQL Server, PostgreSQL, and Umbra RDBMSs. The Neo4j implementation uses the Cypher query language [8]. SQL Server uses the Transact-SQL language with the graph extension.[11] PostgreSQL and Umbra both use SQL's PostgreSQL dialect. All of these implementations passed cross-validation against each other.

7.2 Auditing

LDBC's benchmarks come with stringent auditing guidelines to ensure that they are implemented correctly and the results derived during benchmark runs are

---

[10] `https://github.com/ldbc/ldbc_snb_interactive_impls`

[11] `https://learn.microsoft.com/en-us/sql/relational-databases/graphs/`
`sql-graph-overview?view=sql-server-ver16`

reproducible. Audits are carried out by independent auditors who are certified by the benchmark task force. The auditor ensures that an implementation is compliant with the LDBC specification by performing a thorough code review, running ACID tests, and executing the benchmark. The results of audits are published as *full disclosure reports* and systems are ranked on the LDBC website according to their throughput.[12] In the following, we highlight important aspects of the SNB auditing guidelines such as rules for precomputation and ACID tests. For the detailed auditing guidelines, we refer the reader to the SNB specification [2, Chapter 7].

Precomputation. The auditing guidelines permit the use of precomputed auxiliary data structures (views, indexes, views, etc.) provided that they are always kept up-to-date upon update operations. A frequent use of precomputations is the creation of a rootPost edge for each Message, which points to the root Post of the Message's thread. Implementers may decide to store information redundantly, e.g. by adding a property to the Forum–hasMember–Person edge that contains the number of Posts in the Forum, for improved locality during query execution.

ACID tests. To ensure thorough testing of transactional guarantees, SNB Interactive v2 uses a separate ACID suite [29], which tests for 10 transactional anomalies. While Interactive v1 only requires systems to guarantee the *read committed* isolation level, the inclusion of delete operations necessitates *snapshot isolation* to ensure queries read a consistent database state. To illustrate this consider a graph with four nodes n_1, n_2, n_3, n_4, and three edges $n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_4$. Assume transaction T_a begins traversing from n_1, reading n_1, n_2, and n_3. Then, T_b deletes n_2 and commits. Then, T_c inserts n_5, connecting n_3 and n_4, $(n_3 \rightarrow n_5 \rightarrow n_4)$, and commits. T_a then reads n_5 and (incorrectly) concludes that n_4 is reachable from n_1 – when in fact at no point in time was this a valid database state. The ACID tests also include a *durability test:* during a benchmark run, the system-under-test is shut down abruptly and restarted afterward. The system is expected to guarantee durability, which is verified by the auditor who checks whether the last update operations issued by the driver are reflected in the database's state after recovery.

Full disclosure report (FDR). Audited benchmark results must be accompanied by a full disclosure report (FDR). The FDR documents the benchmark setup for reproducibility and contains the detailed results of the benchmark run (including statistics of the individual query runtimes).

8 Related work: LDBC FinBench

The LDBC FinBench (Financial Benchmark) targets distributed scale-out transactional graph database management systems. It is set in the financial domain and uses concepts such as Account, Loan, and transfer. Its data distribution follows the characteristics of the financial domain, where a hub vertex (e.g. a large e-commerce vendor) may have billions of edges. To make queries tractable

---

[12] https://ldbcouncil.org/benchmarks/snb-interactive/

on such vertices, the workload employs *truncation*, i.e. a traversal only uses a truncated set of edges, e.g. the 5,000 most recent edges. A requirement directly derived from the financial domain is having *strict latency requirements* for some queries, e.g. 99% of a given query's executions have to finish in 100 ms. The workload also includes path-finding queries that can be expressed as *regular queries with memory* [14].

9 Conclusion

Summary. In this paper, we summarized the LDBC SNB Interactive v1 workload and explained its shortcoming to motivate its renewal. We then presented the draft version of the Interactive v2 workload, which is expected to be very close to the final version of the workload. The new workload uses a data generator producing deep cascading delete operations, includes a completely reworked driver and workload scheduler, and a scalable parameter generator. We compared the workload against other benchmarks and highlighted its key novel features that allow the incorporation of delete operations while keeping important guarantees such as stable query runtimes. While the benchmark was substantially reworked, we made an effort to keep the user-facing changes minimal and only replaced a single read query, `CR14`. We believe users with an existing v1 implementation can adopt the new version with reasonable development cost and extend their experiments to use larger scale factors in a matter of days. Therefore, we expect users to quickly migrate to the new version upon its release.

Future work. As the next step in the Interactive v2 workload's development process, the SNB task force will finalize the workload, conduct *standard-establishing audits* on two reference implementations and submit the workload for acceptance by the LDBC Members Policy Council. Audits are expected to commence in 2024. The task force will keep maintaining the workload in the coming years. In future versions of SNB Interactive, we plan to incorporate long-running transactions, schema constraints [3], and regular path queries [14].

Acknowledgements

References

1. R. Angles et al. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017.
2. R. Angles et al. The LDBC Social Network Benchmark. *CoRR*, abs/2001.02299, 2020. `http://arxiv.org/abs/2001.02299`.

3. R. Angles et al. PG-Keys: Keys for property graphs. In *SIGMOD*, 2021.

4. P. A. Boncz et al. TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark. In *TPCTC*, 2013.

5. A. Bonifati et al. Graph generators: State of the art and open challenges. *ACM Comput. Surv.*, 53(2):36:1–36:30, 2020.

6. A. Deutsch et al. Graph pattern matching in GQL and SQL/PGQ. In *SIGMOD*, 2022.

7. O. Erling et al. The LDBC Social Network Benchmark: Interactive workload. In *SIGMOD*, 2015.

8. N. Francis et al. Cypher: An evolving query language for property graphs. In *SIGMOD*, pages 1433–1445. ACM, 2018.

9. J. Gray, editor. *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann, 2nd edition, 1993.

10. J. Gray et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

11. A. Green et al. Updating graph databases with Cypher. *PVLDB*, 2019.

12. A. Gubichev and P. A. Boncz. Parameter curation for benchmark queries. In *TPCTC*, 2014.

13. L. Lőrincz, J. Koltai, A. F. Győr, and K. Takács. Collapse of an online social network: Burning social capital to create it? *Soc. Networks*, 57:43–53, 2019.

14. L. Libkin and D. Vrgoc. Regular path queries on graphs with data. In *ICDT*, 2012.

15. M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, pages 415–444, 2001.

16. M. E. J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemp. Phys.*, 46(5):323–351, 2005.

17. D. Püroja. LDBC Social Network Benchmark Interactive v2. Master's thesis, Universiteit van Amsterdam, 2023. `https://ldbcouncil.org/docs/papers/msc-thesis-david-puroja-snb-interactive-v2-2023.pdf`.

18. M. Raasveldt and H. Mühleisen. DuckDB: An embeddable analytical database. In *SIGMOD*, 2019.

19. L. Roditty and U. Zwick. On dynamic shortest paths problems. In *ESA*, 2004.

20. S. Sahu et al. The ubiquity of large graphs and surprising challenges of graph processing: Extended survey. *VLDB J.*, 2020.

21. S. Sakr et al. The future is big graphs: A community view on graph processing systems. *Commun. ACM*, 2021.

22. S. Shastri et al. Understanding and benchmarking the impact of GDPR on database systems. *VLDB*, 13(7):1064–1077, 2020.

23. C. L. Staudt, A. Sazonovs, and H. Meyerhenke. NetworKit: A tool suite for large-scale complex network analysis. *Netw. Sci.*, 4(4):508–530, 2016.

24. G. Szárnyas et al. The LDBC Social Network Benchmark: Business Intelligence workload. *PVLDB*, 2022.

25. TPC. TPC Benchmark H, revision 2.18.0. pages 1–138, 2017. `http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf`.

26. J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the Facebook social graph. *CoRR*, abs/1111.4503, 2011. `http://arxiv.org/abs/1111.4503`.

27. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, (393):440–442, 1998.

28. J. Waudby et al. Supporting dynamic graphs and temporal entity deletions in the LDBC Social Network Benchmark's data generator. In *GRADES-NDA*, 2020.

29. J. Waudby et al. Towards testing ACID compliance in the LDBC Social Network Benchmark. In *TPCTC*, 2020.