

---

## **The LDBC Social Network Benchmark (version 2.2.5-SNAPSHOT, commit dd444ea)**

---

The specification was built on the source code available at  
[https://github.com/ldbc/ldbc\\_snb\\_docs/tree/main](https://github.com/ldbc/ldbc_snb_docs/tree/main)



This work is licensed under a Creative Commons Attribution 4.0 License.

---

## ABSTRACT

LDBC's Social Network Benchmark (LDBC SNB) is an effort intended to test various functionalities of systems used for graph-like data management. For this, LDBC SNB uses the recognizable scenario of operating a social network, characterized by its graph-shaped data.

LDBC SNB consists of two workloads that focus on different functionalities: the Interactive workload (interactive transactional queries) and the Business Intelligence workload (analytical queries).

This document contains the definition of both workloads. This includes a detailed explanation of the data used in the LDBC SNB benchmark, a detailed description for all queries, and instructions on how to generate the data and run the benchmark with the provided software.

---

## EXECUTIVE SUMMARY

The new data economy era, based on complexly structured, distributed and large datasets, has brought on new demands on data management and analytics. As a consequence, new industry actors have appeared, offering technologies specially built for the management of graph-like data. Also, traditional database technologies, such as relational databases, are being adapted to the new demands to remain competitive.

LDBC's Social Network Benchmark (LDBC SNB) is an industrial and academic initiative, formed by principal actors in the field of graph-like data management. Its goal is to define a framework where different graph based technologies can be fairly tested and compared, that can drive the identification of systems' bottlenecks and required functionalities, and can help researchers to open new research frontiers.

The philosophy around which LDBC SNB is designed is to be easy to understand, flexible and cheap to adopt. For all these reasons, LDBC SNB will propose different workloads representing all the usage scenarios of graph-like database technologies, hence, targeting systems of different nature and characteristics. In order increase its adoption by industry and research institutions, LDBC SNB provides all necessary software, which are designed to be easy to use and deploy at a small cost.

This document contains:

- A detailed specification of the data used in the whole LDBC SNB benchmark.
- A detailed specification of the workloads.
- A detailed specification of the execution rules of the benchmark.
- A detailed specification of the auditing rules and the full disclosure report's required contents.

# TABLE OF CONTENTS

1	INTRODUCTION	14
1.1	Motivation for the Benchmark	14
1.2	Relevance to the Industry	14
1.3	General Benchmark Overview	15
1.4	Related Projects	16
1.5	Participation of Industry and Academia	16
1.6	Technical Report	17
2	BENCHMARK SPECIFICATION	18
3	DATA SETS AND DATA GENERATION	19
3.1	Data Types	19
3.2	Data Schema	19
3.2.1	Entities (Nodes)	20
3.2.2	Relations (Edges)	23
3.2.3	Domain Concepts	24
3.3	Data Generation	24
3.3.1	Resource Files	25
3.3.2	Graph Generation	25
3.3.3	Distributions, Parameters, and Quirks	27
3.3.4	Implementation Details	28
3.4	Output Data	28
3.4.1	Scale Factors	29
3.4.2	Serializers	30
3.4.3	Interactive Update Streams (Inserts)	32
3.4.4	Substitution Parameters	32
3.5	Introducing Delete Operations	32
3.6	Lifespan Management	34
3.6.1	General Rules	35
3.6.2	Person	35
3.6.3	Forum and Message	36
3.6.4	Forum	36
3.6.5	Message	38
3.6.6	Complex Example	39
3.7	Ensuring Realism	39
3.8	Converting Delete Events into Delete Operations	41
4	WORKLOADS	44
4.1	Query Description Format	44
4.2	Conventions for Query Definitions	44
4.3	Substitution Parameters	46
4.4	Return Values	47
5	UPDATE OPERATIONS	48
5.1	Insert Operations	48
5.2	Delete Operations	53

6	INTERACTIVE V1 WORKLOAD	58
6.1	Complex Reads . . . . .	59
6.2	Short Reads . . . . .	73
6.3	Workload Definition . . . . .	77
7	INTERACTIVE V2 WORKLOAD	80
7.1	Overview . . . . .	80
7.2	Operations . . . . .	80
7.2.1	Complex Reads . . . . .	81
7.2.2	Short Reads . . . . .	82
7.2.3	Insert Operations . . . . .	82
7.2.4	Delete Operations . . . . .	82
7.3	Parameter Curation . . . . .	82
7.3.1	Building Blocks for Parameter Curation . . . . .	82
7.3.2	Parameter Curation for Relational Queries . . . . .	82
7.3.3	Parameter Curation for Path-Finding Queries . . . . .	83
7.3.4	Query Variants . . . . .	83
7.3.5	Parameter Generator Implementation . . . . .	84
7.4	Workload Scheduling and Benchmark Driver . . . . .	84
7.4.1	Scheduling Operations . . . . .	84
7.4.2	Driver . . . . .	85
8	BUSINESS INTELLIGENCE WORKLOAD	86
8.1	Overview . . . . .	86
8.2	Read Query Templates . . . . .	86
8.2.1	Choke Point-Based Design Methodology . . . . .	87
8.2.2	Analysis of Selected Queries . . . . .	87
8.3	Parameter Curation for BI Queries . . . . .	88
8.3.1	The Need for Parameter Curation . . . . .	88
8.3.2	Parameter Generation Steps . . . . .	88
8.3.3	Parameter Curation for Graph Queries . . . . .	88
8.3.4	Query Variants . . . . .	89
8.3.5	Scalability and Reproducibility . . . . .	89
8.4	Reads . . . . .	90
8.5	Insert Operations . . . . .	110
8.6	Delete Operations . . . . .	110
9	AUDITING POLICIES	111
9.1	Rationale and General Principles . . . . .	111
9.2	Auditing Rules Overview . . . . .	111
9.2.1	Auditor Training, Certification, and Selection . . . . .	111
9.2.2	Auditing Process Stages . . . . .	112
9.2.3	Challenge Procedure . . . . .	112
9.3	Auditable Properties of Systems and Benchmark Implementations . . . . .	113
9.3.1	Validation of Query Results . . . . .	113
9.3.2	ACID Compliance . . . . .	113
9.3.3	Data Schema . . . . .	114
9.3.4	Data Format and Preprocessing . . . . .	115
9.3.5	Data Access Transparency . . . . .	115
9.3.6	Query Languages . . . . .	115
9.3.7	Materialization . . . . .	116
9.3.8	Steady State . . . . .	116

9.3.9	Query Mix	116
9.3.10	System Configuration and System Pricing	117
9.3.11	Benchmark Specifics	118
9.4	Auditing Rules for the Interactive Workload	118
9.4.1	Scaling	119
9.4.2	Data Model and Data Loading	120
9.4.3	Precomputation	121
9.4.4	Benchmark Software Components	121
9.4.5	Implementation Language and Data Access Transparency	122
9.4.6	Correctness of Benchmark Implementation	123
9.4.7	Benchmarking Workflow	124
9.4.8	Full Disclosure Report	125
9.5	Auditing Rules for the Business Intelligence Workload	126
9.5.1	Overview	126
9.5.2	Workflow	126
9.5.3	Runtimes	127
9.5.4	Scoring Metrics	127
9.5.5	Implementation Rules	128
9.5.6	Scaling	128
9.5.7	Full Disclosure Report	128
10	ACID TEST SUITE	129
10.1	Background	129
10.2	Atomicity	129
10.3	Isolation	130
10.3.1	System Model	131
10.3.2	General Design	132
10.3.3	Dirty Write	132
10.3.4	Dirty Reads	133
10.3.5	Cut Anomalies	134
10.3.6	Observed Transaction Vanishes	136
10.3.7	Fractured Read	136
10.3.8	Lost Update	137
10.3.9	Write Skew	137
10.4	Consistency and Durability Tests	138
11	RELATED WORK	139
11.1	ACID Tests in Other Benchmarks	139
11.2	Graph Processing Benchmarks	139
11.3	Scalable Graph Generators	139
	BIBLIOGRAPHY	140
A	CHOKE POINTS	145
A.1	Aggregation Performance	145
A.2	Join Performance	146
A.3	Data Access Locality	148
A.4	Expression Calculation	148
A.5	Correlated Sub-Queries	149
A.6	Parallelism and Concurrency	150
A.7	Graph Specifics	150
A.8	Language Features	151

A.9	Update Operations . . . . .	153
B	SCALE FACTOR STATISTICS . . . . .	154
B.1	Number of Entities for SNB Interactive v1.0 . . . . .	154
B.2	Number of Entities for SNB BI v1.0 . . . . .	154
B.3	Factor Tables . . . . .	154
C	BENCHMARK CHECKLIST . . . . .	157
D	LEGACY DATA SETS FOR THE INTERACTIVE WORKLOAD . . . . .	158
D.1	Output Data . . . . .	158
D.1.1	Scale Factors . . . . .	158
D.1.2	Serializers . . . . .	159
D.1.3	Update Streams . . . . .	161
D.1.4	Substitution Parameters . . . . .	161
E	EXAMPLE GRAPH . . . . .	164

## LIST OF FIGURES

2.1	High-level overview of the frameworks implementing each LDBC Social Network Benchmark workload. Legend: <span style="background-color: #FFD700;">Software component</span> <span style="background-color: #ADD8E6;">Data artifact</span> . . . . .	18
3.1	UML class diagram-style depiction of the LDBC SNB graph schema. Note that the knows edges should be treated as undirected (but are serialized only in a single direction). The cardinality of the hasModerator edge has changed between version 1 (where it was exactly 1) and version 2 (where it is 0..1). . . . .	20
3.2	The Datagen generation process. . . . .	27
3.3	The power-law used to generate comments. . . . .	28
3.4	The distribution used to generate posts during flashmob events. . . . .	29
3.5	Example graph and its intervals. . . . .	36
3.6	Example graph and time intervals for selecting lifespan attributes, <i>creation</i> and <i>deletion dates</i> . . . . .	40
3.7	Distribution for determining the probability a Person is deleted given their number of connections. . . . .	41
3.8	Probability a post is deleted given the number of comments in its thread. . . . .	42
3.9	Cumulative probability density function of when a post, comment, or like is deleted after it is created ( $x = 0$ ). . . . .	43
3.10	Possible dynamic entity <i>creation</i> ● and <i>deletion</i> ● dates with respect to simulation start, bulk load cut off, simulation end, and network collapse. . . . .	43
4.1	Example graph pattern. . . . .	45
7.1	Components and workflow of the Interactive v2 workload. The corresponding sections are shown in green circles (§). Legend: <span style="background-color: #FFD700;">Software component</span> <span style="background-color: #ADD8E6;">Data artifact</span> . . . . .	80
7.2	Example graph and distribution for path curation. . . . .	83
7.3	Workflow of driver modes in SNB Interactive v2. . . . .	85
8.1	Main software components and data artifacts of the benchmark and their connection to the workflow executed by the BI benchmark driver. . . . .	86
9.1	Benchmark execution and auditing workflow. For non-audited runs, the implementers perform the steps of the auditor. . . . .	119
9.2	Warm-up and measurement window for benchmark run. . . . .	125
9.3	Tests and batches (power and throughput) executed in the BI workload's workflow. . . . .	126
10.1	Graph schema for the ACID test queries. . . . .	130
10.2	Hierarchy of isolation levels as described in [11]. All anomalies are covered except <b>G-Cursor(x)</b> . . . . .	130
E.1	Example graph snapshot (without update operations). . . . .	164
E.2	Example graph with update operations. . . . .	165



## LIST OF TABLES

3.1	Description of the data types. Some types such as 32-bit Float and 64-bit Integer are currently not used in the benchmark. . . . .	19
3.2	Attributes of the Forum entity. . . . .	21
3.3	Attributes of the Message interface. . . . .	21
3.4	Attributes of the Organisation entity. . . . .	21
3.5	Attributes of the Person entity. . . . .	22
3.6	Attributes of the Place entity. . . . .	22
3.7	Attributes of the Post entity. . . . .	22
3.8	Attributes of the Tag entity. . . . .	22
3.9	Attributes of the TagClass entity. . . . .	23
3.10	Description of the data relations. Type – D: directed edge, U: undirected edge. . . . .	24
3.11	Resource files. . . . .	26
3.12	Properties of data sets for each scale factor for the <i>raw data sets</i> produced the Spark-based generator, used as a basis of the data sets of SNB Interactive v2 and SNB BI. . . . .	30
3.13	Attributes and edges serialized to separate files the different CSV serializers. . . . .	30
3.14	Files output by the <code>csv-composite-projected-fk</code> serializer (31 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. . . . .	31
3.15	Files output by the <code>csv-composite-merged-fk</code> serializer (18 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. . . . .	32
3.16	Directories created by the raw serializer (18 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. The entities with the <code>explicitlyDeleted</code> attribute – Comment, Forum, Post nodes, and <code>hasMember</code> , <code>knows</code> , <code>likes</code> (Comment/Post) edges – denote whether the entity is deleted as part of an explicit delete operation or implicitly through a cascading delete operation. . . . .	33
3.17	Mapping of [2] message types to LDBC’s schema. . . . .	41
6.1	Frequencies for each Interactive complex query and SF. . . . .	78
6.2	Short read queries (columns) potentially triggered after given complex/short read queries (rows). . . . .	79
A.1	Coverage of choke points by queries. . . . .	145
B.1	The number of entities per SF and per file in the Interactive workload (produced by the Hadoop-based generator and measured based on the output of the <code>CsvBasic</code> serializer). To derive these numbers, 100% of the network was generated as an initial bulk data set with no update streams. Notation – C: entity category, N: node, E: edge. . . . .	154
B.2	The number of entities per SF and per file in the <i>initial data set</i> used in the BI workload. Notation – C: entity category, N: node, E: edge. . . . .	155
B.3	The number of entities per SF and per file in the <i>update data sets</i> used in the BI workload. Notation – T: update type, I: insert, D: delete; C: entity category, N: node, E: edge. . . . .	155
B.4	The total size of the factor tables. . . . .	156
D.1	Properties of data sets for each scale factor in the Interactive workload produced by the Hadoop-based generator. For detailed statistics, see Table B.1 . . . . .	159
D.2	Update stream statistics for SNB Interactive v1.0 . . . . .	159
D.3	Attributes and edges serialized to separate files the different CSV serializers. . . . .	159

D.4	Files output by the CsvBasic serializer (33 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. . . . .	160
D.5	Files output by the CsvMergeForeign serializer (20 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. . . . .	161
D.6	Files output by the CsvComposite serializer (31 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. . . . .	162
D.7	Files output by the CsvCompositeMergeForeign serializer (18 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. . . . .	162
D.8	Generic schema of update (insert) stream files. The start time ( $t_s$ ) is identical to the createDate attribute (repeated later in the row). . . . .	163
D.9	Schemas of the lines in the update stream (insert stream) files. . . . .	163

## ACKNOWLEDGMENTS

Special thanks to all the people that have contributed to the development of this benchmark suite:

- Renzo Angles (Universidad de Talca)
- János Benjamin Antal (Budapest University of Technology and Economics)
- Alex Averbuch (Neo4j)
- Altan Birler (TUM)
- Peter Boncz (Vrije Universiteit Amsterdam, CWI)
- Márton Búr (McGill University)
- Orri Erling (OpenLink Software)
- Andrey Gubichev (Technische Universität München)
- Vlad Haprian (Oracle Labs)
- Moritz Kaufmann (Technische Universität München)
- Josep Lluís Larriba Pey (Universitat Politècnica de Catalunya)
- Norbert Martínez (Huawei Technologies)
- József Marton (Budapest University of Technology and Economics)
- Marcus Paradies (SAP, DLR)
- Minh-Duc Pham (Altran)
- Arnau Prat-Pérez (DAMA UPC, Sparsity Technologies)
- David Püroja (CWI)
- Mirko Spasić (OpenLink Software)
- Benjamin A. Steer (Queen Mary University of London, Pometry)
- Dávid Szakállas
- Gábor Szárnyas (MTA-BME Lendület Research Group on Cyber-Physical Systems, Budapest University of Technology and Economics, CWI)
- Jack Waudby (Newcastle University)
- Mingxi Wu (TigerGraph)
- Yuchen Zhang (TigerGraph)

## DEFINITIONS

This section defines fundamental concepts used in the LDBC benchmark terminology. Part of the definitions below are repeated from the LDBC benchmark specification document.

**LDBC SNB** The Linked Data Benchmark Council’s Social Network Benchmark suite which currently consists of the Interactive workload and a preliminary version of the Business Intelligence workload.

**System Under Test (SUT)** This is the totality of the hardware and software that participates in a benchmark run, excluding parts that are exclusively used for driving the workload. If the parts driving the workload are collocated on the same operating system instance as the SUT, then this is also considered a part of the SUT. In client-server configurations where the test driver is not on a machine hosting any DBMS function the SUT is not considered to encompass the hardware or software which exclusively serves to drive the test workload.

**Datagen** This module is provided by LDBC SNB and produces the standard benchmark datasets to be loaded into the SUT for the benchmark. The data generation phase is not part of running the benchmark.

**Test Driver (Benchmark Driver, Driver)** The test driver refers to the parts of the benchmark run that coordinate query execution and, if prescribed by a given benchmark, data loading.

**Workload (Benchmark)** This is the totality of the tasks a particular benchmark performs against an SUT. This includes data loading as well as the query/update workload. This does not include preparatory stages such as generating benchmark data with a data generator or transferring the data to the platform constituting the SUT. The terms workload and benchmark are synonyms in this context.

**Time Compression Ratio (TCR)** This parameter of the Interactive workload compresses (or stretches) durations between operation start times to increase (or decrease) operation rate, thereby allowing systems to reach their maximum throughput for a given workload. The smaller this number is, the higher compression ratio it represents (e.g. 2.0 means run benchmark 2× slower, while 0.1 = run benchmark 10× faster). Systems are expected to compete on achieving the *lowest possible* TCR (i.e. the highest  $TCR^{-1} = \frac{1}{TCR}$ ).

**Query mix** The ratio of read and update queries of a workload, and the frequency at which they are issued.

**Scale Factor (SF)** The LDBC SNB is designed to target systems of different size and scale. The scale factor determines the size of the data used to run the benchmark. The scale factor refers to the measured size of the data in Gigabytes when serialised in CsvSingularProjectedFK.

**Validation Step** The benchmark specifies a scale factor for which ACID test cases are executed and the query results are compared to a reference result set (i.e. expected output). This step is required to use the very same set of queries and data structures (this includes both PDS, IADS and EADS – defined below) that are used in the actual benchmark runs.

**Schema (Database Schema)** A schema is the totality of the non-built-in declarations which are fed into the SUT prior to running a workload. For a relational system, the schema consists of tables, indices, views, materialised views and declarative constraints (e.g. foreign key and not null constraints). An ontology for an RDF system counts as a schema if it is loaded on the SUT. An RDF SUT may have no schema at all and still run the workload. However, any declaration or setting (e.g. indices) that is not on by default in the SUT, but is used in at least one case of the benchmark run counts as part of the schema. The schema does not include stored procedures, triggers, or other imperative (procedural) application specific code that may reside on the SUT and could impact the benchmark results. The schema is required to be the same across all benchmark runs using the same scale factor for a given workload.

**Primary Data Structure (PDS)** This is anything that may influence the result of a database query or may be changed by an update of the database. These may be resident in RAM or durable media or both. Examples of data structures are database base tables and adjacency lists.

**Implicit Auxiliary Data Structure (IADS)** This is a data structure for providing more efficient access to all or parts of the primary data structure. IADS are created by the DBMS automatically and the system may allow them to be turned off.

Some systems, such as many RDF stores have multiple covering indices on the primary data structure. The definition in this case is that the primary data structure consists of all the dif-

ferently ordered full copies of the base table; a table of subject predicate object graph (SPOG) in the RDF case. In this same instance, Auxiliary data structures comprise any data structure which materialise a subset of the SPOG.

**Explicit Auxiliary Data Structure (EADS)** These are any application or workload profile specific structures that are declared in addition to the PDSs and IADSs managed by the SUT. These duplicate the data and are created with explicit statements. Secondary indices, materialised views, with or without aggregates, are all examples of this in a relational context. The decision about the used EADS is always part of the schema declaration.

In the case of relational systems, an ADS may be an index from primary key values to a heap table, if the system in question has such concepts. A secondary index of a relational table, in its memory based and durable media based manifestations is an example for EADS. Such a secondary index is not considered an ADS since it must be declared, which makes its creation explicit. An ADS must be implicit and not created by any specific DDL statement or directive. In the case of RDF systems, if the implementation supports user definable index schemes, as long as these are defined once and apply to all triples/quads, such structures are designated as ADS. If an RDF system selectively makes data structures which apply to some quads but not to others, then such structures are designated as EADS.

**SUT-Resident Logic** This is any application specific code that is resident on the SUT, whether by static linking, dynamic loading, JIT, interpretation or any other means of embedding application specific logic into a generic DBMS. Examples of this are stored procedures, hosting Java, CLR or other run times in the SUT process (or processes), loading application specific libraries to extend native functions or data structures etc. A special case is that of a database exclusively accessed via an in-process API. In these cases, any code that is not the test driver or a workload implementation expressed against a generally supported API of the DBMS is deemed SUT resident logic in addition to any other code which may fit the above definitions.

**Test Sponsor** The party which initiates an audit of a benchmark implementation over an SUT. This is typically the vendor of a key component of the SUT, e.g. DBMS or hardware.

**Full Disclosure Report (FDR)** This is a document which allows reproduction of any audited benchmark result by a third party. It contains complete description of the circumstances of the benchmark run, including version and configuration of SUT, dataset and test driver.

## 1 INTRODUCTION

### 1.1 Motivation for the Benchmark

The new era of data economy, based on large, distributed, and complexly structured datasets, has brought on new and complex challenges in the field of data management and analytics. These datasets, usually modeled as large graphs, have attracted both industry and academia, due to new opportunities in research and innovation they offer. This situation has also opened the door for new companies to emerge, offering new non-relational and graph-like technologies that are called to play a significant role in upcoming years.

The change in the data paradigm calls for new benchmarks to test these new emerging technologies, as they set a framework where different systems can compete and be compared in a fair way, they let technology providers identify the bottlenecks and gaps of their systems and, in general, drive the research and development of new information technology solutions. Without them, the uptake of these technologies is at risk by not providing the industry with clear, user-driven targets for performance and functionality.

The Linked Data Benchmark Council's [81] Social Network Benchmark (LDBC SNB) aims at being a comprehensive benchmark by setting the rules for the evaluation of graph-like data management technologies. LDBC SNB is designed to be a plausible look-alike of all the aspects of operating a social network site, as one of the most representative and relevant use cases of modern graph-like applications.

LDBC SNB includes the Interactive workload [24], which consists of user-centric transactional-like interactive queries, and the Business Intelligence workload, which includes analytic queries to respond to business-critical questions. Initially, a graph analytics workload was also included in the roadmap of LDBC SNB, but this was finally delegated to the Graphalytics benchmark project [38, 39], which was adopted as an official LDBC graph analytics benchmark. LDBC SNB and Graphalytics combined target a broad range of systems with different nature and characteristics. LDBC SNB and Graphalytics aim at capturing the essential features of these scenarios while abstracting away details of specific business deployments.

This document contains the definition of the Interactive workload and the first draft of the Business Intelligence workload. This includes a detailed explanation of the data used in the LDBC SNB benchmark, a detailed description for all queries, and instructions on how to generate the data and run the benchmark with the provided software.

### 1.2 Relevance to the Industry

LDBC SNB is intended to provide the following value to different stakeholders:

- For **end users** facing graph processing tasks, LDBC SNB provides a recognizable scenario against which it is possible to compare merits of different products and technologies. By covering a wide variety of scales and price points, LDBC SNB can serve as an aid to technology selection.
- For **vendors** of graph database technology, LDBC SNB provides a checklist of features and performance characteristics that helps in product positioning and can serve to guide new development.
- For **researchers**, both industrial and academic, the LDBC SNB dataset and workload provide interesting challenges in multiple choke point areas, such as query optimization, (distributed) graph analysis, transactional throughput, and provides a way to objectively compare the effectiveness and efficiency of new and existing technology in these areas.

The technological scope of LDBC SNB comprises all systems that one might conceivably use to perform social network data management tasks:

- **Graph database management systems** (e.g. Neo4j, TigerGraph, AWS Neptune) are novel technologies aimed at storing property graphs, i.e. graphs with labels and properties (attributes) on nodes and edges. They support graph traversals, typically by means of APIs, though some of them also support dedicated graph-oriented query languages (e.g. Neo4j's Cypher and TigerGraph's GSQL, as well as the GQL and SQL/PGQ standards). These systems' internal structures are typically designed to store dynamic graphs

that change over time. They offer support for transactional queries with some degree of consistency, and value-based indices to quickly locate nodes and edges. Finally, their architecture is typically single-machine (non-cluster). These systems can potentially implement all three workloads, though Interactive and Business Intelligence workloads are where they will presumably be more competitive.

- **Graph processing frameworks** (e.g. Giraph, Signal/Collect, GraphLab, Green Marl) are designed to perform global graph computations, executed in parallel or in a lockstep fashion. These computations are typically long latency, involving many nodes and edges and often consist of approximation answers to NP-complete problems. These systems expose an API, sometimes following a vertex-centric paradigm, and their architecture targets both single-machine and cluster systems. These systems will likely implement the Graph Analytics workload.
- **RDF database systems** (e.g. OWLIM, Virtuoso, Stardog, AWS Neptune) are systems that implement the SPARQL 1.1 query language, similar in complexity to SQL-92, which allows for structured queries, and simple traversals. RDF database systems often come with additional support for simple reasoning (sameAs, subClass), text search, and geospatial predicates. RDF database systems generally support transactions, but not always with full concurrency and serializability and their supposed strength is integrating multiple data sources (e.g. DBpedia). Their architecture is both single-machine and clustered, and they will likely target Interactive and Business Intelligence workloads.
- **Relational database systems** (e.g. PostgreSQL, MySQL, Oracle, IBM Db2, Microsoft SQL Server, Virtuoso, MonetDB, Vectorwise, Vertica, DuckDB but also Hive and Impala) treat graph data relationally, and queries are formulated in SQL and/or PL/SQL. Both single-machine and cluster systems exist. They do not normally support recursion or stateful recursive algorithms, which makes them not at home in the Graph Analytics workloads.

## 1.3 General Benchmark Overview

LDBC SNB aims at being a complete benchmark, designed with the following goals in mind:

- **Rich coverage.** LDBC SNB is intended to cover most demands encountered in the management of complexly structured data.
- **Modularity.** LDBC SNB is broken into parts that can be individually addressed. In this manner LDBC SNB stimulates innovation without imposing an overly high threshold for participation.
- **Reasonable implementation cost.** For a product offering relevant functionality, the effort for obtaining initial results with SNB should be small, in the order of days.
- **Relevant selection of challenges.** Benchmarks are known to direct product development in certain directions. LDBC SNB is informed by the state-of-the-art in database research so as to offer optimization challenges for years to come while not having a prohibitively high threshold for entry.
- **Reproducibility and documentation of results.** LDBC SNB will specify the rules for full disclosure of benchmark execution and for auditing of benchmark runs in accordance with the LDBC Byelaws [44]. The workloads may be run on any equipment but the exact configuration and price of the hardware and software must be disclosed.

LDBC SNB benchmark is modeled around the operation of a real social network site. A social network site represents a relevant use case for the following reasons:

- It is simple to understand for a large audience, as it is arguably present in our every-day life in different shapes and forms.
- It allows testing a complete range of interesting challenges, by means of different workloads targeting systems of different nature and characteristics.
- A social network can be scaled, allowing the design of a scalable benchmark targeting systems of different sizes and budgets.

Chapter 2 summarizes LDBC's benchmark design philosophy.

In Chapter 3, we define the schema of the data used in the benchmark. The schema represents a realistic social network, including people and their activities in the social network during a period of time. Personal information of each person, such as name, birthday, interests or places where people work or study, is included. A person's activity is represented in the form of friendship relationships and content sharing (i.e. messages and pictures). LDBC SNB provides a scalable synthetic data generator based on the MapReduce paradigm, which produces networks with the described schema with distributions and correlations similar to those expected in a real social network. Furthermore, the data generator is designed to be user-friendly. The proposed data schema is shared by all the different proposed workloads, those we currently have, and those that will be proposed in the future.

In Chapter 4, an overview of the workloads is provided. All SNB workloads are designed to mimic the different usage scenarios found in operating a real social network site, and each of them targets one or more types of systems. Each workload defines a set of queries and query mixes, designed to stress the SUTs in different choke point areas, while being credible and realistic. The Interactive workload reproduces the interaction between the users of the social network by including lookups and transactions, which update small portions of the database. These queries are designed to be interactive and target systems capable of responding to such queries with low latency for multiple concurrent users. The Business Intelligence workload represents analytic queries a social network company would like to perform in the social network, to take advantage of the data and to discover new business opportunities. This workload explores moderate to large portions of the graph from different entities, and performs more resource-intensive operations.

All workloads provide an execution test driver, which is responsible for executing the workloads and gathering the results. The driver is designed with simplicity and portability in mind to ease the implementation on systems with different nature and characteristics at a low implementation cost. Furthermore, it automatically handles the validation of the queries by means of a validation dataset provided by LDBC. The overall philosophy of LDBC SNB is to provide the necessary software tools to run the benchmark, and therefore to reduce the benchmark's entry point as much as possible.

Chapter 5 defines the update operations used in the SNB workloads. Chapter 6, Chapter 7, and Chapter 8 define the SNB Interactive v1, Interactive v2, and BI workloads, respectively. Chapter 9 contains the SNB auditing policies. Chapter 10 defines the ACID test suite. Chapter 11 summarized the related work on graph processing benchmarks.

## 1.4 Related Projects

Along the Social Network Benchmark, LDBC [7] provides other benchmarks as well:

- The Semantic Publishing Benchmark (SPB) [76] measures the performance of *semantic databases* operating on RDF datasets.
- The Graphalytics benchmark [38] measures the performance of *graph analysis* operations (e.g. PageRank, local clustering coefficient).
- The Financial benchmark (FinBench) suite is set in the financial domain with multiple planned workloads. Currently, the *Transactional* workload is available, which focuses on low-latency operations.

## 1.5 Participation of Industry and Academia

The list of institutions that take part in the definition and development of LDBC SNB is formed by relevant actors from both the industry and academia in the field of linked data management. All the participants have contributed with their experience and expertise in the field, making a credible and relevant benchmark, which meets all the desired needs.



## 1.6 Technical Report

This technical report is available on arXiv [6] and is updated upon new releases of the SNB.

## 2 BENCHMARK SPECIFICATION

**Overview** The LDBC Social Network Benchmark workloads require several components to generate data and updates, produce query substitution parameters, run the benchmark on the system under test, etc. Figure 2.1 shows a blueprint for the frameworks implementing the LDBC SNB benchmark workloads.

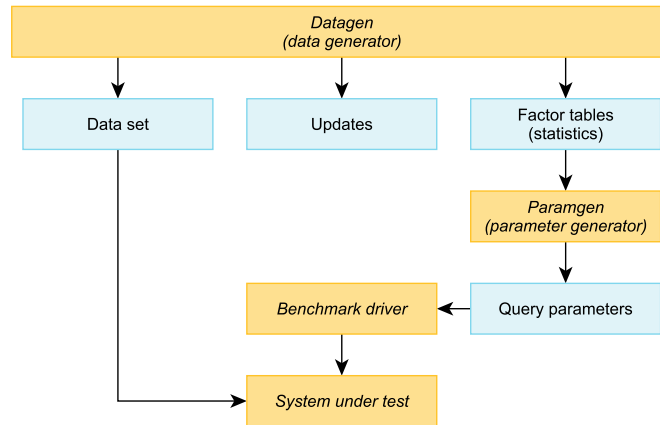


Figure 2.1: High-level overview of the frameworks implementing each LDBC Social Network Benchmark workload. Legend: Software component Data artifact

**Portability** LDBC SNB is designed to be flexible and to have an affordable entry point. LDBC SNB aims to accommodate systems from single node in-memory systems to large distributed multi-node clusters. Therefore, the requirements to fulfill for executing LDBC SNB are limited to pure software requirements to be able to run the tools. While the benchmark specification aims to be portable, the software provided by LDBC SNB have been primarily tested on Linux-based operating systems (e.g. Ubuntu LTS). The driver and clients for the reference implementations were implemented in Java. The generator has two versions: the Hadoop-based one was written in Java, while the Spark-based one is written in a mix of Java and Scala.

**Auditable systems** LDBC SNB does not impose the usage of any specific type of system, as it targets systems of different nature and characteristics, from graph databases, graph processing frameworks and RDF systems, to traditional relational database management systems. Therefore, data can be stored in the most convenient manner the test sponsor may decide, as long as it conforms with the execution rules.

### 3 DATA SETS AND DATA GENERATION

This chapter introduces the data used by LDBC SNB. This includes the different data types, the data schema, how it is generated and the different scale factors.

**Warning.** This chapter describes the latest variant of the data set. If you are looking for information on the Interactive workload, please also check Appendix D.

#### 3.1 Data Types

Table 3.1 describes the different data types used in the benchmark.

Type	Description
ID	integer type with 64-bit precision. All IDs within a single entity type (e.g. Person, Message) are unique, but different entity types (e.g. a Forum and a Tag) might have the same ID.
32-bit Integer	integer type with 32-bit precision
64-bit Integer	integer type with 64-bit precision
32-bit Float	integer type with 32-bit precision
64-bit Float	integer type with 64-bit precision
String	variable length text of size 80 Unicode characters
Long String	variable length text of size 256 Unicode characters
Text	variable length text of size 2000 Unicode characters
Date	date with a precision of a day, encoded as a string with the following format: yyyy-mm-dd, where yyyy is a four-digit integer representing the year, the year, mm is a two-digit integer representing the month and dd is a two-digit integer representing the day.
DateTime	date with a precision of milliseconds, encoded as a string with the following format: yyyy-mm-ddTHH:MM:ss.sss+00:00, where yyyy is a four-digit integer representing the year, the year, mm is a two-digit integer representing the month and dd is a two-digit integer representing the day, HH is a two-digit integer representing the hour, MM is a two digit integer representing the minute and ss.sss is a five digit fixed point real number representing the seconds up to millisecond precision. Finally, the +00:00 of the end represents the timezone, which should always be GMT (both for inputs and outputs).
Boolean	logical type, taking the value of either True Of False

Table 3.1: Description of the data types. Some types such as 32-bit Float and 64-bit Integer are currently not used in the benchmark.

#### 3.2 Data Schema

Figure 3.1 shows the data schema in UML. The schema defines the structure of the data used in the benchmark in terms of entities and their relations. Data represents a snapshot of the activity of a social network during a period of time. Data includes entities such as Persons, Organisations, and Places. The schema also models the way persons interact, by means of the friendship relations established with other persons, and the sharing of content such as Messages (both textual and images), replies to Messages and likes to Messages. People form groups to talk about specific topics, which are represented as Tags<sup>1</sup>. An example graph conforming the SNB schema is shown in Appendix E.

<sup>1</sup>Tags are similar to *hashtags* on contemporary social media sites (see <https://en.wikipedia.org/wiki/Hashtag>). In this document, we occasionally use the term *topic* to refer to tags.

LDBC SNB has been designed to be flexible and to target systems of different nature and characteristics. As such, it does not force any particular internal representation of the schema. The Datagen component supports multiple output data formats to fit the needs of different types of systems, including RDF, relational DBMS and graph DBMS.

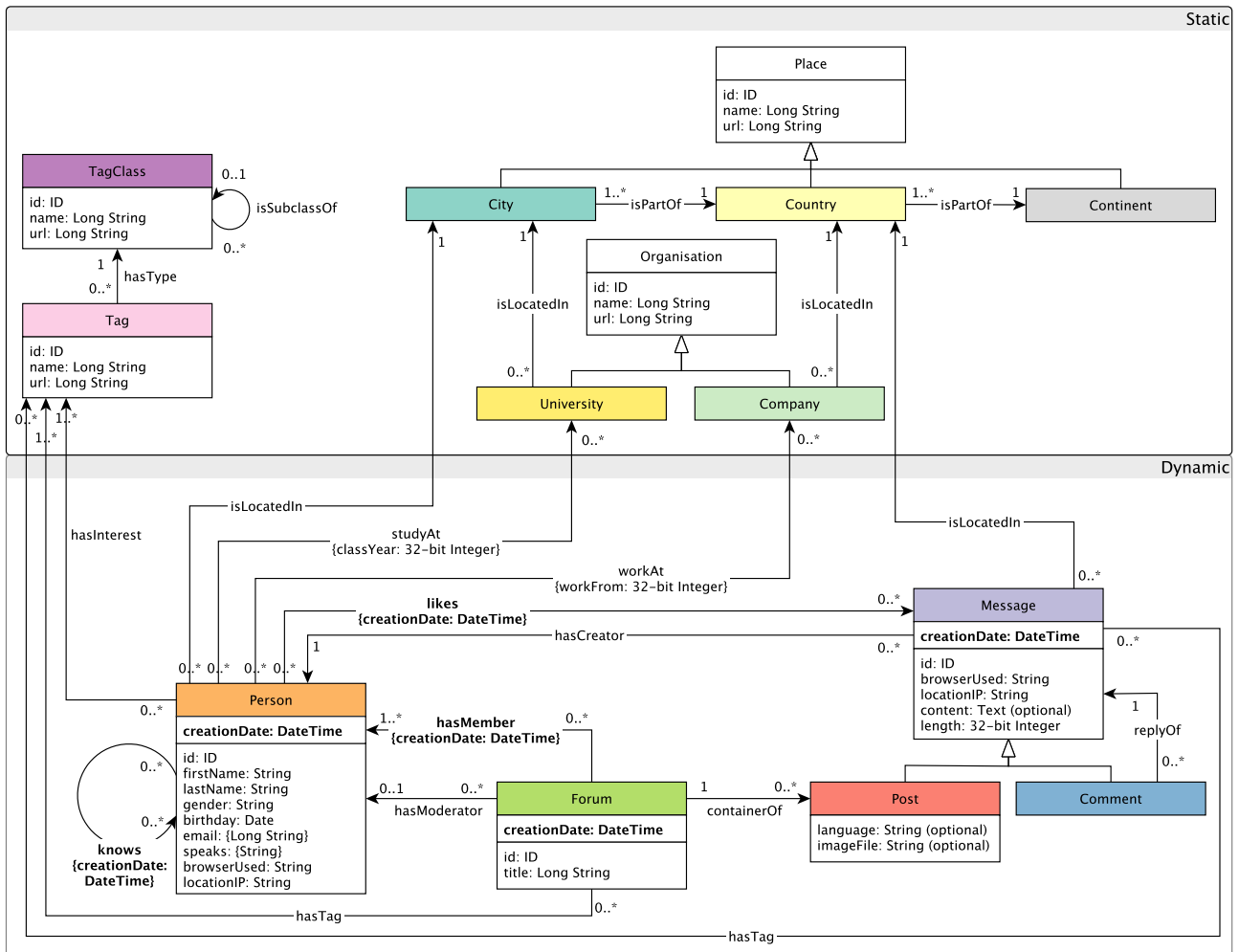


Figure 3.1: UML class diagram-style depiction of the LDBC SNB graph schema. Note that the knows edges should be treated as undirected (but are serialized only in a single direction). The cardinality of the hasModerator edge has changed between version 1 (where it was exactly 1) and version 2 (where it is 0..1).

The schema specifies different entities, their attributes and their relations. All of them are described in the following sections.

### Textual Restrictions and Notes

- Messages always have a non-empty `content` attribute.
- Posts have either a `content` or an `imageFile` attribute (i.e. they always have exactly one of them.) The one they do not have is represented with an empty string or with `NULL`.
- Posts in a forum can be created by a non-member person if and only if that person is the moderator of the Forum.

#### 3.2.1 Entities (Nodes)

**City:** a sub-class of a `Place`, and represents a city of the real world. City entities are used to specify where persons live, as well as where universities operate.

**Comment:** a sub-class of a Message, and represents a comment made by a person to an existing message (either a Post or a Comment).

**Company:** a sub-class of an Organisation, and represents a company where persons work.

**Continent:** a sub-class of a Place, and represents a continent of the real world.

**Country:** a sub-class of a Place, and represents a country of the real world. Countries are selected as the place of operation for Companies as well as the location of Messages.

**Forum:** a meeting point where people post messages. Forums are characterized by the topics (represented as tags) people in the forum are talking about. Although from the schema's perspective it is not evident, there exist three different types of forums. They are distinguished by their titles:

- personal walls: "Wall of ..."
- image albums: "Album  $k$  of ..."
- groups: "Group for ..."

Table 3.2 shows the attributes of the Forum entity.

Attribute	Type	Description
id	ID	The identifier of the forum.
title	Long String	The title of the forum.
creationDate	DateTime	The date the forum was created.

Table 3.2: Attributes of the Forum entity.

**Message:** an abstract entity that represents a message created by a person. Table 3.3 shows the attributes of the Message abstract entity.

Attribute	Type	Description
id	ID	The identifier of the message.
browserUsed	String	The browser used by the Person to create the message.
creationDate	DateTime	The date the message was created.
locationIP	String	The IP of the location from which the message was created.
content	Text (optional)	The content of the message.
length	32-bit Integer	The length of the content.

Table 3.3: Attributes of the Message interface.

**Organisation:** an institution of the real world. Table 3.4 shows the attributes of the Organisation entity.

Attribute	Type	Description
id	ID	The identifier of the organisation.
name	Long String	The name of the organisation.
url	Long String	The URL of the organisation.

Table 3.4: Attributes of the Organisation entity.

**Person:** the avatar a real world person creates when he/she joins the network, and contains various information about the person as well as network related information. Table 3.5 shows the attributes of the Person entity.

Attribute	Type	Description
id	ID	The identifier of the person.
firstName	String	The first name of the person.
lastName	String	The last name of the person.
gender	String	The gender of the person.
birthday	Date	The birthday of the person.
email	{Long String}	The set of emails the person has (cardinality: at least one).
speaks	{String}	The set of languages the person speaks (cardinality: at least one).
browserUsed	String	The browser used by the person when he/she registered to the social network.
locationIP	String	The IP of the location from which the person was registered to the social network.
creationDate	DateTime	The date the person joined the social network.

Table 3.5: Attributes of the Person entity.

**Place:** a place in the world. Table 3.6 shows the attributes of the Place entity. Note, each Place has additional parameters: longitude and latitude, which are not exposed. These are used internally for sorting places.

Attribute	Type	Description
id	ID	The identifier of the place.
name	Long String	The name of the place.
url	Long String	The URL of the place.

Table 3.6: Attributes of the Place entity.

**Post:** a sub-class of Message, that is posted in a forum. Posts are created by persons into the forums where they belong. Posts contain either content or imageFile, always one of them but never both. The one they do not have is an empty string. Table 3.7 shows the attributes of the Post entity.

Attribute	Type	Description
language	String (optional)	The language of the post. Mutually exclusive with imageFile.
imageFile	String (optional)	The image file of the post. Mutually exclusive with language.

Table 3.7: Attributes of the Post entity.

**Tag:** a topic or a concept. Tags are used to specify the topics of forums and posts, as well as the topics a person is interested in. Table 3.8 shows the attributes of the Tag entity.

Attribute	Type	Description
id	ID	The identifier of the tag.
name	Long String	The name of the tag.
url	Long String	The URL of the tag.

Table 3.8: Attributes of the Tag entity.

**TagClass:** a class used to build a hierarchy of tags. Table 3.9 shows the attributes of the TagClass entity.

Attribute	Type	Description
id	ID	The identifier of the tagclass.
name	Long String	The name of the tagclass.
url	Long String	The URL of the tagclass.

Table 3.9: Attributes of the TagClass entity.

**University:** a sub-class of Organisation, and represents an institution where persons study.

### 3.2.2 Relations (Edges)

Relations (edges) connect entities of different types. The endpoint entities are defined by their “id” attribute. Edge instances starting from or ending in a given node are treated as a set, i.e. no ordering is defined on the edges. Multiple edges (i.e. edges of the same type between two entity instances) are not allowed in SNB graphs.

Name	Source	Destination	Type	Description												
containerOf	Forum[1]	Post[0..*]	D	A Forum and a Post contained in it												
hasCreator	Message[0..*]	Person[1]	D	A Message and its creator (Person)												
hasInterest	Person[0..*]	Tag[1..*]	D	A Person and a Tag representing a topic the person is interested in												
hasMember	Forum[0..*]	Person[1..*]	D	A Forum and its member (Person) In version 1: <table><tr><td>Attribute</td><td>joinDate</td></tr><tr><td>Type</td><td>DateTime</td></tr><tr><td>Description</td><td>The Date the person joined the Forum</td></tr></table> In version 2: <table><tr><td>Attribute</td><td>creationDate</td></tr><tr><td>Type</td><td>DateTime</td></tr><tr><td>Description</td><td>The Date the person joined the Forum</td></tr></table>	Attribute	joinDate	Type	DateTime	Description	The Date the person joined the Forum	Attribute	creationDate	Type	DateTime	Description	The Date the person joined the Forum
Attribute	joinDate															
Type	DateTime															
Description	The Date the person joined the Forum															
Attribute	creationDate															
Type	DateTime															
Description	The Date the person joined the Forum															
hasModerator	Forum[0..*]	In version 1: Person[1] In version 2: Person[0..1]	D	A Forum and its moderator (Person)												
hasTag	Message[0..*]	Tag[0..*]	D	A Message and a Tag representing the message's topic												
hasTag	Forum[0..*]	Tag[1..*]	D	A Forum and a Tag representing the forum's topic												
hasType	Tag[0..*]	TagClass[1]	D	A Tag and a TagClass the tag belongs to												
isLocatedIn	Company[0..*]	Country[1]	D	A Company and its home Country												
isLocatedIn	Message[0..*]	Country[1]	D	A Message and the Country from which it was issued												
isLocatedIn	Person[0..*]	City[1]	D	A Person and their home City												
isLocatedIn	University[0..*]	City[1]	D	A University and the City where the university is												
isPartOf	City[1..*]	Country[1]	D	A City and the Country it is part of												
isPartOf	Country[1..*]	Continent[1]	D	A Country and the Continent it is part of												
isSubclassOf	TagClass[0..*]	TagClass[0..1]	D	A TagClass and its parent TagClass												

knows	Person[0..*]	Person[0..*]	U	<p>Two Persons that know each other. Note that (1) the knows edges are undirected (all other edge types are directed and (2) to avoid duplications, these edges are only serialized to one direction and it is the responsibility of the loader/implementation component to treat them as undirected. In this specification document, we use the terms “knows” and “friends (with/of/etc.)” interchangeably.</p> <table><tr><td>Attribute</td><td>creationDate</td></tr><tr><td>Type</td><td>DateTime</td></tr><tr><td>Description</td><td>The date the knows relation was established</td></tr></table>	Attribute	creationDate	Type	DateTime	Description	The date the knows relation was established
Attribute	creationDate									
Type	DateTime									
Description	The date the knows relation was established									
likes	Person[0..*]	Message[0..*]	D	<p>A Person that likes a Message</p> <table><tr><td>Attribute</td><td>creationDate</td></tr><tr><td>Type</td><td>DateTime</td></tr><tr><td>Description</td><td>The date the like was issued</td></tr></table>	Attribute	creationDate	Type	DateTime	Description	The date the like was issued
Attribute	creationDate									
Type	DateTime									
Description	The date the like was issued									
replyOf	Comment[0..*]	Message[1]	D	<p>A Comment and the Message it replies</p>						
studyAt	Person[0..*]	University[0..*]	D	<p>A Person and a University it has studied</p> <table><tr><td>Attribute</td><td>classYear</td></tr><tr><td>Type</td><td>32-bit Integer</td></tr><tr><td>Description</td><td>The year the person graduated</td></tr></table>	Attribute	classYear	Type	32-bit Integer	Description	The year the person graduated
Attribute	classYear									
Type	32-bit Integer									
Description	The year the person graduated									
workAt	Person[0..*]	Company[0..*]	D	<p>A Person and a Company it works</p> <table><tr><td>Attribute</td><td>workFrom</td></tr><tr><td>Type</td><td>32-bit Integer</td></tr><tr><td>Description</td><td>The year the person started to work at that Company</td></tr></table>	Attribute	workFrom	Type	32-bit Integer	Description	The year the person started to work at that Company
Attribute	workFrom									
Type	32-bit Integer									
Description	The year the person started to work at that Company									

Table 3.10: Description of the data relations. Type – D: directed edge, U: undirected edge.

### 3.2.3 Domain Concepts

A *thread* consists of Messages, starting with a single Post and the Comments that – either directly or transitively – reply to that Post.

## 3.3 Data Generation

LDBC SNB provides Datagen (Data Generator), which produces synthetic datasets following the schema described above. Data produced mimics a social network’s activity during a period of time. Three parameters determine the generated data: the number of persons, the number of years simulated, and the starting year of simulation. Datagen is defined by the following characteristics:



- **Realism.** Data generated by Datagen mimics the characteristics of those found in a real social network. In Datagen, output attributes, cardinalities, correlations and distributions have been finely tuned to reproduce a real social network in each of its aspects. On the one hand, it is aware of the data and link distributions found in a real social network such as Facebook. On the other hand, it uses real data from DBpedia, such as property dictionaries, which are used to ensure that attribute values are realistic and correlated.
- **Scalability.** Since LDBC SNB targets systems of different scales and budgets, Datagen is capable of generating datasets of different sizes, from a few Gigabytes to Terabytes. Datagen is implemented following the MapReduce parallel paradigm, allowing the generation of small datasets in single node machines, as well as large datasets on commodity clusters.
- **Determinism.** Datagen is deterministic regardless of the number of cores/machines used to produce the data. This important feature guarantees that all Test Sponsors will face the same dataset, thus, making the comparisons between different systems fair and the benchmarks' results reproducible.
- **Usability.** LDBC SNB is designed to have an affordable entry point. As such, Datagen's design is severely influenced by this philosophy, and therefore it is designed to be as easy to use as possible.

### 3.3.1 Resource Files

Datagen uses a set of resource files with data extracted from DBpedia. Conceptually, Datagen generates attribute's values following a property dictionary model that is defined by

- a dictionary  $D$
- a ranking function  $R$
- a probability function  $F$

Dictionary  $D$  is a fixed set of values. The ranking function  $R$  is a bijection that assigns to each value in a dictionary a unique rank between 1 and  $|D|$ . The probability density function  $F$  specifies how the data generator chooses values from dictionary  $D$  using the rank for each term in the dictionary. The idea to have a separate ranking and probability function is motivated by the need of generating correlated values: in particular, the ranking function is typically parameterized by some parameters: different parameter values result in different rankings. For example, in the case of a dictionary of property `firstName`, the popularity of first names might depend on the gender, country and birthday properties. Thus, the fact that the popularity of first names in different countries and times is different, is reflected by the different ranks produced by function  $R$  over the full dictionary of names. Datagen uses a dictionary for each literal property, as well as ranking functions for all literal properties. These are materialized in a set of resource files, which are described in Table 3.11.

### 3.3.2 Graph Generation

Figure 3.2 conceptually depicts the full data generation process. The first step loads all the dictionaries and resource files, and initializes the Datagen parameters. Second, it generates all the Persons in the graph, and the minimum necessary information to operate. Part of this information are the interests of the persons, and the number of knows relationships of every person, which is guided by a degree distribution function similar to that found in Facebook [86].

The next three steps are devoted to the creation of knows relationships. An important aspect of real social networks, is the fact that similar persons (with similar interests and behaviors) tend to be connected. This is known as the Homophily principle [48, 14], and implies the presence of a larger amount of triangles than that expected in a random network. In order to reproduce this characteristic, Datagen generates the edges by means of correlation dimensions. Given a person, the probability to be connected to another person is typically skewed with respect to some similarity between the persons. That is, for a person  $p$  and for a small set of persons that are somehow similar to it, there is a high connectivity probability, whereas for most other persons, this probability is quite low. This knowledge is exploited by Datagen to reproduce correlations.

Given a similarity function  $M(p) : p \rightarrow [0, \infty)$  that gives a score to a person, with the characteristic that two similar persons will have similar scores, we can sort all the persons by function  $M$  and compare a person  $p$

Resource Name	Description
Browsers	Contains a list of web browsers and their probability to be used. It is used to set the browsers used by the users.
Cities by Country	Contains a list of cities and the country they belong. It is used to assign cities to users and universities.
Companies by Country	Contains the set of companies per country. It is used to set the countries where companies operate.
Countries	Contains a list of countries and their populations. It is used to obtain the amount of people generated for each country.
Emails	Contains the set of email providers. It is used to generate the email accounts of persons.
IP Zones	Contains the set of IP ranges assigned to each country. It is used to assign the IP addresses to users.
Languages by Country	Contains the set of languages spoken in each country. It is used to set the languages spoken by each user.
Name by Country	Contains the set of names and the probability to appear in each country. It is used to assign names to persons, correlated with their countries.
Popular places by Country	Contains the set of popular places per country. These are used to set where images attached to posts are taken from.
Surnames' by Country	Contains the set of surnames and the probability to appear in each country. It is used to assign surnames to persons, correlated with their countries.
Tags by Country	Contains a set of tags and their probability to appear in each country. It is used to assign the interests to persons and forums.
Tag Classes	Contains, for each tag, the classes it belongs to.
Tag Hierarchies	Contains, for each tagClass, their parent tagClass.
Tag Matrix	Contains, for each tag, the correlation probability with the other tags. It is used to enrich the tags associated to messages.
Tag Text	Contains, for each tag, a text. This is used to generate the text for messages.
Universities by City	Contains the set of universities per city. It is used to set the cities where universities operate.

Table 3.11: Resource files.

against only the  $K$  neighbouring persons in the sorted array. The consequence of this approach is that similar persons are grouped together, and the larger the distance between two persons indicates a monotonic increase in their similarity difference. In order to choose the persons to connect, Datagen uses a geometric probability distribution that provides a probability for picking persons to connect, that are between 1 and  $K$  positions apart in the similarity ranking.

Similarity functions and probability distribution functions over ranked distance drive what kind of persons will be connected with an edge, not how many. As stated above, the number of friends of a person is determined by a Facebook-like distribution. The edges that will be connected to a person  $p$ , are selected by randomly picking the required number of edges according to the correlated probability distributions as discussed before. In the case that multiple correlations exist, another probability function is used to divide the intended number of edges between the various correlation dimensions. In Datagen, three correlated dimensions are chosen: the first one depends on where the person studied and when, and the second correlation dimension depends on the interests of the person, and the third one is random (to reproduce the random noise present in real data). Thus, Datagen has a Facebook-like distributed node degree, and a predictable (but not fixed) average split between the reasons for creating edges.

In the next step, a person's activity, in the form of forums, posts and comments is created. Datagen reproduces the fact that people with a larger number of friends have a higher activity, and hence post more photos

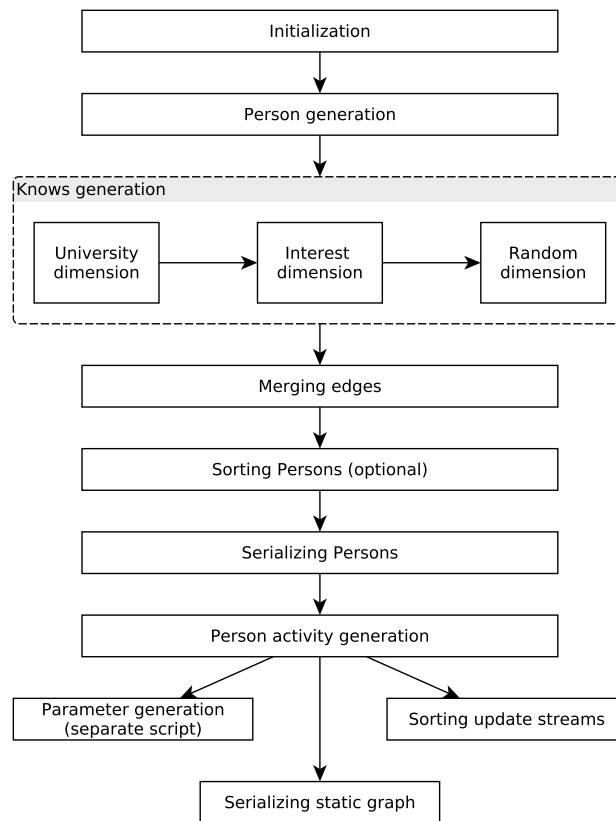


Figure 3.2: The Datagen generation process.

and comments to a larger number of posts. Another important characteristic of real persons' activity in social network, are time correlations. Usually, person's posts creation in a social network is driven by real world events. For instance, one may think about an important event such as the elections in a country, or a natural disaster. Around the time these events occur, network activity about these events' topics sees an increase in volume. Datagen reproduces these characteristics with the simulation of what we name as flashmob events. Several events are generated randomly at the beginning of the generation process, which are assigned a random tag, and are given a time and an intensity which represents the repercussion of the event in the real world. When persons' posts are created, some of them are classified as flashmob posts, and their topics and dates are assigned based on the generated flashmob events. The volume of activity around this events is modeled following a model similar to that described in [45]. Furthermore, in order to reproduce the more uniform every day person activity, Datagen also generates posts uniformly distributed along all the simulated time.

Finally, in the last step the data is serialized into the output files.

### 3.3.3 Distributions, Parameters, and Quirks

Interesting quirks:

- A Person is not a member of their Wall but they are its moderator, they do not have a hasMember edge.
- Each Album generated for Person will have approximately 70% of their friends as members.
- A given Person has a 5% chance of being a moderator of a set of groups.
- Group membership is composed of 30% from the moderator's friends and the remainder is chosen randomly (from the block the person is in).
- Comments are only made in Walls and Groups.
- Messages can only receive likes during a 7-day window after their creation.

- Comments always occur within one day of Message they are replying to. The creation date is generated following the power-law distribution in Figure 3.3. The mean delay between Comments and their parent Message is 6.85 hours.
- Flashmob events span a 72-hour time window with a specific event time in the middle of the window; there are 36 hours each side of the specific event time. Following the distribution in Figure 3.4 posts are generated either side of flashmob event time, posts are clustered around the specific event time.

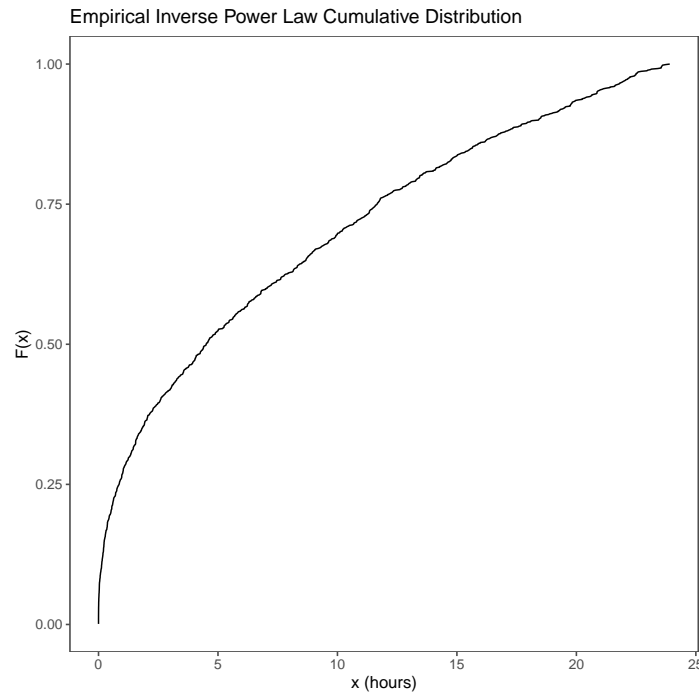


Figure 3.3: The power-law used to generate comments.

### 3.3.4 Implementation Details

Datagen is implemented using the MapReduce parallel paradigm. In MapReduce, a Map function runs on different parts of the input data, in parallel and on many node clusters. This function processes the input data and produces for each result a key. Reduce functions then obtain this data and Reducers run in parallel on many cluster nodes. The produced key simply determines the Reducer to which the results are sent. The use of the MapReduce paradigm allows the generator to scale considerably, allowing the generation of huge datasets by using clusters of machines.

In the case of Datagen, the overall process is divided into three MapReduce jobs. In the first job, each mapper generates a subset of the persons of the graph. A key is assigned to each person using one of the similarity functions described above. Then, reducers receive the key-value pairs sorted by the key, generate the knows relations following the described windowing process, and assign to each person a new key based on another similarity function, for the next MapReduce pass. This process can be successively repeated for additional correlation dimension. Finally, the last reducer generates the remaining information such as forums, posts and comments.

## 3.4 Output Data

For each scale factor, Datagen produces three different artefacts:

- **Dataset:** The dataset to be bulk loaded by the SUT. In the Interactive workload, it corresponds to roughly the 90% of the total generated network.

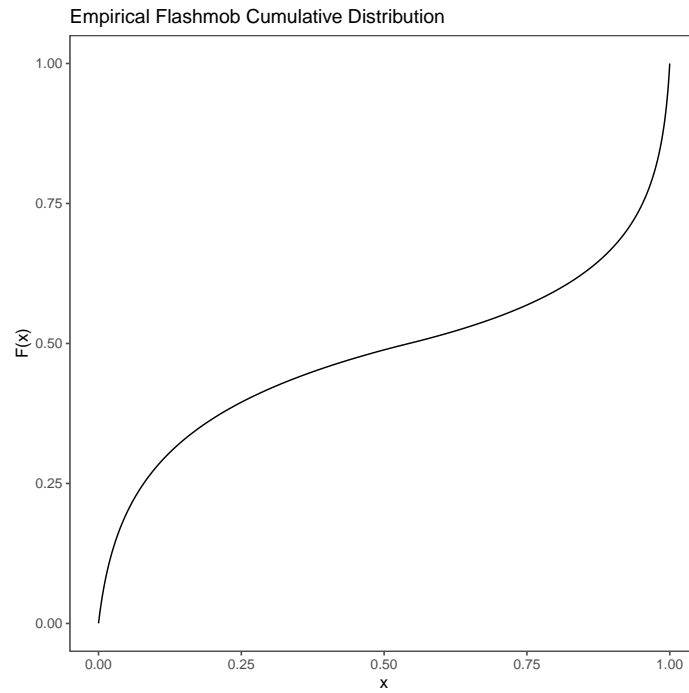


Figure 3.4: The distribution used to generate posts during flashmob events.

- **Update Streams:** A set of update streams containing update queries, which are used by the driver to generate the update queries of the workloads. This update streams correspond to the remaining 10% of the generated dataset.
- **Substitution Parameters:** A set of files containing the different parameter bindings that will be used by the driver to generate the read queries of the workloads.

### 3.4.1 Scale Factors

LDBC SNB defines a set of scale factors (SFs), targeting systems of different sizes and budgets. SFs are computed based on the ASCII size *in Gibibytes* of the generated output files using the csv-singular-merged-fk serializer (see Section 3.4.2).<sup>2</sup> For both workloads, the SF1 data set is 1 GiB, SF100 is 100 GiB, and SF10 000 is 10 000 GiB (not 10 TiB).

However, **note that the two SNB workloads have different data sets** due with different folder structures.

The data sets sizes are established as follows: For both workloads, we use the default settings for the splitting the data into an initial (bulk-loaded) data set and the later update operations (“streams”). For Interactive, both the 90% initial data and the 10% update streams count towards the total size and the csv-singular-merged-fk serializer is used. For BI, the sum of the initial snapshot (97%) and the update operations (daily inserts and deletes) are measured and the default CSV serializers (composite-merged-fk) is used.

It is important to note that for a given workload and scale factor, data sets generated using different serializers contain exactly the same data, the only difference is in how they are represented.

The currently available SFs are the following: 1, 3, 10, 30, 100, 300, 1 000, 3 000, 10 000, 30 000. Additionally, three small SFs, 0.003, 0.1, and 0.3 are provided to help initial testing and validation efforts.

The Test Sponsor may select the SF that better fits their needs, by properly configuring the Datagen, as described in Section 3.3. The size of the resulting dataset is mainly affected by the following configuration parameters: the number of persons and the number of years simulated. By default, all SFs are defined over a period of three years, starting from 2010, and SFs are computed by scaling the number of Persons in the network. Table 3.12 shows the number of entities for SFs 1, ..., 30 000 data sets.

<sup>2</sup>This way of characterizing the size of the data set is identical to the scaling of TPC-H and TPC-DS.

File	SF1	SF3	SF10	SF30	SF100	SF300	SF1000	SF3000	SF10000	SF30000
static/Organisation	7955	7955	7955	7955	7955	7955	7955	7955	7955	7955
static/Place	1460	1460	1460	1460	1460	1460	1460	1460	1460	1460
static/Tag	16080	16080	16080	16080	16080	16080	16080	16080	16080	16080
static/TagClass	71	71	71	71	71	71	71	71	71	71
dynamic/Comment	2 391 707	7 275 929	24 318 240	71 971 437	238 859 896	698 717 507	2 305 141 269	6 788 314 573	22 203 530 429	68 078 584 186
dynamic/Comment_hasTag_Tag	2 903 970	8 957 968	30 193 298	90 186 505	300 936 421	885 843 849	2 934 823 389	8 669 809 939	28 414 179 030	87 250 551 072
dynamic/Forum	106 594	259 629	705 629	1 754 332	4 876 750	12 314 071	35 084 033	92 411 437	272 234 669	770 847 855
dynamic/Forum_hasMember_Person	3 260 692	9 831 062	33 637 572	100 176 831	336 799 532	992 219 233	3 299 845 513	9 734 943 439	31 952 684 743	98 131 214 167
dynamic/Forum_hasTag_Tag	342 040	841 153	2 294 050	5 682 315	15 787 515	39 868 135	113 622 479	299 293 084	881 501 639	2 495 628 126
dynamic/Person	10 620	25 870	70 800	175 950	487 700	1 230 500	3 505 000	9 232 000	27 200 000	77 000 000
dynamic/Person_hasInterest_Tag	246 066	607 394	1 659 221	4 103 933	11 398 465	28 784 564	82 043 446	216 113 647	636 466 970	1 801 780 271
dynamic/Person_knows_Person	219 450	668 431	2 304 951	6 880 584	23 116 805	68 313 982	227 125 539	670 962 543	2 201 852 957	6 763 316 230
dynamic/Person_likes_Comment	1 616 891	5 469 630	20 401 119	66 391 084	243 335 846	776 234 551	2 796 244 391	8 801 761 184	30 518 383 179	97 396 567 634
dynamic/Person_likes_Post	844 544	2 659 885	9 328 362	29 137 595	105 650 858	335 953 318	1 210 202 589	3 822 741 245	13 258 168 236	42 113 297 722
dynamic/Person_studyAt_University	8 562	20 755	56 777	140 829	390 266	984 945	2 804 285	7 386 305	21 760 681	61 607 278
dynamic/Person_workAt_Company	22 766	55 826	154 122	383 107	1 061 627	2 678 190	7 627 121	20 093 569	59 188 556	167 544 307
dynamic/Post	1 192 942	3 056 157	8 781 335	22 948 816	67 764 850	181 024 990	548 192 276	1 516 905 453	4 693 293 319	13 820 145 527
dynamic/Post_hasTag_Tag	778 511	2 384 596	8 112 750	24 116 550	80 572 324	237 819 624	789 063 560	2 330 311 354	7 634 983 368	23 442 869 026

Table 3.12: Properties of data sets for each scale factor for the *raw data sets* produced the Spark-based generator, used as a basis of the data sets of SNB Interactive v2 and SNB BI.

Serializer name (v2.x)	Legacy serializer name (v0.x and v1.x)	Nodes	Attributes		Edges	
			single-valued	multi-valued	one-to-many	many-to-many
csv-singular-projected-fk	CsvBasic	⊗	○	⊗	⊗	⊗
csv-composite-projected-fk	CsvComposite	⊗	○	○	⊗	⊗
csv-singular-merged-fk	CsvMergeForeign	⊗	○	⊗	○	⊗
csv-composite-merged-fk	CsvCompositeMergeForeign	⊗	○	○	○	⊗

Table 3.13: Attributes and edges serialized to separate files the different CSV serializers.

Table 3.13 shows how each CSV serializer handles attributes/edges of different cardinalities, demonstrating why *csv-singular-projected-fk* has the most files and *csv-composite-merged-fk* has the least number of files.

### 3.4.2 Serializers

The datasets are generated in the `social_network/` directory, split into static and dynamic parts (Figure 3.1). The filenames (without the extension) end in `_i_j` where `i` is the block id and `j` is the partition id (set by `numThreads`). The SUT has to take care only of the generated Dataset to be bulk loaded. Using `NULL` values for storing optional values is allowed.

Datagen currently only supports CSV-based serializers. These produce CSV-like text files which use the pipe character “|” as the primary field separator and the semicolon character “;” as a separator for multi-valued attributes (for the composite serializers). The CSV files are stored in two subdirectories: `static/` and `dynamic/`. Depending on the number of threads used for generating the dataset, the number of files varies, since there is a file generated per thread. We indicate this with “`part-*`” in the specification.

The following CSV variants are supported:

- *csv-composite-projected-fk*: Each relation with a cardinality larger than one are output in a separate file. Generated files and their schemas as shown in Table 3.14.
- *csv-composite-merged-fk*: This serializer is similar to *csv-composite-projected-fk*, but relations that have a cardinality of 1-to-N are merged in the entity files as a foreign keys. There are 13 such relations in total:
  - `Comment_hasCreator_Person`, `Comment_isLocatedIn_Country`, `Comment_replyOf_Comment`, `Comment_replyOf_Post` (merged to `Comment`);
  - `Forum_hasModerator_Person` (merged to `Forum`);
  - `Organisation_isLocatedIn_Place` (merged to `Organisation`);
  - `Person_isLocatedIn_City` (merged to `Person`);
  - `Place_isPartOf_Place` (merged to `Place`);
  - `Post_hasCreator_Person`, `Post_isLocatedIn_Country`, `Forum_containerOf_Post` (merged to `Post`);
  - `Tag_hasType_TagClass` (merged to `Tag`);
  - `TagClass_isSubclassOf_TagClass` (merged to `TagClass`)

C	File	Content
N	static/Organisation/part-*.csv	id   type   name   url
E	static/Organisation_isLocatedIn_Place/part-*.csv	OrganisationId   PlaceId
N	static/Place/part-*.csv	id   name   url   type
E	static/Place_isPartOf_Place/part-*.csv	Place1Id   Place2Id
N	static/Tag/part-*.csv	id   name   url
E	static/Tag_hasType_TagClass/part-*.csv	TagId   TagClassId
N	static/TagClass/part-*.csv	id   name   url
E	static/TagClass_isSubclassOf_TagClass/part-*.csv	TagClass1Id   TagClass2Id
N	dynamic/Comment/part-*.csv	creationDate   id   locationIP   browserUsed   content   length
E	dynamic/Comment_hasCreator_Person/part-*.csv	creationDate   CommentId   PersonId
E	dynamic/Comment_hasTag_Tag/part-*.csv	creationDate   CommentId   TagId
E	dynamic/Comment_isLocatedIn_Country/part-*.csv	creationDate   CommentId   CountryId
E	dynamic/Comment_replyOf_Comment/part-*.csv	creationDate   Comment1Id   Comment2Id
E	dynamic/Comment_replyOf_Post/part-*.csv	creationDate   CommentId   PostId
N	dynamic/Forum/part-*.csv	creationDate   id   title
E	dynamic/Forum_containerOf_Post/part-*.csv	creationDate   ForumId   PostId
E	dynamic/Forum_hasMember_Person/part-*.csv	creationDate   ForumId   PersonId
E	dynamic/Forum_hasModerator_Person/part-*.csv	creationDate   ForumId   PersonId
E	dynamic/Forum_hasTag_Tag/part-*.csv	creationDate   ForumId   TagId
N	dynamic/Person/part-*.csv	creationDate   id   firstName   lastName   gender   birthday   locationIP   browserUsed   language   email
E	dynamic/Person_hasInterest_Tag/part-*.csv	creationDate   personId   interestId
E	dynamic/Person_isLocatedIn_City/part-*.csv	creationDate   PersonId   CityId
E	dynamic/Person_knows_Person/part-*.csv	creationDate   Person1Id   Person2Id
E	dynamic/Person_likes_Comment/part-*.csv	creationDate   PersonId   CommentId
E	dynamic/Person_likes_Post/part-*.csv	creationDate   PersonId   PostId
E	dynamic/Person_studyAt_University/part-*.csv	creationDate   PersonId   UniversityId   classYear
E	dynamic/Person_workAt_Company/part-*.csv	creationDate   PersonId   CompanyId   workFrom
N	dynamic/Post/part-*.csv	creationDate   id   imageFile   locationIP   browserUsed   language   content   length
E	dynamic/Post_hasCreator_Person/part-*.csv	creationDate   PostId   PersonId
E	dynamic/Post_hasTag_Tag/part-*.csv	creationDate   PostId   TagId
E	dynamic/Post_isLocatedIn_Country.csv	creationDate   PostId   CountryId

Table 3.14: Files output by the csv-composite-projected-fk serializer (31 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge.

Generated files and their schemas as shown in Table 3.15.

- **csv-singular-merged-fk:** Similar to the csv-composite-merged-fk but multi-valued attributes (Person.email and Person.speaks) are stored as separate directories (Person\_email\_EmailAddress and Person\_speaks\_Language, resp.).
- **csv-singular-projected-fk:** Similar to the csv-composite-projected-fk but multi-valued attributes (Person.email and Person.speaks) are stored as separate directories (Person\_email\_EmailAddress and Person\_speaks\_Language, resp.).
- **raw mode:** The file names are the same as in composite-merged-fk but there are two important differences: (1) additional attributes, e.g. deletionDate, explicitlyDeleted, and weight (used for weighted graphs in Graphalytics [39]), are included, (2) all data is included, i.e. if a Forum is created and deleted before sampling the initial data set, it is included in this data set. Generated files and their schemas as shown in Table 3.16.

**Inheritance** The inheritance hierarchies in the schema have two important characteristics (1) all subclasses use the same id space, e.g. there cannot be a Comment and a Post with id 1 at the same time, (2) they are serialized to CSVs using either the *map hierarchy to single table* or *map each concrete class to its own table* strategies<sup>3</sup>:

**Message = Comment | Post** *Map each concrete class to its own table* is used i.e. separate CSV files are used for the Comment and the Post classes.

<sup>3</sup><http://www.agiledata.org/essays/mappingObjects.html>

C	File	Content
N	static/Organisation/part-*.csv	id   type   name   url   LocationPlaceId
N	static/Place/part-*.csv	id   name   url   type   PartOfPlaceId
N	static/Tag/part-*.csv	id   name   url   TypeTagClassId
N	static/TagClass/part-*.csv	id   name   url   SubclassOfTagClassId
N	dynamic/Comment/part-*.csv	creationDate   id   locationIP   browserUsed   content   length   CreatorPersonId   LocationCountryId   ParentPostId   ParentCommentId
E	dynamic/Comment_hasTag_Tag/part-*.csv	creationDate   CommentId   TagId
N	dynamic/Forum/part-*.csv	creationDate   id   title   ModeratorPersonId
E	dynamic/Forum_hasMember_Person/part-*.csv	creationDate   ForumId   PersonId
E	dynamic/Forum_hasTag_Tag/part-*.csv	creationDate   ForumId   TagId
N	dynamic/Person/part-*.csv	creationDate   id   firstName   lastName   gender   birthday   locationIP   browserUsed   LocationCityId   language   email
E	dynamic/Person_hasInterest_Tag/part-*.csv	creationDate   personId   interestId
E	dynamic/Person_knows_Person/part-*.csv	creationDate   Person1Id   Person2Id
E	dynamic/Person_likes_Comment/part-*.csv	creationDate   PersonId   CommentId
E	dynamic/Person_likes_Post/part-*.csv	creationDate   PersonId   PostId
E	dynamic/Person_studyAt_University/part-*.csv	creationDate   PersonId   UniversityId   classYear
E	dynamic/Person_workAt_Company/part-*.csv	creationDate   PersonId   CompanyId   workFrom
N	dynamic/Post/part-*.csv	creationDate   id   imageFile   locationIP   browserUsed   language   content   length   CreatorPersonId   ContainerForumId   LocationCountryId
E	dynamic/Post_hasTag_Tag/part-*.csv	creationDate   PostId   TagId

Table 3.15: Files output by the csv-composite-merged-fk serializer (18 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge.

**Place = City | Country | Continent** *Map hierarchy to single table* is used, i.e. all Place node are serialized in a single file. A discriminator attribute “type” is used with the value denoting the concrete class, e.g. “Country”.

**Organisation = Company | University** *Map hierarchy to single table* is used, i.e. all Organisation nodes are serialized in a single file. A discriminator attribute “type” is used with the value denoting the concrete class, e.g. “Company”.

### 3.4.3 Interactive Update Streams (Inserts)

The generic schema for the Interactive update streams is given in Table D.8, while the concrete schemas of each insert operations is given in Table D.9. The update stream files are generated in the `social_network/` directory and are grouped as follows:

- updateStream\_\*\_person.csv files contain update operation 1: **INS 1**
- updateStream\_\*\_forum.csv files contain update operations 2–8: **INS 2** **INS 3** **INS 4** **INS 5** **INS 6** **INS 7** **INS 8**

Remark: update streams in version 1 only contain inserts, while in version 2, they contain both inserts and deletes.

### 3.4.4 Substitution Parameters

The substitution parameters are generated in the `substitution_parameters/` directory. Each parameter file is named `{interactive|bi}<id>_param.txt`, corresponding to an operation of Interactive complex reads ( **IC 1** – **IC 14v2** ) and BI reads ( **BI 1** – **BI 20** ). The schemas of these files are defined by the operator, e.g. the schema of **IC 1** is “personId|firstName”.

## 3.5 Introducing Delete Operations

**Challenge for systems** To support deletion operations graph processing systems need to solve numerous technical challenges:



C	File	Content
N	static/Organisation/part-*.csv	id   type   name   url   LocationPlaceId
N	static/Place/part-*.csv	id   name   url   type   PartOfPlaceId
N	static/Tag/part-*.csv	id   name   url   TypeTagClassId
N	static/TagClass/part-*.csv	id   name   url   SubclassOfTagClassId
N	dynamic/Comment/part-*.csv	creationDate   deletionDate   explicitlyDeleted   id   locationIP   browserUsed   content   length   CreatorPersonId   LocationCountryId   ParentPostId   ParentCommentId
E	dynamic/Comment_hasTag_Tag/part-*.csv	creationDate   deletionDate   CommentId   TagId
N	dynamic/Forum/part-*.csv	creationDate   deletionDate   explicitlyDeleted   id   title   ModeratorPersonId
E	dynamic/Forum_hasMember_Person/part-*.csv	creationDate   deletionDate   explicitlyDeleted   ForumId   PersonId
E	dynamic/Forum_hasTag_Tag/part-*.csv	creationDate   deletionDate   ForumId   TagId
N	dynamic/Person/part-*.csv	creationDate   deletionDate   explicitlyDeleted   id   firstName   lastName   gender   birthday   locationIP   browserUsed   LocationCityId   language   email
E	dynamic/Person_hasInterest_Tag/part-*.csv	creationDate   deletionDate   personId   interestId
E	dynamic/Person_knows_Person/part-*.csv	creationDate   deletionDate   explicitlyDeleted   Person1Id   Person2Id
E	dynamic/Person_likes_Comment/part-*.csv	creationDate   deletionDate   explicitlyDeleted   PersonId   CommentId
E	dynamic/Person_likes_Post/part-*.csv	creationDate   deletionDate   explicitlyDeleted   PersonId   PostId
E	dynamic/Person_studyAt_University/part-*.csv	creationDate   deletionDate   PersonId   UniversityId   classYear
E	dynamic/Person_workAt_Company/part-*.csv	creationDate   deletionDate   PersonId   CompanyId   workFrom
N	dynamic/Post/part-*.csv	creationDate   deletionDate   explicitlyDeleted   id   imageFile   locationIP   browserUsed   language   content   length   CreatorPersonId   ContainerForumId   LocationCountryId
E	dynamic/Post_hasTag_Tag/part-*.csv	creationDate   deletionDate   PostId   TagId

Table 3.16: Directories created by the raw serializer (18 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge. The entities with the `explicitlyDeleted` attribute – Comment, Forum, Post nodes, and `hasMember`, `knows`, `likes` (Comment/Post) edges – denote whether the entity is deleted as part of an explicit delete operation or implicitly through a cascading delete operation.

1. Users should be able to *express deletion operations* using the database API, preferably using a high-level declarative query language with clear semantics [31].
2. Deletion operations *limit the algorithms and data structures* that can be used by a system. Certain dynamic graph algorithms are significantly more expensive to recompute in the presence of deletes [71] or only support either insert or deletions but not both [72]. A number of updatable matrix storage formats only support efficient insertions but not deletions [18]. Meanwhile some graph databases might be able to exploit indices to speed up deletions [15, Sec. 4.4.2]
3. *Distributed graph databases* need to employ specialized protocols to enforce consistency of deletions [88].

**Challenge for benchmarks** Due to their importance and challenging nature, we found it necessary to incorporate delete operations into LDBC benchmarks. However, doing so is a non-trivial task as it impacts on each component in the benchmark workflow: workload specifications, data generation, parameter curation, and the workload driver. This section focuses primarily on data generation.

The initial step in generating delete operations is to define the semantics of the desired operations. To understand common behaviour of deletes we informally surveyed several social networks, the findings of which motivated the design of 8 delete operations described in Section 8.6.

The next step was to generate *delete events* within LDBC’s synthetic data generator and ensure that they follow a logic order in the social network. For example, a delete `knows` edge event can only occur after both Persons join the network and become friends, but before either Person leaves the network. To achieve this Datagen was extended to support *dynamic entities*. Dynamic entities have a *creation date* and a *deletion date*, which together comprise an entity’s *lifespan*. Once generated this allows for the extraction of deletion operations, which can be utilized by LDBC workloads. Deriving valid lifespans for dynamic entities was the subject of a short paper published at the GRADES-NDA 2020 workshop [89] and is presented in Section 3.6.

Next it was important to distinguish between *implicit* and *explicit* delete events. Continuing with the `knows` edge example, once created the connection exists until either Person leaves the network, at which point the connection is implicitly deleted, as per the semantics of delete Person (Section 5.2). Alternatively, at any time

up until this event, the friendship can be explicitly deleted, i.e. two people have a disagreement and “unfriend” each other, but both continue using the social network. Distinguishing between these types of events is important as only explicit delete events should become delete operations in the workload.

To achieve this each dynamic entity is assigned a probability of being explicitly deleted, if selected the entity is marked as such; this is used to ensure the correct serialization of delete events into delete operations. For entities selected for explicit deletion the next step is to determine a realistic time at which the event occurs. For example, a post has a higher probability of being deleted soon after it was posted compared to after 5 days. To achieve this each dynamic entity is assigned a realistic distribution to select delete event timestamps from, which respects the bounds imposed by the valid lifespans. The probability distributions used to determine if a dynamic entities is explicitly deleted and then when that event occurs is discussed in Section 3.7.

Once generated dynamic entities must be correctly serialized. Depending on its creation date, deletion date, and if the entity is explicitly deleted it can, (i) spawn an insert and delete operation, (ii) be included in the bulk load component and spawn a delete operation, (iii) just be included in the bulk load component, (iv) spawn only an insert operation, and (v) not be serialized at all! The approach for doing this is presented in Section 3.8.

We summarize the numerous challenges supporting the generation of dynamic entities and thus delete operations poses below:

1. **Validity.** The generator should produce *valid lifespans*, where each generated dynamic entity guarantees that (a) events in the graph follow a logical order: e.g. in a social network, two people can become friends only after both persons joined the network and before either person leaves the network, (b) the graph never violates the cardinality constraints prescribed by its schema, and (c) the graph continuously satisfies the semantic constraints required by the application domain (e.g. no isolated comments in a social network).
2. **Realism.** The generator should create a graph with a realistic correlations and distribution of entities over time. For example, in a social network the distribution of activity is non-uniform over time, real-world events such as elections or controversial posts can drive spikes of posts and unfollowings respectively [55]. In addition, deletions can be correlated with certain attributes: e.g. the likelihood a person leaves the network may be correlated with their number of friends [47]. Also, there are often temporal correlations between entity creation and deletion: e.g. posts have an increased chance of deletion immediately following creation compared to after a 3 month period.
3. **Serialization.** Care must be taken to distinguish between implicit and explicit delete events when creating the bulk load component, insert operations, and delete operations.
4. **Scalability.** A graph with dynamic entities should be generated at scale (up to billions of edges).

## 3.6 Lifespan Management

This section is based on the short paper published at the GRADES-NDA 2020 workshop [89] authored by the task force members.

In this section, we define the constraints for generating dynamic entities in a social network. Each dynamic entity gets a *lifespan*, represented by two *lifespan attributes*, a *creation date* and a *deletion date*. We first briefly review the data generator, introduce our notation and define the parameters of the generation process. Then, we define the semantic constraints which regulate the participation in certain relationships along with the constraints for selecting intervals. We illustrate an application of these with two examples, shown in Figure 3.5 and Figure 3.6.

**Graph schema** The LDBC Datagen component [62, 63] is responsible for generating the graph used in the benchmarks. It produces a synthetic dataset modelling a social network’s activity. Its graph schema has 11 concrete node types connected by 20 edge types, and its entities (nodes/edges) are classified as either dynamic or static (Figure 3.1). The dynamic part of the graph comprises of a fully connected Person graph and a number of Message trees under Forums.

**Notation** To describe lifespans and related constraints, we use the following notation. Constants are uppercase bold, e.g. **NC**. Entity types are monospaced, e.g. `Person`, `hasMember`. Variables are lowercase italic, e.g. *pers*, *hm*. Entities are sans-serif, e.g. `P1`, `HM`. For an entity  $x$ ,  $*x$  denotes its creation date, while  $\dagger x$  denotes its deletion date. In most cases, both the creation and the deletion date are selected from an interval, e.g.  $*x \in [d_1, d_2)$  means that entity  $x$  should be created between dates  $d_1$  (inclusive) and  $d_2$  (exclusive). The selected creation and deletion dates together form an interval that represents the lifespan of its entity. If any of the intervals for selecting the lifespan attributes of an entity are empty, i.e.  $d_2 \leq d_1$ , the entity should be discarded. As illustrated later, this interval is often used to determine the intervals where the creation and deletion dates of dependant entities are selected.

**Parameters** We parameterize the generator as follows. The network is created in 2010 and exists for 10 years at which point the network collapses (**NC** = 2020). Data is simulated for a 3-year period, between the simulation start, **SS** = 2010 and the simulation end, **SE** = 2013. In order to allow *windowed execution* by the LDBC SNB driver (used for multi-threaded and distributed operation), we define a sufficiently large amount of time that needs to pass between consecutive operations on an entity as  $\Delta = 10s$ .

### 3.6.1 General Rules

In this section, we define general rules that must be satisfied by all entities in the graph. In subsequent sections, we refine these with domain-specific constraints. For a node  $n_1$ , we always require that:

- $*n_1 \in [\text{SS}, \text{SE})$ , the node must be created between the simulation start and the simulation end.
- $\dagger n_1 \in [*n_1 + \Delta, \text{NC})$ , the node must exist for at least  $\Delta$  time and must be deleted before the network collapse.

To enforce referential integrity constraints (i.e. prevent dangling edges), the lifespan of edge  $e$  between nodes  $n_1$  and  $n_2$  must always satisfy the following criteria:

- $*e \in [\max(*n_1, *n_2), \min(\dagger n_1, \dagger n_2, \text{SE}))$ , in other terms, the edge must be created no sooner than both of its endpoints but before any of its endpoints are deleted.
- $\dagger e \in [*e + \Delta, \min(\dagger n_1, \dagger n_2))$ , i.e. the edge must exist for at least  $\Delta$  time and deleted no later than any of its endpoints.

To further refine the constraints for edges, we distinguish between two main cases.

(1) The endpoints of edge  $e$  are existing node  $n_1$  and node  $n_2$  which is inserted at the same time as the edge:

- $*e = *n_2$
- $\dagger e = \min(\dagger n_1, \dagger n_2)$ . In case of edges with *containment semantics* (node  $n_1$  contains  $n_2$  through edge  $e$ ), node  $n_2$  must always be deleted at the same time as edge  $e$ , therefore  $\dagger e = \dagger n_2$  and  $\dagger n_2 \leq \dagger n_1$ .

(2) In other cases, the edge must be created when both of its endpoints already exist and must be deleted no later than them:

- $*e \in [\max(*n_1, *n_2) + \Delta, \min(\dagger n_1, \dagger n_2, \text{SE}))$
- $\dagger e \in [*e + \Delta, \min(\dagger n_1, \dagger n_2))$

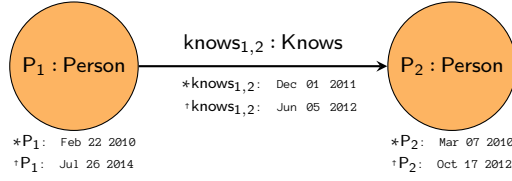
These constraints capture the “minimum” (i.e. most relaxed) set of constraints that must be enforced in all domains. Next, we introduce additional constraints specific to our social network schema.

### 3.6.2 Person

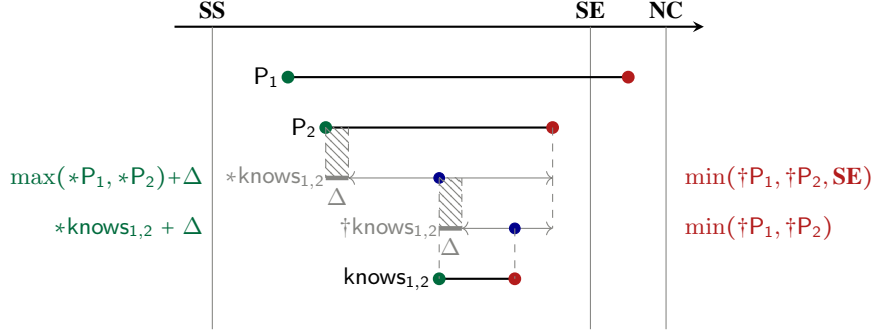
A Person  $p$  is the avatar a real-world person creates when they join the network. A Person joins the network,  $*p$ , during the simulation period and they leave the network,  $\dagger p$ , during the network lifetime:

- $*p \in [\text{SS}, \text{SE})$
- $\dagger p \in [*p + \Delta, \text{NC})$

For the edges of Person nodes pointing to a static node (isLocatedIn, studyAt, workAt, and hasInterest), we assign the creation and deletion date from  $*p$  and  $\dagger p$ , resp.



(a) An instance of a knows edge connecting two Person nodes. *Creation* and *deletion* dates are shown for each entity.



(b) Illustration of the intervals in which the *creation dates*  $\bullet$  and the *deletion dates*  $\bullet$  can be selected. Thick black lines represent entity lifespans and thin grey lines represent valid intervals that dates can be selected in;  $\bullet$  indicates the selected times (spanning the lifespan interval of the given entity). On the thin grey lines, thicker sections represent the minimal amount of time that must pass before selecting a value. In case of creation dates, this is used to ensure that the dependant entity exists for at least  $\Delta$  time. In case of deletion dates, it is used to ensure that the entity exists for at least  $\Delta$  time.

Figure 3.5: Example graph and its intervals.

### 3.6.2.1 Knows

The knows edge connects two Persons  $p_i$  and  $p_j$  that know each other in the network. The intervals where the creation and deletion dates can be generated in are illustrated in Figure 3.5b and defined below:

- $*knows_{i,j} \in [\max(*p_i, *p_j) + \Delta, \min(\dagger p_i, \dagger p_j, SE))$
- $\dagger knows_{i,j} \in [*knows_{i,j} + \Delta, \min(\dagger p_i, \dagger p_j))$

### 3.6.3 Forum and Message

The rules for Forum and Message nodes along with their edges are given in Section 3.6.4 and Section 3.6.5, respectively, and illustrated in Figure 3.6.

### 3.6.4 Forum

A Forum is a meeting point where people post Messages. There exists three categories of Forums: Wall ( $forum_w$ ), Album ( $forum_a$ ), and Group ( $forum_g$ ). Each Forum has a set of Persons connected via hasMember edges, a set of Tags connected via hasTag edges, a single moderator connected by a hasModerator edge and a set of Messages (discussed in Section 3.6.5). For all Forums the outgoing hasTag edges get their creation date and deletion date from  $*forum$  and  $\dagger forum$ , respectively.

#### 3.6.4.1 Groups

Groups are public places for people that share interests, any Person can create a Group  $forum_g$  during their lifespan. A Group can be deleted anytime after it was created.

- $*forum_g \in [*p + \Delta, \min(\dagger p, SE))$

- $\dagger forum_g \in [ *forum_g + \Delta, \text{NC})$

**Group Moderator** The initial hasModerator  $hmd_g$  is the Group creator. If the moderator leaves the Group, the Group will have no moderator (this is allowed in the schema of version 0.4.0+, see Figure 3.1).

- $*hmd_g \in [ *forum_g + \Delta, \min(\dagger forum_g, \dagger p, \text{SE}))$
- $\dagger hmd_g \in [ *hmd_g + \Delta, \min(\dagger forum_g, \dagger p))$

**Group Membership** Any Person  $p$  can become a member of a given group. The hasMember  $hm_g$  creation is generated from the interval in which the Person and Forum lifespans overlap. The deletion date is generated from the interval between the membership creation date (incremented by  $\Delta$ ) and the minimum of the Person and Forum deletion dates.

- $*hm_g \in [ \max(*forum_g, *p) + \Delta, \min(\dagger forum, \dagger p, \text{SE}))$
- $\dagger hm_g \in [ *hm_g + \Delta, \min(\dagger forum_g, \dagger p))$

### 3.6.4.2 Walls

Every Person  $p$ , has a Wall  $forum_w$  which is created when the Person joins the social network. The wall is deleted when the Person is deleted.

- $*forum_w = *p + \Delta$
- $\dagger forum_w = \dagger p$

**Wall Moderator** Each Person has a hasModerator  $hmd_w$  edge to their wall, which gets the creation date (incremented by  $\Delta$ ) and deletion date from  $forum_w$ . Note, only the moderator can create Post nodes on the wall and the connecting Tag nodes are set based on the interest of the moderator.

- $*hmd_w = *forum_w + \Delta$
- $\dagger hmd_w = \dagger forum_w$

**Wall Membership** For a Person  $p_i$ , all their friends  $p_j$  (Person nodes connected via a knows edge) become members of  $forum_w$  at the time the knows edge is created. Hence, a hasMember  $hm_w$  edge gets the creation date of knows incremented by  $\Delta$ . The deletion date is derived from the minimum of the Forum deletion date and knows deletion date.

- $*hm_w = *knows_{i,j} + \Delta$
- $\dagger hm_w = \min(\dagger forum_w, \dagger knows_{i,j})$

### 3.6.4.3 Albums

A Person can create multiple Albums ( $forum_a$ ) containing a set of Photos. Albums can be created and then deleted at any point during the lifespan of the Person.

- $*forum_a \in [ *p + \Delta, \min(\dagger p, \text{SE}))$
- $\dagger forum_a \in [ *forum_a + \Delta, \dagger p)$

**Album Moderator** The Person is the moderator for any Album they create. Album ownership cannot change hence hasModerator  $hmd_a$  gets the creation date (incremented by  $\Delta$ ) and deletion date from  $*forum_a$  and  $\dagger forum_a$  respectively.

- $*hmd_a = *forum_a + \Delta$
- $\dagger hmd_a = \dagger forum_a$

**Album Membership** Only friends  $p_i$  of a Person  $p_j$  can become members of Albums created by  $p_j$ . The `hasMember`  $hm_a$  edge creation date is derived from the Album and knows creation dates. The deletion is derived from the Forum and knows deletion dates.

- $*hm_a = \max(*forum_a, *knows_{i,j}) + \Delta$
- $\dagger hm_w = \min(\dagger forum_a, \dagger knows_{i,j})$

### 3.6.5 Message

A Message is an abstract entity that represents a message created by a Person. There are two Message subtypes: Post and Comment. A Post is created in a Forum and a Comment represents a comment made by a Person to an existing Message (either a Post or a Comment). In a Forum the set of Message nodes form a *tree* with a Post node at the root and Comment nodes for the rest.

#### 3.6.5.1 Post

A Post can be created by a Person in a Forum. Only the moderator (i.e. owner) can post on a Wall or in an Album (`hasModerator`), whereas all members including the moderator (`hasMember/hasModerator`) can post in a Group. These relationships are captured with the  $hm$  variable in the formulas. Posts are divided in three categories, *regular posts*, *photos*, and *flashmob posts*.

**Regular Posts and Photos** Regular posts capture the standard daily activity in a Group or on a Wall. Photos are created in Albums. (Interaction with Photos is limited to likes, see details in Section 3.6.5.3). The creation date for these is determined as follows:

$$*post \in [*hm + \Delta, \min(\dagger hm, SE))$$

**Flashmob Posts** Flashmob posts are generated around events that attract significant interest (such as elections) that result in a spike in activity. These events span a  $2\phi$ -hour time window centered around a specific event time, flashmob event  $fme$ , in the middle of the window; there are  $\phi$  hours each side of the specific event time.

$$*post \in [\max(*hm + \Delta, fme - \phi h), \min(\dagger hm, fme + \phi h, SE))$$

The deletion dates for all categories of Posts are determined as:

$$\dagger post \in [*post + \Delta, \dagger hm)$$

**containerOf edge** Each Post node has an incoming `containerOf` edge which gets the same lifespan attributes as the Post.

#### 3.6.5.2 Comment

A Comment  $comm$  is created by Person  $p$  as a reply to Message  $m$ . Comments are only made in Walls and Groups. Comment always occur within  $\gamma$  days of their parent message following a power-law distribution with mean 6.85 hours.

- $*comm \in [\max(*m, *hm) + \Delta, \min(\dagger m, \dagger hm, *m + \gamma d, SE))$
- $\dagger comm \in [*comm + \Delta, \min(\dagger m, \dagger hm))$

**replyOf edge** Comments always have an outgoing `replyOf` edge with containment semantics, i.e. the target Message contains the Comment. These edges get the same lifespan as their source Comment.

### 3.6.5.3 likes

A likes edge *likes* can exist between Person *p* and Message *m*. Messages can only receive likes during a  $\mu$ -day window after their creation at which point no more activity is generated.

- $*likes \in [\max(*p, *m) + \Delta, \min(\dagger p, \dagger m, *m + \mu d, SE))$
- $\dagger likes \in [*likes + \Delta, \min(\dagger p, \dagger m))$

### 3.6.6 Complex Example

In Figure 3.6, a complex example graph is shown with the corresponding intervals. Both *the intervals for selecting the creation and deletion date* attributes and the selected *lifespan intervals* are shown.

## 3.7 Ensuring Realism

Capturing realistic deletion behaviour was broken down into two dimensions. Firstly, each dynamic entity is assigned a probability of being explicitly deleted. Second, if selected for explicit deletion, a deletion event date is selected using a distribution bound by the valid lifespan of that entity. To make informed choices of deletion probabilities and deletion date distributions, where possible, real-world data was used.

**Delete Person** Lorincz et al. [47] have analyzed iWiW, a now-defunct Hungarian social network, observing that people with more connections are less likely to leave a social network. When a Person is generated they are assigned a *maxKnows* value which indicates the amount of knows connections they will make across the lifetime of the network. This information is then utilized to determine the probability a person is explicitly deleted using the distribution provided in [47], reproduced in Figure 3.7. A deletion event date is then selected uniformly from the person’s valid lifespan. On average 3.5% of Persons are deleted across the simulation period.

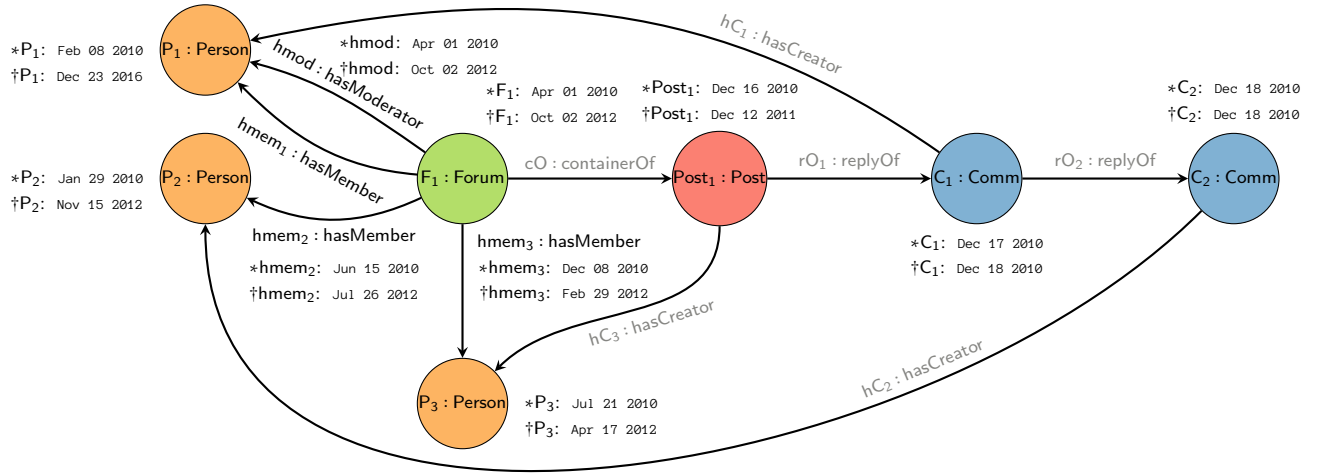
**Delete Knows** Myers and Leskovec [55] analysed 1.2 billion tweets from 13.1 million Twitter users. These users made 112.3 million new connections, and deleted 39.2 million connections; a 3:1 follow:unfollow ratio. As Datagen models a generic social media platform we have chosen a different ratio of 20:1 (on average 5% of knows edges are deleted), rather than overcapture behavior that may be unique to a single site. [55] also finds a constant background flux of follows and unfollows interleaved with bursts in such activity. Currently, Datagen has no follow bursts, thus, we have decided not to incorporate unfollow bursts. They also find less similar friends have a high probability of being unfollowed; modelling this relationship is work in progress. If a knows edge is selected for explicit deletion then a deletion date is then selected uniformly from the edge’s valid lifespan.

**Delete Post/Comment and Delete Post/Comment Like** Posts in groups and walls are produced via a uniform generator and a flashmob generator, capturing background events and bursts in events respectively. A comment generator is then used to produce a tree of comments on each post. Posts in albums are referred to as photos, they are produced by a different generator and do not have flashmob events nor do they have comment trees. Additionally, all posts and comments have a number of likes generated for it.

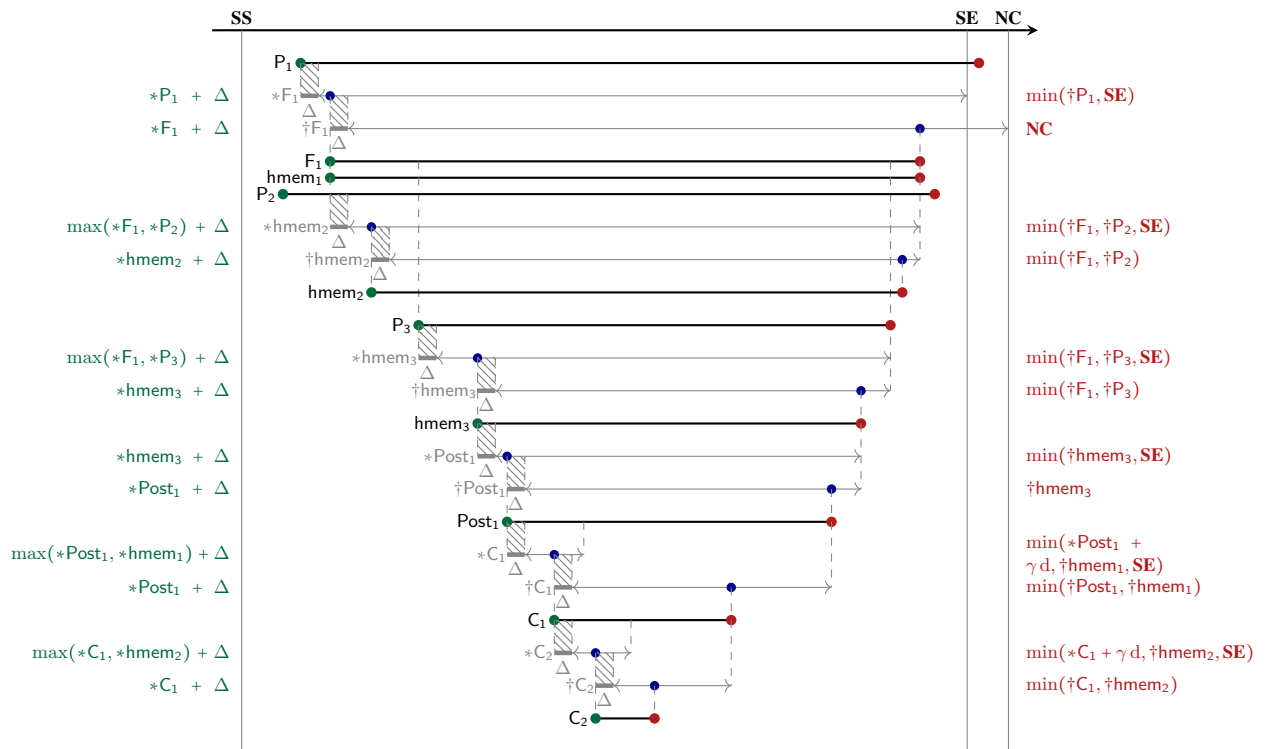
Almuhimedi et al. [2] tracked 292K Twitter users for 1 week. They found 2.4% of 67.2M tweets were deleted across 4 categories: status posts, retweets, replies, and mentions of other users that were not replies. In order to apply these findings to Datagen and obtain the average percentage of Messages and likes deleted across the simulation period, tweet categories were mapped to Datagen Message types. Table 3.17 gives the mapping and the percentage deleted across the simulation period within each category.

Additionally, [2] identified not all users delete messages, with around 50% of users doing so. Thus, each Person in the network has a 50% chance of being marked a *messageDeleter*, who subsequently, may or may not, delete post, comments, or likes. [2] also identify a relationship between the depth of replies to a tweet and the chance the tweet is deleted – a tweet with less replies is more likely to be deleted. We apply this relationship to





(a) Example graph with an instance of a Forum containing a Message tree of depth 3 and its Person members. Lifespan attributes (*creation* and *deletion* dates) are shown for each dynamic entity. Edges in grey get their lifespan attributes as per Figure 3.1 and Section 3.6.4.



(b) Illustration of the intervals in which the *creation* dates (green dots) and the *deletion* dates (red dots) of entities can be selected. Thick black lines represent entity lifespans and thin grey lines represent valid intervals that dates can be selected in; • indicates the selected times (spanning the lifespan interval of the given entity). On the thin grey lines, thicker sections represent the minimal amount of time that must pass before selecting a value. In case of creation dates, this is used to ensure that the dependant entity exists for at least  $\Delta$  time. In case of deletion dates, it is used to ensure that the entity exists for at least  $\Delta$  time.

Figure 3.6: Example graph and time intervals for selecting lifespan attributes, *creation* and *deletion* dates.



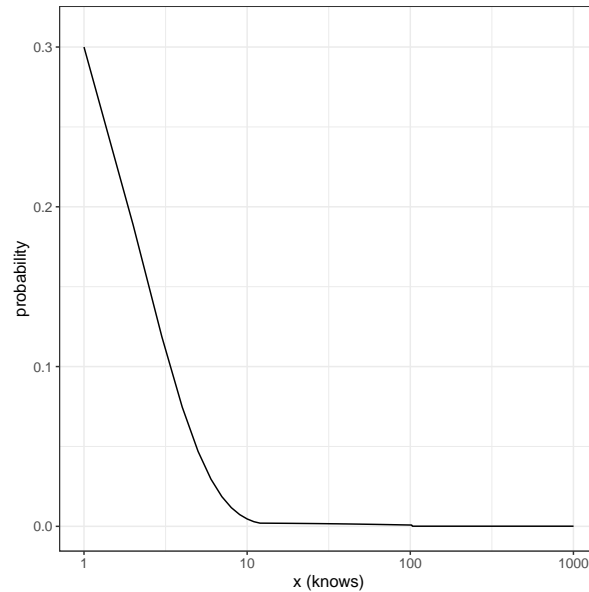


Figure 3.7: Distribution for determining the probability a Person is deleted given their number of connections.

Message type [2]	Datagen	% Deleted
Status updates	Post/Photo	2.7
Non-reply mentions	Post/Photo	2.7
Replies	Comment	1.8
Retweets	Post/Photo/Comment Likes	2.4

Table 3.17: Mapping of [2] message types to LDBC’s schema.

the number of Comments in a Posts thread using the distribution in Figure 3.8. Note, this distribution has an average of 2.7% aligning with Table 3.17.

Almuhimedi et al. also observe a temporal relationship for when a tweet is deleted – a tweet has a higher chance of being deleted soon after it was created. They found 50% of all deleted tweets were removed within 8 minutes of creation. We have recreated the temporal distribution in [2] and use it to generate deletion dates from the valid lifespan intervals for posts, comments, and likes that are selected for explicit deletion Figure 3.9.

**Delete Forum and Delete Forum Membership** We currently do not have empirical evidence to motivate realistic behaviour of Forum deletion. Forums have 3 types: walls, groups, and albums. Groups and albums can be explicitly deleted, walls cannot. The target proportion of groups and albums that are deleted across the simulation period is 1%.

Additionally, we currently do not have empirical evidence to motivate realistic behaviour of hasMember edge deletion. Only membership of groups can be explicitly deleted. The target proportion of group memberships that are deleted across the simulation period is 5%.

### 3.8 Converting Delete Events into Delete Operations

Datagen supports 3 modes, each having different output:

- **Interactive.** Produces the data necessary for the Interactive workload. Includes a set of bulk load csv files and a number of update streams, which contain only insert operations.
- **BI.** Produces the data necessary for the Business Intelligence workload. Includes a set of bulk load csv files and a number of update batches, which contain insert and delete operations.

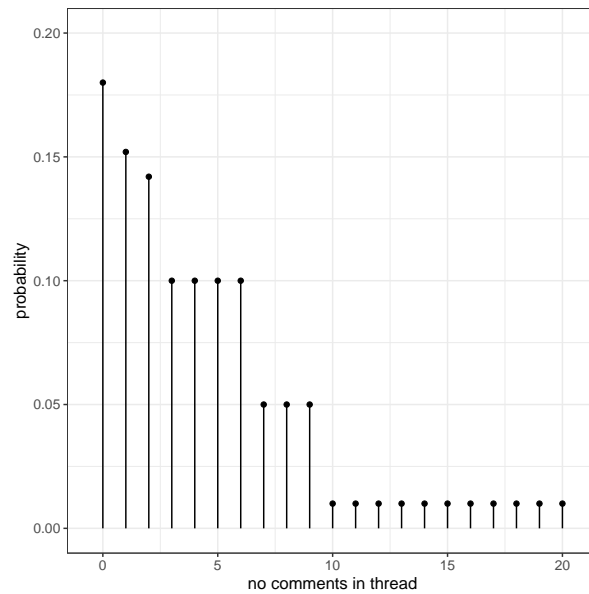


Figure 3.8: Probability a post is deleted given the number of comments in its thread.

- **Raw.** Produces a fully dynamic graph without insert or delete operations. Includes a set of bulk load csv files (covering whole simulation period), with each dynamic entity having creation and deletion date attributes serialized. This mode is not intended for use with any LDBC workload.

When run in Interactive mode Datagen produces a graph that monotonically increases in size over the simulation period with insert-only operations, e.g. once Person joins the network they never leave, not delete a post nor unlike a picture. This mode is supported for backward compatibility with the Interactive workload.

The modes BI and raw use the dynamic graph containing creation events and deletion events. Raw mode effectively serializes the graph to a bulk component and has a slightly different schema, with each entity having creation date and deletion date fields. This mode was developed for testing, yet may be useful to users that require a dynamic graph data set for purposes other than benchmarking.

For the BI mode the generated data must be converted into a bulk load component and a series of update batches (containing insert and delete operations). Figure 3.10 displays the possible creation and deletion dates a dynamic entity can have with respect to the bulk load cut off, simulation end, and network collapse, which determines the target file the entity should be serialized to. For example, if a Post is created after the bulk load and deleted before the simulation end this should result in an insert and a delete operation in the update batch data set. If an entity is marked for explicit deletion then, if the conditions in Figure 3.10 are satisfied then a deletion operation is serialized into the update batches.

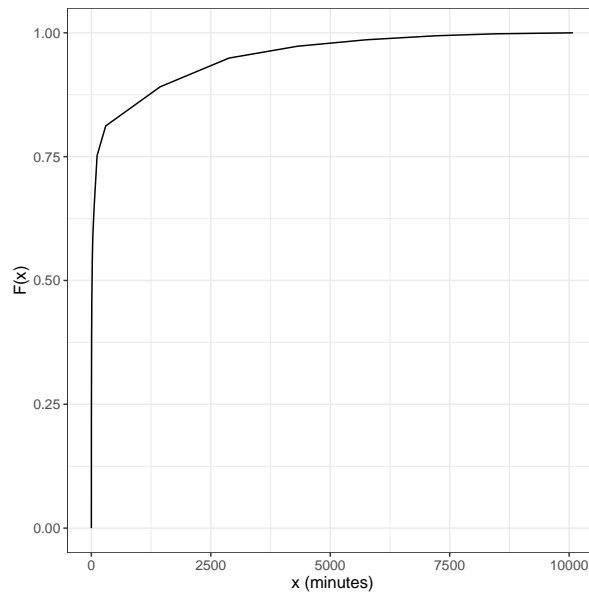
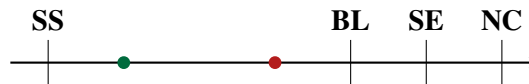
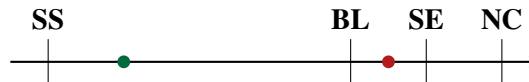


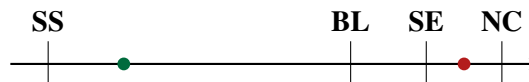
Figure 3.9: Cumulative probability density function of when a post, comment, or like is deleted after it is created ( $x = 0$ ).



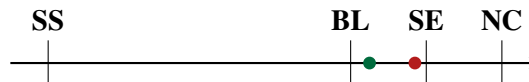
(a) Dynamic entity has creation and deletion dates before the bulk load cut off. This entity is not serialized.



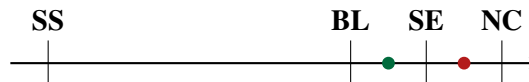
(b) Dynamic entity has creation date before the bulk load cut off and a deletion date after the bulk load cut off, but before the simulation end. Such an entity is serialized into the bulk load component and spawns a delete operation.



(c) Dynamic entity has creation date before the bulk load cut off and a deletion date after the simulation end. Such an entity is in serialized only into the bulk load component.



(d) Dynamic entity has creation date after the bulk load cut off and a deletion date before the simulation end. Such an entity produces an insert operation and a delete operation.



(e) Dynamic entity has creation date after the bulk load cut off, but before the simulation end, and a deletion date after the simulation end. Such an entity produces only an insert operation.

Figure 3.10: Possible dynamic entity *creation* ● and *deletion* ● dates with respect to simulation start, bulk load cut off, simulation end, and network collapse.

## 4 WORKLOADS

### 4.1 Query Description Format

Queries are described in natural language using a well-defined structure that consists of three sections: *description*, a concise textual description of the query, *parameters*, a list of input parameters and their types; *results*, a list of expected results and their types. Additionally, queries returning multiple results specify *sorting criteria* and a *limit* (to return top- $k$  results). For strings, the sorting criteria should be interpreted as a binary comparison of the strings.<sup>12</sup>

We use the following notation:

- **Node type:** node type in the dataset. One word, possibly constructed by appending multiple words together, starting with an uppercase character and following the camel case notation, e.g. TagClass represents an entity of type “TagClass”.
- **Edge type:** edge type in the dataset. One word, possibly constructed by appending multiple words together, starting with a lowercase character and following the camel case notation e.g. workAt represents an edge of type “workAt”.
- **Attribute:** attribute of a node or an edge in the dataset. One word, possibly constructed by appending multiple words together, starting with a lowercase character and following the camel case notation, and prefixed by a “.” to dereference the node/edge, e.g. person.firstName refers to “firstName” attribute on the “person” entity, and studyAt.classYear refers to “classYear” attribute on the “studyAt” edge.
- **Unordered Set:** an unordered collection of distinct elements. Surrounded by { and } braces, with the element type between them, e.g. {String} refers to a set of strings.
- **Ordered List:** an ordered collection where duplicate elements are allowed. Surrounded by [ and ] braces, with the element type between them, e.g. [String] refers to a list of strings.
- **Ordered Tuple:** a fixed-length, fixed-order list of elements, where elements at each position of the tuple have predefined, possibly different, types. Surrounded by < and > braces, with the element types between them in a specific order e.g. <String, Boolean> refers to a 2-tuple containing a string value in the first element and a boolean value in the second, and [<String, Boolean>] is an ordered list of those 2-tuples.

**Categorization of results.** Results are categorized according to their source of origin:

- **Raw (R)**, if the result attribute is returned with an unmodified value and type.
- **Calculated (C)**, if the result is calculated from attributes using arithmetic operators, functions, boolean conditions, etc.
- **Aggregated (A)**, if the result is an aggregated value, e.g. a count or a sum of another value. If a result is both calculated and aggregated (e.g.  $\text{count}(x) + \text{count}(y)$  or  $\text{avg}(x + y)$ ), it is considered an aggregated result.
- **Meta (M)**, if the result is based on type information, e.g. the type of a node.

### 4.2 Conventions for Query Definitions

**Interval notations.** Closed interval boundaries are denoted with [ and ], while open interval boundaries are denoted with ( and ). For example,  $[0, 1)$  denotes an interval between 0 and 1, closed on the left and open on the right.

**Comparing Date and DateTime values.** Some query specifications (e.g. BI 1) require implementations to compare a DateTime value with a Date value. In these cases, the Date value should be implicitly converted DateTime value with a time of 00:00:00.000+00:00 (i.e. with the timezone of GMT).

<sup>1</sup>C or POSIX collation in PostgreSQL, see <https://www.postgresql.org/docs/13/locale.html>

<sup>2</sup>BINARY collation in DuckDB, see <https://duckdb.org/docs/sql/expressions/collations>

**Matching semantics.** Unless noted otherwise, the specification uses *homomorphic* matching semantics [4], i.e. both nodes and edges can occur multiple times in a match. Note that for variable-length path, duplicate edges are not allowed.

**Aggregation semantics.** The `count` aggregation always requires the query to determine the number of *distinct* elements (nodes or edges). For example, this can be achieved in the Cypher, SPARQL and SQL query languages with the `count(DISTINCT ...)` construct.

**Graph patterns.** To illustrate queries, we use graph patterns such as Figure 4.1 with the following notation:

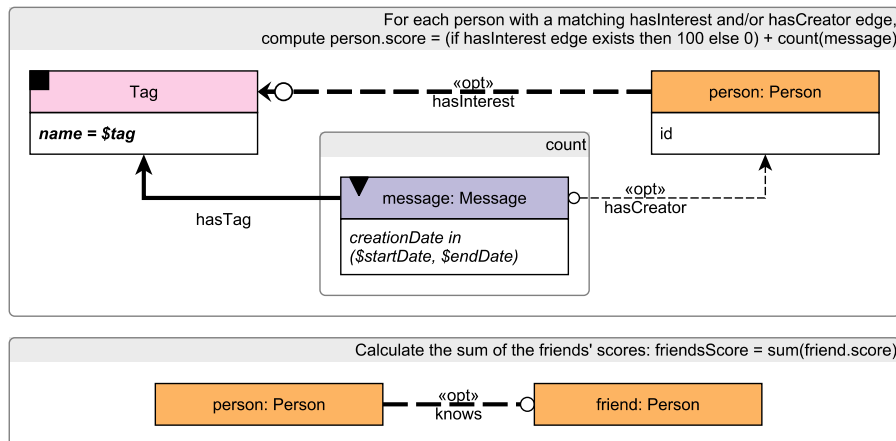


Figure 4.1: Example graph pattern.

- Nodes in the pattern are shown with rectangular boxes with their type name stated at the top and emphasized with colour coding.
- A black square ■ in the node's top left corner and a ***bold italic*** condition denote that the node is uniquely specified by the query parameters (e.g. by using an identifier or a unique attribute such as URL).
- Attributes of nodes and edges can be subject to range constraints (e.g. date within a given range, birthday larger than a given date, etc.). These are denoted with the ▼ symbol.
- Nodes in the pattern are captioned with `entityName: EntityType` (camel case notation for both, starting with a lowercase character for the first and an uppercase character for the second). If the `entityName` is neither returned in the query results (in raw, aggregated, or calculated form), nor referenced in the query specification, the `entityName` can be omitted.
- Edges in the graph pattern use the following notation:
  - Regular edges, i.e. edges that must be present in the subgraph, are denoted with solid black lines.
  - Negative edges, i.e. edges that must not be present in the subgraph, are denoted with dashed red lines and the «neg» keyword.
  - Optional edges, i.e. edges that may or may not be in the subgraph, are denoted with dashed black lines, the «opt» keyword, and a circle symbol ○ at the optional end of the edge.
  - Edges without direction have no arrows. Their semantics is that there must be an edge in *the least one of the (incoming, outgoing) directions*.



- Edges with many-to-many cardinalities are denoted with thicker lines, emphasizing that they may contribute more results in the result set.

- Filtering conditions are typeset in *italic*, e.g. *id = \$tag*.
- Attributes that should be returned are denoted in sans-serif font, e.g. `name`.
- Variable length paths, i.e. edges that can be traversed multiple times are denoted with *\*min. . . max*, e.g. *replyOf\** or *knows\*1. . . 2*. By default, the value of min is 1, and the value of max is unlimited.
- Aggregations are shown in boxes with a grey strip on their top describing the type of aggregation (*count*, *sum*, *average*, etc.).

**Keywords.** The pattern notation uses a small set of keywords:

- Aggregation operations: *avg*, *count*, *sum*.
- Functions:
  - *floor(x)*: returns  $\lfloor x \rfloor$ ,
  - *year(date)*: extracts the year from a given date,
  - *month(date)*: extracts the month from a given date.
  - *day(date)*: extracts the day (of the month) from a given date.

**Deletions.** Deletions of a single element are denoted with a red cross ✖, while recursive deletions are denoted with a purple cross ✖.

**Resolving ambiguity.** Note that if the textual description and the graph pattern are different for a particular query (either due to an error or the lack of sophistication in the graphical syntax), *the textual description takes precedence*.

## 4.3 Substitution Parameters

Together with the dataset, Datagen produces a set of parameters per query type. Parameter generation is designed in such a way that for each query type, all of the generated parameters yield similar runtime behaviour of that query.

Specifically, the selection of parameters for a query template guarantees the following properties of the resulting queries:

- P1: the query runtime has a bounded variance: the average runtime corresponds to the behavior of the majority of the queries
- P2: the runtime distribution is stable: different samples of (e.g. 10) parameter bindings used in different query streams result in an identical runtime distribution across streams
- P3: the optimal logical plan (optimal operator order) of the queries is the same: this ensures that a specific query template tests the system's behavior under the well-chosen technical difficulty (e.g. handling voluminous joins or proper cardinality estimation for subqueries, etc.)

As a result, the amount of data that the query touches is roughly the same for every parameter binding, assuming that the query optimizer figures out a reasonable execution plan for the query. This is done to avoid bindings that cause unexpectedly long or short runtimes of queries, or even result in a completely different optimal execution plan. Such effects could arise due to the data skew and correlations between values in the generated dataset.

In order to get the parameter bindings for each of the queries, we have designed a *Parameter Curation* procedure that works in two stages:

1. for each query template for all possible parameter bindings, we determine the size of intermediate results in the *intended* query plan. Intermediate result size heavily influences the runtime of a query, so two queries with the same operator tree and similar intermediate result sizes at every level of this operator tree are expected to have similar runtimes. This analysis is effectively a side effect of data generation, that is

we keep all the necessary counts (number of friends per user, number of posts of friends etc.) as we create the dataset.

2. then, a greedy algorithm selects (“curates”) those parameters with similar intermediate result counts from the domain of all the parameters.

Parameter bindings are stored in the `substitution_parameters` folder inside the data generator directory. Each query gets its bindings in a separate file. Every line of a parameter file is a JSON-formatted collection of key-value pairs (name of the parameter and its value). For example, the Query 1 parameter bindings are stored in file `query_1_param.txt`, and one of its lines may look like this:

```
{"PersonID": 1, "Name": "Lei", "PersonURI": "http://www.ldbc.eu/ldbc_socialnet/1.0/data/pers1"}
```

Depending on implementation, the SUT may refer to persons either by IDs (relational and graph databases) or URIs (RDF systems), so we provide both values for the Person parameter. Finally, parameters for short reads are taken from those in complex reads and inserts.

## 4.4 Return Values

Return values are subject to the following rules:

- DateTime and Date values should use GMT timezone (or they should be converted by the client to GMT).

## 5 UPDATE OPERATIONS

This chapter contains the specifications of the Insert and Delete operations in the SNB suite. Inserts are used in the BI workload as well as the Interactive v1 and v2 workloads. Deletes are only used in the Interactive v2 and BI workloads.

### 5.1 Insert Operations

Each insert operations creates

1. either a single node of a certain type, along with its edges to other existing nodes
2. or a single edge of a certain type between two existing nodes.

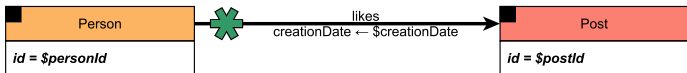
In Interactive v1, these operations were called “updates”. In Interactive v2, they are called “inserts”.

#### Updates / insert / 1

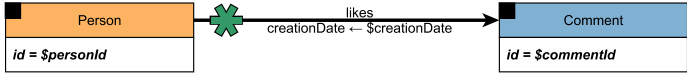
INS 1	query	Updates / insert / 1			
INS 2	title	Add person			
INS 3	pattern				
INS 4					
INS 5					
INS 6					
INS 7					
INS 8					
	description	Add a Person <i>node</i> , connected to the network by 4 possible <i>edge</i> types.			
	params	1	\$personId	ID	
		2	\$personFirstName	String	
		3	\$personLastName	String	
		4	\$gender	String	
		5	\$birthday	Date	
		6	\$creationDate	DateTime	
		7	\$locationIP	String	
		8	\$browserUsed	String	
		9	\$cityId	ID	
		10	\$languages	{String}	
		11	\$emails	{Long String}	
		12	\$tagIds	{ID}	
		13	\$studyAt	{<ID, 32-bit Integer>}	{<universityId, classYear>}
		14	\$workAt	{<ID, 32-bit Integer>}	{<companyId, workFrom>}
	CPs	9.1, 9.2			



**Updates / insert / 2**

INS 1	query	Updates / insert / 2														
INS 2	title	Add like to post														
INS 3	pattern															
INS 4																
INS 5																
INS 6	description	Add a likes <i>edge</i> to a Post.														
INS 7	params	<table><tr><td>1</td><td>\$personId</td><td>ID</td><td></td></tr><tr><td>2</td><td>\$postId</td><td>ID</td><td></td></tr><tr><td>3</td><td>\$creationDate</td><td>DateTime</td><td></td></tr></table>			1	\$personId	ID		2	\$postId	ID		3	\$creationDate	DateTime	
1	\$personId	ID														
2	\$postId	ID														
3	\$creationDate	DateTime														
INS 8																
	CPs	9.2														

**Updates / insert / 3**

INS 1	query	Updates / insert / 3												
INS 2	title	Add like to comment												
INS 3	pattern													
INS 4														
INS 5														
INS 6	description	Add a likes <i>edge</i> to a Comment.												
INS 7	params													
INS 8		<table><tr><td>1</td><td>\$personId</td><td>ID</td><td></td></tr><tr><td>2</td><td>\$commentId</td><td>ID</td><td></td></tr><tr><td>3</td><td>\$creationDate</td><td>DateTime</td><td></td></tr></table>	1	\$personId	ID		2	\$commentId	ID		3	\$creationDate	DateTime	
1		\$personId	ID											
2	\$commentId	ID												
3	\$creationDate	DateTime												
	CPs	9.2												

## Updates / insert / 4

INS 1	query	Updates / insert / 4		
INS 2	title	Add forum		
INS 3	pattern	<pre> graph LR     Forum[Forum] -- hasModerator --&gt; Person[Person]     Forum -- hasTag --&gt; Tag[Tag]     Forum -- id --&gt; ID1[id]     Forum -- title --&gt; Title[title]     Forum -- creationDate --&gt; CD1[creationDate]     Person -- id --&gt; ID2[id]     Tag -- tag --&gt; TagAttr[tag]           </pre>		
INS 4				
INS 5				
INS 6				
INS 7				
INS 8				
	description	Add a Forum <i>node</i> , connected to the network by 2 possible <i>edge</i> types.		
	params	1	\$forumId	ID
		2	\$forumTitle	Long String
		3	\$creationDate	DateTime
		4	\$moderatorId	ID
		5	\$tagIds	{ID}
	CPs	9.1, 9.2		

## Updates / insert / 5

INS 1	query	Updates / insert / 5		
INS 2	title	Add forum membership		
INS 3	pattern	<pre> graph LR     Person[Person] -- hasMember --&gt; Forum[Forum]     Person -- id --&gt; ID1[id]     Forum -- id --&gt; ID2[id]     hasMember -- creationDate --&gt; CD[creationDate]           </pre>		
INS 4				
INS 5				
INS 6				
INS 7	description	Add a Forum membership <i>edge</i> (hasMember) to a Person.		
INS 8	params	1	\$personId	ID
		2	\$forumId	ID
		3	\$creationDate	DateTime
	CPs	9.1, 9.2		

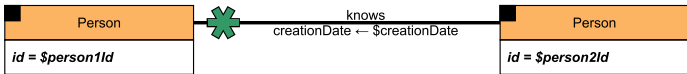
## Updates / insert / 6

INS 1	query	Updates / insert / 6		
INS 2	title	Add post		
INS 3	pattern			
INS 4				
INS 5				
INS 6				
INS 7				
INS 8				
	description	Add a Post <i>node</i> connected to the network by 4 possible <i>edge</i> types (hasCreator, containerOf, isLocatedIn, hasTag).		
	params	1	\$postId	ID
		2	\$imageFile	String
		3	\$creationDate	DateTime
		4	\$locationIP	String
		5	\$browserUsed	String
		6	\$language	String
		7	\$content	Text
		8	\$length	32-bit Integer
		9	\$authorPersonId	ID
		10	\$forumId	ID
		11	\$countryId	ID
		12	\$tagIds	{ID}
	CPs	9.1, 9.2		

## Updates / insert / 7

INS 1	query	Updates / insert / 7			
INS 2	title	Add comment			
INS 3	pattern	<div><div>The parent Message is either a Post or a Comment.</div><div><div><div>Post</div><div><math>id = \\$replyToPostId</math></div></div><div><div>Comment</div><div><math>id = \\$replyToCommentId</math></div></div></div><div><div><div>Country</div><div><math>id = \\$countryId</math></div></div><div><div>Tag</div><div><math>id\ in\ \\$tagIds</math></div></div><div><div>Comment</div><div><math>id \leftarrow \\$commentId</math> <math>creationDate \leftarrow \\$creationDate</math> <math>locationIP \leftarrow \\$locationIP</math> <math>browserUsed \leftarrow \\$browserUsed</math> <math>content \leftarrow \\$content</math> <math>length \leftarrow \\$length</math></div></div><div><div>Person</div><div><math>id = \\$authorPersonId</math></div></div></div><div><div>replyOf</div><div>replyOf</div><div>isLocatedIn</div><div>hasTag</div><div>hasCreator</div></div></div>			
INS 4					
INS 5					
INS 6					
INS 7					
INS 8					
	description	Add a Comment <i>node</i> replying to a Post/Comment, connected to the network by 4 possible <i>edge</i> types (replyOf, hasCreator, isLocatedIn, hasTag).			
	params	1	\$commentId	ID	
		2	\$creationDate	DateTime	
		3	\$locationIP	String	
		4	\$browserUsed	String	
		5	\$content	Text	
		6	\$length	32-bit Integer	
		7	\$authorPersonId	ID	
		8	\$countryId	ID	
		9	\$replyToPostId	ID	<i>old version:</i> -1 if the Comment is a reply of a Comment; <i>new version:</i> null if the Comment is a reply of a Post
		10	\$replyToCommentId	ID	<i>old version:</i> -1 if the Comment is a reply of a Post; <i>new version:</i> null if the Comment is a reply of a Post
		11	\$tagIds	{ID}	
	CPs	9.1, 9.2			

## Updates / insert / 8

INS 1	query	Updates / insert / 8			
INS 2	title	Add friendship			
INS 3	pattern				
INS 4					
INS 5					
INS 6					
INS 7					
INS 8	description	Add a friendship <i>edge</i> (knows) between two Persons.			
	params	1	\$person1Id	ID	
		2	\$person2Id	ID	
		3	\$creationDate	DateTime	
	CPs	9.2			

## 5.2 Delete Operations

Each delete operation removes

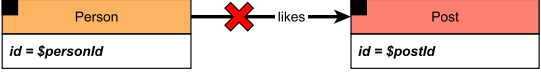
1. a single edge between two existing nodes
2. or a node, all its edges and, in certain cases, nodes and edges that are transitively reachable on a certain path (thus performing a cascading delete).

### Updates / delete / 1

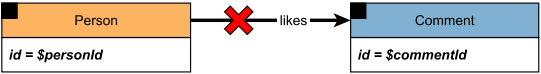
DEL 1  
DEL 2  
DEL 3  
DEL 4  
DEL 5  
DEL 6  
DEL 7  
DEL 8

query	Updates / delete / 1
title	Remove person and its personal forums and message (sub)threads
pattern	
description	Remove a Person with ID \$personId and its edges (isLocatedIn, studyAt, workAt, hasInterest, likes, knows, hasMember, hasModerator, hasCreator). Additionally, remove the Album and Wall Forums whose moderator is the Person and remove all Messages the Person has created in the rest of the Forums (Groups).
params	<div>1</div> <div>\$personId</div> <div>ID</div>
CPs	9.3, 9.4, 9.5
relevance	<ul style="list-style-type: none"> <li>• Removal of a Person removes Forums of type “Walls” and “Albums” but not “Groups”, which can continue if even the founder has left the network. For Groups, the hasModerator edge is deleted. We have discussed various approaches to appoint a new moderator, e.g. <ol style="list-style-type: none"> <li>1. choose member at random from the set of existing group members or</li> <li>2. the member with the oldest group join date becomes the moderator. However, to keep the generator and the workload simple, currently no moderator is selected, leaving the group without a moderator.</li> </ol> </li> <li>• Removal of a Person removes all Posts/Comments they are creator of this could result in the removal of a Comment in the middle of a thread.</li> </ul>

**Updates / delete / 2**

DEL 1	query	Updates / delete / 2									
DEL 2	title	Remove post like									
DEL 3	pattern										
DEL 6	description	Given a Person with ID \$personId and a Post with ID \$postId, remove the likes edge between them.									
DEL 7	params	<table border="1"> <tr> <td>1</td><td>\$personId</td><td>ID</td><td></td></tr> <tr> <td>2</td><td>\$postId</td><td>ID</td><td></td></tr> </table>		1	\$personId	ID		2	\$postId	ID	
1	\$personId	ID									
2	\$postId	ID									
DEL 8	CPs	9.4									
	relevance	Removal of a likes edge is a rare event, e.g. people accidentally liking a Post, this can be reflected by the relative frequency of the operation.									

**Updates / delete / 3**

DEL 1	query	Updates / delete / 3									
DEL 2	title	Remove comment like									
DEL 3	pattern										
DEL 6	description	Given a Person with ID \$personId and a Comment with ID \$commentId, remove the likes edge between them.									
DEL 7	params	<table border="1"> <tr> <td>1</td><td>\$personId</td><td>ID</td><td></td></tr> <tr> <td>2</td><td>\$commentId</td><td>ID</td><td></td></tr> </table>		1	\$personId	ID		2	\$commentId	ID	
1	\$personId	ID									
2	\$commentId	ID									
DEL 8	CPs	9.4									
	relevance	Removal of a likes edge is a rare event, e.g. people accidentally liking a Comment, this can be reflected by the relative frequency of the operation.									

**Updates / delete / 4**

DEL 1	query	Updates / delete / 4				
DEL 2	title	Remove forum and its content				
DEL 3	pattern					
DEL 4						
DEL 5						
DEL 6						
DEL 7						
DEL 8						
	description	Remove a Forum with ID \$forumId and its edges (hasModerator, hasMember, hasTag) and all Posts in the Forum (connected by containerOf edges) and their direct and transitive Comments.				
	params	<table><tr><td>1</td><td>\$forumId</td><td>ID</td><td></td></tr></table>	1	\$forumId	ID	
1	\$forumId	ID				
	CPs	9.3, 9.4, 9.5				
	relevance	n/a				

**Updates / delete / 5**

DEL 1

DEL 2

DEL 3

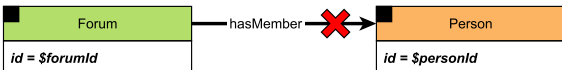
DEL 4

DEL 5

DEL 6

DEL 7

DEL 8

query	Updates / delete / 5			
title	Remove forum membership			
pattern				
description	Given a Forum with ID \$forumId and a Person with ID \$personId, remove the hasMember edge between them.			
params	<div>1</div>	\$forumId	ID	
	<div>2</div>	\$personId	ID	
CPs	9.4			
relevance	n/a			

## Updates / delete / 6

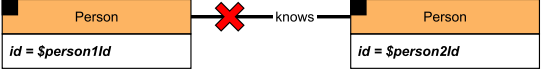
DEL 1	query	Updates / delete / 6
DEL 2	title	Remove post thread
DEL 3	pattern	
DEL 4	description	Remove a Post node with ID \$postId and its edges (isLocatedIn, likes, hasCreator, hasTag, containerOf). Remove all replies to the Post and the connecting replyOf edges. In addition, remove all transitive reply Comments to the Post and their edges.
DEL 5	params	1 \$postId ID
DEL 6	CPs	9.3, 9.4, 9.5
DEL 7	relevance	n/a
DEL 8		

## Updates / delete / 7

DEL 1	query	Updates / delete / 7
DEL 2	title	Remove comment subthread
DEL 3	pattern	
DEL 4	description	Remove a Comment node with ID \$commentId and its edges (isLocatedIn, likes, hasCreator, hasTag). In addition, remove all replies to the Comment connected by replyOf and their edges.
DEL 5	params	1 \$commentId ID
DEL 6	CPs	9.3, 9.4, 9.5
DEL 7	relevance	n/a
DEL 8		



**Updates / delete / 8**

DEL 1	query	Updates / delete / 8		
DEL 2	title	Remove friendship		
DEL 3	pattern			
DEL 4	description	Given two Person nodes with IDs \$person1Id and \$person2Id, remove the knows edge between them.		
DEL 5	params	1	\$person1Id	ID
DEL 6		2	\$person2Id	ID
DEL 7	CPs	9.4		
DEL 8	relevance	n/a		

## 6 INTERACTIVE v1 WORKLOAD

The Interactive v1 workload consists of a set of relatively complex read-only queries, that touch a significant amount of data – often the two-step friendship neighbourhood and associated messages –, but typically in close proximity to a single node. Hence, the query complexity is sublinear to the dataset size.

The LDBC SNB Interactive workload consists of three query classes:

- **Complex read-only queries.** See Section 6.1.
- **Short read-only queries.** See Section 6.2.
- **Insert operations.** See Section 5.1.

### Related Publications

A detailed description of the workload (covering reads and inserts) is available in the paper published at SIGMOD 2015 [24]. The ACID Test Suite was first published at TPCTC 2020 [90]. It is part of this specification in Chapter 10.

### Related Software Components

- **Datagen (Hadoop-based):** [https://github.com/ldbc/ldbc\\_snb\\_datagen\\_hadoop](https://github.com/ldbc/ldbc_snb_datagen_hadoop)
- **Driver:** [https://github.com/ldbc/ldbc\\_snb\\_interactive\\_v1\\_driver](https://github.com/ldbc/ldbc_snb_interactive_v1_driver)
- **Reference implementations:** [https://github.com/ldbc/ldbc\\_snb\\_interactive\\_v1\\_impls](https://github.com/ldbc/ldbc_snb_interactive_v1_impls)

## 6.1 Complex Reads

## Interactive / complex / 1

IC 1

IC 2

IC 3

IC 4

IC 5

IC 6

IC 7

IC 8

IC 9

IC 10

IC 11

IC 12

IC 13

IC 14v1

IC 14v2

query	Interactive / complex / 1																																																																					
title	Transitive friends with a certain name																																																																					
pattern	<pre>graph LR     StartPerson["person: Person id = \$personId"] -- knows*1..3 --&gt; OtherPerson["otherPerson: Person firstName = \$firstName id lastName birthday creationDate gender browserUsed locationIP email speaks"]     OtherPerson -- isLocatedIn --&gt; LocationCity["locationCity: City name"]     OtherPerson -- «opt» workAt --&gt; Company["company: Company name"]     OtherPerson -- «opt» studyAt --&gt; University["university: University name"]     LocationCity -- isLocatedIn --&gt; CompanyCountry["companyCountry: Country name"]     University -- isLocatedIn --&gt; UniversityCity["universityCity: City name"]</pre>																																																																					
description	Given a start Person with ID \$personId, find Persons with a given first name (\$firstName) that the start Person is connected to (excluding start Person) by at most 3 steps via the knows relationships. Return Persons, including the distance (1..3), summaries of the Persons workplaces and places of study.																																																																					
params	<table><tr><td>1</td><td>\$personId</td><td>ID</td><td></td></tr><tr><td>2</td><td>\$firstName</td><td>String</td><td></td></tr></table>				1	\$personId	ID		2	\$firstName	String																																																											
1	\$personId	ID																																																																				
2	\$firstName	String																																																																				
result	<table><tr><td>1</td><td>otherPerson.id</td><td>ID</td><td>R</td><td></td></tr><tr><td>2</td><td>otherPerson.lastName</td><td>String</td><td>R</td><td></td></tr><tr><td>3</td><td>distanceFromPerson</td><td>32-bit Integer</td><td>C</td><td></td></tr><tr><td>4</td><td>otherPerson.birthday</td><td>Date</td><td>R</td><td></td></tr><tr><td>5</td><td>otherPerson.creationDate</td><td>DateTime</td><td>R</td><td></td></tr><tr><td>6</td><td>otherPerson.gender</td><td>String</td><td>R</td><td></td></tr><tr><td>7</td><td>otherPerson.browserUsed</td><td>String</td><td>R</td><td></td></tr><tr><td>8</td><td>otherPerson.locationIP</td><td>String</td><td>R</td><td></td></tr><tr><td>9</td><td>otherPerson.email</td><td>{Long String}</td><td>R</td><td></td></tr><tr><td>10</td><td>otherPerson.speaks</td><td>{String}</td><td>R</td><td></td></tr><tr><td>11</td><td>locationCity.name</td><td>String</td><td>R</td><td></td></tr><tr><td>12</td><td>universities</td><td>{&lt;String, 32-bit Integer, String&gt;}</td><td>A</td><td>{&lt;university.name, studyAt.classYear, universityCity.name&gt;}</td></tr><tr><td>13</td><td>companies</td><td>{&lt;String, 32-bit Integer, String&gt;}</td><td>A</td><td>{&lt;company.name, workAt.workFrom, companyCountry.name&gt;}</td></tr></table>					1	otherPerson.id	ID	R		2	otherPerson.lastName	String	R		3	distanceFromPerson	32-bit Integer	C		4	otherPerson.birthday	Date	R		5	otherPerson.creationDate	DateTime	R		6	otherPerson.gender	String	R		7	otherPerson.browserUsed	String	R		8	otherPerson.locationIP	String	R		9	otherPerson.email	{Long String}	R		10	otherPerson.speaks	{String}	R		11	locationCity.name	String	R		12	universities	{<String, 32-bit Integer, String>}	A	{<university.name, studyAt.classYear, universityCity.name>}	13	companies	{<String, 32-bit Integer, String>}	A	{<company.name, workAt.workFrom, companyCountry.name>}
1	otherPerson.id	ID	R																																																																			
2	otherPerson.lastName	String	R																																																																			
3	distanceFromPerson	32-bit Integer	C																																																																			
4	otherPerson.birthday	Date	R																																																																			
5	otherPerson.creationDate	DateTime	R																																																																			
6	otherPerson.gender	String	R																																																																			
7	otherPerson.browserUsed	String	R																																																																			
8	otherPerson.locationIP	String	R																																																																			
9	otherPerson.email	{Long String}	R																																																																			
10	otherPerson.speaks	{String}	R																																																																			
11	locationCity.name	String	R																																																																			
12	universities	{<String, 32-bit Integer, String>}	A	{<university.name, studyAt.classYear, universityCity.name>}																																																																		
13	companies	{<String, 32-bit Integer, String>}	A	{<company.name, workAt.workFrom, companyCountry.name>}																																																																		
sort	<table><tr><td>1</td><td>distanceFromPerson</td><td>↑</td><td></td></tr><tr><td>2</td><td>otherPerson.lastName</td><td>↑</td><td></td></tr><tr><td>3</td><td>otherPerson.id</td><td>↑</td><td></td></tr></table>				1	distanceFromPerson	↑		2	otherPerson.lastName	↑		3	otherPerson.id	↑																																																							
1	distanceFromPerson	↑																																																																				
2	otherPerson.lastName	↑																																																																				
3	otherPerson.id	↑																																																																				
limit	20																																																																					
CPs	2.1, 5.3, 8.2																																																																					
relevance	This query is a representative of a simple navigational query. It is interesting for several aspects. (1) It requires for a complex aggregation for returning the concatenation of universities, companies, languages and email information of the Person. (2) It tests the ability of the optimizer to move the evaluation of sub-queries functionally dependant on the Person, after the evaluation of the top-k. (3) Its performance is highly sensitive to properly estimating the cardinalities in each transitive path, and paying attention not to explore already visited Persons.																																																																					

**Interactive / complex / 2**

IC 1	query	Interactive / complex / 2			
IC 2	title	Recent messages by your friends			
IC 3	pattern				
IC 4	description	Given a start Person with ID \$personId, find the most recent Messages from all of that Person's friends (friend nodes). Only consider Messages created before the given \$maxDate (excluding that day).			
IC 5	params	1	\$personId	ID	
IC 6		2	\$maxDate	Date	
IC 7	result	1	friend.id	ID	R
IC 8		2	friend.firstName	String	R
IC 9		3	friend.lastName	String	R
IC 10		4	message.id	ID	R
IC 11		5	message.content or message.imageFile (for photos)	Text	R
IC 12		6	message.creationDate	DateTime	R
IC 13	sort	1	message.creationDate	↓	
IC 14v1		2	message.id	↑	
IC 14v2	limit	20			
	CPs	1.1, 2.2, 2.3, 3.2, 8.5			
	relevance	This is a navigational query looking for paths of length two, starting from a given Person, going to their friends and from them, moving to their published Posts and Comments. This query exercises both the optimizer and how data is stored. It tests the ability to create execution plans taking advantage of the orderings induced by some operators to avoid performing expensive sorts. This query requires selecting Posts and Comments based on their creation date, which might be correlated with their identifier and therefore, having intermediate results with interesting orders. Also, messages could be stored in an order correlated with their creation date to improve data access locality. Finally, as many of the attributes required in the projection are not needed for the execution of the query, it is expected that the query optimizer will move the projection to the end.			

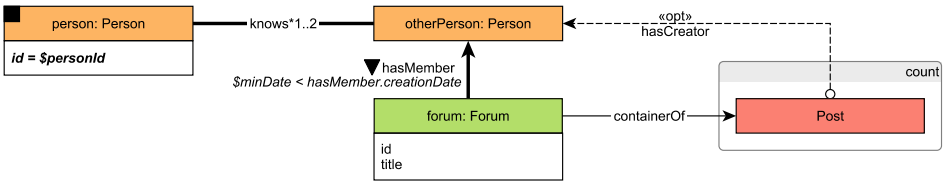
## Interactive / complex / 3

query	Interactive / complex / 3				
title	Friends and friends of friends that have been to given countries				
pattern					
description	Given a start Person with ID \$personId, find Persons that are their friends and friends of friends (excluding the start Person) that have made Posts / Comments in both of the given Countries (named \$countryXName and \$countryYName), within [\$startDate, \$startDate + \$durationDays) (closed-open interval). Only Persons that are foreign to these Countries are considered, that is Persons whose location Country is neither named \$countryXName nor \$countryYName.				
params	1	\$personId	ID		
	2	\$countryXName	String	In SNB Interactive v2, this query has two variants: (a) Correlated Countries (b) Anti-correlated Countries	
	3	\$countryYName	String		
	4	\$startDate	Date	Beginning of requested period	
	5	\$durationDays	32-bit Integer	Duration of requested period, in days. The interval [\$startDate, \$startDate + \$durationDays) is closed-open	
result	1	otherPerson.id	ID	R	
	2	otherPerson.firstName	String	R	
	3	otherPerson.lastName	String	R	
	4	xCount	32-bit Integer	A	Number of Messages from Country named \$countryXName created by the Person within the given time
	5	yCount	32-bit Integer	A	Number of Messages from Country named \$countryYName created by the Person within the given time
	6	count	32-bit Integer	A	count = xCount + yCount
sort	1	count	↓		
	2	otherPerson.id	↑		
limit	20				
CPs	2.1, 3.1, 5.1, 8.2, 8.5				
relevance	This query looks for paths of length two and three, starting from a Person, going to friends or friends of friends, and then moving to Messages. This query tests the ability of the query optimizer to select the most efficient join ordering, which will depend on the cardinalities of the intermediate results. Many friends of friends can be duplicate, then it is expected to eliminate duplicates and those people prior to access the Post and Comments, as well as eliminate those friends from Countries named \$countryXName and \$countryYName, as the size of the intermediate results can be severely affected. A possible structural optimization could be to materialize the number of Posts and Comments created by a Person, and progressively filter those people that could not even fall in the top 20 even having all their posts in the Countries named \$countryXName and \$countryYName.				

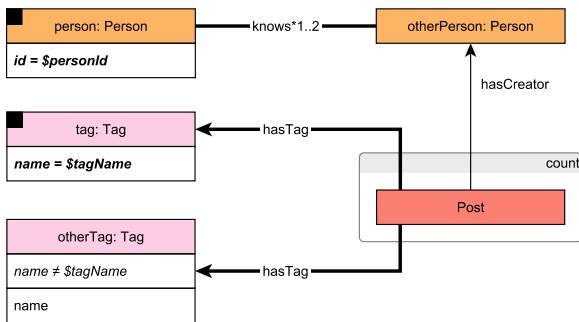
**Interactive / complex / 4**

IC 1	query	Interactive / complex / 4			
IC 2	title	New topics			
IC 3	pattern	<pre> graph LR     P1[Person] -- knows --&gt; P2[person: Person]     P2 -- knows --&gt; F[friend: Person]     F -- "«opt» hasCreator" --&gt; P3[Post]     P3 -- "creationDate &lt; \$startDate" --&gt; P4[Post]     P4 -- "startDate ≤ creationDate &lt; startDate + durationDays" --&gt; P5[Post]     P5 -- "postCount = count" --&gt; P6[Post]     P6 -- "hasTag" --&gt; T[tag: Tag]     P3 -- "«neg» hasTag" --&gt; T     </pre>			
IC 4					
IC 5					
IC 6					
IC 7					
IC 8					
IC 9					
IC 10					
IC 11					
IC 12	description	<p>Given a start Person with ID <code>\$personId</code>, find Tags that are attached to Posts that were created by that Person's friends. Only include Tags that were attached to friends' Posts created within a given time interval <code>[\$startDate, \$startDate + \$durationDays)</code> (closed-open) and that were never attached to friends' Posts created before this interval.</p>			
IC 13	params	1	<code>\$personId</code>	ID	
IC 14v1		2	<code>\$startDate</code>	Date	
IC 14v2		3	<code>\$durationDays</code>	32-bit Integer	Duration of requested period, in days. The interval <code>[\$startDate, \$startDate + \$durationDays)</code> is closed-open
	result	1	<code>tag.name</code>	Long String	R
		2	<code>postCount</code>	32-bit Integer	A
	sort	1	<code>postCount</code>	↓	
		2	<code>tag.name</code>	↑	
	limit	10			
	CPs	2.3, 8.2, 8.5			
	relevance	<p>This query looks for paths of length two, starting from a given Person, moving to Posts and then to Tags. It tests the ability of the query optimizer to properly select the usage of hash joins or index based joins, depending on the cardinality of the intermediate results. These cardinalities are clearly affected by the input Person, the number of friends, the variety of Tags, the time interval and the number of Posts.</p>			

**Interactive / complex / 5**

IC 1	query	Interactive / complex / 5			
IC 2	title	New groups			
IC 3	pattern	 <pre> graph LR     P1[person: Person id = \$personId] -- knows*1..2 --&gt; P2[otherPerson: Person]     P2 -.-&gt; «opt» hasCreator  Post     F[forum: Forum id title] -- hasMember \$minDate &lt; hasMember.creationDate --&gt; P2     F -- containerOf --&gt; Post     subgraph count         Post     end </pre>			
IC 4	description	<p>Given a start Person with ID \$personId, denote their friends and friends of friends (excluding the start Person) as otherPerson.</p> <p>Find Forums that any Person otherPerson became a member of after a given date (\$minDate). For each of those Forums, count the number of Posts that were created by the Person otherPerson.</p>			
IC 5	params	1	\$personId	ID	
IC 6		2	\$minDate	Date	
IC 7	result	1	forum.title	Long String	R
IC 8		2	postCount	32-bit Integer	A
IC 9		Number of Posts made in forum that were created by the Person otherPerson			
IC 10	sort	1	postCount	↓	
IC 11		2	forum.id	↑	
IC 12	limit	20			
IC 13	CPs	2.3, 3.3, 8.2, 8.5			
IC 14v1	relevance	<p>This query looks for paths of length two and three, starting from a given Person, moving to friends and friends of friends, and then getting the Forums they are members of. Besides testing the ability of the query optimizer to select the proper join operator, it rewards the usage of indices, but their accesses will be presumably scattered due to the two/three-hop search space of the query, leading to unpredictable and scattered index accesses. Having efficient implementations of such indices will be highly beneficial.</p>			
IC 14v2					

**Interactive / complex / 6**

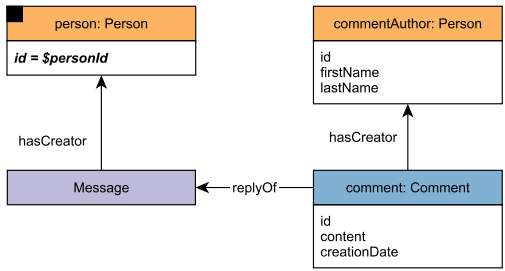
IC 1	query	Interactive / complex / 6			
IC 2	title	Tag co-occurrence			
IC 3	pattern	 <pre> graph TD     person[person: Person id = \$personId] -- knows*1..2 --&gt; otherPerson[otherPerson: Person]     otherPerson -- hasCreator --&gt; Post[Post count]     Post -- hasTag --&gt; tag[tag: Tag name = \$tagName]     Post -- hasTag --&gt; otherTag[otherTag: Tag name ≠ \$tagName name] </pre>			
IC 4					
IC 5					
IC 6					
IC 7					
IC 8					
IC 9					
IC 10					
IC 11					
IC 12					
IC 13					
IC 14v1	description	<p>Given a start Person with ID \$personId and a Tag with name \$tagName, find the other Tags that occur together with this Tag on Posts that were created by start Person's friends and friends of friends (excluding start Person). Return top 10 Tags, and the count of Posts that were created by these Persons, which contain both this Tag and the given Tag.</p>			
IC 14v2	params	1	\$personId	ID	
		2	\$tagName	Long String	
	result	1	otherTag.name	Long String	R
		2	postCount	32-bit Integer	A
	sort	1	postCount	↓	
		2	otherTag.name	↑	
	limit	10			
	CPs	5.1, 8.2			
	relevance	This query looks for paths of lengths three or four, starting from a given Person, moving to friends or friends of friends, then to Posts and finally ending at a given Tag.			



**Interactive / complex / 7**

IC 1	query	Interactive / complex / 7			
IC 2	title	Recent likers			
IC 3	pattern	<pre> graph LR     person[person: Person] -- knows --&gt; person     person -- hasCreator --&gt; message[message: Message]     friend[friend: Person] -- likes --&gt; message     message --&gt; person     message --&gt; friend   </pre>			
IC 4					
IC 5					
IC 6					
IC 7					
IC 8					
IC 9					
IC 10	description	<p>Given a start Person with ID <code>\$personId</code>, find the most recent likes on any of start Person's Messages. Find Persons that liked (<code>likes</code> edge) any of start Person's Messages, the Messages they liked most recently, the creation date of that like, and the latency in minutes (<code>minutesLatency</code>) between creation of Messages and like. Additionally, for each Person found return a flag indicating (<code>isNew</code>) whether the liker is a friend of start Person. In case that a Person liked multiple Messages at the same time, return the Message with lowest identifier.</p> <p><i>Validation rule:</i> Depending on whether the system-under-test supports leap seconds or uses UTC-SLS (UTC with Smoothed Leap Seconds), a difference of 1 minute can occur between the <code>minutesLatency</code> results of two correct implementations when the time interval includes June 30, 2012, when there was a leap second. Therefore, the <code>minutesLatency</code> value is validated using a tolerance of 1 minute.</p>			
IC 11					
IC 12					
IC 13					
IC 14v1	params	1	<code>\$personId</code>	ID	
IC 14v2					
result	1	<code>friend.id</code>	ID	R	<code>friend.id = personId</code> is allowed
	2	<code>friend.firstName</code>	String	R	
	3	<code>friend.lastName</code>	String	R	
	4	<code>likes.creationDate</code>	DateTime	R	
	5	<code>message.id</code>	ID	R	
	6	<code>message.content</code> or <code>message.imageFile</code> (for photos)	Text	R	
	7	<code>minutesLatency</code>	32-bit Integer	C	Duration between the creation of the Message and the creation of the like, in minutes.
	8	<code>isNew</code>	Boolean	C	False if person and friend know each other, True otherwise
sort	1	<code>likes.creationDate</code>	↓		
	2	<code>friend.id</code>	↑		
limit	20				
CPs	2.2, 2.3, 3.3, 5.1, 8.1, 8.3				
relevance	<p>This query looks for paths of length two, starting from a given Person, moving to its published messages and then to Persons who liked them. It tests several aspects related to join optimization, both at query optimization plan level and execution engine level. On the one hand, many of the columns needed for the projection are only needed in the last stages of the query, so the optimizer is expected to delay the projection until the end. This query implies accessing two-hop data, and as a consequence, index accesses are expected to be scattered. We expect to observe variate cardinalities, depending on the characteristics of the input parameter, so properly selecting the join operators will be crucial. This query has a lot of correlated sub-queries, so it is testing the ability to flatten the query execution plans.</p>				

**Interactive / complex / 8**

IC 1	query	Interactive / complex / 8			
IC 2	title	Recent replies			
IC 3	pattern	 <pre> graph TD     P1[person: Person id = \$personId]     M[Message]     C[comment: Comment id content creationDate]     PA[commentAuthor: Person id firstName lastName]      M -- hasCreator --&gt; P1     C -- hasCreator --&gt; PA     M -- replyOf --&gt; C           </pre>			
IC 4	description	Given a start Person with ID \$personId, find the most recent Comments that are replies to Messages of the start Person. Only consider direct (single-hop) replies, not the transitive (multi-hop) ones. Return the reply Comments, and the Person that created each reply Comment.			
IC 5	params	1	\$personId	ID	
IC 6	result	1	commentAuthor.id	ID	R
IC 7		2	commentAuthor.firstName	String	R
IC 8		3	commentAuthor.lastName	String	R
IC 9		4	comment.creationDate	DateTime	R
IC 10		5	comment.id	ID	R
IC 11		6	comment.content	Text	R
IC 12	sort	1	comment.creationDate	↓	
IC 13		2	comment.id	↑	
IC 14v1	limit	20			
IC 14v2	CPs	2.4, 3.3, 5.3			
	relevance	This query looks for paths of length two, starting from a given Person, going through its created Messages and finishing at their replies. In this query there is temporal locality between the replies being accessed. Thus the top-k order by this can interact with the selection, i.e. do not consider older Posts than the 20th oldest seen so far.			

**Interactive / complex / 9**

IC 1	query	Interactive / complex / 9			
IC 2	title	Recent messages by friends or friends of friends			
IC 3	pattern				
IC 4					
IC 5					
IC 6					
IC 7					
IC 8					
IC 9					
IC 10					
IC 11					
IC 12	description	Given a start Person with ID \$personId, find the most recent Messages created by that Person's friends or friends of friends (excluding the start Person). Only consider Messages created before the given \$maxDate (excluding that day).			
IC 13	params	1	\$personId	ID	
IC 14v1		2	\$maxDate	Date	
IC 14v2	result	1	otherPerson.id	ID	R
		2	otherPerson.firstName	String	R
		3	otherPerson.lastName	String	R
		4	message.id	ID	R
		5	message.content or message.imageFile (for photos)	Text	R
		6	message.creationDate	DateTime	R
	sort	1	message.creationDate	↓	
		2	message.id	↑	
	limit	20			
	CPs	1.1, 1.2, 2.2, 2.3, 3.2, 3.3, 8.5			
	relevance	This query looks for paths of length two or three, starting from a given Person, moving to its friends and friends of friends, and ending at their created Messages. This is one of the most complex queries, as the list of choke points indicates. This query is expected to touch variable amounts of data with entities of different characteristics, and therefore, properly estimating cardinalities and selecting the proper operators will be crucial.			

## Interactive / complex / 10

IC 1  
IC 2  
IC 3  
IC 4  
IC 5  
IC 6  
IC 7  
IC 8  
IC 9  
IC 10  
IC 11  
IC 12  
IC 13  
IC 14v1  
IC 14v2

query	Interactive / complex / 10				
title	Friend recommendation				
pattern	<div><div><div><div>person: Person</div><div><code>id = \$personId</code></div></div><div><div>foaf: Person</div><div><code>(month(birthday) = \$month and day(birthday) ≥ 21) or (month(birthday) = \$month+1 and day(birthday) &lt; 22)</code></div><div>id firstName lastName gender</div></div><div><div>city: City</div><div>name</div></div><div>knows*2..2</div><div>isLocatedIn</div></div><div><div><div>common</div><div><div><div>person: Person</div><div>hasInterest</div><div>Tag</div></div><div><div>foaf: Person</div><div>hasCreator</div><div>Post</div></div><div>count</div><div>hasTag</div></div></div><div><div><div>uncommon</div><div><div><div>person: Person</div><div>«neg» hasInterest</div><div>Tag</div></div><div><div>foaf: Person</div><div>hasCreator</div><div>Post</div></div><div>count</div><div>hasTag</div></div></div></div></div></div>				
description	<p>Given a start Person with ID <code>\$personId</code>, find that Person’s friends of friends (<code>foaf</code>) – excluding the start Person and his/her immediate friends –, who were born on or after the 21st of a given <code>\$month</code> (in any year) and before the 22nd of the following month. Calculate the similarity between each friend and the start person, where <code>commonInterestScore</code> is defined as follows:</p> <ul style="list-style-type: none"><li>• <code>common</code> = number of Posts created by friend, such that the Post has a Tag that the start person is interested in</li><li>• <code>uncommon</code> = number of Posts created by friend, such that the Post has no Tag that the start person is interested in</li><li>• <code>commonInterestScore</code> = <code>common</code> - <code>uncommon</code></li></ul>				
params	1	<code>\$personId</code>	ID		
	2	<code>\$month</code>	32-bit Integer	Between 1 and 12. Implementations may also pass the next month as an additional <code>\$nextMonth</code> parameter	
result	1	<code>foaf.id</code>	ID	R	
	2	<code>foaf.firstName</code>	String	R	
	3	<code>foaf.lastName</code>	String	R	
	4	<code>commonInterestScore</code>	32-bit Integer	A	
	5	<code>foaf.gender</code>	String	R	
	6	<code>city.name</code>	String	R	
sort	1	<code>commonInterestScore</code>	↓		
	2	<code>foaf.id</code>	↑		
limit	10				
CPs	2.3, 3.3, 4.1, 4.2, 5.1, 5.2, 6.1, 7.1, 8.6				
relevance	<p>This query looks for paths of length two, starting from a Person and ending at the friends of their friends. It does widely scattered graph traversal, and one expects no locality of in friends of friends, as these have been acquired over a long time and have widely scattered identifiers. The join order is simple but one must see that the anti-join for “not in my friends” is better with hash. Also the last pattern in the scalar sub-queries joining or anti-joining the Tags of the candidate’s Posts to interests of self should be by hash.</p>				

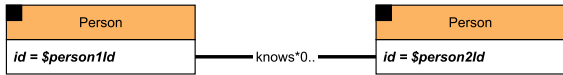
**Interactive / complex / 11**

IC 1	query	Interactive / complex / 11			
IC 2	title	Job referral			
IC 3	pattern	<pre> graph TD     P1[person: Person id = \$personId] -- knows*1..2 --&gt; P2[otherPerson: Person id firstName lastName]     P2 -- "workAt workAt.year(workFrom) &lt; \$year" --&gt; C[company: Company name]     C -- isLocatedIn --&gt; CO[country: Country name = \$name]           </pre>			
IC 14v1	description	Given a start Person with ID \$personId, find that Person's friends and friends of friends (excluding start Person) who started working in some Company in a given Country with name \$countryName, before a given date (\$workFromYear).			
IC 14v2	params	1	\$personId	ID	
		2	\$countryName	String	
		3	\$workFromYear	32-bit Integer	
	result	1	otherPerson.id	ID	R
		2	otherPerson.firstName	String	R
		3	otherPerson.lastName	String	R
		4	company.name	String	R
		5	workAt.workFrom	32-bit Integer	R
	sort	1	workAt.workFrom	↑	
		2	otherPerson.id	↑	
		3	company.name	↓	
	limit	10			
	CPs	1.3, 2.3, 2.4, 3.3, 4.2			
	relevance	This query looks for paths of length two or three, starting from a Person, moving to friends or friends of friends, and ending at a Company. In this query, there are selective joins and a top-k order by that can be exploited for optimizations.			

**Interactive / complex / 12**

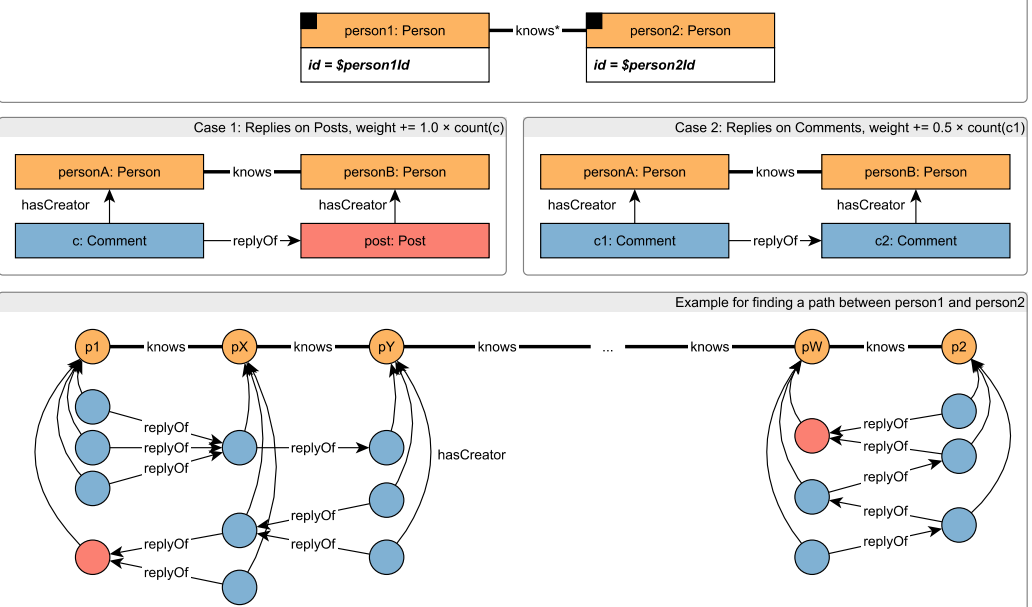
query	Interactive / complex / 12				
title	Expert search				
pattern					
description	Given a start Person with ID \$personId, find the Comments that this Person’s friends made in reply to Posts, considering only those Comments that are direct (single-hop) replies to Posts, not the transitive (multi-hop) ones. Only consider Posts with a Tag in a given TagClass with name \$tagClassName or in a descendant of that TagClass. Count the number of these reply Comments, and collect the Tags that were attached to the Posts they replied to, but only collect Tags with the given TagClass or with a descendant of that TagClass. Return Persons with at least one reply, the reply count, and the collection of Tags.				
params	1	\$personId	ID		
	2	\$tagClassName	Long String		
result	1	friend.id	ID	R	
	2	friend.firstName	String	R	
	3	friend.lastName	String	R	
	4	tagNames	{Long String}	A	
	5	replyCount	32-bit Integer	A	
sort	1	replyCount	↓		
	2	friend.id	↑		
limit	20				
CPs	3.3, 7.2, 7.3, 8.2				
relevance	This query starts at a Person, moves to its friends, and the to their Comments and their root Posts. Then, it gets the Tag of each Post and checks whether it (directly or transitively) belongs to the specified TagClass. This can be thought of a bidirectional search between the Person and the TagClass. The difficulty of this query is determining the optimal direction of this traversal.				

**Interactive / complex / 13**

IC 1	query	Interactive / complex / 13			
IC 2	title	Single shortest path			
IC 3	pattern				
IC 4	description	<p>Given two Persons with IDs \$person1Id and \$person2Id, find the shortest path between these two Persons in the subgraph induced by the knows edges. Return the length of this path:</p> <ul style="list-style-type: none"> <li>• -1: no path found</li> <li>• 0: start person = end person</li> <li>• &gt; 0: path found (start person ≠ end person)</li> </ul>			
IC 5					
IC 6					
IC 7					
IC 8					
IC 9	params	1	\$person1Id	ID	<p>In SNB Interactive v2, this query has two variants:</p> <p>(b) Guaranteed that there is no path between the two Persons</p> <p>(b) Guaranteed that there is a 4-hop path between the two Persons</p>
IC 10		2	\$person2Id	ID	
IC 11	result	1	shortestPathLength	32-bit Integer	C
IC 12	CPs	3.3, 7.2, 7.3, 7.5, 7.8, 8.1, 8.6			
IC 13	relevance	<p>This query looks for a variable length path, starting at a given Person and finishing at an another given Person. Proper cardinality estimation and search space pruning, will be crucial. This query also allows for possible parallel implementations.</p>			

## Interactive / complex / 14v1

IC 1  
IC 2  
IC 3  
IC 4  
IC 5  
IC 6  
IC 7  
IC 8  
IC 9  
IC 10  
IC 11  
IC 12  
IC 13  
IC 14v1  
IC 14v2

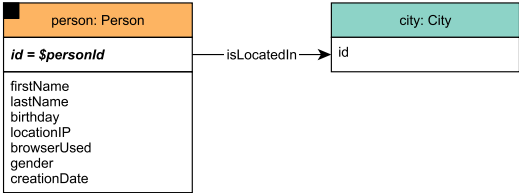
query	Interactive / complex / 14v1			
title	Trusted connection paths (v1)			
pattern	<p>Enumerate all unweighted shortest paths on knows edges from person1 to person2. For each edge on the path, calculate a weight based on interactions between the pair of Persons of the edge as a sum of cases #1 and #2 for the Persons (both ways), and the sum of these weights determine the total weight of each path.</p> 			
description	<p><i>This query is used in SNB Interactive v1.</i></p> <p>Given two Persons with IDs \$person1Id and \$person2Id, find all (unweighted) shortest paths between these two Persons, in the subgraph induced by the knows relationship.</p> <p>Then, for each path calculate a weight. The nodes in the path are Persons, and the weight of a path is the sum of weights between every pair of consecutive Person nodes in the path.</p> <p>The weight for a pair of Persons is calculated based on their interactions:</p> <ul style="list-style-type: none"> <li>• Every direct reply (by one of the Persons) to a Post (by the other Person) is 1.0.</li> <li>• Every direct reply (by one of the Persons) to a Comment (by the other Person) is 0.5.</li> </ul> <p>Note that interactions are counted both ways (e.g. if Alice writes 2 Post replies and 1 Comment reply to Bob, while Bob writes 3 Post replies and 4 Comment replies to Alice, their interaction score is <math>2 \times 1.0 + 1 \times 0.5 + 3 \times 1.0 + 4 \times 0.5 = 7.5</math>).</p> <p>Return all the paths with shortest length and their weights. Do not return any rows if there is no path between the two Persons.</p>			
params	1	\$person1Id	ID	
	2	\$person2Id	ID	
result	1	personIdsInPath	[ID]	C
	2	pathWeight	64-bit Float	C
sort	1	pathWeight	↓	The order of paths with the same weight is unspecified
CPs	3.3, 5.3, 7.2, 7.3, 7.5, 7.7, 8.1, 8.2, 8.3, 8.6			
relevance	<p>This query looks for a variable length path, starting at a given Person and finishing at an another given Person. This is a more complex query as it not only requires computing the path length, but returning it and computing a weight. To compute this weight one must look for smaller sub-queries with paths of length three, formed by the two Persons at each step, a Post and a Comment.</p>			



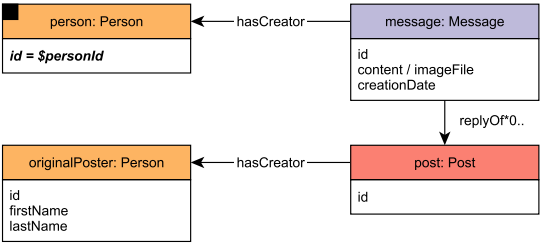
## 6.2 Short Reads

## Interactive / short / 1

IS 1  
IS 2  
IS 3  
IS 4  
IS 5  
IS 6  
IS 7

query	Interactive / short / 1			
title	Profile of a person			
pattern				
description	Given a start Person with ID \$personId, retrieve their first name, last name, birthday, IP address, browser, and city of residence.			
params	1	\$personId	ID	
result	1	person.firstName	String	R
	2	person.lastName	String	R
	3	person.birthday	Date	R
	4	person.locationIP	String	R
	5	person.browserUsed	String	R
	6	city.id	ID	R
	7	person.gender	String	R
	8	person.creationDate	DateTime	R

**Interactive / short / 2**

IS 1	query	Interactive / short / 2			
IS 2	title	Recent messages of a person			
IS 3	pattern	 <pre> graph TD     P1[person: Person id = \$personId] -- hasCreator --&gt; M[message: Message id content / imageFile creationDate]     M -- replyOf*0.. --&gt; P2[post: Post id]     P2 -- hasCreator --&gt; OP[originalPoster: Person id firstName lastName]           </pre>			
IS 4	description	<p>Given a start Person with ID \$personId, retrieve the last 10 Messages created by that user. For each Message, return that Message, the original Post in its conversation (post), and the author of that Post (originalPoster). If any of the Messages is a Post, then the original Post (post) will be the same Message, i.e. that Message will appear twice in that result.</p>			
IS 5	params	1	\$personId	ID	
IS 6	result	1	message.id	ID	R
IS 7		2	message.content or message.imageFile (for photos)	Text	R
		3	message.creationDate	DateTime	R
		4	post.id	ID	R
		5	originalPoster.id	ID	R
		6	originalPoster.firstName	String	R
		7	originalPoster.lastName	String	R
	sort	1	message.creationDate	↓	
		2	message.id	↓	
	limit	10			

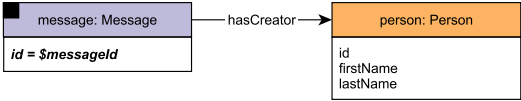
**Interactive / short / 3**

IS 1	query	Interactive / short / 3				
IS 2	title	Friends of a person				
IS 3	pattern	<div><div><div></div><div>person: Person</div><div><i>id = \$personId</i></div></div><div>— knows — creationDate</div><div><div>friend: Person</div><div>id firstName lastName</div></div></div>				
IS 4						
IS 5						
IS 6						
IS 7	description	Given a start Person with ID \$personId, retrieve all of their friends, and the date at which they became friends.				
	params	<div><div>1</div><div>\$personId</div><div>ID</div><div></div></div>				
	result	<div><div>1</div><div>friend.id</div><div>ID</div><div>R</div><div></div></div>				
		<div><div>2</div><div>friend.firstName</div><div>String</div><div>R</div><div></div></div>				
		<div><div>3</div><div>friend.lastName</div><div>String</div><div>R</div><div></div></div>				
		<div><div>4</div><div>knows.creationDate</div><div>DateTime</div><div>R</div><div></div></div>				
	sort	<div><div>1</div><div>knows.creationDate</div><div>↓</div><div></div></div>				
		<div><div>2</div><div>friend.id</div><div>↑</div><div></div></div>				

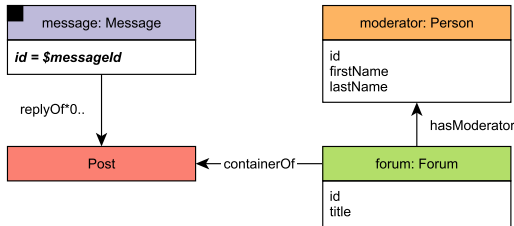
**Interactive / short / 4**

IS 1	query	Interactive / short / 4				
IS 2	title	Content of a message				
IS 3	pattern	<div><div>message: Message</div><div><i>id = \$messageId</i></div><div>creationDate content / imageFile</div></div>				
IS 4						
IS 5						
IS 6						
IS 7	description	Given a Message with ID \$messageId, retrieve its content and creation date.				
	params	<div><div>1</div><div>\$messageId</div><div>ID</div><div></div></div>				
	result					
		<div><div>1</div><div>message.creationDate</div><div>DateTime</div><div>R</div><div>messageCreationDate</div></div>				
		<div><div>2</div><div>message.content or message.imageFile (for photos)</div><div>Text</div><div>R</div><div>messageContent</div></div>				

**Interactive / short / 5**

IS 1	query	Interactive / short / 5			
IS 2	title	Creator of a message			
IS 3	pattern	 <pre> graph LR     msg["message: Message id = \$messageId"] -- hasCreator --&gt; person["person: Person id firstName lastName"]           </pre>			
IS 5	description	Given a Message with ID \$messageId, retrieve its author.			
IS 6	params	1	\$messageId	ID	
IS 7	result	1	person.id	ID	R
		2	person.firstName	String	R
		3	person.lastName	String	R

**Interactive / short / 6**

IS 1	query	Interactive / short / 6			
IS 2	title	Forum of a message			
IS 3	pattern	 <pre> graph TD     msg["message: Message id = \$messageId"] -- replyOf*0.. --&gt; post["Post"]     post -- containerOf --&gt; forum["forum: Forum id title"]     forum -- hasModerator --&gt; mod["moderator: Person id firstName lastName"]           </pre>			
IS 5	description	Given a Message with ID \$messageId, retrieve the Forum that contains it and the Person that moderates that Forum. Since Comments are not directly contained in Forums, for Comments, return the Forum containing the original Post in the thread which the Comment is replying to.			
IS 6	params	1	\$messageId	ID	
IS 7	result	1	forum.id	ID	R
		2	forum.title	Long String	R
		3	moderator.id	ID	R
		4	moderator.firstName	String	R
		5	moderator.lastName	String	R

**Interactive / short / 7**

IS 1	query	Interactive / short / 7			
IS 2	title	Replies of a message			
IS 3	pattern	<pre> graph TD     Message["message: Message&lt;br/&gt;id = \$messageId"] -- hasCreator --&gt; messageAuthor["messageAuthor: Person"]     Comment["comment: Comment&lt;br/&gt;id&lt;br/&gt;content&lt;br/&gt;creationDate"] -- replyOf --&gt; Message     Comment -- hasCreator --&gt; replyAuthor["replyAuthor: Person&lt;br/&gt;id&lt;br/&gt;firstName&lt;br/&gt;lastName"]     messageAuthor -.-&gt; «opt» knows  replyAuthor           </pre>			
IS 4	description	<p>Given a Message with ID \$messageId, retrieve the (1-hop) Comments that reply to it.</p> <p>In addition, return a boolean flag knows indicating if the author of the reply (replyAuthor) knows the author of the original message (messageAuthor). If author is same as original author, return False for knows flag.</p>			
IS 5	params	1	\$messageId	ID	
IS 6	result	1	comment.id	ID	R
IS 7		2	comment.content	Text	R
		3	comment.creationDate	DateTime	R
		4	replyAuthor.id	ID	R
		5	replyAuthor.firstName	String	R
		6	replyAuthor.lastName	String	R
		7	knows	Boolean	C
		True if the knows edge exists between the replyAuthor and the messageAuthor nodes, False otherwise (including the case when the two nodes are the same)			
	sort	1	comment.creationDate	↓	
		2	replyAuthor.id	↑	

## 6.3 Workload Definition

The *Test Driver* is in charge of the execution of the Interactive Workload. At the beginning of the execution, the Test Driver creates a query mix by assigning to each query instance, a query issue time and a set of parameters taken from the generated substitution parameter set described above.

Query issue times have to be carefully assigned. Although substitution parameters are chosen in such a way that queries of the same type take similar time, not all query types have the same complexity and touch the same amount of data, which causes them to scale differently for the different scale factors. Therefore, if all query instances, regardless of their type, are issued at the same rate, those more complex queries will dominate the execution's result, making faster query types purposeless. To avoid this situation, each query type is executed at a different rate. The way the execution rate is decided, also depends on the nature of the query: complex read, short read or update.

Update queries' issue times are taken from the update streams generated by the data generator. These are the times where the actual event happened during the simulation of the social network. Complex reads' times are expressed in terms of update operations. For each complex read query type, a frequency value is assigned

which specifies the relation between the number of updates performed per complex read. Table 6.1 shows the frequencies for each complex query and SF used in the Interactive v1 workload (Chapter 6).

Query	SF1	SF3	SF10	SF30	SF100	SF300	SF1 000	SF3 000
1	26	26	26	26	26	26	26	26
2	37	37	37	37	37	37	37	37
3	69	79	92	106	123	142	165	189
4	36	36	36	36	36	36	36	36
5	57	61	66	72	78	84	91	98
6	129	172	236	316	434	580	796	1063
7	87	72	54	48	38	32	25	21
8	45	27	15	9	5	3	1	1
9	157	209	287	384	527	705	967	1292
10	30	32	35	37	40	44	47	51
11	16	17	19	20	22	24	26	28
12	44	44	44	44	44	44	44	44
13	19	19	19	19	19	19	19	19
14	49	49	49	49	49	49	49	49

Table 6.1: Frequencies for each Interactive complex query and SF.

Finally, short reads are inserted in order to balance the ratio between reads and writes, and to simulate the behavior of a real user of the social network. For each complex read instance, a sequence of short reads is planned. There are two types of short read sequences: Person centric and Message centric. Depending on the type of the complex read, one of them is chosen. Each sequence consists of a set of short reads which are issued in a row. The issue time assigned to each short read in the sequence is determined at run time, and is based on the completion time of the complex read it depends on. The substitution parameters for short reads are taken from the results of previously executed queries, including both complex and short reads:

- Complex reads: IC 1 IC 2 IC 3 IC 7 IC 8 IC 9 IC 10 IC 11 IC 12 IC 14v1 IC 14v2
- Short reads: IS 2 IS 3 IS 5 IS 6 IS 7

To see which short and complex queries can potentially trigger additional short query queries, see Table 6.2.

Once a short read sequence is issued (and provided that sufficient substitution parameters exist), there is a probability that another short read sequence is issued. This probability decreases for each new sequence issued.<sup>1</sup> Since the same random number generator seed is used across executions, the workload is deterministic.

The specified frequencies, implicitly define the query ratios between queries of different types, as well as a default target throughput. However, the Test Sponsor may specify a different target throughput to test, by “squeezing” together or “stretching” apart the queries of the workload. This is achieved by means of the “Time Compression Ratio” that is multiplied by the frequencies (see Table 6.1). Therefore, different throughputs can be tested while maintaining the relative ratios between the different query types.

**Warning.** Note that in the current implementation of SNB Interactive v1, short queries are only produced if updates are enabled. In the absence of updates, no short queries will be executed.

<sup>1</sup>The probability can be adjusted using the `ldbc.snb.interactive.short_read_dissipation` configuration option.

	IS 1	IS 2	IS 3	IS 4	IS 5	IS 6	IS 7
IC 1	⊗	⊗	⊗				
IC 2	⊗	⊗	⊗	⊗	⊗	⊗	⊗
IC 3	⊗	⊗	⊗				
IC 7	⊗	⊗	⊗	⊗	⊗	⊗	⊗
IC 8	⊗	⊗	⊗	⊗	⊗	⊗	⊗
IC 9	⊗	⊗	⊗	⊗	⊗	⊗	⊗
IC 10	⊗	⊗	⊗				
IC 11	⊗	⊗	⊗				
IC 12	⊗	⊗	⊗				
IC 14	⊗	⊗	⊗				
IS 2	⊗	⊗	⊗	⊗	⊗	⊗	⊗
IS 3	⊗	⊗	⊗				
IS 5	⊗	⊗	⊗				
IS 6	⊗	⊗	⊗				
IS 7	⊗	⊗	⊗	⊗	⊗	⊗	⊗

Table 6.2: Short read queries (columns) potentially triggered after given complex/short read queries (rows).

## 7 INTERACTIVE v2 WORKLOAD

*This chapter is based on the TPCTC 2023 paper “The LDBC Social Network Benchmark Interactive Workload v2: A Transactional Graph Query Benchmark with Deep Delete Operations” [65], co-authored by several members of the SNB task force.*

### Work-in-Progress

The Interactive v2 workload is currently work-in-progress. As of January 2024, commissioning audits for this workload is not yet possible.

### Related Software Components

- Datagen (Spark-based): [https://github.com/ldbc/ldbc\\_snb\\_datagen\\_spark](https://github.com/ldbc/ldbc_snb_datagen_spark)
- Driver: [https://github.com/ldbc/ldbc\\_snb\\_interactive\\_v2\\_driver](https://github.com/ldbc/ldbc_snb_interactive_v2_driver)
- Reference implementations: [https://github.com/ldbc/ldbc\\_snb\\_interactive\\_v2\\_impls](https://github.com/ldbc/ldbc_snb_interactive_v2_impls)

## 7.1 Overview

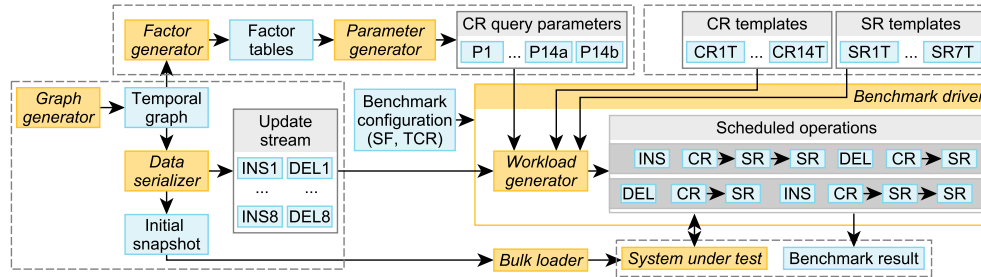


Figure 7.1: Components and workflow of the Interactive v2 workload. The corresponding sections are shown in green circles (§). Legend: Software component Data artifact

## 7.2 Operations

The LDBC SNB Interactive v2 workload uses four types of operations. There are 14 complex and 7 short read queries. Update operations include 8 inserts and, newly introduced in the Interactive v2 workload, 8 deletes. The workload mix consists of approximately 8% complex read, 72% short read, 20% insert, and 0.2% delete operations. The complex reads and the short reads are identical to the ones in Interactive v1, except for query 14, which was replaced to cover the *Cheapest path-finding* choke point.<sup>1</sup>

**Cheapest path-finding** While we strived to keep the changes to the queries minimal, we replaced Q14 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*, a key computational kernel and a language opportunity for GQL [20]. Therefore, we changed Q14 to use *cheapest paths* instead of *all shortest paths*.

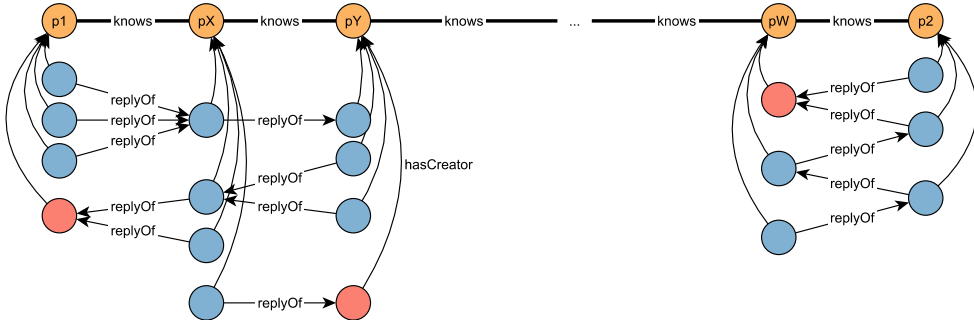
<sup>1</sup>The term *shortest paths* refers to the problem of finding *unweighted shortest paths*, which can be computed with BFS. The term *cheapest paths* refers to the *weighted shortest paths* problem, which can be solved using e.g. Dijkstra’s algorithm.



## 7.2.1 Complex Reads

## Interactive / complex / 14v2

IC 1  
IC 2  
IC 3  
IC 4  
IC 5  
IC 6  
IC 7  
IC 8  
IC 9  
IC 10  
IC 11  
IC 12  
IC 13  
IC 14v1  
IC 14v2

query	Interactive / complex / 14v2				
title	Trusted connection paths (v2)				
pattern	<div><div><p>Find a cheapest path on edges where numInteractions ≥ 1, using edge weight = max(round(40 - sqrt(numInteractions)), 1)</p><div><div>person1: Person</div><div>id = \$person1Id</div></div><div>knows*</div><div><div>person2: Person</div><div>id = \$person2Id</div></div></div><div><p>numInteractions = count(c)</p><div><div>personA: Person</div><div>hasCreator</div><div>c: Comment</div></div><div>knows</div><div><div>personB: Person</div><div>hasCreator</div><div>m: Message</div></div><div>replyOf</div></div><div><p>Example for finding a path between person1 and person2</p></div></div>				
description	<p><i>This query is used in SNB Interactive v2.</i></p> <p>Find a cheapest path between two given Persons with IDs \$person1Id and \$person2Id in the interaction subgraph. If there are multiple cheapest paths, any of them can be returned. Do not return any rows if there is no path between the Persons. The interaction subgraph is based on a projection of the Person-knows-Person graph. In this projection, only those knows edges are kept whose endpoint Persons have at least one interaction between them. An interaction is defined as a direct reply Comment (by one of the Persons) to a Message (by the other Person). The weights are defined as: <math>\max(\text{round}(40 - \sqrt{\text{numInteractions}}), 1)</math></p> <p><i>Note:</i> Interactions are counted both ways, e.g. if Alice knows Bob, Alice writes 2 reply Comments to Bob’s Messages and Bob writes 3 reply Comments to Alice’s Messages, their total number of interactions is 5 and the weight of the knows edge is 38.</p> <p><i>Remark:</i> Determinism is ensured by using square root followed by rounding. For all integers between 1 and 100 000, the square root’s fractional part is more than 10e-5 from 0.5, where the rounding could be non-deterministic based on floating point inaccuracies. As 10e-5 is significantly larger than the machine epsilon of IEEE 754 floats (both 32- and 64-bit), the floating point inaccuracies have no chance to affect the derived integer edge weights.</p>				
params	1	\$person1Id	ID	(b) There are no paths between the two Persons (b) There is a 4-hop path between the two Persons	
	2	\$person2Id	ID		
result	1	personIdsInPath	[ID]	C	Identifiers representing an ordered sequence of the Persons in the path
	2	pathWeight	64-bit Integer	C	
CPs	3.3, 5.3, 7.6, 7.7, 7.8, 8.1, 8.2, 8.3, 8.6				
relevance	This query tests the performance of cheapest path (weighted shortest path) computation.				

### 7.2.2 Short Reads

The short reads operations are identical to the ones in Interactive v1, see Section 6.2.

### 7.2.3 Insert Operations

See Section 5.1.

### 7.2.4 Delete Operations

See Section 5.2.

## 7.3 Parameter Curation

To prevent caching query results, the SNB Interactive v2 driver instantiates the parameterized complex read (IC<sub>3</sub>) query templates with different *substitution parameters* (a.k.a. parameter bindings). However, 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 [32, 80].

### 7.3.1 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 7.1, the parameter generation is a two-step process. The *factor generator* produces *factor tables*, which contain data cube-like summary statistics [29] 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)`.

### 7.3.2 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 IC<sub>3</sub>): we select values near a high percentile for the correlated and a low percentile for the anti-correlated case.

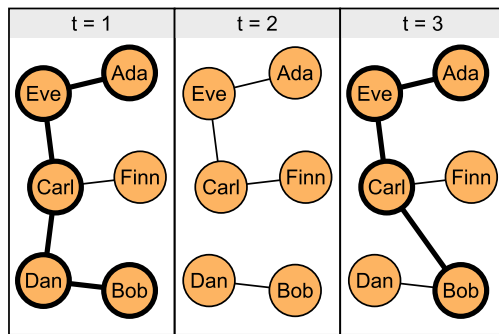
**Generating the parameters** The parameter candidates discovered by the previous approaches are stored in temporary tables. The parameter generation step uses these tables to select parameters for each day in the update stream.

### 7.3.3 Parameter Curation for Path-Finding Queries

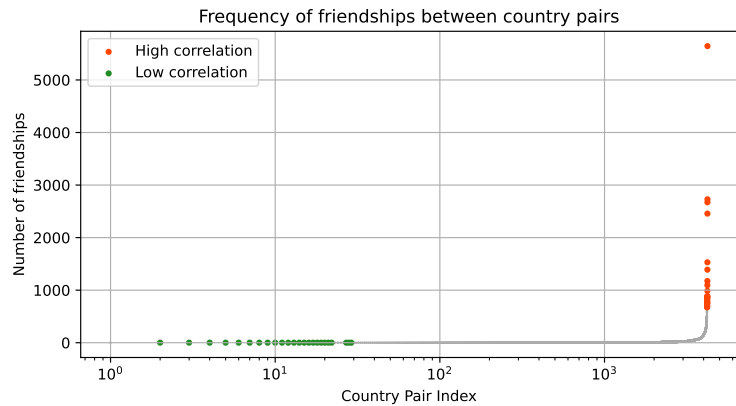
**The effect of deletes** A key distinguishing feature of graph data management systems is their first-class support for path queries [4]. We demonstrate why ensuring stable query runtimes for path queries is particularly challenging through the example of Figure 7.2a, where we query for the (unweighted) 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  $k_{\text{upper}}$  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  $k_{\text{lower}}$  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 [77].



(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.

Figure 7.2: Example graph and distribution for path curation.

### 7.3.4 Query Variants

The new workload introduces variants for three queries: `IC 3` , `IC 13` , `IC 14v2` .

**Complex read 3: Correlated vs. anti-correlated Countries** For `IC 3` , variant `IC 3(a)` starts from Countries that have a high correlation in the friendship network, while variant `IC 3(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 7.2b and select values at percentile 1.00 for variant (a) and percentile 0.01 for variant (b).

**Complex reads 13 and 14: 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 `IC 13` and `IC 14v2`. 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.

### 7.3.5 Parameter Generator Implementation

The parameter generator is implemented in Python using `NetworkKit` [77] and SQL queries executed by `DuckDB` [68]. Based on our experiments in [64, 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.

## 7.4 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 determined. 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.

### 7.4.1 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, 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,<sup>2</sup> we balance them such that each query type is expected to take the same amount of time to execute. For example, `IC 14 (new)` is expected to be more difficult than `IC 13`, 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 `IC 3` and `IC 14` trigger `IS 2`, 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. To see which short read queries are potentially triggered after given short read and complex read queries, see Table 6.2.

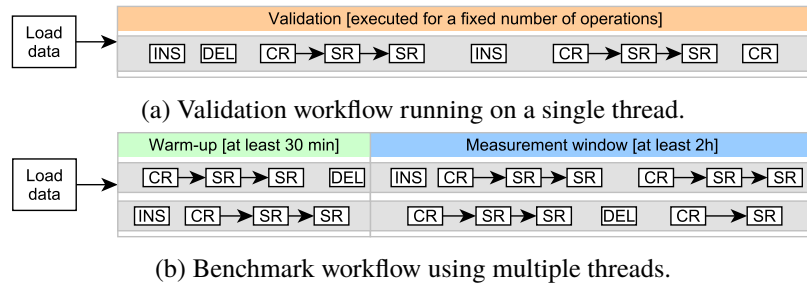


Figure 7.3: Workflow of driver modes in SNB Interactive v2.

### 7.4.2 Driver

**Driver modes** The SNB driver has two key modes of operation. In *cross-validation mode* (Figure 7.3a) the driver tests an implementation against the output of another implementation. To ensure deterministic results, operations in this mode are executed sequentially with no overlap between queries and updates. In *benchmark mode* (Figure 7.3b), 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 [24]: 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 (IC, IS, INS, DEL) executed per second. A lower TCR value implies a higher throughput.

**Individual execution times** To facilitate deeper analysis, the benchmark driver also collects all individual query execution times. Based on these, the benchmark reports must include statistics 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.

<sup>2</sup>Unlike in TPC-H [84] and SNB BI [80], which use *geometric mean* in their metrics.

## 8 BUSINESS INTELLIGENCE WORKLOAD

The Business Intelligence (BI) workload is the SNB’s analytical (OLAP) workload. As such, it defines complex read queries that touch a significant portion of the data (see Section 8.4). Additionally, it defines daily batches of updates over a 33-day period (see Section 8.5 for inserts and Section 8.6 for deletes).

### Related Publications

The BI workload was published in PVLDB 2022 [80].

### Related Software Components

- Datagen (Spark-based): [https://github.com/ldbc/ldbc\\_snb\\_datagen\\_spark](https://github.com/ldbc/ldbc_snb_datagen_spark)
- Driver and reference implementations: [https://github.com/ldbc/ldbc\\_snb\\_bi](https://github.com/ldbc/ldbc_snb_bi)

## 8.1 Overview

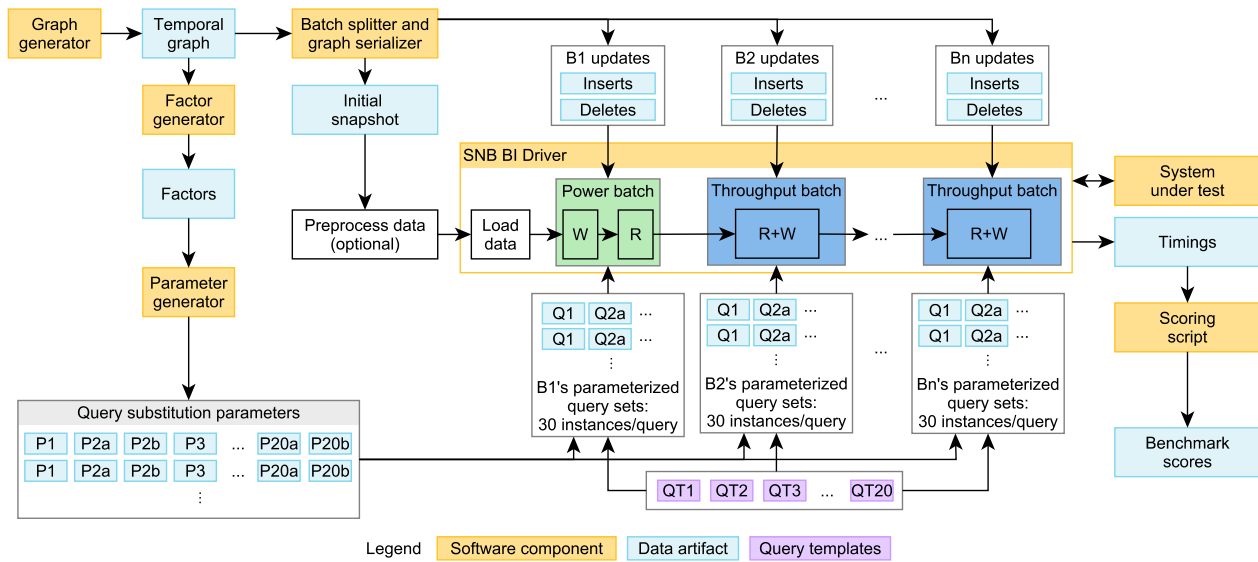


Figure 8.1: Main software components and data artifacts of the benchmark and their connection to the workflow executed by the BI benchmark driver.

An overview of the BI workload is shown in Figure 8.1. The rules for auditing workload implementations are given in Section 9.5.

## 8.2 Read Query Templates

SNB BI consists of 20 parameterized *read query templates*, referred to as *queries*. These search for graph patterns (often implying join-heavy operations on many-to-many edges), traverse hierarchies, and compute cheapest paths (a.k.a. weighted shortest paths). Additionally, they include filtering, grouping, aggregation, and sorting operators. While all queries explore a large portion of the graph, they only return the top- $k$  (typically 20 or 100) results, keeping their result sizes compact to avoid emphasizing the client-server network protocol’s role in the benchmark [67].

### 8.2.1 Choke Point-Based Design Methodology

LDBC’s query design process relies on the use of *choke points* (Appendix A), i.e. challenging aspects of query processing. SNB BI includes 38 choke points divided into 9 categories: aggregation performance, join performance, data access locality, expression calculation, correlated subqueries, parallelism and concurrency, graph specifics, language features, and update operations. Their coverage is shown in Table A.1. In the following, we discuss two challenges that are particularly prevalent in graph workloads.

#### 8.2.1.1 Explosive and redundant multi-joins

In recent years it has become clear that graph pattern matching, or equivalent multi-join queries over many-to-many relationships, typically generate very large intermediate results when executed with traditional join algorithms. This is especially the case for cyclical join graphs (corresponding to cyclic graph queries). It was proven in theory [58] and shown in practice [87, 50, 27] that “worst-case optimal” *multi-join* algorithms can avoid these large intermediates and outperform traditional joins. Following this, there has been increased attention on *redundancy* in join results (even when produced by worst-case optimal joins), which can be eliminated using *factorized* query processing techniques [12, 59, 35]. Graph pattern matching queries that contain large join patterns will trigger these phenomena.

#### 8.2.1.2 Expressive path finding

SNB BI contains queries that require an efficient implementation of shortest path finding between many pairs. Expressing such queries requires a query language which supports either path finding or recursion. The underlying system implementation must then handle this with an optimized execution strategy, as recursing to try all paths will not scale. As some of this path finding includes on-the-fly computed edges (joins) between nodes, the queries can benefit from *path expressions*, as proposed in Oracle’s PGQL language [70] and as part of the GQL and SQL/PGQ languages [20]. The path finding required by SNB BI not only tests connectivity (as supported in SPARQL), but also requires returning the *cheapest cost* along weighted paths (necessitating SPARQL extensions [52]).

### 8.2.2 Analysis of Selected Queries

In order to defeat trivializing complex query performance by query caching, benchmarks can use both frequent updates (which require invalidating caches or maintaining cached intermediates) as well as parameterized query templates. The BI workload features update batches, so parametrized *read query templates* are necessary to guard against this between the batches. In this section, we analyze four read query templates.

*Notation:* We denote the query parameters with the \$ symbol and discuss their generation in Section 8.3.

#### 8.2.2.1 Q11: Friend triangles

**BI 11** imposes two key difficulties. First, systems should efficiently filter the knows edges based on the location of their endpoint Persons (Country \$country) and the date range. Second, given a large number of knows edges even after filtering, efficient enumeration of personA–personB–personC triangles (a cyclic subgraph query) requires worst-case optimal multi-joins.

#### 8.2.2.2 Q14: International dialog

**BI 14** imposes different challenges depending on whether Countries \$country1 and \$country2 are correlated or anti-correlated (Section 8.3.3.1). For the ranking, *top-k pushdown* can be exploited: once a result for a City in \$country1 is obtained, extra restrictions in a selection can be added based on the value of this element. As the score of two Persons does not depend on any query parameters, precomputing and maintaining it as an attribute on the knows edge can be beneficial.



### 8.2.2.3 Q18: Friend recommendation

BI 18 is inspired by Twitter’s recommendation algorithm [34]. Implementations of this query can exploit factorization: systems can count the number of mutual friends without explicitly enumerating all  $\langle \text{person1}, \text{personM}, \text{person2} \rangle$  tuples.

### 8.2.2.4 Q20: Recruitment

BI 20 performs *graph projection* [5]. Instead of materializing this graph in the database, systems may represent it using a compact in-memory structure such as CSR (Compressed Sparse Row) [73]. To perform the cheapest path computation, a single-source shortest path algorithm (starting from  $\text{\$person2}$ ), such as Dijkstra’s algorithm, can be used. As the projected graph is independent of query parameters, precomputing and maintaining it can be beneficial.

## 8.3 Parameter Curation for BI Queries

### 8.3.1 The Need for Parameter Curation

A disadvantage of executing the same read query template with different parameters is that the intermediate results and runtimes can be severely influenced by the parameter values. This is particularly the case in SNB BI with its explosive joins, skewed out-degrees, skewed value distributions, correlated value distributions, and structural correlations. Moreover, the updates (including cascading deletes) can significantly change the portion of the graph reached by the same query executed at different times. In order to keep query performance understandable we need to actively *curate* parameters, such that different parameters executed at different logical times still lead to stable and, therefore, understandable results. We achieve this through *parameter curation* [32, 24], a data mining process of looking for parameter values with suitably similar characteristics.

### 8.3.2 Parameter Generation Steps

Our parameter curation process is a two-step process: we first generate *factors* followed by the *parameters* (Figure 8.1). These components are executed for each scale factor and are independent of the serialization format/layout of the data set.

#### 8.3.2.1 Factor Generator

The factor generator produces 21 *factor tables* containing summary statistics from the temporal graph, e.g. the number of Persons per City or the number of Messages per day for each Tag.

#### 8.3.2.2 Parameter Generation

To find suitable substitution parameters that (presumably) lead to the same amount of data access and thus similar runtimes, we first identify the factor table containing the summary statistics of the query’s parameters. For example, Q14’s template uses the parameters Country  $\text{\$country1}$  and Country  $\text{\$country2}$ . Therefore, we use the `countryPairsNumFriends` factor table which contains  $\text{\$country1}, \text{\$country2}$  pairs and the number of friendships on Person lives in  $\text{\$country1}$  and the other lives in  $\text{\$country2}$ . Using this table, we select the  $p$ th percentile from the distribution as the *anchor*, then rank the rest of the distribution based on their absolute difference from the anchor and take the top- $k$  values. We shuffle the values using a hash function to avoid introducing artificial locality, where e.g. subsequent queries start in nodes from the same ID range. Listing 8.1 shows the SQL query implementing the parameter generation for Q14a.

### 8.3.3 Parameter Curation for Graph Queries

We discuss two parameter curation cases that are particularly important in graph data management.



### 8.3.3.1 Correlated vs. Anti-Correlated Parameters

Our parameter curation provides a straightforward way of selecting start entities which are affected by (structural or attribute-level) correlation vs. anti-correlation: corresponding parameters can be found by selecting a high vs. low percentile as the anchor in the parameter generation query. For example, for Q14 (Section 8.4), we selected variant *a* to  $p = 0.98$  (correlated) and variant *b* to  $p = 0.03$  (anti-correlated).

### 8.3.3.2 Path Queries

SNB BI queries Q15, Q19, Q20 include cheapest path finding queries computed between given (sets of) Persons. These queries are particularly challenging for parameter curation: if there is no path between the two endpoints, query runtimes are significantly higher as the search has to traverse an entire connected component to ensure that no path exists. Moreover, the presence of a path between two nodes *at a given time* does not guarantee that it will always exist during the benchmark execution as deletions can render the endpoints of a path unreachable.

### 8.3.4 Query Variants

12 queries have a single variant, while 8 queries have two variants, yielding a total of 28 query variants. As a rule of thumb, variants *a* are expected to produce a longer runtime while variants *b* are expected to be simpler. Variants of Q2, Q8, Q16 are parametrized with a flashmob vs. a non-flashmob date. Variants of Q14 and Q19 select correlated vs. non-correlated Countries/Cities. Q10's variants differ in degree (a start Person with an average number of friends vs. only a few friends), while Q15's variants have different path lengths and time intervals (4 hops and one week vs. 2 hops and one month). Q20*a* selects endpoints where it is guaranteed that *no path exists*, while Q20*b* selects ones where there is guaranteed that a path exists.

### 8.3.5 Scalability and Reproducibility

#### 8.3.5.1 Scalability

The *factor generator* is part of the SNB Datagen and runs after the *temporal graph* has been created. It is implemented in Spark for distributed execution. While its computations use expensive, aggregation-heavy queries, the derived factor tables are *compact*, e.g. SF10 000 has only 20 GiB of factors in compressed Parquet format, the equivalent of approximately 100 GiB in CSV format, i.e. 1% of the total data set size. The *parameter generator* queries are executed in DuckDB [68], which supports vertical scalability and is capable of running the parameter generation for SF10 000 using less than 512 GiB memory.

```
SELECT country1, country2
FROM (
  SELECT
    country1,
    country2,
    abs(frequency - (
      SELECT percentile_disc(0.98) WITHIN GROUP (ORDER BY frequency) AS anchor FROM countryPairsNumFriends
    )) AS diff
  FROM countryPairsNumFriends
  ORDER BY diff, country1, country2
)
ORDER BY md5(concat(country1, country2))
LIMIT 50
```

Listing 8.1: Parameter generation SQL query for Q14*a*.

### 8.3.5.2 Reproducibility

It is important to guarantee that the parameter curation process is reproducible. To this end, we leverage that the Datagen and, consequently, the factor generator are reproducible. To ensure that the parameter generation queries yield deterministic results we define a total ordering in each query. To provide deterministic shuffling we base the ordering on MD5 hashes (instead of the actual attribute values), see Listing 8.1.

## 8.4 Reads

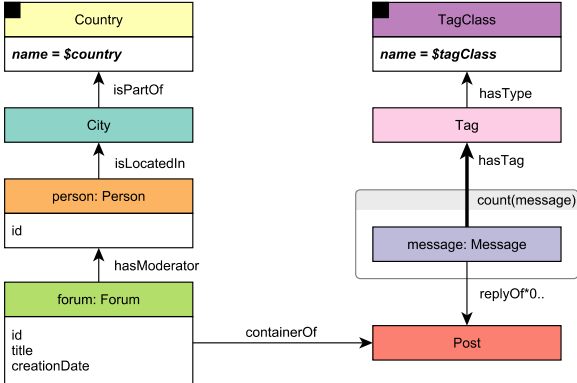
### BI / read / 1

BI 1	query	BI / read / 1				
BI 2	title	Posting summary				
BI 3	pattern	<div><div>▼message: Message</div><div>creationDate &lt; \$datetime</div><div>length year(creationDate)</div></div>				
BI 4						
BI 5						
BI 6						
BI 7						
BI 8	description	Given a \$datetime, find all Messages created before that moment. Group them by a 3-level grouping:				
BI 9						
BI 10		1. by year of creation				
BI 11		2. for each year, group into Message types: is Comment or not				
BI 12		3. for each year-type group, split into four groups based on length of their content				
BI 13		• 0: 0 ≤ length < 40 (short)				
BI 14		• 1: 40 ≤ length < 80 (one liner)				
BI 15		• 2: 80 ≤ length < 160 (tweet)				
BI 16	• 3: 160 ≤ length (long)					
BI 17	params	1	\$datetime	DateTime		
BI 20						
	result	1	year	32-bit Integer	R	year(message.creationDate)
		2	isComment	Boolean	M	True for Comments, False for Posts
		3	lengthCategory	32-bit Integer	C	0 for short, 1 for one-liner, 2 for tweet, 3 for long
		4	messageCount	64-bit Integer	A	Total number of Messages in that group
		5	averageMessageLength	32-bit Float	A	Average length of the Message content in that group
		6	sumMessageLength	64-bit Integer	A	Sum of all Message content lengths
		7	percentageOfMessages	32-bit Float	A	Number of Messages in group as a percentage of all messages created before the given date
	sort	1	year	↓		
		2	isComment	↑	False < True, i.e. Posts come first and Comments second	
		3	lengthCategory	↑		
	limit	n/a				
	CPs	1.2, 3.2, 4.1, 4.2, 8.5				

## BI / read / 2

query	BI / read / 2				
title	Tag evolution				
pattern	<pre>graph TD     TagClass -- hasType --&gt; Tag     Tag -- hasTag --&gt; Message1     Tag -- hasTag --&gt; Message2     subgraph Window1 [countWindow1 = count(message)]         Message1["message: Message creationDate in [\$date, \$date+100 days]"]     end     subgraph Window2 [countWindow2 = count(message)]         Message2["message: Message creationDate in [\$date+100 days, \$date+200 days]"]     end</pre>				
description	Find the Tags under a given \$tagClass that were used in Messages during in the 100-day time window starting at \$date and compare it with the 100-day time window that follows. For the Tags and for both time windows, compute the count of Messages.				
params	1	\$date	Date	Based on the creation day – TagClass – number of Messages factor table: (a) A flashmob date (b) A non-flashmob date	
	2	\$tagClass	Long String	For both (a) and (b), TagClasses with a similar amount of Messages are selected	
result	1	tag.name	Long String	R	
	2	countWindow1	32-bit Integer	A	Occurrences of the tag during the first time window
	3	countWindow2	32-bit Integer	A	Occurrences of the tag during the second time window
	4	diff	32-bit Integer	A	Absolute difference of countWindow1 and countWindow2
sort	1	diff	↓		
	2	tag.name	↑		
limit	100				
CPs	2.4, 3.1, 3.2, 4.1, 4.2, 4.3, 5.3, 6.1, 8.2, 8.5				

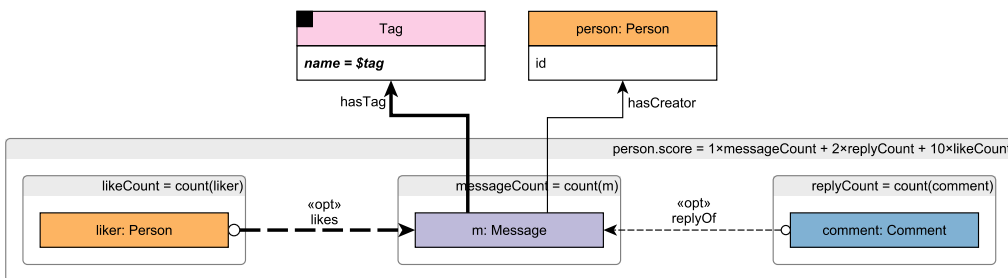
## BI / read / 3

query	BI / read / 3																													
title	Popular topics in a country																													
pattern																														
description	<p>Given a \$tagClass and a \$country, find all the Forums created in the given \$country, containing at least one Message with Tags belonging directly to the given \$tagClass, and count the Messages by the Forum which contains them.</p> <p>The location of a Forum is identified by the location of the Forum’s moderator.</p>																													
params	<table><tr><td>1</td><td>\$tagClass</td><td>Long String</td><td>TagClasses with a similar amount of Messages are selected</td></tr><tr><td>2</td><td>\$country</td><td>Long String</td><td>Big Countries are selected</td></tr></table>					1	\$tagClass	Long String	TagClasses with a similar amount of Messages are selected	2	\$country	Long String	Big Countries are selected																	
1	\$tagClass	Long String	TagClasses with a similar amount of Messages are selected																											
2	\$country	Long String	Big Countries are selected																											
result	<table><tr><td>1</td><td>forum.id</td><td>ID</td><td>R</td><td></td></tr><tr><td>2</td><td>forum.title</td><td>Long String</td><td>R</td><td></td></tr><tr><td>3</td><td>forum.creationDate</td><td>DateTime</td><td>R</td><td></td></tr><tr><td>4</td><td>person.id</td><td>ID</td><td>R</td><td></td></tr><tr><td>5</td><td>messageCount</td><td>32-bit Integer</td><td>A</td><td></td></tr></table>					1	forum.id	ID	R		2	forum.title	Long String	R		3	forum.creationDate	DateTime	R		4	person.id	ID	R		5	messageCount	32-bit Integer	A	
1	forum.id	ID	R																											
2	forum.title	Long String	R																											
3	forum.creationDate	DateTime	R																											
4	person.id	ID	R																											
5	messageCount	32-bit Integer	A																											
sort	<table><tr><td>1</td><td>messageCount</td><td>↓</td><td></td></tr><tr><td>2</td><td>forum.id</td><td>↑</td><td></td></tr></table>					1	messageCount	↓		2	forum.id	↑																		
1	messageCount	↓																												
2	forum.id	↑																												
limit	20																													
CPs	1.1, 1.2, 1.3, 2.1, 2.2, 2.4, 3.3, 8.2																													

## BI / read / 4

query	BI / read / 4				
title	Top message creators by country				
pattern	<div><div><div>1. select top 100 forums based on memberCount in country</div><div><div>Country</div><div>name</div></div><div>isPartOf</div><div><div>City</div></div><div>isLocatedIn</div><div><div>memberCount = count(member)</div><div><div>member: Person</div></div></div><div>hasMember</div><div><div>forum: Forum</div><div>creationDate &gt; \$date</div></div></div></div> <div><div>2. for each country, for each of the top 100 forums (topForum1), count the Messages made by Persons who are members of any of the top 100 forums (topForum2)</div><div><div><div>topForum1: Forum</div><div>containerOf</div><div>Post</div></div><div><div>messageCount = count(message)</div><div><div>Message</div></div></div><div>replyOf*0..</div><div><div>topForum2: Forum</div><div>is in top 100 forum, can be equal to topForum1</div></div><div>hasMember</div><div><div>person: Person</div><div>id firstName lastName creationDate</div></div><div>hasCreator</div></div></div>				
description	<p>Find the most popular Forums by Country, where the popularity of a Forum is measured by the number of members that Forum has from a given Country and the Forum was created after a given \$date.</p> <p>Calculate the top 100 most popular Forums. If a Forum is popular in multiple countries, it should only be calculated once with its largest membership. In case of a tie, the Forum with the smaller id value should be selected.</p> <p>For each member Person of the 100 most popular Forums, count the number of Messages (messageCount) they made in any of those (most popular) Forums. Also include those member Persons who have not posted any Messages (have a messageCount of 0).</p>				
params	1	\$date	Date	Selected from the first 30 days of the network	
result	1	person.id	ID	R	
	2	person.firstName	String	R	
	3	person.lastName	String	R	
	4	person.creationDate	DateTime	R	
	5	messageCount	32-bit Integer	A	
sort	1	messageCount	↓		
	2	person.id	↑		
limit	100				
CPs	1.2, 1.3, 2.1, 2.2, 2.3, 2.4, 3.3, 5.3, 6.1, 8.2, 8.4				

## BI / read / 5

query	BI / read / 5				
title	Most active posters of a given topic				
pattern	 <pre>graph TD     Tag[Tag: name = \$tag] -- hasTag --&gt; Message[m: Message]     Person1[person: Person] -- hasCreator --&gt; Message     Liker[liker: Person] -- "«opt» likes" --&gt; Message     Comment[comment: Comment] -- "«opt» replyOf" --&gt; Message     subgraph LIKER_BOX [likeCount = count(liker)]         Liker     end     subgraph MESSAGE_BOX [messageCount = count(m)]         Message     end     subgraph COMMENT_BOX [replyCount = count(comment)]         Comment     end     Person1 -- "person.score = 1*messageCount + 2*replyCount + 10*likeCount" --&gt; Score[ ]</pre>				
description	<p>Get each Person (person) who has created a Message (message) with a given \$tag (direct relation, not transitive). Considering only these Messages, for each Person node:</p> <ul style="list-style-type: none"><li>Count its Messages (messageCount).</li><li>Count likes (likeCount) to its Messages.</li><li>Count Comments (replyCount) in reply to its Messages.</li></ul> <p>The score is calculated according to the following formula: <math>1 \times \text{messageCount} + 2 \times \text{replyCount} + 10 \times \text{likeCount}</math>.</p>				
params	1	\$tag	Long String	Tags with a similar amount of Messages are selected. To avoid caching, different Tags should be used than the ones in Q6 and Q7.	
result	1	person.id	ID	R	
	2	replyCount	32-bit Integer	A	
	3	likeCount	32-bit Integer	A	
	4	messageCount	32-bit Integer	A	
	5	score	32-bit Integer	A	
sort	1	score	↓		
	2	person.id	↑		
limit	100				
CPs	1.2, 2.3, 2.6, 8.2				

## BI / read / 6

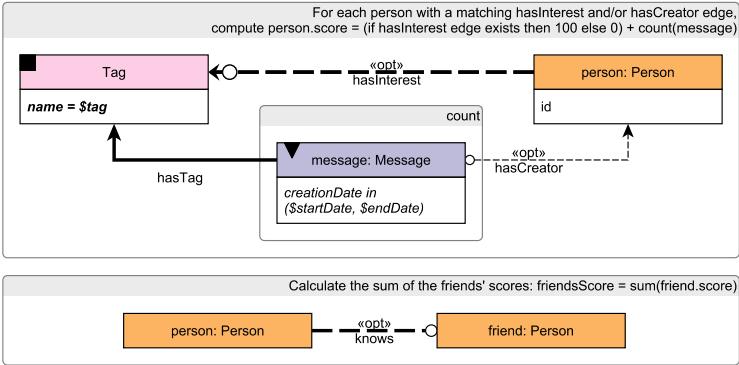
BI 1	query	BI / read / 6			
BI 2	title	Most authoritative users on a given topic			
BI 3	pattern				
BI 4	description	<p>Given a \$tag, find all Persons (person1) that ever created a Message with the \$tag. For each of these Persons (person1) compute their “authority score” as follows:</p> <ul style="list-style-type: none"> <li>• The “authority score” is the sum of “popularity scores” of the Persons (person2) that liked any of that Person’s Messages with the given \$tag (same criterion as for message1).</li> <li>• A Person’s (person2) “popularity score” is defined as the total number of likes (by any Person person3) on any of their Messages (message2).</li> </ul>			
BI 5	params	1	\$tag	Long String	Tags with a similar amount of Messages are selected. To avoid caching, different Tags should be used than the ones in Q5 and Q7.
BI 6	result	1	person1.id	ID	R
BI 7		2	authorityScore	32-bit Integer	A
BI 8	sort	1	authorityScore	↓	
BI 9		2	person1.id	↑	
BI 10	limit	100			
BI 11	CPs	1.2, 2.3, 2.6, 3.3, 6.1, 8.2			
BI 12	relevance	Computing the authority scores might involve computing the popularity score for the same Person multiple times. Implementations are advised to avoid such redundant computations.			

**BI / read / 7**

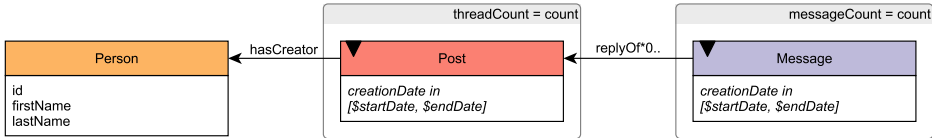
BI 1	query	BI / read / 7			
BI 2	title	Related topics			
BI 3	pattern				
BI 4	description	Find all Messages that have a given \$tag. Find the related Tags attached to (direct) reply Comments of these Messages, but only of those reply Comments that do not have the given \$tag. Group the related Tags by name, and get the count of replies in each group.			
BI 5	params	1	\$tag	Long String	Tags with a similar amount of Messages are selected. To avoid caching, different Tags should be used than the ones in Q5 and Q6.
BI 6	result	1	relatedTag.name	Long String	R
BI 7		2	count	32-bit Integer	A
BI 8	sort	1	count	↓	
BI 9		2	relatedTag.name	↑	
BI 10	limit	100			
BI 11	CPs	1.4, 3.3, 5.2, 8.1			



## BI / read / 8

query	BI / read / 8			
title	Central person for a tag			
pattern	<div><div>For each person with a matching hasInterest and/or hasCreator edge, compute <math>\text{person.score} = (\text{if hasInterest edge exists then } 100 \text{ else } 0) + \text{count}(\text{message})</math></div><div>Calculate the sum of the friends' scores: <math>\text{friendsScore} = \text{sum}(\text{friend.score})</math></div></div>			
description	<p>Given a \$tag, find all Persons that are interested in the \$tag and/or have written a Message (Post or Comment) with a creationDate after a given \$startDate and that has a given \$tag. For each Person, compute the score as the sum of the following two aspects:</p> <ul style="list-style-type: none"><li>• 100, if the Person has this \$tag as their interest, or 0 otherwise</li><li>• number of Messages by this Person with the given \$tag</li></ul> <p>Also, for each Person, compute the sum of the score of the Person’s friends (friendsScore).</p>			
params	<div><div>1</div><div>\$tag</div><div>Long String</div><div>Tags with a similar amount of Messages are selected</div></div> <div><div>2</div><div>\$startDate</div><div>Date</div><div>(a): A range during which a flashmob event happened (it should yield at least a 5× difference)</div></div> <div><div>3</div><div>\$endDate</div><div>Date</div><div>(b): A regular range (does not include a flashmob event)</div></div>			
result	<div><div>1</div><div>person.id</div><div>ID</div><div>R</div><div></div></div> <div><div>2</div><div>score</div><div>32-bit Integer</div><div>A</div><div></div></div> <div><div>3</div><div>friendsScore</div><div>32-bit Integer</div><div>A</div><div>The sum of the score of the person’s friends</div></div>			
sort	<div><div>1</div><div>score + friendsScore</div><div>↓</div><div></div></div> <div><div>2</div><div>person.id</div><div>↑</div><div></div></div>			
limit	100			
CPs	1.2, 2.1, 2.3, 3.2, 5.3, 8.2, 8.4, 8.5			
relevance	Similarly to BI 16, there are two major ways to compute this query: (1) creating an induced subgraph of the interested Persons and their friends and performing the scoring on this graph or (2) performing the scoring without creating an induced subgraph and scoring the friends of a Person on-the-fly. The first approach is more efficient as it avoids redundant computations, however, specifying it needs support for composable graph queries.			

**BI / read / 9**

query	BI / read / 9				
title	Top thread initiators				
pattern					
description	<p>For each Person, count the number of Posts they created in the time interval [\$startDate, \$endDate] (equivalent to the number of threads they initiated) and the number of Messages in each of their (transitive) reply trees, including the root Post of each tree. When calculating Message counts only consider Messages created within the given time interval.</p> <p>Return each Person, number of Posts they created, and the count of all Messages that appeared in the reply trees (including the Post at the root of tree).</p>				
params	1	\$startDate	Date	Selected around the same date	
	2	\$endDate	Date	80-100 days after the \$startDate	
result	1	person.id	ID	R	
	2	person.firstName	String	R	
	3	person.lastName	String	R	
	4	threadCount	32-bit Integer	A	The number of Posts created by that Person (the number of threads initiated)
	5	messageCount	32-bit Integer	A	The number of Messages created in all the threads this Person initiated
sort	1	messageCount	↓		
	2	person.id	↑		
limit	100				
CPs	1.2, 2.2, 2.3, 2.6, 3.2, 7.2, 7.3, 7.4, 8.1, 8.5				

## BI / read / 10

query	BI / read / 10				
title	Experts in social circle				
pattern					
description	<p>Given a Person startPerson with ID \$personID, find all other Persons (expertCandidatePerson) that live in a given \$country and are connected to the startPerson on a <i>shortest path</i> with length in range [\$minPathDistance, \$maxPathDistance] through the knows relation.</p> <p>For each of these expertCandidatePerson nodes, retrieve all of their Messages that contain at least one Tag belonging to a given \$tagClass (direct relation not transitive). For each Message, retrieve all of its Tags.</p> <p>Group the results by Persons and Tags, then count the Messages by a certain Person having a certain Tag.</p>				
params	1	\$personId	ID	(a) Persons with an average degree of knows edges are selected (b) Persons who have only one friend and that Person has two friends in total (including the original Person)	
	2	\$country	String	Select mid-sized Countries	
	3	\$tagClass	Long String	TagClasses with a similar degree of hasType edges are selected	
	4	\$minPathDistance	32-bit Integer	3	
	5	\$maxPathDistance	32-bit Integer	4	
result	1	expertCandidatePerson.id	ID	R	
	2	tag.name	Long String	R	
	3	messageCount	32-bit Integer	A	Number of Messages created by that Person containing that Tag
sort	1	messageCount	↓		
	2	tag.name	↑		
	3	expertCandidatePerson.id	↑		
limit	100				
CPs	1.2, 1.3, 2.3, 2.4, 2.6, 3.3, 5.3, 7.1, 7.2, 7.3, 8.1, 8.6				

**BI / read / 11**

query	BI / read / 11				
title	Friend triangles				
pattern					
description	<p>For a given \$country, count all the distinct triples of Persons such that:</p> <ul style="list-style-type: none"><li>• personA is friend of personB,</li><li>• personB is friend of personC,</li><li>• personC is friend of personA,</li></ul> <p>and these friendships were created in the range [\$startDate, \$endDate].</p> <p>Distinct means that given a triple <math>t_1</math> in the result set <math>R</math> of all qualified triples, there is no triple <math>t_2</math> in <math>R</math> such that <math>t_1</math> and <math>t_2</math> have the same set of elements.</p>				
params	<div>1</div>	<div>\$country</div>	<div>Long String</div>	<div>Selected from the largest Countries (India, China)</div>	
	<div>2</div>	<div>\$startDate</div>	<div>Date</div>	<div>Selected from a 30-day interval towards the end of the simulation time</div>	
	<div>3</div>	<div>\$endDate</div>	<div>Date</div>	<div>Selected to yield around a 100-day interval</div>	
result	<div>1</div>	<div>count</div>	<div>64-bit Integer</div>	<div>A</div>	
limit	n/a				
CPs	2.3, 2.5, 3.2				

**BI / read / 12**

query	BI / read / 12				
title	How many persons have a given number of messages				
pattern					
description	<p>For each Person, count the number of Messages they made (messageCount). Only count Messages with the following attributes:</p> <ul style="list-style-type: none"><li>• Its content is not empty (and consequently, the imageFile attribute is empty for Posts).</li><li>• Its creationDate is after \$startDate (exclusive, equality is not allowed).</li><li>• Its length is below the \$lengthThreshold (exclusive, equality is not allowed).</li><li>• It is written in any of the given \$languages.</li></ul> <ul style="list-style-type: none"><li>– The language of a Post is defined by its language attribute.</li><li>– The language of a Comment is that of the Post that initiates the thread where the Comment replies to.</li></ul> <p>The Post and Comments in the reply tree’s path (from the Message to the Post) do not have to satisfy the constraints for content, length, and creationDate.</p> <p>For each messageCount value, count the number of Persons with exactly messageCount Messages (with the required attributes).</p>				
params	<div>1</div>	<div>\$startDate</div>	<div>Date</div>	<div>Selected randomly from a 60-day interval.</div>	
	<div>2</div>	<div>\$lengthThreshold</div>	<div>32-bit Integer</div>	<div>Balanced against startDate to filter around 30% of the Messages within a language and keep the variance low.</div> <div>The selection of this parameter uses a factor table of bucketed Message lengths and creation dates.</div>	
	<div>3</div>	<div>\$languages</div>	<div>{String}</div>	<div>Only the most frequently used languages</div>	
result	<div>1</div>	<div>messageCount</div>	<div>32-bit Integer</div>	<div>A</div>	<div>Number of Messages created</div>
	<div>2</div>	<div>personCount</div>	<div>32-bit Integer</div>	<div>A</div>	<div>Number of Persons with messageCount Messages</div>
sort	<div>1</div>	<div>personCount</div>	<div>↓</div>		
	<div>2</div>	<div>messageCount</div>	<div>↓</div>		
limit	n/a				
CPs	1.1, 1.2, 1.4, 2.6, 3.2, 4.2, 4.3, 8.1, 8.2, 8.3, 8.4, 8.5				

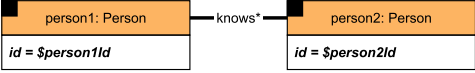
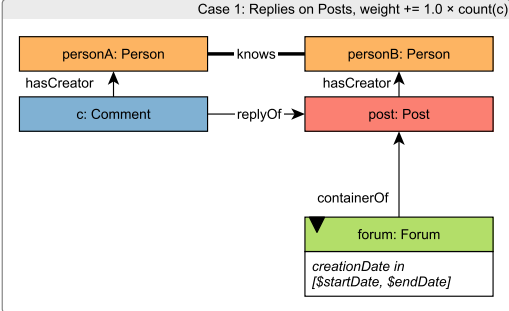
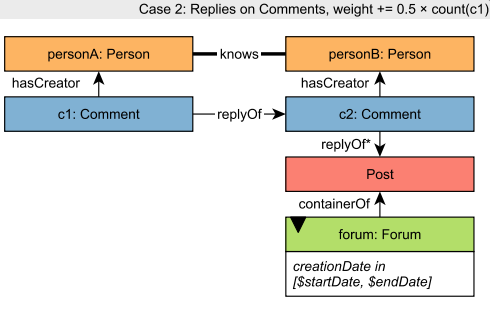
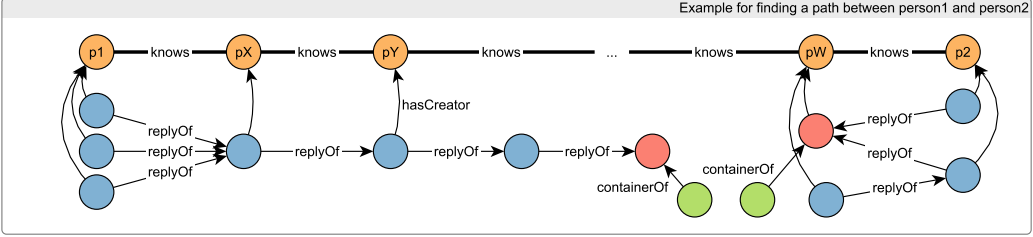
## BI / read / 13

query	BI / read / 13				
title	Zombies in a country				
pattern	<div><div>1. zombies = collect(zombie)</div><div><div><div>Country</div><div>name = \$country</div></div><div><div>City</div><div>isPartOf</div><div>Country</div></div><div><div>zombie: Person</div><div>creationDate &lt; \$endDate and (messageCount / months &lt; 1)</div></div><div><div>message: Message</div><div>messageCount = count(message) creationDate &lt; \$endDate</div></div><div><div>zombie: Person</div><div>hasCreator</div><div>message: Message</div></div></div></div> <div><div>2. For each zombie IN zombies, calculate: zombieScore = zombieLikeCount / totalLikeCount</div><div><div><div>likerPerson: Person</div><div>totalLikeCount = count(likerPerson) creationDate &lt; \$endDate</div></div><div><div>Message</div><div>hasCreator</div><div>zombie: Person</div></div><div><div>likerZombie: Person</div><div>zombieLikeCount = count(likerZombie) creationDate &lt; \$endDate and likerZombie IN zombies</div></div><div><div>likerPerson: Person</div><div>likes</div><div>Message</div></div><div><div>Message</div><div>likes</div><div>likerZombie: Person</div></div></div></div>				
description	<p>Find zombies within the given \$country, and return their zombie scores. A zombie is a Person created before the given \$endDate, which has created an average of [0, 1) Messages per month, during the time range between profile’s creationDate and the given \$endDate. The number of months spans the time range from the creationDate of the profile to the \$endDate with partial months on both end counting as one month (e.g. a creationDate of Jan 31 and an \$endDate of Mar 1 result in 3 months).</p> <p>For each zombie, calculate the following:</p> <ul style="list-style-type: none"><li>• zombieLikeCount: the number of likes received from other zombies.</li><li>• totalLikeCount: the total number of likes received.</li><li>• zombieScore: zombieLikeCount / totalLikeCount. If the value of totalLikeCount is 0, the zombieScore of the zombie should be 0.0.</li></ul> <p>For both zombieLikeCount and totalLikeCount, only consider likes received from profiles that were created before the given \$endDate.</p>				
params	1	\$country	Long String	Selected from the largest Countries (India, China)	
	2	\$endDate	Date	Selected from the last days of the initial data set	
result	1	zombie.id	ID	R	
	2	zombieLikeCount	32-bit Integer	A	
	3	totalLikeCount	32-bit Integer	A	
	4	zombieScore	32-bit Float	A	Determined as zombieLikeCount / totalLikeCount
sort	1	zombieScore	↓		
	2	zombie.id	↑		
limit	100				
CPs	1.2, 2.1, 2.3, 2.4, 2.6, 3.2, 3.3, 4.2, 5.1, 5.3, 8.2, 8.4, 8.5				

## BI / read / 14

BI 1	query	BI / read / 14			
BI 2	title	International dialog			
BI 3	pattern	<p>For each pair of countries, calculate the cost as a sum of cases #1–4. Cases that have a match add to the final score with the specified value. Each case only counts once, multiple matches do not increase to the score.</p>			
BI 5					
BI 6					
BI 7					
BI 8					
BI 9					
BI 10					
BI 11					
BI 12					
BI 13					
BI 14	description	<p>Consider all pairs of people (person1, person2) such that (1) they know each other, (2) one is located in a City of \$country1, and (3) the other is located in a City of \$country2. For each City of \$country1, return the highest scoring pair. If there are multiple top-scoring pairs in a city, return the pair with the lowest (person1.id, person2.id) using a lexicographical ordering.</p> <p>The score of a pair is defined as the sum of the subscores awarded for the following kinds of interaction. The initial value is score = 0.</p> <ol style="list-style-type: none"> <li>1. person1 has created a reply Comment to at least one Message by person2: score += 4</li> <li>2. person1 has created at least one Message that person2 has created a reply to: score += 1</li> <li>3. person1 liked at least one Message by person2: score += 10</li> <li>4. person1 has created at least one Message that was liked by person2: score += 1</li> </ol> <p>Consequently, the maximum score a pair can obtain is: 4 + 1 + 10 + 1 = 16.</p>			
BI 15					
BI 16					
BI 17					
BI 18					
BI 19					
BI 20					
BI 21	params	1	\$country1	Long String	(a) Correlated with parameter country2, i.e. the Countries are close and there are many Persons knowing each other
		2	\$country2	Long String	(b) Uncorrelated with parameter country2, i.e. the Countries are afar and there are few Persons knowing each other
BI 22	result	1	person1.id	ID	R
		2	person2.id	ID	R
		3	city1.name	Long String	R
		4	score	32-bit Integer	C
BI 23	sort	1	score	↓	
		2	person1.id	↑	
		3	person2.id	↑	
BI 24	limit	100			
BI 25	CPs	1.3, 1.4, 2.1, 3.1, 3.3, 5.1, 5.2, 5.3, 8.3, 8.4			

## BI / read / 15

BI 1	query	BI / read / 15			
BI 2	title	Trusted connection paths through forums created in a given timeframe			
BI 3	pattern	<p>Calculate the weight of the shortest path on knows edges between person1 and person2. Edge weights are determined as <math>1 / (\text{interaction score} + 1)</math>, where interaction score is the sum of cases #1 and #2 for the Person endpoints of the edge (tried both ways).</p> 			
BI 4		<p>Case 1: Replies on Posts, weight += 1.0 × count(c)</p> 			
BI 5		<p>Case 2: Replies on Comments, weight += 0.5 × count(c1)</p> 			
BI 6		<p>Example for finding a path between person1 and person2</p> 			
BI 7					
BI 8					
BI 9					
BI 10					
BI 11					
BI 12					
BI 13	description	<p>Given two Persons with IDs \$person1Id and \$person2Id, calculate the cost of the weighted shortest path between these two Persons, in the subgraph induced by the knows relationship. The interaction score of a knows edge is calculated based on the interactions of its Person endpoints:</p> <ul style="list-style-type: none"> <li>• Every direct reply (by one of the Persons) to a Post (by the other Person) is 1.0 point.</li> <li>• Every direct reply (by one of the Persons) to a Comment (by the other Person) is 0.5 points.</li> </ul> <p>Only consider Messages that were created in a Forum that was created within the timeframe (interval) [\$startDate, \$endDate]. Note that for Comments, the containing Forum is that of the Post that the comment (transitively) replies to. Also note that interactions are counted both ways.</p> <p>The weight for the shortest path algorithm is determined as <math>\frac{1}{\text{interaction score} + 1}</math>.</p> <p>The result of the query is a single number, the cost of the weighted shortest path. If no such path exists, the query should return -1.0.</p>			
BI 14					
BI 15					
BI 16					
BI 17					
BI 18					
BI 19					
BI 20					
	params	1	\$person1Id	ID	(a) \$person1Id – \$person2Id pair with a distance of 4 hops
		2	\$person2Id	ID	(b) \$person1Id – \$person2Id pair with a distance of 2 hops
		3	\$startDate	Date	(a) Small interval (approx. one week)
		4	\$endDate	Date	(b) Big interval (approx. one month)
	result	1	weight	32-bit Float	C
	limit	n/a			
	CPs	1.2, 2.1, 2.2, 2.4, 3.3, 5.1, 5.3, 7.2, 7.3, 7.6, 7.7, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6			



## BI / read / 16

query	BI / read / 16																				
title	Fake news detection																				
pattern	<div><div>For \$tagX/\$dayX in [tagA/dateA, tagB/dateB], compute scoreX = count(messageX)</div><div><div>1. Create an induced subgraph of Persons who created a Message with Tag \$tagX on \$dateX</div><div><div><div>tag: Tag</div><div>name = \$tagX</div></div><div>hasTag</div><div><div>Message</div><div>day(creationDate) = \$dateX</div></div><div>hasCreator</div><div><div>person: Person</div></div></div></div><div><div>2. In the subgraph, count the Messages (using the same conditions) from People with ≤ \$maxKnowsLimit friends</div><div><div><div>tag: Tag</div><div>name = \$tagX</div></div><div>hasTag</div><div><div>messageX: Message</div><div>day(creationDate) = \$dateX</div></div><div>hasCreator</div><div><div>person: Person</div><div>count ≤ \$maxKnowsLimit</div><div>«opt» knows</div><div>Person</div></div></div></div></div>																				
description	<p>Given two Tag/date pairs (\$tagA/\$dateA and \$tagB/\$dateB), for each pair \$tagX/\$dateX:</p> <ul style="list-style-type: none"><li>• Create an induced subgraph between Persons where for each pair of Persons person1/person2, both have created a Message on the day of \$dateX with Tag \$tagX.</li><li>• In the induced subgraph, only keep pairs of Persons who have at most maxKnowsLimit friends (in the induced subgraph).</li><li>• For these Persons, count the number of Messages created on \$dateX with Tag \$tagX.</li></ul> <p>Return Persons who had at least one Messages for both \$tagA/\$dateA and \$tagB/\$dateB ranked by their total number of Messages (descending).</p>																				
params	<table><tr><td>1</td><td>\$tagA</td><td>Long String</td><td>(a) \$tagA/\$dateA, \$tagB/\$dateB are both selected to be a flashmob Tag/date combination (b) \$tagA/\$dateA, \$tagB/\$dateB are both selected to be a non-flashmob Tag/date combination</td></tr><tr><td>2</td><td>\$dateA</td><td>Date</td><td></td></tr><tr><td>3</td><td>\$tagB</td><td>Long String</td><td></td></tr><tr><td>4</td><td>\$dateB</td><td>Date</td><td></td></tr><tr><td>5</td><td>\$maxKnowsLimit</td><td>32-bit Integer</td><td>Selected between 3 and 6</td></tr></table>	1	\$tagA	Long String	(a) \$tagA/\$dateA, \$tagB/\$dateB are both selected to be a flashmob Tag/date combination (b) \$tagA/\$dateA, \$tagB/\$dateB are both selected to be a non-flashmob Tag/date combination	2	\$dateA	Date		3	\$tagB	Long String		4	\$dateB	Date		5	\$maxKnowsLimit	32-bit Integer	Selected between 3 and 6
1	\$tagA	Long String	(a) \$tagA/\$dateA, \$tagB/\$dateB are both selected to be a flashmob Tag/date combination (b) \$tagA/\$dateA, \$tagB/\$dateB are both selected to be a non-flashmob Tag/date combination																		
2	\$dateA	Date																			
3	\$tagB	Long String																			
4	\$dateB	Date																			
5	\$maxKnowsLimit	32-bit Integer	Selected between 3 and 6																		
result	<table><tr><td>1</td><td>person.id</td><td>ID</td><td>R</td><td></td></tr><tr><td>2</td><td>messageCountA</td><td>32-bit Integer</td><td>A</td><td>Message count for \$tagA/\$dateA</td></tr><tr><td>3</td><td>messageCountB</td><td>32-bit Integer</td><td>A</td><td>Message count for \$tagB/\$dateB</td></tr></table>	1	person.id	ID	R		2	messageCountA	32-bit Integer	A	Message count for \$tagA/\$dateA	3	messageCountB	32-bit Integer	A	Message count for \$tagB/\$dateB					
1	person.id	ID	R																		
2	messageCountA	32-bit Integer	A	Message count for \$tagA/\$dateA																	
3	messageCountB	32-bit Integer	A	Message count for \$tagB/\$dateB																	
sort	<table><tr><td>1</td><td>messageCountA + messageCountB</td><td>↓</td><td></td></tr><tr><td>2</td><td>person.id</td><td>↑</td><td></td></tr></table>	1	messageCountA + messageCountB	↓		2	person.id	↑													
1	messageCountA + messageCountB	↓																			
2	person.id	↑																			
limit	20																				
CPs	5.3, 8.4, 8.5																				
relevance	<p>There are two major ways to compute this query: (1) create the induced subgraph as suggested by the specification (either as a view or in materialized form), or (2) skip creating the induced subgraph and perform on-the-fly check for the number of friends (who also posted at least one Message with the given Tag on the given date). The latter approach is easier to express in systems which do not provide graph views but might result in redundant computations (the query engine might repeatedly check whether a Person has at least one Message that satisfies the conditions).</p>																				

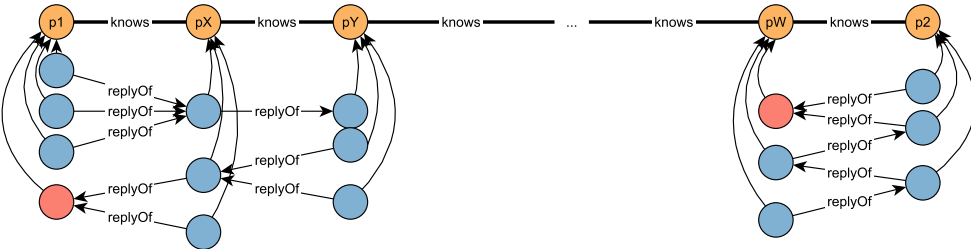
## BI / read / 17

BI 1	query	BI / read / 17		
BI 2	title	Information propagation analysis		
BI 3	pattern			
BI 4				
BI 5				
BI 6				
BI 7				
BI 8				
BI 9				
BI 10				
BI 11				
BI 12				
BI 13				
BI 14				
BI 15				
BI 16				
BI 17				
BI 18	description	<p>This query aims to identify instances of “information propagation” when a Person (person1) submits a Message (message1) with a given \$tag to a Forum (forum1). This is read by other members of forum1, Persons person2 and person3 (who must be different Persons). Some time later (specified by the \$delta parameter), these persons have a discussion with the same \$tag in a different Forum (forum2) where person1 is not a member. The discussion consists of a Message (message2) by person2 and a direct reply Comment (comment) by person3.</p>		
BI 19		<p>Return IDs of person1 with the number of interactions their Messages (might have) caused.</p>		
BI 20				
params	1	\$tag	Long String	Tags with a similar amount of Messages are selected
	2	\$delta	32-bit Integer	Measured in hours, selected to be between 8 and 16 hours.
result	1	person1.id	ID	R
	2	messageCount	32-bit Integer	A
sort	1	messageCount	↓	
	2	person1.id	↑	
limit	10			
CPs	2.1, 2.3, 2.5, 2.6, 8.1			

**BI / read / 18**

query	BI / read / 18				
title	Friend recommendation				
pattern	<div>For each person1 compute top-k(person2) based on mutualFriendCount</div>				
description	<p>For a given \$tag, for each person1 interested in \$tag, recommend new friends (person2) who</p> <ul style="list-style-type: none"><li>• do not yet know person1</li><li>• have at least one mutual friend with person1</li><li>• are also interested in \$tag.</li></ul> <p>Rank Persons person2 based on the number of mutual friends with person1.</p>				
params	1	\$tag	Long String	Tags with a similar amount of Persons are selected	
result	1	person1.id	ID	R	
	2	person2.id	ID	R	
	3	mutualFriendCount	32-bit Integer	A	
sort	1	mutualFriendCount	↓		
	2	person1.id	↑		
	3	person2.id	↑		
limit	20				
CPs	2.5, 2.6, 8.1				

## BI / read / 19

query	BI / read / 19			
title	Interaction path between cities			
pattern	<div><div><p>Find the shortest paths between all pairs of Persons in city1 and city2. The weight of a knows edge is based on the number of interactions between its Persons: <math>\text{knows.weight} = \max(\text{round}(40 - \sqrt{\text{numInteractions}}), 1)</math></p><div><div><div>city1: City</div><div><math>\text{id} = \\$city1Id</math></div><div>isLocatedIn</div><div>person1: Person</div></div><div><div>city2: City</div><div><math>\text{id} = \\$city2Id</math></div><div>isLocatedIn</div><div>person2: Person</div></div><div>compute weighted shortest paths on knows.weight</div></div></div><div><p>Reply from personA to personB's Message, or personB to personA's Message</p><div><div><div>personA: Person</div><div>knows</div><div>personB: Person</div></div><div><div>hasCreator</div><div>m1: Message</div></div><div><div>hasCreator</div><div>m2: Message</div></div><div><div>replyOf</div><div>m1: Message</div></div><div><div>replyOf</div><div>m2: Message</div></div></div></div><div><p>Example for finding a path between person1 and person2</p></div></div>			
description	<p>Given two Cities with IDs <math>\\$city1Id</math>, <math>\\$city2Id</math>, find Persons <math>person1</math>, <math>person2</math> living in these Cities (respectively) with the <i>cheapest</i> interaction path between them.</p> <p>The cheapest path is equivalent to the <i>weighted shortest</i> path. It is computed on a subgraph of the Person-knows-Person graph with the edge weights based on the number of interactions. An <i>interaction</i> is a direct reply Comments from one Person to Messages by the other Person. Only knows edges with at least one interaction between their endpoint Persons are considered. For these, the weight of a knows edge is defined as: <math>\max(\text{round}(40 - \sqrt{\text{numInteractions}}), 1)</math></p> <p>If there are multiple pairs of people with cheapest paths that have the same total weight, return all of them.</p> <p><i>Note:</i> Interactions are counted both ways, e.g. if Alice knows Bob, Alice writes 2 reply Comments to Bob's Messages and Bob writes 3 reply Comments to Alice's Messages, their total number of interactions is 5 and the weight of the knows edge is 38.</p> <p><i>Remark:</i> Determinism is ensured by using square root followed by rounding. For all integers between 1 and 100 000, the square root's fractional part is more than 10e-5 from 0.5, where the rounding could be non-deterministic based on floating point inaccuracies. As 10e-5 is significantly larger than the machine epsilon of IEEE 754 floats (both 32- and 64-bit), the floating point inaccuracies have no chance to affect the derived integer edge weights.</p>			
params	<div><div>1</div><div><math>\\$city1Id</math></div><div>ID</div></div> <div><div>2</div><div><math>\\$city2Id</math></div><div>ID</div></div>	<div>(a) Small Cities within the same Country</div> <div>(b) Larger Cities from different Countries</div>		
result	<div><div>1</div><div>person1.id</div><div>ID</div><div>R</div></div> <div><div>2</div><div>person2.id</div><div>ID</div><div>R</div></div> <div><div>3</div><div>totalWeight</div><div>32-bit Integer</div><div>C</div></div>			
sort	<div><div>1</div><div>person1.id</div><div>↑</div></div> <div><div>2</div><div>person2.id</div><div>↑</div></div>			
limit	n/a			
CPs	3.3, 7.6, 7.7, 8.4, 8.6			
relevance	To find the weighted shortest paths efficiently, the system can use e.g. a bidirectional Dijkstra algorithm. As the edge weights do not depend on any parameter, systems can pre-compute them (if they do not interleave reads and writes).			

## BI / read / 20

query	BI / read / 20			
title	Recruitment			
pattern	<div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <p>Compute weighted shortest path between all Persons who work in the Company to Person person2 on knows.weight</p> <pre> graph TD     company[company: Company]     person1[person1: Person]     person2[person2: Person]     company -- "name = \$company" --&gt; person1     person1 -- "workAt" --&gt; company     person1 -- "compute weighted shortest path on knows.weight" --&gt; person2     person2 -- "id = \$person2Id" --&gt; person2         </pre> </div> <div style="width: 48%;"> <p>knows.weight: <math>\min(\text{abs}(\text{studyAtA.classYear} - \text{studyAtB.classYear})) + 1</math></p> <pre> graph LR     personA[personA: Person] -- knows --&gt; personB[personB: Person]     personA -- "studyAtA: studyAt" --&gt; universityA[University]     personB -- "studyAtB: studyAt" --&gt; universityB[University]         </pre> </div> </div> <div style="margin-top: 10px;"> <p>Example for finding a path between person1 and person2</p> <pre> graph LR     p1((p1)) -- knows --&gt; px((pX))     px -- knows --&gt; py((pY))     py -- knows --&gt; dots[...]     dots -- knows --&gt; pw((pW))     pw -- knows --&gt; p2((p2))     p1 -- studyAt --&gt; u1(( ))     px -- studyAt --&gt; u1     px -- studyAt --&gt; u2(( ))     py -- studyAt --&gt; u2     pw -- studyAt --&gt; u3(( ))     p2 -- studyAt --&gt; u3     u1 -- studyAt --&gt; u2     u2 -- studyAt --&gt; u3         </pre> </div>			
description	<p>Consider knows edges where the endpoint Persons attended the same University and set the weight of the edge to the absolute difference between the year of enrolment plus 1. If the Persons attended multiple universities, we select the smallest (<math>\min</math>) value. Formally:</p> $w = \min_{\text{studyAt}_A, \text{studyAt}_B}  \text{studyAt}_A.\text{classYear} - \text{studyAt}_B.\text{classYear}  + 1$ <p>Given a \$company and a Person person2 with ID \$person2Id (who is not working and has not worked at \$company), find a different Person (person1) who works or at some point worked in \$company and is reachable from person2 through people who have studied together through the shortest weighted path.</p> <p>If there are multiple Person person1 nodes with the same shortest path length, return all of them.</p>			
params	1	\$company	Long String	Companies with a similar number of employees (former or current) are selected
	2	\$person2Id	ID	(a) There is guaranteed to be no path between any person1 working at company and person2 (b) There is guaranteed to be a 2-hop path between at least one person1 working at company and person
result	1	person1.id	ID	R
	2	totalWeight	32-bit Integer	C
sort	1	totalWeight	↑	
	2	person1.id	↑	
limit	20			
CPs	3.3, 7.6, 7.7, 7.8, 8.4, 8.6			
relevance	To find the weighted shortest paths efficiently, the system can use e.g. a bidirectional Dijkstra algorithm. As the edge weights do not depend on any parameter, systems can pre-compute them (if they do not interleave reads and writes).			

## 8.5 Insert Operations

Insert operations consist of individual inserts for each entity type. Implementations typically use the same format as the one for loading the initial snapshot of the data set.

## 8.6 Delete Operations

See Section 5.2.

## 9 AUDITING POLICIES

This chapter contains the auditing policies for the LDBC Social Network Benchmark. The initial draft of the auditing policies were published in the EU project deliverable D6.3.3 “LDBC Benchmark Auditing Policies”.

Auditing is defined as “*An examination of the implementation, execution, and results of a test or benchmark, generally performed by an independent third-party, and resulting in a report of findings*” [66]. It is an essential step towards achieving benchmark results in a reliable, reproducible, and independent manner. This chapter is divided in the following parts:

- Motivation of benchmark result auditing
- General discussion of auditable aspects of benchmarks
- Specific checklists and running rules for the Social Network Benchmark’s workloads (Interactive, Business Intelligence)

Many definitions and general considerations are shared between the benchmarks, hence it is justified to present the principles first and to refer to these in the context of the benchmark specific rules.

The auditing process, including the auditor certification exams, the possibility of challenging audited results, etc., are defined in the LDBC Byelaws [44]. Please refer to the latest Byelaws document when conducting audits.

### 9.1 Rationale and General Principles

The purpose of benchmark auditing is to improve the *credibility* and *reproducibility* of benchmark claims by involving a set of detailed execution rules and third party verification of compliance with these.

Rules may exist separately of auditing but auditing is not meaningful unless the rules are adequately precise. Aspects like auditor training and qualification cannot be addressed separately from a discussion of the matters the auditor is supposed to verify. Thus the credibility of the entire process hinges on clear and shared understanding of what a benchmark is expected to demonstrate and on the auditor being capable of understanding the process and of verifying that the benchmark execution is fair and does not abuse the rules or pervert the objectives of the benchmark.

Due to the open-ended nature of technology and the agenda of furthering innovation via measurement, it is not feasible or desirable to over-specify the limits of benchmark implementation. Hence there will always remain judgement calls for borderline cases. In this respect auditing and the LDBC are not separate. It is expected that issues of compliance as well as of maintenance of rules will come before the LDBC as benchmark claims are made.

### 9.2 Auditing Rules Overview

#### 9.2.1 Auditor Training, Certification, and Selection

##### 9.2.1.1 Auditor Training

Auditor training consists of familiarisation with the benchmark and existing implementations thereof. This involves the auditor candidate running the reference implementations of the benchmark in order to see what is normal behaviour and practice in the workload. The training and practice may involve communication with the benchmark task force for clarifying intent and details of the benchmark rules. This produces feedback for the task force for further specification of the rules.

##### 9.2.1.2 Auditor Certification

The auditor certification and qualification is done in the form of an examination administered by the task force responsible for the benchmark being audited. The examination may be carried out by teleconference. The task

force will subsequently vote on accepting each auditor, by simple majority. An auditor is certified for a particular benchmark by the task force maintaining the benchmark in question.

### 9.2.1.3 Auditor Selection

In the default auditor selection, the task force responsible for the benchmark being audited appoints a third party, impartial auditor. The task force may in special cases appoint itself as auditor of a particular result. This is not, however, the preferred course of action but may be done if no suitable third party auditor is available

## 9.2.2 Auditing Process Stages

### 9.2.2.1 Getting Ready for a Benchmark Audit

A benchmark result can be audited if it is a *complete implementation* of an LDBC benchmark workload. This includes implementing all operations (reads and updates) correctly, using official data sets, using the official LDBC driver (if available), and complying with the auditing rules of the workload (e.g. workloads may have different rules regarding query languages, the allowance of materialized views, etc.). Workloads may specify further requirements such as ACID-compliance (checked using the LDBC ACID test suite).

### 9.2.2.2 Performing a Benchmark Audit

A benchmark result is to be audited by an LDBC appointed auditor or the LDBC task force managing the benchmark. An LDBC audit may be performed by remote login and does not require the auditor's physical presence on site. The test sponsor shall grant the auditor any access necessary for validating the benchmark run. This will typically include administrator access to the SUT hardware.

### 9.2.2.3 Benchmark-Specific Checklist

Each benchmark specifies a checklist to be verified by the auditor. The benchmark run shall be performed by the auditor. The auditor shall take copies of relevant configuration files and test results for future checking and insertion into the full disclosure report.

### 9.2.2.4 Producing the FDR

The FDR is produced by the auditor or auditors, with any required input from the test sponsor. Each non-default configuration parameter needs to be included in the FDR and justification needs to be provided why the given parameter was changed. The auditor produces an attestation letter that verifies authenticity of the presented results. This letter is to be included into the FDR as an addendum. The attestation letter has no specific format requirements but shall state that the auditor has established compliance with a specified version of the benchmark specification.

### 9.2.2.5 Publishing the FDR

The FDR and any benchmark specific summaries thereof shall be published on the LDBC website, <https://ldbcouncil.org/>.

## 9.2.3 Challenge Procedure

A benchmark result may be *challenged* for non-compliance with LDBC rules. The benchmark task force responsible for maintenance of the benchmark will rule on matters of compliance. A result found to be non-compliant will be withdrawn from the list of official LDBC benchmark results.



## 9.3 Auditable Properties of Systems and Benchmark Implementations

### 9.3.1 Validation of Query Results

A benchmark should be published with a deterministically reproducible validation data set. Validation queries applied to the validation data set will deterministically produce a set of correct answers. This is used in the first stage of benchmark run to test for the correctness of an SUT or benchmark implementation. This validation stage is not timed.

**Inputs for validation** The validation takes the form of a set of data generator parameters, a set of test queries that at least include one instance of each of the workload query templates and the expected results.

**Approximate results and error margin** In certain cases the results may be approximate. This may happen in cases of non-unique result ordering keys, imprecise numeric data types, random behaviours in certain graph analytics algorithms etc. Therefore, a validation set shall specify the degree of allowable error: For example, for counts, the value must be exact, for sums, averages and the like, at least 8 significant digits are needed, for statistical measures like graph centralities, the result must be within 1% of the reference result. Each benchmark shall specify its expectation in an unambiguously verifiable manner.

### 9.3.2 ACID Compliance

As part of the auditing process for the Interactive workload and for certain systems in the BI workload, the auditors ascertain that the SUT satisfies the ACID properties, i.e. it provides atomic transactions, complies with its claimed isolation level, and ensures durability in case of failures. This section outlines transactional behaviours of SUTs which are checked in the course of auditing an SUT in a given benchmark.

A benchmark specifies transactional semantics that may be required for different parts of the workload. The requirements will typically be different for initial bulk load of data and for the workload itself. Different sections of the workload may further be subject to different transactionality requirements.

No finite series of tests can prove that the ACID properties are fully supported. Passing the specified tests is a necessary, but not sufficient, condition for meeting the ACID requirements. However, for fairness of reporting, only the tests specified here are required and must appear in the FDR for a benchmark. (This is taken exactly from the TPC-C specification [83].)

The properties for ACID compliance are defined as follows:

**Atomicity** Either all of the effects of the transaction are in effect after the transaction or none of the effects is in effect. This is by definition only verifiable after a transaction has finished.

**Consistency** ADS such as secondary indices will be consistent among themselves as well as with the table or other PDS, if any. Such a consistency (compliance to all constraints, if these are declared in the schema, e.g. primary key constraint, foreign key constraints and cardinality constraints) may be verified after the commit or rollback of a transaction. If a single thread of control runs within a transaction, then subsequent operations are expected to see consistent state across all data indices pertaining to a table or similar object. Multiple threads which may share a transaction context are not required to observe a consistent state at all times during the execution of the transaction. Consistency will however always be verifiable after the commit or rollback of any transaction, regardless of the number of threads that have either implicitly or explicitly participated in the transaction. Any intra-transaction parallelism introduced by the SUT will preserve transactional semantics statement-by-statement. If explicit, application created sessions share a transaction context, then this definition of consistency does not hold: for example, if two threads insert into the same table at the same time in the same transaction context, these may or may not see a consistent image of (E)ADS for the parts affected by the other thread. All things will be consistent after the commit or rollback, however, regardless of the number of threads, implicit or explicit that have participated in the transaction.

**Isolation** Isolation is defined as the set of phenomena that may (or may not) be observed by operations running within a single transaction context. The levels of isolation are defined as follows:

**Read uncommitted** No guarantees apply.

**Read committed** A transaction will never read a value that has at no point in time been part of a committed state.

**Repeatable read** If a transaction reads a value several times during its execution, then it will see the original state with its own modifications so far applied to it. If the transaction itself consists of multiple reading and updating threads then the ambiguities that may arise are beyond the scope of transaction isolation.

**Serializable** The transactions see values that correspond to a fully serial execution of all client transactions. This is like repeatable read except that if the transaction reads something, and repeats the read, it is guaranteed that no new values will appear for the same search condition on a subsequent read in the same transaction context. For example, a row that was seen not to exist when first checked will not be seen by a subsequent read. Likewise, counts of items will not be seen to change.

**Durability** Durability means that once the SUT has confirmed a successful commit, the committed state will survive any instantaneous failure of the SUT (e.g. a power failure, software crash, reboot or the like). Durability is tied to atomicity in that if one part of the changes made by a transaction survives then all parts must survive.

### 9.3.3 Data Schema

A benchmark may specify restrictions on schema. For example, TPC-H and TPC-DS specify that only certain indices may be declared. In the LDBC context, the matter is more complex since the range of possible SUTs is much broader, including diverse combinations of schema first and schema-less systems and configurations.

#### 9.3.3.1 Schema Declaration

By default, a system may declare no schema at all, as may be the case with RDF or graph DBMSs. If EADSs are declared, then these must be consistently applied to all data within the same workload for a given scale factor. The nature of prohibited EADSs, if any, depends on the benchmark and may be stated in the benchmark specification.

#### 9.3.3.2 Schema-Optional

RDF and graph databases may sometimes be adopted due to their support for schema-last or schema-less operation. It is known that for many cases of RDF with a regular structure, a 1:1 mapping to a relational schema may exist. A benchmark may prohibit the use of such a mapping with the rationale that if the data were purely relational in structure then there would be no point in using RDF or graph DB in the first place. The example of such mapping is Sparqlify (or D2RQ), where SPARQL is directly translated to SQL and run against a relational database.

**Use of EADS in a schema-less data model** A benchmark may allow use of EADS with a schema-less data model such as RDF with the condition that whilst some data structures may become more efficient, no data structure is prohibited. The schema-less nature may persist but some common structures may benefit from more efficient physical representation.

**Benchmarks enforcing schema-first semantics** A benchmark may also state that it allows strict schema-first semantics, e.g. SQL, and that the SUT need not make any specific provisions for schema change during the run. For an RDF system this would mean a priori imposing compliance with a data shape or ontology, not with OWL semantics but with semantics close to those of SQL DDL. In such a case, the ontology or data shape may as such be construed to be a valid hint for creation of application specific EADS.

**Disclosure of data schema in the FDR** In any case, a benchmark must state its policy concerning presence or absence of schema and enforcement thereof. If implementations declare a schema then any schema must be disclosed in full as part of the FDR.

### 9.3.4 Data Format and Preprocessing

When producing the data sets, implementers are allowed to use custom formatting options (e.g. use or omission of quotes, separator character, datetime format, etc.). It is also allowed to convert the output of the Datagen into a format (e.g. Parquet) that is loadable by the test-specific implementation of the data importer. Additional preprocessing steps are also allowed, including adjustments to the CSV files (e.g. with shell scripts), splitting and concatenating files, compressing and decompressing files, etc. However, the preprocessing step shall not include a precomputation of (partial) query results.

### 9.3.5 Data Access Transparency

A benchmark may specify that an implementation is not allowed the use of explicit access paths. For example, explicitly specifying which EADS or IADS should be used for any given operation may be prohibited. Furthermore, in scale-out systems, explicit references to data location (other than via values of partitioning keys) may be prohibited. In general, references to internal data representation of an entity, e.g. row in a table, should be prohibited. Reference should take place via column names in a schema or property URIs in RDF, not via physical offsets or the like.

### 9.3.6 Query Languages

In typical RDBMS benchmarks, online transaction processing (OLTP) benchmarks are allowed to be implemented via stored procedures, effectively amounting to explicit query plans. Meanwhile, online analytical processing (OLAP) benchmarks prohibit the use of using general-purpose programming languages (e.g. C, C++, Java) for query implementations and only allow domain-specific query languages.

In the graph processing space, there is currently (as of 2022) no standard query language and the systems are considerably more heterogeneous. Therefore, the LDBC situation regarding declarativity is not as simple as that of for example the TPC-H (where queries should be specified in SQL with the additional constraint of omitting any hints for OLAP workloads) and individual SNB workloads specify their policy of either requiring a domain-specific query language or allowing the implementation of the queries in a general-purpose programming language.

In the case of domain-specific languages, systems are allowed to implement an SNB query as a sequence of multiple queries. A typical example of this is the following sequence: (1) create projected graph, (2) run query, (3) drop projected graph. However, it is not allowed to use subqueries in an unrealistic and contrived manner, i.e. the goal of overcoming optimization issue, e.g. hard-coding a certain join order in a declarative query language. It is the responsibility of the auditor to determine whether a sequence of queries can be considered realistic w.r.t. how a user would formulate their queries in the language provided by the system.

#### 9.3.6.1 Rules for Imperative Implementations Using a General-Purpose Programming Language

An implementation where the queries are written in a general-purpose programming language (including imperative and “API-based” implementations) may choose between semantically equivalent implementations of an operation based on the query parameters. This simulates the behaviour of a query optimizer in the presence of literal values in the query. If an implementation does this, all the code must be disclosed as part of the FDR and the decision must be based on values extracted from the database, not on hard-coded threshold values in the implementation.

The auditor must be able to reliably assess the compliance of an implementation to guidelines specifying these matters. The actual specification remains benchmark-dependent. Borderline cases may be brought to the task force responsible for arbitration.

### 9.3.6.2 Disclosure of Query Implementations in the FDR

Benchmarks allowing imperative expression of workload should require full disclosure of all query implementation code.

### 9.3.7 Materialization

The mix of read and update operations in a workload will determine to which degree precomputation of results is beneficial. The auditor must check that materialised results are kept consistent at the end of each transaction.

### 9.3.8 Steady State

An online workload must be able to indefinitely keep up the reported throughput. The benchmark definition may put specific restrictions on the duration of individual parts of the workload.

#### 9.3.8.1 Bringing the SUT into Steady State

One implication of this is that an SUT must be able to accommodate inserts at a specific rate for a realistic length of time. For example, if the workload is of an online nature then the SUT should be sized so as not to run out of space for new data for a reasonable duration of time. The TPC-C 180-day rule is an example of this. An analytical benchmark that primarily bulk loads data does not need to reserve as much space for new data. Each benchmark shall state its specific requirements in this respect.

### 9.3.9 Query Mix

A benchmark consists of multiple different operations that may vary in frequency and duration of individual instances of each operation may vary in function of parameter selection. A benchmark must specify an operation mix and a minimum count of operations that constitutes a compliant benchmark execution.

The auditor must ascertain from the records of a benchmark execution that a sufficient number of operations has indeed taken place for the report. For example, a 1000 GB TPC-H must have at least 7 streams in the throughput test and the workload is to be run twice following bulk load. For LDBC SNB, the run must be at least 2 hours of wall clock, measured time and the count of successful transactions of each type must be in a strictly set ratio with the count of other operations.

Benchmarks shall each specify a minimum count of operations and relative frequencies of operations for a qualifying execution.

#### 9.3.9.1 Post-Processing of Query Results and Compression During Transmission

All computing required for a given query needs to happen in the DBMS. The SUT's test driver shall not post-process query results in a way that changes their value. For example, it is not allowed to return floating-point values with a precision of 0.5 that are encoded as integers and divided by 2 on the client side.

Note that *lossless compression* during the communication between the test driver and the DBMS is allowed. For instance, as long as the DBMS uses a data type that conforms with the schema requirements for a given attribute, one can apply compression to send it back to/from the driver and decompress it. For example, for complex query Q14 in the Interactive v1 workload, the implementation should ultimately produce a floating point score.

The same applies for query parameters. At both the client's and the server's endpoint, the correct fully qualified datatype must occur, but during transmission, it is allowed to apply compression.

### 9.3.10 System Configuration and System Pricing

A benchmark execution shall produce a full disclosure report which specifies the hardware and software of the SUT, the benchmark implementation version and any specifics that are detailed in the benchmark specification. This clause gives a general minimum for disclosure for the SUT.

#### 9.3.10.1 Details of Machines Driving and Running the Workload

An SUT may consist of one or more pieces of physical hardware. An SUT may include virtual or bare-metal machines in a cloud service. For each distinct configuration, the FDR shall disclose the number of units of the type as well as the following:

1. The used cloud provider (including the region where machines reside, if applicable).
2. Common name of the item, e.g. Dell PowerEdge xxxx or i3.2xlarge instance.
3. Type and number of CPUs, cores & threads per CPU, clock frequency, cache size.
4. Amount of memory, type of memory and memory frequency, e.g. 64GB DDR3 1333MHz.
5. Disk controller or motherboard type if disk controller is on the motherboard.
6. For each distinct type of secondary storage device, the number and specification of the device, e.g. 4xSeagate Constellation 2TB SATA 6Gbit/s.
7. Number and type of network controllers, e.g. 1x Mellanox QDR InfiniBand HCA, PCIE 2.0, 2x1GbE on motherboard. If the benchmark execution is entirely contained on a single machine, it must be stated, and the description of network controllers can be omitted.
8. Number and type of network switches. If multiple switches are used, the wiring between the switches should be disclosed. Only the network switches and interfaces that participate in the run need to be reported. If the benchmark execution is entirely contained on a single machine, it must be stated, and the description of network switches can be omitted.
9. Date of availability of the system as a whole, i.e. the latest date of availability of any part.

#### 9.3.10.2 System Pricing

The price of the hardware in question must be disclosed. For cloud setups, the price of a dedicated instance for 3 years must be disclosed. The price should reflect the single quantity list price that any buyer could expect when purchasing one system with the given specification. The price may be either an item by item price or a package price if the system is sold as a package. Reported prices should adhere the TPC Pricing Specification 2.9.0 [40, 85]. It is particularly important to ensure that the maintenance contract guarantees 24/7 support and 4 hour response time for problem recognition. If the benchmark driver is running on a separate machine, the price of this machine should not be included in the total system price.

#### 9.3.10.3 Details of Software Components in the System

The SUT software must be described at least as follows:

1. The units of the SUT software are typically the DBMS and operating system.
2. Name and version of each separately priced piece of the SUT software.
3. If the price of the SUT software is tied to platform or count of concurrent users, these parameters must be disclosed.
4. Price of the SUT software.
5. Date of availability.

Reported prices should adhere the TPC Pricing Specification 2.5.0 [40, 85].

The configuration of the SUT must be reported so as to include the following:

1. The used LDBC specification, driver and data generator version.
2. Complete configuration files of the DBMS, including any general server configuration files, any configuration scripts run on the DBMS for setting up the benchmark run etc.

3. Complete schema of the DBMS, including eventual specification of storage layout.
4. Any OS configuration parameters if other than default, e.g. `vm.swappiness`, `vm.max_map_count` in Linux.
5. Complete source code of any server-side logic, e.g. stored procedures, triggers.
6. Complete source code of driver-side benchmark implementation.
7. Description of the benchmark environment, including software versions, OS kernel version, DBMS version as well as versions of other major software components used for running the benchmark (Docker, Java Virtual Machine, Python, etc.).
8. The SUT's highest configurable isolation level and the isolation level used for running the benchmark.

#### 9.3.10.4 Audit of System Configuration

The auditor must ascertain that a reported run has indeed taken place on the SUT in the disclosed configuration. The full disclosure shall contain any relevant parameters of the benchmark execution itself, including:

1. Parameters, switches, configuration file for data generation.
2. Complete text of any data loading script or program.
3. Parameters, switches, configuration files for any test driver. If the test driver is not an LDBC supplied open source package or is a modification of such, then the complete text or diff against a specific LDBC package must be disclosed.
4. Test driver output files shall be part of the disclosure. In general, these must at least detail the following:
  - i) Time and duration of data load and the timed portion of the benchmark execution.
  - ii) Count of each workload item (e.g. query, transaction) successfully executed within the measurement window.
  - iii) Min/average/max execution time of each workload item, the specific benchmark shall specify additional details.

Given this information, the number of concurrent database sessions at each point in the execution must be clearly stated. In the case of a cluster database, the possible spreading of connections across multiple server processes must be disclosed.

All parameters included in this section must be reported in the full disclosure report to guarantee that the benchmark run can be reproduced exactly in the future. Similarly, the test sponsor will inform the auditor the scale factor to test. Finally, a clean test system with enough space to store the initial data set, the update streams, substitution parameters and anything that is part of the input and output as well as the benchmark run must be provided.

#### 9.3.11 Benchmark Specifics

Similarly to TPC benchmarks, the LDBC benchmarks prohibit so-called benchmark specials (i.e. extra software modules implemented in the core DBMS logic just to make a selected benchmark run faster are disallowed). Furthermore, upon request of the auditor, the test sponsor must provide all the source code relevant to the benchmark.

## 9.4 Auditing Rules for the Interactive Workload

This section specifies a checklist (in the form of individual sections) that a benchmark audit shall cover in case of the SNB Interactive workload. An overview of the benchmark audit workflow is shown in Figure 9.1. The three major phases of the audit are preparing the input data and validation query results (captured by *Preparations* in the figure), validating the correctness of query results returned by the SUT using the validation scale factor and running the benchmark with all the prescribed workloads (*Benchmarking*), and creating the FDR (*Finalization*). The colour codes capture the responsibilities of performing a step or providing some data in the workflow.

A key objective of the auditing guidelines for the Interactive workload is to *allow a broad range of systems* to implement the benchmark. Therefore, they do not impose constraints on the data model (graph, relational,

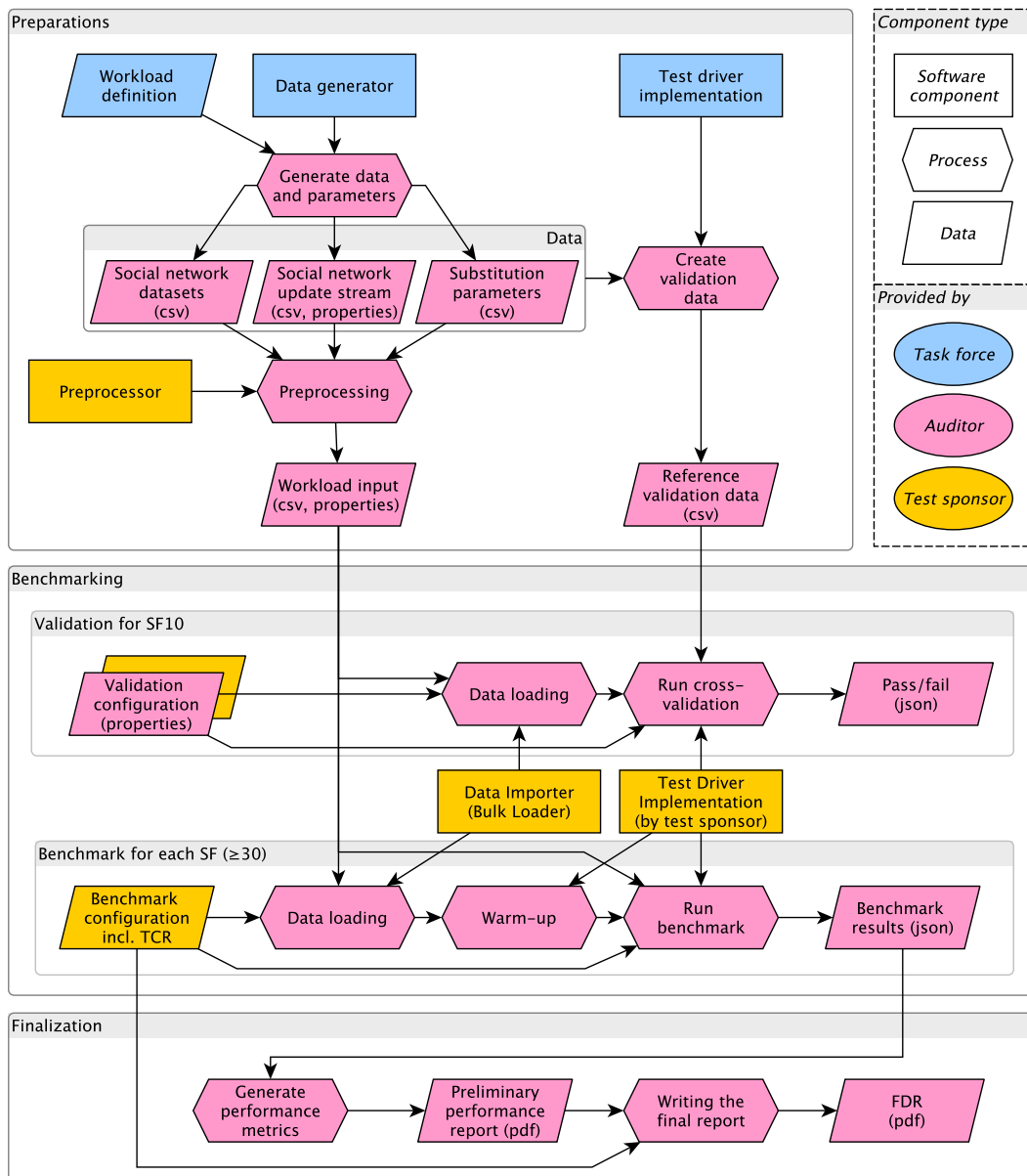


Figure 9.1: Benchmark execution and auditing workflow. For non-audited runs, the implementers perform the steps of the auditor.

triple, etc. representations are allowed) or on the query language (both declarative and imperative languages are allowed).

## 9.4.1 Scaling

### 9.4.1.1 Scale Factors

The scale factor of an SNB data set is the size of the data set in GiB of CSV (comma-separated values) files. The size of a data set is characterized by scale factors: SF10, SF30, SF100 etc. (see Section 3.4.1). All data sets contain data for three years of social network activity.

The *validation run* shall be performed on the SF10 data set (see Section 9.4.6.1) and use at least 100 000 operations. Note that the auditor may perform additional validation runs of the benchmark implementation using

smaller data sets (e.g. SF1) and issue queries.<sup>1</sup>

Audited *benchmark runs* of the Interactive workload shall use SF30 or larger data sets. The rationale behind this decision is to ensure that there is a sufficient number of update operations available to guarantee 2.5 hours of continuous execution (see Section 9.4.7.2).

#### 9.4.1.2 Social Network data sets

**Initial data set** The data set is divided into a bulk loadable initial database population (90%) and an update stream (10%). These are generated by the SNB data generator. The data generator has options for splitting the data set into any number of files.

**Dependencies between messages in the update stream** The update stream contains the latest 10% of the events in the simulated social network. These events form a single serializable sequence in time. Some events will depend on preceding events, for example a message must exist before a reply comment to the message is created. The data generator guarantees that these are separated by at least 10 seconds of simulation time.

**Parallel updates** The update stream may be broken into arbitrarily many sub-streams. The partition scheme is created by the Datagen. During benchmark execution, the driver preserves dependencies between update operations, such as ensuring not to refer to non-existent entities in updates (e.g. a like is not added to a message which has not been inserted yet).

### 9.4.2 Data Model and Data Loading

#### 9.4.2.1 Supported Data Models

SNB may be implemented with different data models (e.g. relational, RDF, and different graph data models). The reference schema is provided in the specification using a UML-like notation.

#### 9.4.2.2 Generated Input Data

**Storage** The data generator produces comma-separated values (CSV) for all data models.

**Data format** A single attribute has a single data type, as follows:

**Identifier** This is an integer value foreign key or a URI in RDF. If this is an integer column, the implementation data type should support at least  $2^{50}$  distinct values.

**Date** A date should support a date range from 0000 to 9999 in the year field.

**DateTime** A datetime should support a date range from 0000 to 9999 in the year field, with at least millisecond precision.

**Short string** The string column for names may have a variable length and may have a declared maximum length, e.g. 40 characters.

**Long string** For example a message content may be a long string that is often short in the data but may not declare a maximum length and must support data sizes of up to 1 MB.

The above is stated in further detail in the benchmark specification, and it shall take precedence over the above in the case of conflict.

A single attribute in the reference schema may not be divided into multiple attributes in the target schema.

---

<sup>1</sup> An example test could be to issue complex reads with parameters such as `personId` and `messageId` selected from the `Person/Message` entities inserted from the update streams and cross-validate these against other systems. (The substitution parameters are taken from the initial snapshot of the graph so these nodes are not targeted by the regular workload executed by the driver.)



**Database schema** A schema on the DBMS is optional. An RDF implementation for example may work without one. An RDF implementation is allowed to load the RDF reference schema and to take advantage of the data type and cardinality statements therein.

**Configuration parameters** Datagen configuration parameters, including SF, distributions, number of persons, serialiser (e.g. CsvSingularMergedFK) should be reported.

**Primary data structures** An RDF, relational, or graph schema may specify system specific options affecting DBMS storage layout. These may for example specify vertical partitioning. Vertical partitioning means anything from a column store layout with per-column allocated storage space to use of explicit column groups. Any mix of row or column-wise storage structures is allowed as long as this is declaratively specified on a per data structure-basis.

**Auxiliary data structures** Covering indices and clustered indices are allowed. If these are defined, then all replications of data implied by these must be maintained statement by statement, i.e. each auxiliary data structure must be consistent with any other data structures of the table after each data manipulation operation.

A covering index is an index which materialises a specific order of a specific subset or possibly all columns of a table. A clustered index is an index which materialises all columns of a table in a specific order, which order may or may not be that of the primary key of the table. A clustered or covering index may be the primary or only representation of a table.

Any subset of the columns on a covering or clustered index may be used for ordering the data. A hash based index or a combination of a hash based and tree based index are all allowed, in row or column-wise or hybrid forms.

**Loading the data** We expect the SUT to provide some means to bulk load the data set either in the form of a dedicated offline loader component or an online loader that allows bulk inserting into a database. The total of the bulk load time and the time for subsequent operations (indexing, computing statistics, etc.) must be reported in the FDR (see Section 9.4.7). As loading can be an expensive operation, it is allowed to conduct the audit such that the loading is only performed once, and the validation/benchmarking phases use the resulting database instance. In practice, this can look like as follows: (1) load the data, (2) compute statistics, uniqueness constraints, keys, indices, etc., (3) shut down the SUT, (4) create a backup of the database (e.g. by copying the directory of the database). For all subsequent runs, the database shall be restored from the backup.

### 9.4.3 Precomputation

Precomputation of query results (both interim and end results) is allowed. However, systems must ensure that precomputed results (e.g. materialized views) are kept consistent upon updates.

### 9.4.4 Benchmark Software Components

LDBC provides a test driver, data generator, and summary reporting scripts. Benchmark implementations shall use a stable version (e.g. 0.3.6) of the test driver. The SUT's database software should be a stable version that is available publicly or can be purchased at the time of the release of the audit.

#### 9.4.4.1 Adaptation of the Test Driver to a DBMS

A qualifying run must use a test driver that adapts the provided test driver to interface with the SUT. Such an implementation, if needed, must be provided by the test sponsor. The parameter generation, result recording, and workload scheduling parts of the test driver should not be changed. The auditor must be given access to the test driver source code used in the reported run.

The test driver produces the following artefacts for each execution as a by product of the run: Start and end timestamps in wall clock time, recorded with microsecond precision. The identifier of the operation and any substitution parameters.

#### 9.4.4.2 Summary of Benchmark Results

A separate test summary tool provided with the test driver analyses the test driver log(s) after a measurement window is completed.

The tool produces for each of the distinct queries and transactions the following summary:

- Run time of query in wall clock time.
- Count of executions.
- Minimum/mean/percentiles/maximum execution time.
- Standard deviation from the average execution time.

The tool produces for the complete run the following summary:

- Operations per second for a given SF (throughput). This is the primary metric of this workload.
- The total execution time in wall clock time.
- The total number of completed operations.

### 9.4.5 Implementation Language and Data Access Transparency

The queries and updates may be implemented in a domain-specific query language or as procedural code written in a general-purpose programming language (e.g. using the API of the database).

#### 9.4.5.1 Implementations Using a Domain-Specific Query Language

If a domain-specific query language is used, e.g. GQL, SPARQL, SQL, SQL/PGQ, Cypher, or Gremlin, then explicit query plans are prohibited in all the read-only queries.<sup>2</sup> The update transactions may still consist of multiple statements, effectively amounting to explicit plans.

Explicit query plans include but are not limited to:

- Directives or hints specifying a join order or join type
- Directives or hints specifying an access path, e.g. which index to use
- Directives or hints specifying an expected cardinality, selectivity, fanout or any other information that pertains to the expected number of results or cost of all or part of the query.

*Rationale behind the applied restrictions.* The updates are effectively OLTP and, therefore, the customary freedoms apply, including the use of stored procedures, however subject to access transparency. Declarative queries in a benchmark implementation should be such that they could plausibly be written by an application developer. Therefore, their formulation should not contain system specific aspects that an application developer would be unlikely to know. In other words, making a benchmark implementation should not require uncommon sophistication on behalf of the developer. This is regular practice in analytical benchmarks, e.g. TPC-H.

#### 9.4.5.2 Implementations Using a General-Purpose Programming Language

Implementations using a general-purpose programming language for specifying the queries (including procedural, imperative, and API-based implementations) are expected to respect the rules described in Section 9.3.6. For these implementations, the rules in Section 9.4.5.1 do not apply.

<sup>2</sup>If the queries are not clearly declarative, the auditor must ensure that they do not specify explicit query plans by investigating their source code and experimenting with the query planner of the system (e.g. using SQL's EXPLAIN command).

## 9.4.6 Correctness of Benchmark Implementation

### 9.4.6.1 Validation data set

The scale factor 10 shall be used as validation data set.

### 9.4.6.2 ACID Compliance

The Interactive workload requires full ACID support (Section 9.3.2) from the SUT. This is tested using the LDBC ACID test suite. For the specification of this test suite, see Chapter 10 and the related software repository at [https://github.com/ldbc/ldbc\\_acid](https://github.com/ldbc/ldbc_acid).

**Expected level of isolation** If a transaction reads the database with intent to update, the DBMS must guarantee that repeating the same read within the same transaction will return the same data. This also means that no more and no less data rows must be returned. In other words, this corresponds to snapshot or to serializable isolation. If the database is accessed without transaction context or without intent to update, then the DBMS should provide read committed semantics, e.g. repeating the same read may produce different results but these results may never include effects of pending uncommitted transactions.

**Durability and checkpoints** A checkpoint is defined as the operation which causes data persisted in a transaction log to become durable outside of the transaction log. Specifically, this means that an SUT restart after instantaneous failure following the completion of the checkpoint may not have recourse to transaction log entries written before the end of the checkpoint.

A checkpoint typically involves a synchronisation barrier at which all data committed prior too the moment is required to be in durable storage that does not depend on the transaction log. Not all DBMSs use a checkpointing mechanism for durability. For example a system may rely on redundant storage of data for durability guarantees against instantaneous failure of a single server.

The measurement window may contain a checkpoint. If the measurement window does not contain one, then the restart test will involve redoing all the updates in the window as part of the recovery test.

The timed window ends with an instantaneous failure of the SUT. Instantaneously killing all the SUT process(es) is adequate for simulating instantaneous failure. All these processes should be killed within one second of each other with an operating system action equivalent to the Unix `kill -9`. If such is not available, then powering down each separate SUT component that has an independent power supply is also possible.

The restart test consists of restarting the SUT process(es) and finishes when the SUT is back online with all its functionality and the last successful update logged by the driver can be seen to be in effect in the database.

If the SUT hardware was powered down, the recovery period does not include the reboot and possible file system check time. The recovery time starts when the DBMS software is restarted.

**Recovery** The SUT is to be restarted after the measurement window and the auditor will verify that the SUT contains the entirety of the last update recorded by the test driver(s) as successfully committed. The driver or the implementation have to make this information available. The auditor may also check the *audit log* of the SUT (if available) to confirm that the operations issued by the driver were saved.

Once an official run has been validated, the recovery capabilities of the system must be tested. The system and the driver must be configured in the same way as in during the benchmark execution. After a warm-up period, an execution of the benchmark will be performed under the same terms as in the previous measured run.

**Measuring recovery time** At an arbitrary point close to 2 hours of wall clock time during the run, the machine will be shut down. Then, the auditor will restart the database system and will check that the last committed update (in the driver log file) is actually in the database. The auditor will measure the time taken by the system to recover from the failure. Also, all the information about how durability is ensured must be disclosed. If checkpoints are used, these must be performed with a period of 10 minutes at most.

### 9.4.7 Benchmarking Workflow

A benchmark execution is divided into the following processes (these processes are also shown in Figure 9.1):

**Generate data** This includes running the data generator, placing the generated files in a staging area, configuring storage, setting up the SUT configuration and preparing any data partitions in the SUT. This may include preallocating database space but may not include loading any data or defining any schema having to do with the benchmark. The `ldbc.snb.interactive.update_interleave` driver parameter must come from the `updateStream.properties` file, which is created by the data generator. That parameter should never be set manually. This parameter signifies the average distance of update operations in the workload.

**Preprocessing** If needed, the output from the data generator is to preprocess the data set (Section 9.3.4).

**Create validation data** Using one of the reference implementations of the benchmark, the reference validation data is obtained in `.json` format.

**Data loading** The test sponsor must provide all the necessary documentation and scripts to load the data set into the database to test. This includes defining the database schema, if any, loading the initial database population, making this durably stored and gathering any optimiser statistics. The system under test must support the different data types needed by the benchmark for each of the attributes at their specified precision. No data can be filtered out, everything must be loaded. The test sponsor must provide a tool to perform arbitrary checks of the data or a shell to issue queries in a declarative language if the system supports it.

**Run cross-validation** This step uses the data loader to populate the database, but the load is not timed. The validation data set is used to verify the correctness of the SUT. The auditor must load the provided data set and run the driver in validation mode, which will test that the queries provide the official results. The benchmarking workflow will not go beyond this point unless results match the expected output.

**Warm-up** Benchmark runs are preceded by a warm-up which must be performed using the LDBC driver.

**Run benchmark** The bulk load time is reported and is equal to the amount of elapsed wall clock time between starting the schema definition and receiving the confirmation message of the end of statistics gathering. The workflow runs begin after the bulk load is completed. If the run does not directly follow the bulk load, it must start at a point in the update stream that has not previously been played into the database. In other words, a run may only include update events whose timestamp is later than the latest message creation date in the database prior to start of run. The run starts when the first of the test drivers send its first message to the SUT. If the SUT is running in the same process as the driver, the window starts when the driver starts. Also, make sure that the `-r1/--results_log` is enabled. Make sure that all operations are enabled and the frequencies are those for the selected scale factor (see the exact specification of the frequencies in Appendix B).

#### 9.4.7.1 Query Timing During Benchmark Run

A valid benchmark run must last at least 2 hours of wall clock time and at most 2 hours and 15 minutes. In order to be valid, a benchmark run needs to meet the “95% on-time requirement”. The `results_log.csv` file contains the `actual_start_time` and the `scheduled_start_time` of each of the issued queries. In order to have a valid run, 95% of the queries must meet the following condition:

$$\text{actual\_start\_time} - \text{scheduled\_start\_time} < 1 \text{ second}$$

If the execution of the benchmark is valid, the auditor must retrieve all the files from directory specified by `--results_dir` which includes configuration settings used, results log and results summary. All of which must be disclosed.

#### 9.4.7.2 Measurement Window

Benchmark runs execute the workload on the SUT in two phases (Figure 9.2). First, the SUT must undergo a warm-up period that takes at least 30 minutes and at most 35 minutes. The goal of this is to put the system in

a steady state which reflects how it would behave in a normal operating environment. The performance of the operations during warm-up is not considered. Next, the SUT is benchmarked during a two-hour measurement window. Operation times are recorded and checked to ensure the “95% on-time requirement” is satisfied.

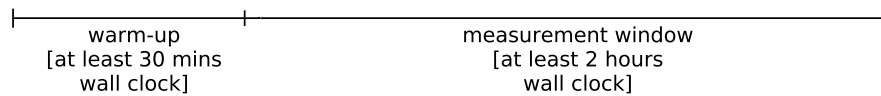


Figure 9.2: Warm-up and measurement window for benchmark run.

The SNB Datagen produces 3 years worth data of which 10% is used for updates (Section 9.4.1.2), i.e. approximately  $3 \times 365 \times 0.1 = 109.5$  days = 2628 hours. To ensure that the 2.5 hour wall clock period has enough input data, the lower bound of TCR is defined as 0.001 (if 2628 hours of updates are played back at more than  $1000\times$  speed, the benchmark framework runs out of updates to execute). System that can achieve a better compression (i.e. lower TCR value) on a given scale factor should use larger SFs for their benchmark runs – otherwise their total runs will be less than 2.5 hours, making them unsuitable for auditing.

### 9.4.8 Full Disclosure Report

Upon successful completion of the audit, an FDR is compiled. In addition to the general requirements, the full disclosure shall cover the following:

- General terms: an executive summary and declaration of the credibility of the audit
- System description and pricing summary: see Section 9.3.10
- Data generation and data loading: see Section 9.4.2.2
- Test driver details: see Section 9.4.4.1
- Performance metrics: see Section 9.4.4.2
- Validation results: see Section 9.4.6.1
- ACID compliance: see Section 9.3.2
- List of supplementary materials

To ensure reproducibility of the audited results, a supplementary package is attached to the full disclosure report. This package should contain:

- A README file with instructions specifying how to set up the system and run the benchmark
- Configuration files of the database, including database-level configuration such as buffer size and schema descriptors (if necessary)
- Source code or binary of a generic driver that can be used to interact with the DBMS
- SUT-specific LDBC driver implementation (similarly to the projects in [https://github.com/ldbc/ldbc\\_snb\\_interactive\\_v1\\_impls](https://github.com/ldbc/ldbc_snb_interactive_v1_impls), [https://github.com/ldbc/ldbc\\_snb\\_interactive\\_v2\\_impls](https://github.com/ldbc/ldbc_snb_interactive_v2_impls), [https://github.com/ldbc/ldbc\\_snb\\_bi](https://github.com/ldbc/ldbc_snb_bi))
- Script or instructions to compile the LDBC Java driver implementation
- Instructions on how to reach the server through CLI and/or web UI (if applicable), e.g. the URL (including port number), user name and password
- LDBC configuration files (.properties), including the `time_compression_ratio` values used in the audited runs
- Scripts to preprocess the input files (if necessary) and to load the data sets into the database
- Scripts to create validation data sets and to run the benchmark
- The implementations of the queries and the update operations, including their complete source code (e.g. declarative queries specifications, stored procedures, etc.)
- Implementation of the ACID test suite
- Binary package of the DBMS (e.g. .deb or .rpm)

## 9.5 Auditing Rules for the Business Intelligence Workload

The following section describes the auditing rules specific to the Business Intelligence (BI) workload.

### 9.5.1 Overview

Implementing the BI workload requires the following key capabilities:

- Loading the initial snapshot of the social network graph
- Evaluating the BI read queries (Section 8.4)
- Evaluating the BI write operations: inserts (Section 8.5) and deletes (Section 8.6)
- Performing concurrent reads and writes (Section 9.5.2) (optional, only allowed if ACID compliance is guaranteed)

### 9.5.2 Workflow

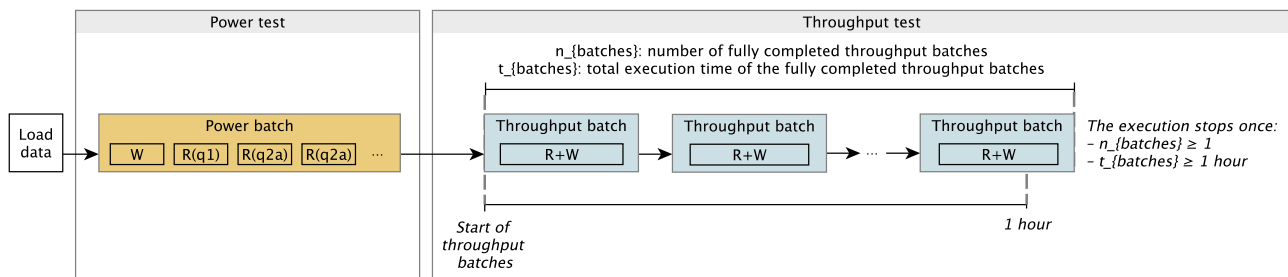


Figure 9.3: Tests and batches (power and throughput) executed in the BI workload's workflow.

The write operations and read queries are run in *daily batches*. In each batch, each query variant (Q1, Q2a, Q2b, Q3, ..., Q20a, Q20b) is executed using 30 different substitution parameters. The BI workflow (Figure 9.3) consists of two key parts: the *power test* (Section 9.5.2.1) and the *throughput test* (Section 9.5.2.2).

#### 9.5.2.1 Power Test

The *power test* runs a single *power batch*. This test first runs the write operations, followed by a sequential execution of individual read query variants. The writes perform a day of inserts and deletes in the simulated social network, while a total of  $28 \times 30 = 840$  read queries are executed.

#### 9.5.2.2 Throughput Test

The *throughput test* consists of multiple *throughput batches*. Each throughput batch runs the same type and the same amount of operations as the power batch. However, they allow concurrent execution of the write operations and read queries in a given batch.

The execution of the throughput test during audits is the *throughput measurement window*. This window must span at least for 1 hour and it must include at least one fully completed batch (see Figure 9.3).

The workload defines two execution modes for the *throughput batches*:

**Disjoint RW mode** In *disjoint RW (read-write) mode* the system performs the reads and writes separately. It first executes the writes, then evaluates the read queries. Concurrency between the read operations is allowed.

This mode is aimed at *read-optimized data analytical systems* which do not support concurrent reads and writes. Implementations may also opt to use this mode for simplicity. For these implementations, passing the *LDBC ACID compliance test* (Chapter 10) is not required.

**Concurrent RW mode** In *concurrent RW (read-write) mode* the system is allowed to run reads and writes concurrently. This requires the systems to be capable of handling *transactions*. Implementations using this mode are required to pass the *LDBC ACID compliance test* (Chapter 10).

### 9.5.3 Runtimes

The runtimes should be reported as follows:

- The *load time* ( $t_{load}$ ) denotes the time to load the data into the SUT and initialize auxiliary data structures (if applicable). For audited runs, we require that  $t_{load} < 24$  hours.
- The *power test time* ( $t_{power\ test}$ ) denotes the time to perform the power test.
- The *throughput measurement window time* ( $t_{throughput\ measurement}$ ) denotes the time to perform the throughput test, including the last (potentially unfinished) batch.
- The *full throughput batches time* ( $t_{full\ throughput\ batches}$ ) denotes the time to evaluate the fully completed batches during the throughput measurement window.

Note that a warm-up period is not allowed (unlike the Interactive workload where such a period is required, see Section 9.4.7.2).

### 9.5.4 Scoring Metrics

SNBI BI provides four scoring metrics: the *power score*, the *throughput score*, and their price-adjusted variants, the *per-\$ power score* and the *per-\$ throughput score*. All scores include the scale factor, denoted with “@SF”.

#### 9.5.4.1 Price

We follow TPC’s specification for reporting prices [85]. The price is established as the *total cost of ownership (TCO)* for the SUT used in the benchmark, reported as a breakdown of machine cost, software license costs, and maintenance costs for 3 years. In case of cloud deployments, the cost of running a 3-year reserved instance should be reported. When establishing the price, the “upfront payment” option available at certain cloud providers should not be considered.

#### 9.5.4.2 Power Scores

The definition of SNB BI’s power score follows TPC-H in using a geometric mean, ensuring that there is an incentive to improve all queries, no matter their running time. Formally, the power score is based on the time to perform the writes and the time spent for executing each variant with 30 different substitution parameters, measured in seconds:

$$power@SF = \frac{3600}{\sqrt[29]{w \cdot q_1 \cdot q_{2a} \cdot q_{2b} \cdot \dots \cdot q_{18} \cdot q_{19a} \cdot q_{19b} \cdot q_{20a} \cdot q_{20b}}} \cdot SF$$

To determine the price-adjusted power score, we factor in the *TCO*:

$$power@SF/\$ = power@SF \cdot \frac{1000}{TCO}$$

#### 9.5.4.3 Throughput Scores

The throughput score is based on  $t_{load}$ , measured in hours, and the cumulative execution time and number of the throughput batches executed:

$$throughput@SF = (24\text{ hours} - t_{load}) \cdot \frac{n_{batches}}{t_{batches}} \cdot SF$$

The subtraction of  $t_{load}$  ensures that the scoring rewards systems with efficient bulk loaders (unlike in TPC-H and TPC-DS which do not include load performance in their metrics). The price-adjusted throughput score is determined analogously:

$$throughput@SF/\$ = throughput@SF \cdot \frac{1\,000}{TCO}$$

### 9.5.5 Implementation Rules

#### 9.5.5.1 Correctness

The SUT shall evaluate all operators correctly. The auditor shall ascertain correctness on the SF10 data set. However, they are allowed to also use data sets of different scale factors, as well as issue custom operations (both reads and writes) to test for the correctness of the implementation.

The validation of correctness is performed on the output of the *power test* step. The rationale for using this only step is that during concurrent execution of R/W operations in the *throughput test*, it is not possible to guarantee deterministic query results, making validation impossible. Moreover, this step already includes a write batch, therefore the query results indirectly test the correctness of the implementation of write operations.

#### 9.5.5.2 Auxiliary Data Structures

Using auxiliary data structures (e.g. indices, materialized views) is allowed if they are kept in an up-to-date state after each write operation. The full disclosure report should enumerate the auxiliary data structures used by the SUT.

#### 9.5.5.3 Query Declarativity

Systems should use a domain-specific query language (e.g. Cypher, Gremlin, GQL, GSQL SQL/PGQ) for the implementation, including the read queries and the update operations. General-purpose programming languages (e.g. C, C++, Java, Julia) are not allowed.

Implementations shall not use *query-specific stored procedures written in a general-purpose programming language* (e.g. a given procedure which implements BI Q5). Using the stored procedure libraries considered to be the “standard libraries” of the SUT is allowed.<sup>3</sup> Implementations may use *stored procedures written in a domain-specific language*. In cases when the categorization of the approach used by the SUT’s query implementations is uncertain, it is the auditor’s responsibility to decide whether the SUT complies with this rule.

#### 9.5.5.4 Query Variants

Several queries (e.g. BI 14 ) use *a* and *b* variants with different sets of input parameters. The SUT should not receive any hints on which variant it is currently evaluating (e.g. Q14a or Q14b). Moreover, it is not allowed for the query implementations to contain code that aims to detect the query variant used.

### 9.5.6 Scaling

Audited *benchmark runs* of the BI workload shall use SF30 or larger data sets. The rationale behind this decision is to ensure that there should be a sufficient number of write operations available to guarantee the execution during the duration of the measurement window (see Figure 9.3).

### 9.5.7 Full Disclosure Report

The *full disclosure report* (FDR) and the *supplementary package* shall contain the same information as for SNB Interactive (Section 9.4.8), including, if applicable (Section 9.5.5), the ACID compliance report (Section 9.3.2).

<sup>3</sup>These libraries often include features such as weighted shortest path algorithms.



## 10 ACID TEST SUITE

*This chapter is based on the TPCTC 2020 paper “Towards Testing ACID Compliance in the LDBC Social Network Benchmark” [90], co-authored by several members of the SNB task force.*

*The framework and reference implementations of the ACID test suite are available at [https://github.com/ldbc/ldbc\\_acid](https://github.com/ldbc/ldbc_acid).*

Verifying ACID compliance is an important step in the benchmarking process for enabling fair comparison between systems. The performance benefits of operating with weaker safety guarantees are well established [30] but this can come at the cost of application correctness. To enable apples vs. apples performance comparisons between systems it is expected they uphold the ACID properties. Currently, LDBC provides no mechanism for validating ACID compliance within the SNB Interactive workflow. A simple solution would be to outsource the responsibility of demonstrating ACID compliance to benchmark implementors. However, the safety properties claimed by a system often do not match observable behaviour [41]. To mitigate this problem, benchmarks such as TPC-C [83] include a number of ACID tests to be executed as part of the benchmarking auditing process. However, we found these tests cannot readily be applied to our context, as they assume lock-based concurrency control and an interactive query API that provides clients with explicit control over a transaction’s lifecycle. Modern data systems often use optimistic concurrency control mechanisms [61] and offer a restricted query API, such as only executing transactions as stored procedures [78]. Further, tests that trigger and test row-level locking phenomena, for instance, do not readily map on graph database systems. Lastly, we found these tests are limited in the range of isolation anomalies they cover.

This chapter presents the design of an implementation-agnostic ACID-compliance test suite for the Interactive workload<sup>1</sup>. Our guiding design principle was to be agnostic of system-level implementation details, relying solely on client observations to determine the occurrence of non-transactional behaviour. Thus all systems can be subjected to the same tests and fair comparisons between SNB Interactive performance results can be drawn. Tests are described in the context of a graph database employing the property graph data model [4]. Reference implementations are given in Cypher [26], the *de facto* standard graph query language.

Particular emphasis is given to testing isolation, covering 10 known anomalies including recently discovered anomalies such as *Observed Transaction Vanishes* [10] and *Fractured Reads* [11]. The test suite has been implemented for 5 database systems.<sup>2</sup> A conscious decision was made to keep tests relatively lightweight, as to not add significant overhead to the benchmarking process.

### 10.1 Background

The tests presented in this chapter are defined on a small core of LDBC SNB schema (extended with properties for versioning) given in Figure 10.1.

### 10.2 Atomicity

*Atomicity* ensures that either all of a transaction’s actions are performed, or none are. Two atomicity tests have been developed. **Atomicity-C** checks for every successful commit message a client receives that any data items inserted or modified are subsequently visible. **Atomicity-RB** checks for every aborted transaction that all its modifications are not visible. Tests are executed as follows: (i) load a graph of `Person` nodes (Listing 10.1) each with a unique `id` and a set of `emails`; (ii) a client executes a full graph scan counting the number of nodes, edges and emails (Listing 10.4) using the result to initialize a counter `committed`; (iii)  $N$  transaction instances (Listing 10.2, Listing 10.3) of the required test are then executed, `committed` is incremented for each successful

---

<sup>1</sup>We acknowledge verifying ACID-compliance with a finite set of tests is not possible. However, the goal is not an exhaustive quality assurance test of a system’s safety properties but rather to demonstrate that ACID guarantees are supported.

<sup>2</sup>Available at [https://github.com/ldbc/ldbc\\_acid](https://github.com/ldbc/ldbc_acid).

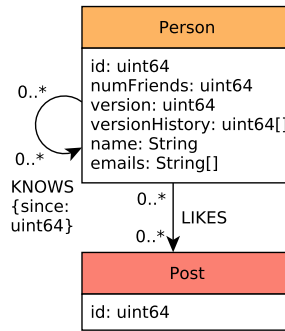
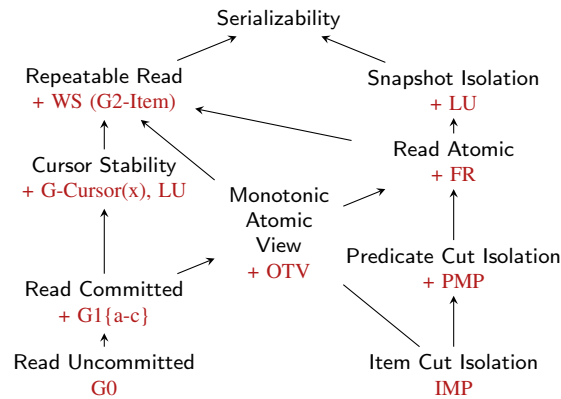


Figure 10.1: Graph schema for the ACID test queries.

Figure 10.2: Hierarchy of isolation levels as described in [11]. All anomalies are covered except **G-Cursor(x)**.

commit; (iii) repeat the full graph scan, storing the result in the variable `finalState`; (iv) perform the anomaly check: `committed=finalState`.

The **Atomicity-C** transaction (Listing 10.2) randomly selects a `Person`, creates a new `Person`, inserts a `KNOWS` edge and appends an `email`. The **Atomicity-RB** transaction (Listing 10.3) randomly selects a `Person`, appends an `email` and attempts to insert a `Person` only if it does not exist. Note, for **Atomicity-RB** if the query API does not offer a `ROLLBACK` statement constraints such as node uniqueness can be utilized to trigger an abort.

### 10.3 Isolation

The gold standard isolation level is *Serializability*, which offers protection against all possible *anomalies* that can occur from the concurrent execution of transactions. Anomalies are occurrences of non-serializable behaviour. Providing *Serializability* can be detrimental to performance [30]. Thus systems offer numerous weak isolation levels such as *Read Committed* and *Snapshot Isolation* that allow a higher degree of concurrency at the cost of potential non-serializable behaviour. As such, isolation levels are defined in terms of the anomalies they prevent [30, 10]. Figure 10.2 relates isolation levels to the anomalies they proscribe.

SNB Interactive does not require systems to provide *Serializability*. However, to allow fair comparison systems must disclose the isolation level used during benchmark execution. The purpose of these isolation tests is to verify that the claimed isolation level matches the expected behaviour. To this end, tests have been developed for each anomaly presented in [11]. Formal definitions for each anomaly are reproduced from [1, 11] using their system model which is described below. General design considerations are discussed before each test is described.

```
CREATE (:Person {id: 1, name: 'Alice', emails: ['alice@aol.com']}),
      (:Person {id: 2, name: 'Bob', emails: ['bob@hotmail.com', 'bobby@yahoo.com']})
```

Listing 10.1: Cypher query for creating initial data for the Atomicity transactions.

```
«BEGIN»
MATCH (p1:Person {id: $person1Id})
CREATE (p1)-[k:KNOWS]->(p2:Person)
SET
  p1.emails = p1.emails + [$newEmail],
  p2.id = $person2Id,
  k.creationDate = $creationDate
«COMMIT»
```

Listing 10.2: Atomicity-C Tx.

```
«BEGIN»
MATCH (p1:Person {id: $person1Id})
SET p1.emails = p1.emails + [$newEmail]
«IF» MATCH (p2:Person {id: $person2Id}) exists
«THEN» «ABORT» «ELSE»
CREATE (p2:Person {id: $person2Id, emails: []})
«END»
«COMMIT»
```

Listing 10.3: Atomicity-RB Tx.

```
MATCH (p:Person)
RETURN count(p) AS numPersons, count(p.name) AS numNames, sum(size(p.emails)) AS numEmails
```

Listing 10.4: Atomicity-C/Atomicity-RB: counting entities in the graph.

### 10.3.1 System Model

Transactions consist of an ordered sequence of read and write operations to an arbitrary set of data items, book-ended by a `BEGIN` operation and a `COMMIT` or an `ABORT` operation. In a graph database data items are nodes, edges and properties. The set of items a transaction reads from and writes to is termed its *item read set* and *item write set*. Each write creates a *version* of an item, which is assigned a unique timestamp taken from a totally ordered set (e.g. natural numbers) version  $i$  of item  $x$  is denoted  $x_i$ . All data items have an initial *unborn* version  $\perp$  produced by an initial transaction  $T_\perp$ . The unborn version is located at the start of each item's version order. An execution of transactions on a database is represented by a *history*,  $H$ , consisting of (i) each transaction's read and write operations, (ii) data item versions read and written and (iii) commit or abort operations.

There are three types of dependencies between transactions, which capture the ways in which transactions can *directly* conflict. *Read dependencies* capture the scenario where a transaction reads another transaction's write. *Antidependencies* capture the scenario where a transaction overwrites the version another transaction reads. *Write dependencies* capture the scenario where a transaction overwrites the version another transaction writes. Their definitions are as follows:

**Read-Depends** Transaction  $T_j$  *directly read-depends* (wr) on  $T_i$  if  $T_i$  writes some version  $x_k$  and  $T_j$  reads  $x_k$ .

**Anti-Depends** Transaction  $T_j$  *directly anti-depends* (rw) on  $T_i$  if  $T_i$  reads some version  $x_k$  and  $T_j$  writes  $x$ 's next version after  $x_k$  in the version order.

**Write-Depends** Transaction  $T_j$  *directly write-depends* (ww) on  $T_i$  if  $T_i$  writes some version  $x_k$  and  $T_j$  writes  $x$ 's next version after  $x_k$  in the version order.

Using these definitions, from a history  $H$  a *direct serialization graph*  $DSG(H)$  is constructed. Each node in the  $DSG$  corresponds to a committed transaction and edges correspond to the types of direct conflicts between transactions. Anomalies can then be defined by stating properties about the  $DSG$ .

The above *item-based* model can be extended to handle *predicate-based* operations [1]. Database operations are frequently performed on set of items provided a certain condition called the *predicate*,  $P$  holds. When a transaction executes a read or write based on a predicate  $P$ , the database selects a version for each item to which  $P$  applies, this is called the version set of the predicate-based denoted as  $Vset(P)$ . A transaction  $T_j$  changes the matches of a predicate-based read  $r_i(P_i)$  if  $T_i$  overwrites a version in  $Vset(P_i)$ .

### 10.3.2 General Design

Isolation tests begin by loading a *test graph* into the database. Configurable numbers of *write clients* and *read clients* then execute a sequence of transactions on the database for some configurable time period. After execution, results from read clients are collected and an *anomaly check* is performed. In some tests an additional full graph scan is performed after the execution period in order to collect information required for the anomaly check.

The guiding principle behind test design was the preservation of data item’s version history – the key ingredient needed in the system model formalization which is often not readily available to clients, if preserved at all. Several anomalies are closely related, tests therefore had to be constructed such that other anomalies could not interfere with or mask the detection of the targeted anomaly. Test descriptions provide (i) informal and formal anomaly definitions, (ii) the required test graph, (iii) description of transaction profiles write and read clients execute, and (iv) reasoning for why the test works.

### 10.3.3 Dirty Write

Informally, a *Dirty Write* (Adya’s **G0** [1]) occurs when updates by conflicting transactions are interleaved. For example, say  $T_i$  and  $T_j$  both modify items  $\{x, y\}$ . If version  $x_i$  precedes version  $x_j$  and  $y_j$  precedes version  $y_i$  a **G0** anomaly has occurred. Preventing **G0** is especially important in a graph database in to order to maintain *Reciprocal Consistency* [88].

**Definition.** A history  $H$  exhibits phenomenon **G0** if  $DSG(H)$  contains a directed cycle consisting entirely of write-dependency edges.

**Test.** Load a test graph containing pairs of Person nodes connected by a KNOWS edge. Assign each Person a unique id and each Person and KNOWS edge a `versionHistory` property of type list (initially empty). During the execution period, write clients execute a sequence of **G0**  $T_W$  instances, Listing 10.5. This transaction appends its ID to the `versionHistory` property for each entity in the Person pair it matches. Note, transaction IDs are assumed to be globally unique. After execution, a read client issues a **G0**  $T_R$  for each Person pair in the graph, Listing 10.6. Retrieving the `versionHistory` for each entity (2 Persons and 1 KNOWS edge) in a Person pair.

**Anomaly check.** For each Person pair in the test graph: (i) prune each `versionHistory` list to remove any version numbers that do not appear in all lists; needed to account for interference from *Lost Update* anomalies (Section 10.3.8), (ii) perform an element-wise comparison between `versionHistory` lists for each entity, (iii) if lists do not agree a **G0** anomaly has occurred.

**Why it works.** Each **G0**  $T_W$  effectively creates a new version of a Person pair. Appending the transaction ID preserves the version history of each entity in the Person pair. In a system that prevents **G0**, each entity of the Person pair should experience the *same* updates, in the *same* order. Hence, each position in the `versionHistory` lists should be equivalent. The additional pruning step is needed as *Lost Updates* overwrite a version, effectively erasing it from the history of a data item.

```
MATCH
  (p1:Person {id: $person1Id})
  -[k:KNOWS]->(p2:Person {id: $person2Id})
SET p1.versionHistory = p1.versionHistory + [$tId]
SET p2.versionHistory = p2.versionHistory + [$tId]
SET k.versionHistory = k.versionHistory + [$tId]
```

Listing 10.5: Dirty Write (**G0**)  $T_W$ .

```
MATCH (p1:Person {id: $person1Id})
  -[k:KNOWS]->(p2:Person {id: $person2Id})
RETURN
  p1.versionHistory AS p1VersionHistory,
  k.versionHistory AS kVersionHistory,
  p2.versionHistory AS p2VersionHistory
```

Listing 10.6: Dirty Write (**G0**)  $T_R$ .

### 10.3.4 Dirty Reads

#### Aborted Reads

Informally, an *Aborted Read* (G1a) anomaly occurs when a transaction reads the updates of a transaction that later aborts.

**Definition.** A history  $H$  exhibits phenomenon **G1a** if  $H$  contains an aborted transaction  $T_i$  and a committed transaction  $T_j$  such that  $T_j$  reads a version written by  $T_i$ .

**Test.** Load a test graph containing only `Person` nodes into the database. Assign each `Person` a unique `id` and `version` initialized to 1; any odd number will suffice. During execution, write clients execute a sequence of G1a  $T_W$  instances, Listing 10.7. Selecting a random `Person id` to populate each instance. This transaction attempts to set `version=2` (any even number will suffice) but always aborts. Concurrently, read clients execute a sequence of G1a  $T_R$  instances, Listing 10.8. This transaction retrieves the `version` property of a `Person`. Read clients store results, which are pooled after execution has finished.

**Anomaly check.** Each read should return `version=1` (or any odd number). Otherwise, a **G1a** anomaly has occurred.

**Why it works.** Each transaction that attempts to set `version` to an even number *always* aborts. Therefore, if a transaction reads `version` to be an even number, it must have read the write of an aborted transaction.

```
MATCH (p:Person {id: $personId})
SET p.version = 2
«SLEEP($sleepTime)»
«ABORT»
```

Listing 10.7: Aborted Read (G1a)  $T_W$ .

```
MATCH (p:Person {id: $personId})
SET p.version = $even
«SLEEP($sleepTime)»
SET p.version = $odd
```

Listing 10.9: Interm. Read (G1b)  $T_W$ .

```
MATCH (p:Person {id: $personId})
RETURN p.version
```

Listing 10.8: Aborted Read (G1a)  $T_R$ .

```
MATCH (p:Person {id: $personId})
RETURN p.version
```

Listing 10.10: Interm. Read (G1b)  $T_R$ .

#### Intermediate Reads

Informally, an *Intermediate Read* (Adya's **G1b** [1]) anomaly occurs when a transaction reads the intermediate modifications of other transactions.

**Definition.** A history  $H$  exhibits phenomenon **G1b** if  $H$  contains a committed transaction  $T_i$  that reads a version of an object  $x_m$  written by transaction  $T_j$ , and  $T_j$  also wrote a version  $x_n$  such that  $m < n$  in  $x$ 's version order.

**Test.** Load a test graph containing only `Person` nodes into the database. Assign each `Person` a unique `id` and `version` initialized to 1; any odd number will suffice. During execution, write clients execute a sequence of G1b  $T_W$  instances, Listing 10.9. This transaction sets `version` to an even number, then an odd number before committing. Concurrently read-clients execute a sequence of G1b  $T_R$  instances, Listing 10.10. Selecting a `Person` by `id` and retrieving its `version` property. Read clients store results which are collected after execution has finished.

```

MATCH (p1:Person {id: $person1Id}) SET p1.version = $transactionId
MATCH (p2:Person {id: $person2Id}) RETURN p2.version

```

Listing 10.11: G1c  $T_{RW}$ .

**Anomaly check.** Each read of `version` should be an odd number. Otherwise, a **G1b** anomaly has occurred.

**Why it works.** The final version installed by an G1b  $T_W$  instance is *never* an even number. Therefore, if a transaction reads `version` to be an even number it must have read an intermediate version.

### Circular Information Flow

Informally, a *Circular Information Flow* (Adya’s **G1c** [1]) anomaly occurs when two transactions affect each other; i.e. both transactions write information the other reads. For example, transaction  $T_i$  reads a write by transaction  $T_j$  and transaction  $T_j$  reads a write by  $T_i$ .

**Definition.** A history  $H$  exhibits phenomenon **G1c** if  $DSG(H)$  contains a directed cycle that consists entirely of read-dependency and write-dependency edges.

**Test.** Load a test graph containing only `Person` nodes into the database. Assign each `Person` a unique `id` and `version` initialized to 0. Read-write clients are required for this test, executing a sequence of G1c  $T_{RW}$ , Listing 10.11. This transaction selects two different `Person` nodes, setting the `version` of one `Person` to the transaction ID and retrieving the `version` from the other. Note, transaction IDs are assumed to be globally unique. Transaction results are stored in format `(txn.id, versionRead)` and collected after execution.

**Anomaly check.** For each result, check the result of the transaction the `versionRead` corresponds to, did not read the transaction of that result. If so a **G1c** anomaly has occurred.

**Why it works.** Consider the result set:  $\{(T_1, T_2), (T_2, T_3), (T_3, T_2)\}$ .  $T_1$  reads the version written by  $T_2$  and  $T_2$  reads the version written by  $T_3$ . Here information flow is unidirectional from  $T_1$  to  $T_2$ . However,  $T_2$  reads the version written by  $T_3$  and  $T_3$  reads the version written by  $T_2$ . Here information flow is circular from  $T_2$  to  $T_3$  and  $T_3$  to  $T_2$ . Thus a **G1c** anomaly has been detected.

### 10.3.5 Cut Anomalies

#### Item-Many-Preceders

Informally, an *Item-Many-Preceders* (**IMP**) anomaly [10] occurs if a transaction observes multiple versions of the same item (e.g. transaction  $T_i$  reads versions  $x_1$  and  $x_2$ ). In a graph database this can be multiple reads of a node, edge, property or label. Local transactions (involving a single data item) occur frequently in graph databases, e.g. in “Retrieve content of a message” **IS 4**.

**Definition.** A history  $H$  exhibits **IMP** if  $DSG(H)$  contains a transaction  $T_i$  such that  $T_i$  directly *item-reads* on  $x$  by more than one other transaction.

**Test.** Load a test graph containing `Person` nodes. Assign each `Person` a unique `id` and `version` initialized to 1. During execution write clients execute a sequence of IMP  $T_W$  instances, Listing 10.12. Selecting a random `id` and installing a new version of the `Person`. Concurrently read clients execute a sequence of IMP  $T_R$  instances, Listing 10.13. Performing multiple reads of the same `Person`; successive reads can be separated by some artificially injected wait time to make conditions more favourable for detecting an anomaly. Both reads within an IMP  $T_R$  transaction are returned, stored and collected after execution.



**Anomaly check.** Each IMP  $T_R$  result set (*firstRead*, *secondRead*) should contain the *same* Person version. Otherwise, an **IMP** anomaly has occurred.

**Why it works.** By performing successive reads within the same transaction this test checks that a system ensures consistent reads of the same data item. If the version changes then a concurrent transaction has modified the data item and the reading transaction is not protected from this change.

```
MATCH (p:Person {id: $personId})
SET p.version = p.version + 1
```

Listing 10.12: IMP  $T_W$ .

```
MATCH (pe:Person {id: $personId}), (po:Post {id: $postId})
CREATE (pe)-[:LIKES]->(po)
```

Listing 10.14: PMP  $T_W$ .

```
MATCH (p1:Person {id: $personId})
WITH p1.version AS firstRead
«SLEEP($sleepTime)»
MATCH (p2:Person {id: $personId})
RETURN firstRead,
       p2.version AS secondRead
```

Listing 10.13: IMP  $T_R$ .

```
MATCH (po1:Post {id: $postId})<-[:LIKES]-(pe1:Person)
WITH count(pe1) AS firstRead
«SLEEP($sleepTime)»
MATCH (po2:Post {id: $postId})<-[:LIKES]-(pe2:Person)
RETURN firstRead,
       count(pe2) AS secondRead
```

Listing 10.15: PMP  $T_R$ .

## Predicate-Many-Preceders

Informally, a **Predicate-Many-Preceders (PMP)** anomaly [10] occurs if a transaction observes different versions resulting from the same predicate read (e.g.  $T_i$  reads  $Vset(P_i) = \{x_1\}$  and  $Vset(P_i) = \{x_1, y_2\}$ ). Pattern matching is a common predicate read operation in a graph database, e.g. query “*Find friends and friends of friends that have been to given countries*” **IC 3**.

**Definition.** A history  $H$  exhibits the phenomenon **PMP** if, for all predicate-based reads  $r_i(P_i : Vset(P_i))$  and  $r_j(P_j : Vset(P_j))$  in  $T_k$  such that the logical ranges of  $P_i$  and  $P_j$  overlap (call it  $P_o$ ), the set of transactions that change the matches of  $P_o$  for  $r_i$  and  $r_j$  differ.

**Test.** Load a test graph containing *Person* and *Post* nodes. Within each node type assign unique IDs. During execution write clients execute a sequence of PMP  $T_W$  instances, inserting a *LIKES* edge between a randomly selected *Person* and *Post*, shown in Listing 10.14. Concurrently read clients execute a sequence of PMP  $T_R$  instances, Listing 10.15. Performing multiple reads of the pattern  $(po:Post) <-[:LIKES]-(p:Person)$  and counting the number of *LIKES* edges; successive reads can be separated by some artificially injected wait time to make conditions more favourable for detecting an anomaly. Both predicate reads within a PMP  $T_R$  transaction are returned, stored and collected after test execution.

**Anomaly check.** For each PMP  $T_R$  transaction result set (*firstRead*, *secondRead*), the *firstRead* should be equal to *secondRead*. Otherwise, a **PMP** anomaly has occurred.

**Why it works.** By performing successive predicate reads and counting the number of *LIKES* edges within the same transaction this test checks that a system ensures consistent reads of the same predicate. If the number of *LIKES* edges changes then a concurrent transaction has inserted a new *LIKES* edge and the reading transaction is not protected from this change.

### 10.3.6 Observed Transaction Vanishes

Informally, an *Observed Transaction Vanishes* (OTV) anomaly [10] occurs when a transaction observes part of another transaction's updates but not all of them (e.g.  $T_1$  writes  $x_1$  and  $y_1$  and  $T_2$  reads  $x_1$  and  $y_1$ ). Before formally defining OTV the *Unfolded Serialization Graph* (USG) must be introduced [1]. The USG is specified for an individual transaction,  $T_i$  and a history,  $H$  and is denoted by  $USG(H, T_i)$ . In a USG the  $T_i$  node is split into multiple nodes, one for each action read  $r_i(\cdot)$  or write  $w_i(\cdot)$  within the transaction. The dependency edges are now incident on the relevant event of  $T_i$ . Additionally, actions within  $T_i$  are connected by an *order edge* e.g. if  $T_i$  reads object  $y_j$  then immediately writes on object  $x$  an order edge exists from  $w_i(x_i)$  to  $r_i(y_j)$ .

**Definition.** A history  $H$  exhibits phenomenon OTV if  $USG(H, T_i)$  contains a directed cycle consisting of (i) exactly one read dependency edge induced by data item  $x$  from  $T_j$  to  $T_i$  and (ii) a set of edges induced by data item  $y$  containing at least one anti dependency edge from  $T_i$  to  $T_j$ . Additionally,  $T_i$ 's read from  $y$  precedes its read from  $x$ .

**Test.** Load a test graph containing a set of cycles of length 4 of Persons with same name connected by Knows edges. Assign each Person an id, name and version property (initialized to 1). Note, id must be unique across nodes and name must be unique across cycles. During execution write clients select a name, id and executes a sequence of OTV  $T_W$  instances, Listing 10.16. This transaction effectively creates a new version of a given cycle. Concurrently read-clients execute a sequence of OTV  $T_R$  instances, Listing 10.17. Matching a given cycle and performing multiple reads. Both reads within an OTV  $T_R$  are returned, stored and collected after execution.

**Anomaly check.** For each OTV  $T_R$  result set (firstRead, secondRead), the maximum version in the firstRead should be less than or equal to the minimum version in the secondRead. Otherwise, an OTV anomaly has occurred.

**Why it works.** OTV  $T_W$  installs a new version of a cycle by updating the version property of each Person. Therefore when matching a cycle once a transaction has observed some version it should *at least* observe this version for every remaining entity in the cycle. Unfortunately, this cannot be deduced from a single read of the cycle as results from matching cycles often does not preserve the order in which graph entities were read. This is solved by making multiple reads of the cycle. The maximum version of the firstRead determines the minimum version of secondRead. If this condition is violated then a transaction has observed the effects of a transaction in the firstRead then subsequently failed to observe it in the secondRead – the observed transaction has vanished!

```
MATCH path =
  (n:Person {id: $personId})
  -[:KNOWS*..4]->(n)
UNWIND nodes(path)[0..4] AS p
SET p.version = p.version + 1
```

Listing 10.16: OTV/FR  $T_W$ .

```
MATCH p1=(n1:Person {id: $personId})-[:KNOWS*..4]->(n1)
RETURN extract(p IN nodes(p1) | p.version) AS firstRead
«SLEEP($sleepTime)»
MATCH p2=(n2:Person {id: $personId})-[:KNOWS*..4]->(n2)
RETURN extract(p IN nodes(p2) | p.version) AS secondRead
```

Listing 10.17: OTV/FR  $T_R$ .

### 10.3.7 Fractured Read

Informally, a *Fractured Read* (FR) anomaly [11] occurs when a transaction reads *across* transaction boundaries. For example, if  $T_1$  writes  $x_1$  and  $y_1$  and  $T_3$  writes  $x_3$ . If  $T_2$  reads  $x_1$  and  $y_1$ , then repeats its read of  $x$  and reads  $x_3$  a fractured read has occurred.



**Definition.** A transaction  $T_j$  exhibits phenomenon **FR** if transaction  $T_i$  writes versions  $x_a$  and  $y_b$  (in any order, where  $x$  and  $y$  may or may not be distinct items),  $T_j$  reads version  $x_a$  and version  $y_c$ , and  $c < b$ .

**Test.** Same as the **OTV** test.

**Anomaly check.** For each FR  $T_R$  (Listing 10.17) result set (firstRead, secondRead), all versions across both version sets should be equal. Otherwise, an **FR** anomaly has occurred.

**Why it works.** FR  $T_W$  installs a new version of a cycle by updating the version properties on each Person. When FR  $T_R$  observes a version every subsequent read in that cycle should read the *same* version as FR  $T_W$  (Listing 10.16) installs the same version for all Person nodes in the cycle. Thus, if it observes a different version it has observed the effect of a different transaction and has read across transaction boundaries.

### 10.3.8 Lost Update

Informally, a **Lost Update (LU)** anomaly [11] occurs when two transactions concurrently attempt to make conditional modifications to the same data item(s).

**Definition.** A history  $H$  exhibits phenomenon **LU** if  $DSG(H)$  contains a directed cycle having one or more antidependency edges and all edges are induced by the same data item  $x$ .

**Test.** Load a test graph containing Person nodes. Assign each Person a unique id and a property numFriends (initialized to 0). During execution write clients execute a sequence of LU  $T_W$  instances, Listing 10.18. Choosing a random Person and incrementing its numFriends property. Clients store local counters (expNumFriends) for each Person, which is incremented each time a Person is selected *and* the LU  $T_W$  instance successfully commits. After the execution period the numFriends is retrieved for each Person using LU  $T_R$  in Listing 10.19 and expNumFriends are pooled from write clients for each Person.

**Anomaly check.** For each Person its numFriends property should be equal to the (global) expNumFriends for that Person.

**Why it works.** Clients know how many successful LU  $T_W$  instances were issued for a given Person. The observable numFriends should reflect this ground truth, otherwise, an **LU** anomaly must have occurred.

```
MATCH (p:Person {id: $personId})
SET p.numFriends = p.numFriends + 1
```

Listing 10.18: Lost Update  $T_W$ .

```
MATCH (p:Person {id: $personId})
RETURN p.numFriends AS numFriends
```

Listing 10.19: Lost Update  $T_R$ .

### 10.3.9 Write Skew

Informally, **Write Skew (WS)** occurs when two transactions simultaneously attempted to make *disjoint* conditional modifications to the same data item(s). It is referred to as **G2-Item** in [1, 25].

**Definition.** A history  $H$  exhibits **WS** if  $DSG(H)$  contains a directed cycle having one or more antidependency edges.

**Test.** Load a test graph containing  $n$  pairs of Person nodes ( $p_1, p_2$ ) for  $k = 0, \dots, n - 1$ , where the  $k$ th pair gets IDs  $p_1.id = 2k+1$  and  $p_2.id = 2k+2$ , and values  $p_1.value = 70$  and  $p_2.value = 80$ . There is a constraint:  $p_1.value + p_2.value > 0$ . During execution write clients execute a sequence of WS  $T_W$  instances, Listing 10.20. Selecting a random Person pair and decrementing the value property of one Person provided doing so would not violate the constraint. After execution the database is scanned using WS  $T_R$ , Listing 10.21.

**Anomaly check.** For each Person pair the constraint should hold true, otherwise, a **WS** anomaly has occurred.

**Why it works.** Under no Serializable execution of WS  $T_W$  instances would the constraint  $p_1.value + p_2.value > 0$  be violated. Therefore, if WS  $T_R$  returns a violation of this constraint it is clear a **WS** anomaly has occurred.

```
MATCH (p1:Person {id: $person1Id}),
      (p2:Person {id: $person2Id})
«IF (p1.value+p2.value < 100)» «THEN» «ABORT» «END»
«SLEEP($sleepTime)»
pId = «pick randomly between personId1, personId2»
MATCH (p:Person {id: $pId})
SET p.value = p.value - 100
«COMMIT»
```

Listing 10.20: WS  $T_W$ .

```
MATCH (p1:Person),
      (p2:Person {id: p1.id+1})
WHERE p1.value + p2.value <= 0
RETURN
  p1.id AS p1id,
  p1.value AS p1value,
  p2.id AS p2id,
  p2.value AS p2value
```

Listing 10.21: WS  $T_R$ .

## 10.4 Consistency and Durability Tests

While this chapter mainly focused on *atomicity* and *isolation* from the ACID properties, we provide a short overview of the other two aspects.

**Durability** is a hard requirement for SNB Interactive and checking it is part of the auditing process. The durability test requires the execution of the SNB Interactive workload and uses the LDBC workload driver. Note, the database and the driver must be configured in the same way as would be used in the performance run. First, the database is subject to a warm-up period. Then after 2 hours of simulation execution, the database processes will be terminated, possibly by disconnecting the entire machine or by a hard process kill. Note that turning the machine off may not be possible in cloud tests. The database system is then restarted and each client issues a read for the value of the last entity (node or edge) it received a successful commit message for, that should produce a correct response.

**Consistency** is defined in terms of constraints: the database remains consistent under updates; i.e. no constraint is violated. Relational database systems usually support primary- and foreign-key constraints, as well as domain constraints on column values and sometimes also support simple within-row constraints. Graph database systems have a diversity of interfaces and generally do not support constraints, beyond sometimes domain and primary key constraints (in case indices are supported). As such, we leave them out of scope for LDBC SNB. However, we do note that we anticipate that graph database systems will evolve to support constraints in the future. Beyond equivalents of the relational ones, property graph systems might introduce graph-specific constraints, such as (partial) compliance to a schema formulated on top of property graphs, rules that guide the presence of labels or structural graph constraints such as connectedness of the graph, absence of cycles, or arbitrary well-formedness constraints [75].

## 11 RELATED WORK

*A detailed list of LDBC publications is curated at <https://ldbcouncil.org/publications>.*

### 11.1 ACID Tests in Other Benchmarks

The challenge of verifying ACID-compliance has been addressed before by transactional benchmarks. For example, TPC-C [83] provides a suite of ACID tests. However, the isolation tests are reliant on lock-based concurrency control, hence are not generalizable across systems. Also, the transactional anomaly test coverage is limited to only four anomalies. The authors of [21] augment the popular YCSB framework for benchmarking transactional NewSQL systems, including a *validation phase* that detects and quantifies consistency anomalies. They permit the definition of arbitrary integrity constraints, checking they hold before and after a benchmark run. Such an approach is not possible within SNB Interactive due to the restrictive nature of transactional updates and the distinct lack of application-level constraints.

The Hermitage project [43] with the goal of improving understanding of weak isolation, developed a range of hand-crafted isolation tests. This test suite has much higher anomaly coverage but suffers from a problem similar to TPC-C. Test execution is performed by hand, opening multiple terminals to step through the tests.<sup>1</sup> The Jepsen project [41] is not a benchmark rather it addresses correctness testing, traditionally focusing on distributed systems under various failure modes. Most of Jepsen’s transactional tests adopt a similar approach to us, executing a suite of transactions with hand-proven invariants. However recently, the project has spawned Elle [42] a black-box transactional anomaly checker. Elle does not rely on hand-crafted tests and can detect every anomaly in [1] (except for predicate-based anomalies) from an arbitrary transaction history.

### 11.2 Graph Processing Benchmarks

Recent graph benchmarking initiatives focus on three key areas:

1. transactional workloads consisting of interactive read and update queries (OLTP) aiming at graph databases that explore small portions of the graph in each query [13, 8, 19, 24, 46],
2. graph analysis algorithms (e.g. PageRank) computed in bulk, typically expressed in cluster frameworks with graph APIs, rather than high-level queries [9, 23, 56, 38],
3. pattern matching and inferencing on semantic data [33, 74, 54, 3, 82].

The SIGMOD 2014 Programming Contest defined queries on the Social Network Benchmark schema with a mix of subgraph projection and graph analytics [22].

The challenges of using benchmarks correctly are described in [69].

The Interactive queries were used in paper [60] to compare the performance of Gremlin, Cypher, SQL and SPARQL query engines.

The Labelled Subgraph Query Benchmark (LSQB) [51] uses graphs produced by the LDBC SNB Datagen but simplifies them by omitting all attributes. It defines join-heavy subgraph queries to perform graph pattern matching.

### 11.3 Scalable Graph Generators

A recent survey [17] studied 38 graph generators, finding that only 4 of them supported generating updates and, intriguingly, even these generators only yield insertions and simple deletions at best. *LinkBench* [8] defines primitive delete operations targeting a single node or a single edge. *XGDBench* [19] defines an operation that deletes a single node. The *Social Network Intelligence BenchMark* (SIB) [16] (a precursor to LDBC SNB) requires the deletion of individual nodes (posts/photos).

---

<sup>1</sup>We initially experimented with Hermitage but found it difficult to induce anomalies that relied on fast timings due to some graph databases offering limited client-side control over transactions, with all statements submitted in one batch.

## BIBLIOGRAPHY

- [1] Atul Adya. “Weak consistency: A generalized theory and optimistic implementations for distributed transactions”. Ph.D. dissertation. MIT, 1999.
- [2] Hazim Almuhiemedi et al. “Tweets are forever: a large-scale quantitative analysis of deleted tweets”. In: *Computer Supported Cooperative Work, CSCW 2013, San Antonio, TX, USA, February 23-27, 2013*. Ed. by Amy S. Bruckman et al. ACM, 2013, pp. 897–908. DOI: 10.1145/2441776.2441878.
- [3] Günes Aluç et al. “Diversified Stress Testing of RDF Data Management Systems”. In: *ISWC*. 2014, pp. 197–212. DOI: 10.1007/978-3-319-11964-9\_13.
- [4] Renzo Angles et al. “Foundations of Modern Query Languages for Graph Databases”. In: *ACM Comput. Surv.* 50.5 (2017), 68:1–68:40. DOI: 10.1145/3104031.
- [5] Renzo Angles et al. “G-CORE: A Core for Future Graph Query Languages”. In: *SIGMOD*. ACM, 2018, pp. 1421–1432. DOI: 10.1145/3183713.3190654.
- [6] Renzo Angles et al. “The LDBC Social Network Benchmark”. In: *CoRR abs/2001.02299* (2020). URL: <http://arxiv.org/abs/2001.02299>.
- [7] Renzo Angles et al. “The Linked Data Benchmark Council: A graph and RDF industry benchmarking effort”. In: *SIGMOD Record* 43.1 (2014), pp. 27–31. DOI: 10.1145/2627692.2627697.
- [8] Timothy G. Armstrong et al. “LinkBench: A database benchmark based on the Facebook social graph”. In: *SIGMOD*. 2013, pp. 1185–1196. DOI: 10.1145/2463676.2465296.
- [9] David A. Bader and Kamesh Madduri. “Design and Implementation of the HPCS Graph Analysis Benchmark on Symmetric Multiprocessors”. In: *HiPC*. 2005, pp. 465–476. DOI: 10.1007/11602569\_48.
- [10] Peter Bailis et al. “Highly Available Transactions: Virtues and Limitations”. In: *VLDB* (2013). DOI: 10.14778/2732232.2732237.
- [11] Peter Bailis et al. “Scalable Atomic Visibility with RAMP Transactions”. In: *ACM Trans. Database Syst.* (2016). DOI: 10.1145/2909870.
- [12] Nurzhan Bakibayev, Dan Olteanu, and Jakub Zavodny. “FDB: A Query Engine for Factorised Relational Databases”. In: *Proc. VLDB Endow.* 5.11 (2012), pp. 1232–1243. DOI: 10.14778/2350229.2350242.
- [13] Sumita Barahmand and Shahram Ghandeharizadeh. “BG: A Benchmark to Evaluate Interactive Social Networking Actions”. In: *CIDR*. 2013. URL: [http://cidrdb.org/cidr2013/Papers/CIDR13\\_Paper93.pdf](http://cidrdb.org/cidr2013/Papers/CIDR13_Paper93.pdf).
- [14] Mauro Barone and Michele Coscia. “Birds of a feather scam together: Trustworthiness homophily in a business network”. In: *Social Networks* 54 (2018), pp. 228–237. DOI: 10.1016/j.socnet.2018.01.009.
- [15] Maciej Besta et al. “Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries”. In: *CoRR abs/1910.09017* (2019). URL: <http://arxiv.org/abs/1910.09017>.
- [16] Peter Boncz et al. *Social Network Intelligence BenchMark*. 2013. URL: [https://www.w3.org/wiki/Social\\_Network\\_Intelligence\\_BenchMark](https://www.w3.org/wiki/Social_Network_Intelligence_BenchMark).
- [17] Angela Bonifati et al. “Graph Generators: State of the Art and Open Challenges”. In: *ACM Comput. Surv.* 53.2 (2020), 36:1–36:30. DOI: 10.1145/3379445.
- [18] Federico Busato et al. “Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices on GPUs”. In: *HPEC*. IEEE, 2018, pp. 1–7. DOI: 10.1109/HPEC.2018.8547541.
- [19] Miyuru Dayarathna and Toyotaro Suzumura. “Graph database benchmarking on cloud environments with XGDBench”. In: *Autom. Softw. Eng.* 21.4 (2014), pp. 509–533. DOI: 10.1007/s10515-013-0138-7.
- [20] Alin Deutsch et al. “Graph Pattern Matching in GQL and SQL/PGQ”. In: *SIGMOD*. ACM, 2022, pp. 2246–2258. DOI: 10.1145/3514221.3526057.

- [21] Akon Dey et al. “YCSB+T: Benchmarking web-scale transactional databases”. In: *ICDE*. IEEE Computer Society, 2014, pp. 223–230. DOI: 10.1109/ICDEW.2014.6818330.
- [22] Márton Elekes, János Benjamin Antal, and Gábor Szárnyas. “An analysis of the SIGMOD 2014 Programming Contest: Complex queries on the LDBC social network graph”. In: *CoRR* abs/2010.12243 (2020). URL: <https://arxiv.org/abs/2010.12243>.
- [23] Benedikt Elser and Alberto Montresor. “An evaluation study of BigData frameworks for graph processing”. In: *Big Data*. 2013, pp. 60–67. DOI: 10.1109/BigData.2013.6691555.
- [24] Orri Erling et al. “The LDBC Social Network Benchmark: Interactive Workload”. In: *SIGMOD*. 2015, pp. 619–630. DOI: 10.1145/2723372.2742786.
- [25] Alan Fekete et al. “Making snapshot isolation serializable”. In: *ACM Trans. Database Syst.* 30.2 (2005), pp. 492–528. DOI: 10.1145/1071610.1071615.
- [26] Nadime Francis et al. “Cypher: An Evolving Query Language for Property Graphs”. In: *SIGMOD*. ACM, 2018, pp. 1433–1445. DOI: 10.1145/3183713.3190657.
- [27] Michael J. Freitag et al. “Adopting Worst-Case Optimal Joins in Relational Database Systems”. In: *VLDB* 13.11 (2020), pp. 1891–1904. URL: <http://www.vldb.org/pvldb/vol13/p1891-freitag.pdf>.
- [28] Goetz Graefe. “Query Evaluation Techniques for Large Databases”. In: *ACM Comput. Surv.* 25.2 (1993), pp. 73–170. DOI: 10.1145/152610.152611.
- [29] Jim Gray et al. “Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals”. In: *Data Min. Knowl. Discov.* 1.1 (1997), pp. 29–53. DOI: 10.1023/A:1009726021843.
- [30] Jim Gray et al. “Granularity of Locks and Degrees of Consistency in a Shared Data Base”. In: *IFIP Working Conference on Modelling in Data Base Management Systems*. 1976, pp. 365–394.
- [31] Alastair Green et al. “Updating Graph Databases with Cypher”. In: *PVLDB* 12.12 (2019), pp. 2242–2253. DOI: 10.14778/3352063.3352139.
- [32] Andrey Gubichev and Peter A. Boncz. “Parameter Curation for Benchmark Queries”. In: *TPCTC*. Vol. 8904. Lecture Notes in Computer Science. Springer, 2014, pp. 113–129.
- [33] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. “LUBM: A benchmark for OWL knowledge base systems”. In: *J. Web Sem.* 3.2-3 (2005), pp. 158–182. DOI: 10.1016/j.websem.2005.06.005.
- [34] Pankaj Gupta et al. “WTF: The who to follow service at Twitter”. In: *WWW*. International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 505–514. DOI: 10.1145/2488388.2488433.
- [35] Pranjal Gupta, Amine Mhedhbi, and Semih Salihoglu. “Columnar Storage and List-based Processing for Graph Database Management Systems”. In: *Proc. VLDB Endow.* 14.11 (2021), pp. 2491–2504. DOI: 10.14778/3476249.3476297.
- [36] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. “Graph Grammars with Negative Application Conditions”. In: *Fundam. Inform.* 26.3/4 (1996), pp. 287–313. DOI: 10.3233/FI-1996-263404.
- [37] Torsten Hoefler and Roberto Belli. “Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results”. In: *SC*. ACM, 2015, 73:1–73:12. DOI: 10.1145/2807591.2807644.
- [38] Alexandru Iosup et al. “LDBC Graphalytics: A Benchmark for Large-Scale Graph Analysis on Parallel and Distributed Platforms”. In: *VLDB* 9.13 (2016), pp. 1317–1328. DOI: 10.14778/3007263.3007270.
- [39] Alexandru Iosup et al. “The LDBC Graphalytics Benchmark”. In: *CoRR* abs/2011.15028 (2020). URL: <https://arxiv.org/abs/2011.15028>.
- [40] Moritz Kaufmann. *Examining the TPC Pricing Specification 2.0.0*. Presented at the 9th LDBC TUC. 2017. URL: <https://ldbncouncil.org/event/ninth-tuc-meeting/attachments/59277315/75431947.pdf>.
- [41] Kyle Kingsbury. *Jepsen Analyses*. <http://jepsen.io/analyses>. 2020.

- [42] Kyle Kingsbury and Peter Alvaro. “Elle: Inferring Isolation Anomalies from Experimental Observations”. In: *CoRR* abs/2003.10554 (2020). URL: <https://arxiv.org/abs/2003.10554>.
- [43] Martin Kleppmann. *Hermitage: Testing transaction isolation levels*. <https://github.com/ept/hermitage>. 2020.
- [44] LDBC. *Byelaws of the Linked Data Benchmark Council v1.3*. <https://ldbouncil.org/docs/LDBC.Byelaws.1.3.ADOPTED.2021-01-14.pdf>. 2021.
- [45] Jure Leskovec et al. “Microscopic evolution of social networks”. In: *KDD*. 2008, pp. 462–470. DOI: 10.1145/1401890.1401948.
- [46] Matteo Lissandrini, Martin Brugnara, and Yannis Velegrakis. “Beyond Macrobenchmarks: Microbenchmark-based Graph Database Evaluation”. In: *PVLDB* 12.4 (2018), pp. 390–403. URL: <http://www.vldb.org/pvldb/vol12/p390-lissandrini.pdf>.
- [47] László Lőrincz et al. “Collapse of an online social network: Burning social capital to create it?” In: *Soc. Networks* 57 (2019), pp. 43–53. DOI: 10.1016/j.socnet.2018.11.004.
- [48] M. McPherson, L. Smith-Lovin, and J. M. Cook. “Birds of a feather: Homophily in social networks”. In: *Annual Review of Sociology* (2001), pp. 415–444.
- [49] Ulrich Meyer and Peter Sanders. “Delta-stepping: A parallelizable shortest path algorithm”. In: *J. Algorithms* 49.1 (2003), pp. 114–152. DOI: 10.1016/S0196-6774(03)00076-2.
- [50] Amine Mhedhbi and Semih Salihoglu. “Optimizing Subgraph Queries by Combining Binary and Worst-Case Optimal Joins”. In: *Proc. VLDB Endow.* 12.11 (2019), pp. 1692–1704. DOI: 10.14778/3342263.3342643.
- [51] Amine Mhedhbi et al. “LSQB: A large-scale subgraph query benchmark”. In: *GRADES-NDA at SIGMOD*. ACM, 2021, 8:1–8:11. DOI: 10.1145/3461837.3464516.
- [52] David Mizell, Kristyn J. Maschhoff, and Steven P. Reinhardt. “Extending SPARQL with graph functions”. In: *BigData*. IEEE Computer Society, 2014, pp. 46–53. DOI: 10.1109/BigData.2014.7004371.
- [53] Guido Moerkotte. “Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing”. In: *PVLDB*. 1998, pp. 476–487. URL: <http://www.vldb.org/conf/1998/p476.pdf>.
- [54] Mohamed Morsey et al. “DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data”. In: *ISWC*. 2011, pp. 454–469. DOI: 10.1007/978-3-642-25073-6\_29.
- [55] Seth A. Myers and Jure Leskovec. “The bursty dynamics of the Twitter information network”. In: *WWW*. ACM, 2014, pp. 913–924. DOI: 10.1145/2566486.2568043.
- [56] Lifeng Nai et al. “GraphBIG: Understanding graph computing in the context of industrial solutions”. In: *SC*. 2015, 69:1–69:12. DOI: 10.1145/2807591.2807626.
- [57] Thomas Neumann and Guido Moerkotte. “A Framework for Reasoning about Share Equivalence and Its Integration into a Plan Generator”. In: *BTW*. 2009, pp. 7–26. URL: <http://subs.emis.de/LNI/Proceedings/Proceedings144/article5220.html>.
- [58] Hung Q. Ngo, Christopher Ré, and Atri Rudra. “Skew strikes back: New developments in the theory of join algorithms”. In: *SIGMOD Rec.* 42.4 (2013), pp. 5–16. DOI: 10.1145/2590989.2590991.
- [59] Dan Olteanu and Maximilian Schleich. “Factorized Databases”. In: *SIGMOD Rec.* 45.2 (2016), pp. 5–16. DOI: 10.1145/3003665.3003667.
- [60] Anil Pacaci et al. “Do We Need Specialized Graph Databases? Benchmarking Real-Time Social Networking Applications”. In: *GRADES at SIGMOD*. 2017, 12:1–12:7. DOI: 10.1145/3078447.3078459.
- [61] Andrew Pavlo and Matthew Aslett. “What’s Really New with NewSQL?” In: *SIGMOD Rec.* (2016). DOI: 10.1145/3003665.3003674.
- [62] Minh-Duc Pham, Peter A. Boncz, and Orri Erling. “S3G2: A Scalable Structure-Correlated Social Graph Generator”. In: *TPCTC*. Vol. 7755. Springer, 2012, pp. 156–172. DOI: 10.1007/978-3-642-36727-4\_11.

- [63] Arnau Prat-Pérez. “LDBC SNB Datagen: Under the hood”. In: *9th LDBC TUC Meeting*. 2017. URL: <https://ldbouncil.org/event/ninth-tuc-meeting/attachments/59277315/75431942.pdf>.
- [64] David Püroja. “LDBC Social Network Benchmark Interactive v2”. <https://ldbouncil.org/docs/papers/msc-thesis-david-puroja-snb-interactive-v2-2023.pdf>. Master’s thesis. Universiteit van Amsterdam, 2023.
- [65] David Püroja et al. “The LDBC Social Network Benchmark Interactive workload v2: A transactional graph query benchmark with deep delete operations”. In: *CoRR* abs/2307.04820 (2023). DOI: 10.48550/arXiv.2307.04820.
- [66] Francois Raab. “Auditing”. In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Y. Zomaya. Springer, 2019. DOI: 10.1007/978-3-319-63962-8\_125-1.
- [67] Mark Raasveldt and Hannes Mühleisen. “Don’t Hold My Data Hostage - A Case For Client Protocol Redesign”. In: *Proc. VLDB Endow.* 10.10 (2017), pp. 1022–1033. DOI: 10.14778/3115404.3115408.
- [68] Mark Raasveldt and Hannes Mühleisen. “DuckDB: An Embeddable Analytical Database”. In: *SIGMOD*. ACM, 2019, pp. 1981–1984. DOI: 10.1145/3299869.3320212.
- [69] Mark Raasveldt et al. “Fair Benchmarking Considered Difficult: Common Pitfalls In Database Performance Testing”. In: *DBTest at SIGMOD*. ACM, 2018, 2:1–2:6. DOI: 10.1145/3209950.3209955.
- [70] Oskar van Rest et al. “PGQL: a property graph query language”. In: *GRADES at SIGMOD*. 2016. DOI: 10.1145/2960414.2960421.
- [71] Liam Roditty. “Decremental maintenance of strongly connected components”. In: *SODA*. SIAM, 2013, pp. 1143–1150. DOI: 10.1137/1.9781611973105.82.
- [72] Liam Roditty and Uri Zwick. “On Dynamic Shortest Paths Problems”. In: *ESA*. Vol. 3221. Lecture Notes in Computer Science. Springer, 2004, pp. 580–591. DOI: 10.1007/978-3-540-30140-0\_52.
- [73] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003. ISBN: 978-0-89871-534-7. DOI: 10.1137/1.9780898718003.
- [74] Michael Schmidt et al. “SP<sup>2</sup>Bench: A SPARQL Performance Benchmark”. In: *Semantic Web Information Management - A Model-Based Perspective*. Springer, 2009, pp. 371–393. DOI: 10.1007/978-3-642-04329-1\_16.
- [75] Oszkár Semeráth et al. “Formal validation of domain-specific languages with derived features and well-formedness constraints”. In: *Softw. Syst. Model.* 16.2 (2017), pp. 357–392. DOI: 10.1007/s10270-015-0485-x.
- [76] Mirko Spasic, Milos Jovanovik, and Arnau Prat-Pérez. “An RDF Dataset Generator for the Social Network Benchmark with Real-World Coherence”. In: *BLINK at ISWC*. 2016. URL: <http://ceur-ws.org/Vol-1700/paper-02.pdf>.
- [77] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. “NetworKit: A tool suite for large-scale complex network analysis”. In: *Netw. Sci.* 4.4 (2016), pp. 508–530. DOI: 10.1017/nws.2016.20.
- [78] Michael Stonebraker et al. “The End of an Architectural Era (It’s Time for a Complete Rewrite)”. In: *VLDB*. ACM, 2007, pp. 1150–1160. URL: <http://www.vldb.org/conf/2007/papers/industrial/p1150-stonebraker.pdf>.
- [79] Gábor Szárnyas. *LDBC Social Network Benchmark graphs*. <https://hdl.handle.net/11112/e6e00558-a2c3-9214-473e-04a16de09bf8>. DOI: 10.25606/SURF.8f3ac424d6694282.
- [80] Gábor Szárnyas et al. “The LDBC Social Network Benchmark: Business Intelligence Workload”. In: *Proc. VLDB Endow.* 16.4 (2022), pp. 877–890. URL: <https://ldbouncil.org/docs/papers/ldbc-snb-bi-vldb-2022.pdf>.
- [81] Gábor Szárnyas et al. “The Linked Data Benchmark Council (LDBC): Driving competition and collaboration in the graph data management space”. In: *CoRR* abs/2307.04350 (2023). DOI: 10.48550/arXiv.2307.04350.

- [82] Gábor Szárnyas et al. “The Train Benchmark: Cross-technology performance evaluation of continuous model queries”. In: *Softw. Syst. Model.* 17.4 (2018), pp. 1365–1393. DOI: 10.1007/s10270-016-0571-8.
- [83] TPC (Transaction Processing Performance Council). *TPC Benchmark C, revision 5.11*. 2010. URL: [http://www.tpc.org/tpc\\_documents\\_current\\_versions/pdf/tpc-c\\_v5.11.0.pdf](http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf).
- [84] TPC (Transaction Processing Performance Council). “TPC Benchmark H, revision 2.18.0”. In: (2017), pp. 1–138. URL: [http://www.tpc.org/tpc\\_documents\\_current\\_versions/pdf/tpc-h\\_v2.18.0.pdf](http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf).
- [85] TPC (Transaction Processing Performance Council). *TPC Pricing Specification, revision 2.9.0*. 2023. URL: [https://www.tpc.org/tpc\\_documents\\_current\\_versions/pdf/TPC-Pricing\\_v2.9.0.pdf](https://www.tpc.org/tpc_documents_current_versions/pdf/TPC-Pricing_v2.9.0.pdf).
- [86] Johan Ugander et al. “The Anatomy of the Facebook Social Graph”. In: *CoRR* abs/1111.4503 (2011).
- [87] Todd L. Veldhuizen. “Leapfrog Triejoin: a worst-case optimal join algorithm”. In: *CoRR* abs/1210.0481 (2012). URL: <http://arxiv.org/abs/1210.0481>.
- [88] Jack Waudby et al. “Preserving Reciprocal Consistency in Distributed Graph Databases”. In: *PaPoC at EuroSys*. ACM, 2020. DOI: 10.1145/3380787.3393675.
- [89] Jack Waudby et al. “Supporting Dynamic Graphs and Temporal Entity Deletions in the LDBC Social Network Benchmark’s Data Generator”. In: *GRADES-NDA at SIGMOD*. ACM, 2020, 8:1–8:8. DOI: 10.1145/3398682.3399165.
- [90] Jack Waudby et al. “Towards Testing ACID Compliance in the LDBC Social Network Benchmark”. In: *TPCTC*. Ed. by Raghunath Nambiar and Meikel Poess. Vol. 12752. Lecture Notes in Computer Science. Springer, 2020, pp. 1–17. DOI: 10.1007/978-3-030-84924-5\_1.



## A CHOKE POINTS

### Introduction

Choke points capture particularly challenging aspects of queries. The correlations between choke points and read queries are displayed in Table A.1.

	1.1	1.2	1.3	1.4	2.1	2.2	2.3	2.4	2.5	2.6	3.1	3.2	3.3	4.1	4.2	4.3	5.1	5.2	5.3	6.1	7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8	8.1	8.2	8.3	8.4	8.5	8.6	
BI 1		⊗									⊗			⊗	⊗																			⊗	
BI 2								⊗			⊗	⊗		⊗	⊗	⊗			⊗	⊗											⊗			⊗	
BI 3	⊗	⊗	⊗		⊗	⊗		⊗					⊗						⊗	⊗											⊗				
BI 4		⊗	⊗		⊗	⊗	⊗	⊗					⊗						⊗	⊗											⊗		⊗		
BI 5		⊗					⊗			⊗																						⊗			
BI 6		⊗					⊗			⊗			⊗							⊗											⊗				
BI 7				⊗									⊗					⊗												⊗					
BI 8		⊗			⊗		⊗					⊗						⊗													⊗		⊗	⊗	
BI 9		⊗				⊗	⊗			⊗		⊗							⊗				⊗	⊗	⊗					⊗			⊗		
BI 10		⊗	⊗				⊗	⊗		⊗			⊗						⊗			⊗	⊗	⊗						⊗				⊗	
BI 11							⊗		⊗			⊗																							
BI 12	⊗	⊗		⊗						⊗		⊗			⊗	⊗															⊗	⊗	⊗	⊗	⊗
BI 13		⊗			⊗		⊗	⊗		⊗		⊗	⊗		⊗		⊗		⊗												⊗		⊗	⊗	
BI 14			⊗	⊗	⊗		⊗				⊗		⊗				⊗	⊗	⊗													⊗	⊗		
BI 15		⊗			⊗	⊗		⊗				⊗					⊗		⊗			⊗	⊗			⊗	⊗			⊗	⊗	⊗	⊗	⊗	
BI 16																			⊗													⊗	⊗		
BI 17					⊗		⊗		⊗	⊗																					⊗				
BI 18								⊗	⊗																						⊗				
BI 19													⊗													⊗	⊗					⊗		⊗	
BI 20												⊗														⊗	⊗	⊗				⊗		⊗	
IC 1					⊗								⊗						⊗												⊗				
IC 2	⊗					⊗	⊗					⊗																						⊗	
IC 3					⊗						⊗							⊗														⊗		⊗	
IC 4							⊗						⊗																					⊗	
IC 5						⊗							⊗																					⊗	
IC 6																		⊗													⊗				
IC 7						⊗	⊗						⊗					⊗												⊗		⊗			
IC 8							⊗						⊗						⊗															⊗	
IC 9	⊗	⊗				⊗	⊗					⊗	⊗																					⊗	
IC 10						⊗						⊗	⊗	⊗			⊗	⊗		⊗	⊗													⊗	
IC 11			⊗				⊗	⊗					⊗		⊗																				
IC 12													⊗										⊗	⊗							⊗				
IC 13													⊗										⊗	⊗		⊗			⊗	⊗				⊗	
IC 14v1													⊗					⊗				⊗	⊗		⊗		⊗		⊗	⊗	⊗			⊗	
IC 14v2													⊗					⊗									⊗	⊗	⊗	⊗	⊗	⊗	⊗		⊗

Table A.1: Coverage of choke points by queries.

### A.1 Aggregation Performance

#### CP-1.1: [QOPT] Interesting orders

TPC-H 1.2

This choke point tests the ability of the query optimizer to exploit the interesting orders induced by some operators. Apart from clustered indices providing key order, other operators also preserve or even induce tuple orderings. Sort-based operators create new orderings, typically the probe-side of a hash join conserves its order, etc.

**Queries** BI 3 BI 12 IC 2 IC 9

**CP-1.2: [QEXE] High cardinality group-by performance**

TPC-H 1.1

This choke point tests the ability of the execution engine to parallelize group-by operations with a large number of groups. Some queries require performing large group-by operations. In such a case, if an aggregation produces a significant number of groups, intra-query parallelization can be exploited as each thread may make its own partial aggregation. Then, to produce the result, these have to be re-aggregated. In order to avoid this, the tuples entering the aggregation operator may be partitioned by a hash of the grouping key and be sent to the appropriate partition. Each partition would have its own thread so that only that thread would write the aggregation, hence avoiding costly critical sections as well. A high cardinality distinct modifier in a query is a special case of this choke point. It is amenable to the same solution with intra-query parallelization and partitioning as the group-by. We further note that scale-out systems have an extra incentive for partitioning since this will distribute the CPU and memory pressure over multiple machines, yielding better platform utilization and scalability.

**Queries** BI 1 BI 3 BI 4 BI 5 BI 6 BI 8 BI 9 BI 10 BI 12 BI 13 BI 15 IC 9

**CP-1.3: [QOPT] Top-k pushdown**

This choke point tests the ability of the query optimizer to perform optimizations based on top- $k$  selections. Many times queries demand for returning the top- $k$  elements based on some property. Engines can exploit that once  $k$  results are obtained, extra restrictions in a selection can be added based on the properties of the  $k$ th element currently in the top- $k$ , being more restrictive as the query advances, instead of sorting all elements and picking the highest  $k$ .

**Queries** BI 3 BI 4 BI 10 BI 14 IC 11

**CP-1.4: [QEXE] Low cardinality group-by performance**

TPC-H 1.3

This choke point tests the ability to efficiently perform group-by evaluation when only a very limited set of groups is available. This can require special strategies for parallelization, e.g. pre-aggregation when possible. This case also allows using special strategies for grouping like using array lookup if the domain of keys is small.

**Queries** BI 7 BI 12 BI 14

**A.2 Join Performance****CP-2.1: [QOPT] Rich join order optimization**

TPC-H 2.3

This choke point tests the ability of the query optimizer to find optimal join orders. A graph can be traversed in different ways. In the relational model, this is equivalent to different join orders. The execution time of these orders may differ by orders of magnitude. Therefore, finding an efficient join (traversal) order is important, which in general, requires enumeration of all the possibilities. The enumeration is complicated by operators that are not freely re-orderable like semi-, anti-, and outer-joins. Because of this difficulty most join enumeration algorithms do not enumerate all possible plans, and therefore can miss the optimal join order. Therefore, this choke point tests the ability of the query optimizer to find optimal join (traversal) orders.

**Queries** BI 3 BI 4 BI 8 BI 13 BI 14 BI 15 BI 17 IC 1 IC 3

**CP-2.2: [QOPT] Late projection**

TPC-H 2.4

This choke point tests the ability of the query optimizer to delay the projection of unneeded attributes until late in the execution. Queries where certain columns are only needed late in the query. In such a situation, it is better to omit them from initial table scans, as fetching them later by row-id with a separate scan operator, which is

joined to the intermediate query result, can save temporal space, and therefore I/O. Late projection does have a trade-off involving locality, since late in the plan the tuples may be in a different order, and scattered I/O in terms of tuples/second is much more expensive than sequential I/O. Late projection specifically makes sense in queries where the late use of these columns happens at a moment where the amount of tuples involved has been considerably reduced; for example after an aggregation with only few unique group-by keys or a top- $k$  operator.

**Queries** BI 3 BI 4 BI 9 BI 15 IC 2 IC 7 IC 9

### CP-2.3: [QOPT] Join type selection

This choke point tests the ability of the query optimizer to select the proper join operator type, which implies accurate estimates of cardinalities. Depending on the cardinalities of both sides of a join, a hash or an index-based join operator is more appropriate. This is especially important with column stores, where one usually has an index on everything. Deciding to use a hash join requires a good estimation of cardinalities on both the probe and build sides. In TPC-H, the use of hash join is almost a foregone conclusion in many cases, since an implementation will usually not even define an index on foreign key columns. There is a break even point between index and hash based plans, depending on the cardinality on the probe and build sides.

**Queries** BI 4 BI 5 BI 6 BI 8 BI 9 BI 10 BI 11 BI 13 BI 17 IC 2 IC 4 IC 5 IC 7  
IC 9 IC 10 IC 11

### CP-2.4: [QOPT] Sparse foreign key joins

TPC-H 2.2

This choke point tests the performance of join operators when the join is sparse. Sometimes joins involve relations where only a small percentage of rows in one of the tables is required to satisfy a join. When tables are larger, typical join methods can be sub-optimal. Partitioning the sparse table, using Hash Clustered indices or implementing Bloom-filter tests inside the join are techniques to improve the performance in such situations [28].

**Queries** BI 2 BI 3 BI 4 BI 10 BI 13 BI 15 IC 8 IC 11

### CP-2.5: [QEXE] Worst-case optimal joins

This choke point tests the query engine's ability to use multi-way, worst-case optimal joins to evaluate cyclic queries which are required to efficiently compute some dense subgraphs such as the triangle, the 4-cycle, and the diamond (4-cycle with a cross-edge). The absence of multi-way joins (e.g. in systems which only support binary joins), implies that join performance will be provably suboptimal for cyclic queries.

**Queries** BI 11 BI 17 BI 18

### CP-2.6: [QEXE] Factorized query execution

Query results produced by many-to-many joins often have redundancies when represented as tuples. Factorization [59] provides a more compact (relational) representation by eliminating repetitions, while still allowing some operations (e.g. aggregation) to be performed without flattening the relation.

**Queries** BI 5 BI 6 BI 9 BI 10 BI 12 BI 13 BI 17 BI 18

## A.3 Data Access Locality

### CP-3.1: [QOPT] Detecting correlation

TPC-H 3.3

This choke point tests the ability of the query optimizer to detect data correlations and exploiting them. If a schema rewards creating clustered indices, the question then is which of the date or data columns to use as key. In fact it should not matter which column is used, as range-propagation between correlated attributes of the same table is relatively easy. One way is through the creation of multi-attribute histograms after detection of attribute correlation. With MinMax indices, range-predicates on any column can be translated into qualifying tuple position ranges. If an attribute value is correlated with tuple position, this reduces the area to scan roughly equally to predicate selectivity.

**Queries** BI 2 BI 14 IC 3

### CP-3.2: [STORAGE] Dimensional clustering

This choke point tests suitability of the identifiers assigned to entities by the storage system to better exploit data locality. A data model where each entity has a unique synthetic identifier, e.g. RDF or graph models, has some choice in assigning a value to this identifier. The properties of the entity being identified may affect this, e.g. type (label), other dependent properties, e.g. geographic location, date, position in a hierarchy, etc., depending on the application. Such identifier choice may create locality which in turn improves efficiency of compression or index access.

**Queries** BI 1 BI 2 BI 8 BI 9 BI 11 BI 12 BI 13 IC 2 IC 9

### CP-3.3: [QEXE] Scattered index access patterns

This choke point tests the performance of indices when scattered accesses are performed. The efficiency of index lookup is very different depending on the locality of keys coming to the indexed access. Techniques like vectoring non-local index accesses by simply missing the cache in parallel on multiple lookups vectored on the same thread may have high impact. Also detecting absence of locality should turn off any locality dependent optimizations if these are costly when there is no locality. A graph neighbourhood traversal is an example of an operation with random access without predictable locality.

**Queries** BI 3 BI 4 BI 6 BI 7 BI 10 BI 13 BI 14 BI 15 BI 19 BI 20 IC 5 IC 7  
IC 8 IC 9 IC 10 IC 11 IC 12 IC 13 IC 14v1 IC 14v2

## A.4 Expression Calculation

### CP-4.1: [QOPT] Common subexpression elimination

TPC-H 4.2a

This choke point tests the ability of the query optimizer to detect common sub-expressions and reuse their results. A basic technique helpful in multiple queries is common subexpression elimination (CSE). CSE should recognize also that `avg` aggregates can be derived afterwards by dividing a `sum` by the `count` when those have been computed.

**Queries** BI 1 BI 2 IC 10

**CP-4.2: [QOPT] Complex boolean expression joins and selections**

TPC-H 4.2d

This choke point tests the ability of the query optimizer to reorder the execution of boolean expressions to improve the performance. Some boolean expressions are complex, with possibilities for alternative optimal evaluation orders. For instance, the optimizer may reorder conjunctions to test first those conditions with larger selectivity [53].

**Queries** BI 1 BI 2 BI 12 BI 13 IC 10 IC 11

**CP-4.3: [QEXE] Low overhead expressions interpretation**

This choke point tests the ability of efficiently evaluating simple expressions on a large number of values. A typical example could be simple arithmetic expressions, mathematical functions like floor and absolute or date functions like extracting a year.

**Queries** BI 2 BI 12

**A.5 Correlated Sub-Queries****CP-5.1: [QOPT] Flattening sub-queries**

TPC-H 5.1

This choke point tests the ability of the query optimizer to flatten execution plans when there are correlated sub-queries. Many queries have correlated sub-queries and their query plans can be flattened, such that the correlated sub-query is handled using an equi-join, outer-join or anti-join. In TPC-H Q21, for instance, there is an **EXISTS** clause (for orders with more than one supplier) and a **NOT EXISTS** clause (looking for an item that was received too late). To execute this query well, systems need to flatten both sub-queries, the first into an equi-join plan, the second into an anti-join plan. Therefore, the execution layer of the database system will benefit from implementing these extended join variants.

The ill effects of repetitive tuple-at-a-time sub-query execution can also be mitigated if execution systems by using vectorized, or blockwise query execution, allowing to run sub-queries with thousands of input parameters instead of one. The ability to look up many keys in an index in one API call creates the opportunity to benefit from physical locality, if lookup keys exhibit some clustering.

**Queries** BI 13 BI 14 BI 15 IC 3 IC 6 IC 7 IC 10

**CP-5.2: [QEXE] Overlap between outer and sub-query**

TPC-H 5.3

This choke point tests the ability of the execution engine to reuse results when there is an overlap between the outer query and the sub-query. In some queries, the correlated sub-query and the outer query have the same joins and selections. In this case, a non-tree, rather DAG-shaped [57] query plan would allow to execute the common parts just once, providing the intermediate result stream to both the outer query and correlated sub-query, which higher up in the query plan are joined together (using normal query decorrelation rewrites). As such, the benchmark rewards systems where the optimizer can detect this and the execution engine supports an operator that can buffer intermediate results and provide them to multiple parent operators.

**Queries** BI 7 BI 14 IC 10

**CP-5.3: [QEXE] Intra-query result reuse**

TPC-H 5.2

This choke point tests the ability of the execution engine to reuse sub-query results when two sub-queries are mostly identical. Some queries have almost identical sub-queries, where some of their internal results can be reused in both sides of the execution plan, thus avoiding to repeat computations.

**Queries** BI 2 BI 4 BI 8 BI 10 BI 13 BI 14 BI 15 BI 16 IC 1 IC 8 IC 14v1 IC 14v2

## A.6 Parallelism and Concurrency

### CP-6.1: [QEXE] Inter-query result reuse

TPC-H 6.3

This choke point tests the ability of the query execution engine to reuse results from different queries. Sometimes with a high number of streams a significant amount of identical queries emerge in the resulting workload. The reason is that certain parameters, as generated by the workload generator, have only a limited amount of parameters bindings. This weakness opens up the possibility of using a query result cache, to eliminate the repetitive part of the workload. A further opportunity that detects even more overlap is the work on recycling, which does not only cache final query results, but also intermediate query results of a “high worth”. Here, worth is a combination of partial-query result size, partial-query evaluation cost, and observed (or estimated) frequency of the partial-query in the workload.

**Queries** BI 2 BI 4 BI 6 IC 10

## A.7 Graph Specifics

### CP-7.1: [QEXE] Incremental path computation

This choke point tests the ability of the execution engine to reuse work across graph traversals. For example, when computing paths within a range of distances, it is often possible to incrementally compute longer paths by reusing paths of shorter distances that were already computed.

**Queries** BI 10 IC 10

### CP-7.2: [QOPT] Cardinality estimation of transitive paths

This choke point tests the ability of the query optimizer to properly estimate the cardinality of intermediate results when executing transitive paths. A transitive path may occur in a “fact table” or a “dimension table” position. A transitive path may cover a tree or a graph, e.g. descendants in a geographical hierarchy vs. graph neighbourhood or transitive closure in a many-to-many connected social network. In order to decide proper join order and type, the cardinality of the expansion of the transitive path needs to be correctly estimated. This could for example take the form of executing on a sample of the data in the cost model or of gathering special statistics, e.g. the depth and fan-out of a tree. In the case of hierarchical dimensions, e.g. geographic locations or other hierarchical classifications, detecting the cardinality of the transitive path will allow one to go to a star schema plan with scan of a fact table with a selective hash join. Such a plan will be on the other hand very bad for example if the hash table is much larger than the “fact table” being scanned.

**Queries** BI 9 BI 10 BI 15 IC 12 IC 13 IC 14v1

### CP-7.3: [QEXE] Execution of a transitive step

This choke point tests the ability of the query execution engine to efficiently execute transitive steps. Graph workloads may have transitive operations, for example finding a shortest path between nodes. This involves repeated execution of a short lookup, often on many values at the same time, while usually having an end condition, e.g. the target node being reached or having reached the border of a search going in the opposite direction. For the best efficiency, these operations can be merged or tightly coupled to the index operations themselves. Also parallelization may be possible but may need to deal with a global state, e.g. set of visited nodes. There are many possible tradeoffs between generality and performance.

**Queries** BI 9 BI 10 BI 15 IC 12 IC 13 IC 14v1

#### CP-7.4: [QEXE] Efficient evaluation of termination criteria for transitive queries

This tests the ability of a system to express termination criteria for transitive queries so that not the whole transitive relation has to be evaluated as well as efficient testing for termination.

**Queries** BI 9

#### CP-7.5: [QEXE] Unweighted shortest paths

A common problem in graph queries is determining the distance between a node and a set of nodes. To compute the distance values, systems may employ BFS or a single-source shortest path algorithm with uniform weights. To compute the distance between two given node, systems can use bidirectional search algorithms.

**Queries** IC 13 IC 14v1

#### CP-7.6: [QEXE] Cheapest paths (weighted shortest paths)

Computing *cheapest paths* (weighted shortest paths) is a fundamental problem in graph queries. While there are well-known algorithms to compute it, e.g. Dijkstra's algorithm, Bellman–Ford, and delta-stepping [49], system often use naïve approaches such as enumerating all paths which makes these queries unnecessarily complex.

**Queries** BI 15 BI 19 BI 20 IC 14v2

#### CP-7.7: [QEXE] Composition of graph queries

In many cases, it is desirable to specify multiple graph queries, where the first one defines an induced subgraph or an overlay graph on the original graph, which is then passed two the next query, and so on. Expressing such computations as a sequence of composable graph queries would be desirable from both usability, optimization, and execution aspects. However, currently many graph dabases lack support for composable graph queries.

The G-CORE [5] design language tackled problem this by introducing the *path property graph* data model (consisting of nodes, edges, and paths) and defining queries such that they return a graph (while also providing means to return a tabular output).

**Queries** BI 15 BI 19 BI 20 IC 14v1 IC 14v2

#### CP-7.8: [QEXE] Reachability between disconnected components

For path finding queries, the result is often that the specified path does not exist in the graph. For example, for a single-source single-destination search, when one of the endpoints is in a small component (e.g. the endpoint is an isolated node), systems using a bidirectional search algorithm can quickly determine that there is no path to be found.

**Queries** BI 20 IC 13 IC 14v2

## A.8 Language Features

### CP-8.1: [LANG] Complex patterns

**Description.** A natural requirement for graph query systems is to be able to express complex graph patterns.



**Transitive edges.** Transitive closure-style computations are common in graph query systems, both with fixed bounds (e.g. get nodes that can be reached through at least 3 and at most 5 knows edges), and without fixed bounds (e.g. get all Messages that a Comment replies to).

**Negative edge conditions.** Some queries define *negative pattern conditions*. For example, the condition that a certain Message does not have a certain Tag is represented in the graph as the absence of a hasTag edge between the two nodes. Thus, queries looking for cases where this condition is satisfied check for negative patterns, also known as negative application conditions (NACs) in graph transformation literature [36].

**Queries** BI 7 BI 9 BI 10 BI 12 BI 15 BI 17 BI 18 IC 7 IC 13 IC 14v1 IC 14v2

### CP-8.2: [LANG] Complex aggregations

**Description.** BI workloads are heavy on aggregation, including queries with *subsequent aggregations*, where the results of an aggregation serves as the input of another aggregation. Expressing such operations requires some sort of query composition or chaining (see also CP-8.4). It is also common to *filter on aggregation results* (similarly to the **HAVING** keyword of SQL).

**Queries** BI 2 BI 3 BI 4 BI 5 BI 6 BI 8 BI 12 BI 13 BI 15 IC 1 IC 3 IC 4 IC 5  
IC 6 IC 12 IC 14v1 IC 14v2

### CP-8.3: [LANG] Ranking-style queries

**Description.** Along with aggregations, BI workloads often use *window functions*, which perform aggregations without grouping input tuples to a single output tuple. A common use case for windowing is *ranking*, i.e. selecting the top element with additional values in the tuple (nodes, edges or attributes).<sup>1</sup>

**Queries** BI 12 BI 14 BI 15 IC 7 IC 14v1 IC 14v2

### CP-8.4: [LANG] Query composition

**Description.** Numerous use cases require *composition* of queries, including the reuse of query results (e.g. nodes, edges) or using scalar subqueries (e.g. selecting a threshold value with a subquery and using it for subsequent filtering operations).

**Queries** BI 4 BI 8 BI 12 BI 13 BI 14 BI 15 BI 16 BI 19 BI 20

### CP-8.5: [LANG] Dates and times

**Description.** Handling dates and times is a fundamental requirement for production-ready database systems. It is particularly important in the context of BI queries as these often calculate aggregations on certain periods of time (e.g. on entities created during the course of a month).

**Queries** BI 1 BI 2 BI 8 BI 9 BI 12 BI 13 BI 15 BI 16 IC 2 IC 3 IC 4 IC 5 IC 9

<sup>1</sup>PostgreSQL defines the **OVER** keyword to use aggregation functions as window functions, and the **rank()** function to produce numerical ranks, see <https://www.postgresql.org/docs/9.1/static/tutorial-window.html> for details.



**CP-8.6: [LANG] Handling paths**

**Description.** Handling paths as first-class citizens is one of the key distinguishing features of graph database systems [5]. Hence, additionally to reachability-style checks, a language should be able to express queries that operate on elements of a path, e.g. calculate a score on each edge of the path. Also, some use cases specify uniqueness constraints on paths [4]: *arbitrary path*, *shortest path*, *no-repeated-node semantics* (also known as *simple paths*), and *no-repeated-edge semantics* (also known as *trails*). Other variants are also used in rare cases, such as *maximal* (non-expandable) or *minimal* (non-contractable) paths.

**Note on terminology.** The *Glossary of graph theory terms* page of Wikipedia<sup>2</sup> defines *paths* as follows: “A path may either be a walk (a sequence of nodes and edges, with both endpoints of an edge appearing adjacent to it in the sequence) or a simple path (a walk with no repetitions of nodes or edges), depending on the source.” In this work, we use the first definition, which is more common in modern graph database systems and is also followed in a recent survey on graph query languages [4].

**Queries** BI 10 BI 15 BI 19 BI 20 IC 10 IC 13 IC 14v1 IC 14v2

**A.9 Update Operations****CP-9.1: [UPD] Insert node**

This choke point tests the ability of the database to insert a node.

**Queries** INS 1 INS 4 INS 5 INS 6 INS 7

**CP-9.2: [UPD] Insert edge**

This choke point tests the ability of the database to insert an edge.

**Queries** INS 1 INS 2 INS 3 INS 4 INS 5 INS 6 INS 7 INS 8

**CP-9.3: [UPD] Delete node**

This choke point tests the ability of the database to delete a node.

**Queries** DEL 1 DEL 4 DEL 6 DEL 7

**CP-9.4: [UPD] Delete edge**

This choke point tests the ability of the database to delete an edge.

**Queries** DEL 1 DEL 2 DEL 3 DEL 4 DEL 5 DEL 6 DEL 7 DEL 8

**CP-9.5: [UPD] Delete recursively**

This choke point tests the ability of the database to recursively perform a delete operation, e.g. delete an entire message thread.

**Queries** DEL 1 DEL 4 DEL 6 DEL 7

<sup>2</sup>[https://en.wikipedia.org/wiki/Glossary\\_of\\_graph\\_theory\\_terms](https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms)

## B SCALE FACTOR STATISTICS

## B.1 Number of Entities for SNB Interactive v1.0

C	File	SF0.1	SF0.3	SF1	SF3	SF10	SF30	SF100	SF300	SF1000	SF3000
N	organisation	7955	7955	7955	7955	7955	7955	7955	7955	7955	7955
E	organisation_isLocatedIn_place	7955	7955	7955	7955	7955	7955	7955	7955	7955	7955
N	place	1460	1460	1460	1460	1460	1460	1460	1460	1460	1460
E	place_isPartOf_place	1454	1454	1454	1454	1454	1454	1454	1454	1454	1454
N	tag	16080	16080	16080	16080	16080	16080	16080	16080	16080	16080
E	tag_hasType_tagclass	16080	16080	16080	16080	16080	16080	16080	16080	16080	16080
N	tagclass	71	71	71	71	71	71	71	71	71	71
E	tagclass_isSubclassOf_tagclass	70	70	70	70	70	70	70	70	70	70
N	comment	203354	682061	2581736	7882971	26540464	80390821	261475982	767719169	2550634137	6273573790
E	comment_hasCreator_person	203354	682061	2581736	7882971	26540464	80390821	261475982	767719169	2550634137	6273573790
E	comment_hasTag_tag	232524	807266	3145443	9688491	32922873	100818244	330756583	975122821	3253337649	8344430563
E	comment_isLocatedIn_place	203354	682061	2581736	7882971	26540464	80390821	261475982	767719169	2550634137	6273573790
E	comment_replyOf_comment	103552	346553	1310385	3997838	13465094	40789548	132671059	389555963	1294311108	3182724110
E	comment_replyOf_post	99802	335508	1271351	3885133	13075370	39601273	128804923	378163206	1256323029	3090849680
N	forum	16818	38050	110347	271226	727502	1835458	4982966	12560110	36086326	75972136
E	forum_containerOf_post	168873	404531	1237554	3200561	9119229	24346116	70420477	188400071	575768804	1297229130
E	forum_hasMember_person	266965	861079	3345548	10352102	35510056	110335311	362933964	1070304327	3570974603	5564031246
E	forum_hasModerator_person	16818	38050	110347	271226	727502	1835458	4982966	12560110	36086326	75972136
E	forum_hasTag_tag	54288	124186	354943	878307	2364249	5941428	16147466	40642813	116757400	262462978
N	person	1700	3900	11000	27000	73000	184000	499000	1254000	3600000	8324653
A	person_email_emailaddress	3690	8393	23372	57419	155585	392497	1064135	2675881	7681772	17762797
E	person_hasInterest_tag	39170	90036	255596	634081	1709747	4289970	11663500	29336703	84271074	194815501
E	person_isLocatedIn_place	1700	3900	11000	27000	73000	184000	499000	1254000	3600000	8324653
E	person_knows_person	18074	57179	226515	704246	2431407	7514541	24842767	73448777	245296255	594161768
E	person_likes_comment	96865	412010	1946260	6868912	25596818	84821954	301042048	947303146	3357196350	8510597029
E	person_likes_post	97638	328473	1303778	4120299	14228924	44582924	149809880	451827331	1540438666	3282073444
A	personSpeaks_language	3771	8595	24246	59609	160992	405234	1099519	2763100	7933284	18342929
E	person_studyAt_organisation	1337	3089	8808	21586	58439	147527	399487	1003543	2880284	6659035
E	person_workAt_organisation	3732	8561	24079	58912	159511	401230	1086041	2730945	7836570	18114611
N	post	168873	404531	1237554	3200561	9119229	24346116	70420477	188400071	575768804	1297229130
E	post_hasCreator_person	168873	404531	1237554	3200561	9119229	24346116	70420477	188400071	575768804	1297229130
E	post_hasTag_tag	59862	207814	816048	2521635	8584195	26346801	86600144	255541805	852679225	2206210705
E	post_isLocatedIn_place	168873	404531	1237554	3200561	9119229	24346116	70420477	188400071	575768804	1297229130
Total nodes		416311	1154108	3966203	11407324	36485761	106781961	337403991	969958916	3166114833	7655125275
Total edges		2031213	6226978	23031794	69422952	231371359	701455758	2286478782	6729459600	22450588784	51780287988

Table B.1: The number of entities per SF and per file in the Interactive workload (produced by the Hadoop-based generator and measured based on the output of the CsvBasic serializer). To derive these numbers, 100% of the network was generated as an initial bulk data set with no update streams. Notation – C: entity category, N: node, E: edge.

## B.2 Number of Entities for SNB BI v1.0

## B.3 Factor Tables

C	File	SF1	SF3	SF10	SF30	SF100	SF300	SF1 000	SF3 000	SF10 000	SF30 000
N	Organisation	7955	7955	7955	7955	7955	7955	7955	7955	7955	7955
E	Organisation_isLocatedIn_Place	7955	7955	7955	7955	7955	7955	7955	7955	7955	7955
N	Place	1460	1460	1460	1460	1460	1460	1460	1460	1460	1460
E	Place_isPartOf_Place	1454	1454	1454	1454	1454	1454	1454	1454	1454	1454
N	Tag	16080	16080	16080	16080	16080	16080	16080	16080	16080	16080
E	Tag_hasType_TagClass	16080	16080	16080	16080	16080	16080	16080	16080	16080	16080
N	TagClass	71	71	71	71	71	71	71	71	71	71
E	TagClass_isSubclassOf_TagClass	70	70	70	70	70	70	70	70	70	70
N	Comment	1739438	5343582	18196074	54737515	185495476	554017340	1876785283	5656073047	18880439325	58666958815
E	Comment_hasCreator_Person	1739438	5343582	18196074	54737515	185495476	554017340	1876785283	5656073047	18880439325	58666958815
E	Comment_hasTag_Tag	2176131	6754220	23113520	70035650	238074593	714772017	2426657766	7330444735	24505161117	76236094545
E	Comment_isLocatedIn_Country	1739438	5343582	18196074	54737515	185495476	554017340	1876785283	5656073047	18880439325	58666958815
E	Comment_replyOf_Comment	7890202	2425043	8274158	25130258	85829276	258292038	883936628	2688432865	9045050101	28244723682
E	Comment_replyOf_Post	950418	2918539	9921916	29607257	99666200	294572950	992848655	2967640182	9835389224	30422235133
N	Forum	100827	245524	667545	1659632	4611436	11642786	33168124	87364322	257338738	728629666
E	Forum_containerOf_Post	1121226	2873419	8273491	21651342	64029217	171283445	519738978	1440235348	4461342990	13148296221
E	Forum_hasMember_Person	2909768	8780738	30201123	90198118	303838931	898932504	3004740356	8909683066	29398116490	90652090014
E	Forum_hasModerator_Person	100827	245524	667545	1659632	4611436	11642786	33168124	87364322	257338738	728629666
E	Forum_hasTag_Tag	328584	809991	2207525	5467942	15195472	38372330	109341702	288057168	848359157	2401607343
N	Person	10295	25066	68673	170654	473001	1193579	3399580	8955552	26384952	74689437
E	Person_hasInterest_Tag	238052	589533	1608653	3978964	11057039	27923123	79573188	209648434	617405426	1747667501
E	Person_isLocatedIn_City	10295	25066	68673	170654	473001	1193579	3399580	8955552	26384952	74689437
E	Person_knows_Person	173014	528896	1839354	5524302	18655515	55656915	187247788	559360185	1854528925	5734470022
E	Person_likes_Comment	1109813	3826649	14586377	48651549	184325690	605620715	2249224980	7279159053	25779776654	83352563279
E	Person_likes_Post	760455	2417873	8546995	26908834	98423296	314778935	1140808487	3619661715	12593759314	40072928363
E	Person_studyAt_University	8309	20113	55066	136614	378582	955425	2719877	7165145	21108848	59758459
E	Person_workAt_Company	22044	54135	149581	371634	1029492	2598384	7398286	19491928	57416114	162518922
N	Post	1121226	2873419	8273491	21651342	64029217	171283445	519738978	1440235348	4461342990	13148296221
E	Post_hasCreator_Person	1121226	2873419	8273491	21651342	64029217	171283445	519738978	1440235348	4461342990	13148296221
E	Post_hasTag_Tag	751933	2305927	7865279	23426338	78380259	231621916	769380657	2273989086	7454473533	22896875734
E	Post_isLocatedIn_Country	1121226	2873419	8273491	21651342	64029217	171283445	519738978	1440235348	4461342990	13148296221
Total nodes		2997352	8513157	27231349	78244709	254634696	738162716	2433117531	719265835	23625531571	72618599705
Total edges		17196776	51035227	170343945	505722361	1703042944	5078844191	17203259133	51881931133	173439201772	539565683952

Table B.2: The number of entities per SF and per file in the *initial data set* used in the BI workload. Notation – C: entity category, N: node, E: edge.

T	C	File	SF1	SF3	SF10	SF30	SF100	SF300	SF1 000	SF3 000	SF10 000	SF30 000
I	N	Comment	652269	1932347	6122166	17233922	53364420	144700167	428355986	1132241525	3323091103	9411625366
I	E	Comment_hasCreator_Person	652269	1932347	6122166	17233922	53364420	144700167	428355986	1132241525	3323091103	9411625366
I	E	Comment_hasTag_Tag	727839	2203748	7079778	20150855	62861828	171071832	508165623	1339365204	3909017913	11014456527
I	E	Comment_isLocatedIn_Country	652269	1932347	6122166	17233922	53364420	144700167	428355986	1132241525	3323091103	9411625366
I	E	Comment_replyOf_Comment	408491	1216510	3872589	10961292	33981364	92048927	272358361	718336661	2102670076	5946782332
I	E	Comment_replyOf_Post	243778	715837	2249577	6272630	19383056	52651240	155997625	413904864	1220421027	3464843043
I	N	Forum	5767	14105	38084	94700	265314	671285	1915909	5047113	14895929	42218181
I	E	Forum_containerOf_Post	71716	182738	507826	1297451	3735615	9741528	28453210	76669773	231949432	671846867
I	E	Forum_hasMember_Person	350924	1050322	3436445	9978585	32960000	93286265	295103572	825253679	2554550825	7479070111
I	E	Forum_hasModerator_Person	5767	14105	38084	94700	265314	671285	1915909	5047113	14895929	42218181
I	E	Forum_hasTag_Tag	13456	31162	86525	214373	592043	1495805	4280777	11235864	33142429	94020588
I	N	Person	325	804	2127	5296	14699	36921	105420	276448	815048	2310563
I	E	Person_hasInterest_Tag	8014	17861	50568	124969	341426	861441	2470258	6465213	19061544	54112770
I	E	Person_isLocatedIn_City	325	804	2127	5296	14699	36921	105420	276448	815048	2310563
I	E	Person_knows_Person	46436	139535	465597	1356282	4461290	12657067	39877751	111602193	347323797	1028845782
I	E	Person_likes_Comment	507078	1642981	5814742	17739535	59010156	170613836	547019411	1522602131	4738606525	14044004355
I	E	Person_likes_Post	84089	242012	781367	2228761	7227562	21174383	69394102	203079530	664408922	2040369359
I	E	Person_studyAt_University	253	642	1711	4215	11684	29520	84408	221160	651833	1848819
I	E	Person_workAt_Company	722	1691	4541	11473	32135	79806	228835	601641	1772442	5025385
I	N	Post	71716	182738	507826	1297451	3735615	9741528	28453210	76669773	231949432	671846867
I	E	Post_hasCreator_Person	71716	182738	507826	1297451	3735615	9741528	28453210	76669773	231949432	671846867
I	E	Post_hasTag_Tag	26578	78669	247471	690212	2192065	6197708	19682903	56322268	180509835	545993292
I	E	Post_isLocatedIn_Country	71716	182738	507826	1297451	3735615	9741528	28453210	76669773	231949432	671846867
Total insert node operations			730077	2129994	6670203	18631369	57380048	155149901	458830525	1214234859	3570751512	10128000977
Total insert edge operations			3943436	11768787	37898932	108193375	341270307	941500954	2858756557	7708806338	23129878647	66602692431
Total insert operations			4673513	13898781	44569135	126824744	398650355	1096650855	3317587082	8923041197	26700630159	76730693408
D	N	Comment	11966	35147	110712	309712	959810	2597282	7704534	20373985	59821497	169401271
D	N	Forum	212	459	1252	3220	8975	22699	64932	172181	506906	1440270
D	E	Forum_hasMember_Person	2004	5002	12857	31647	86820	221834	609738	1565418	4544009	12773538
D	N	Person	54	122	264	510	1265	2827	7285	18234	48251	123926
D	E	Person_knows_Person	5548	16704	57638	168900	560741	1604444	5140980	14599090	46275390	139258201
D	E	Person_likes_Comment	12220	39660	138268	420001	1394595	4040199	12955551	36066934	112313459	332839378
D	E	Person_likes_Post	1992	5869	18835	52070	169649	498070	1634887	4788019	15655650	48054670
D	N	Post	1908	5004	13566	34948	100375	263354	767998	2067056	6267076	1814667
Total delete node operations			14140	40732	125794	348390	1070425	2886162	8544749	22631456	66643730	189107071
Total delete edge operations			21764	67235	227598	672618	2211805	6364547	20341156	57019461	178788508	532925787
Total delete operations			35904	107967	353392	1021008	3282230	9250709	28885905	79650917	245432238	722032858

Table B.3: The number of entities per SF and per file in the *update data sets* used in the BI workload. Notation – T: update type, I: insert, D: delete, C: entity category, N: node, E: edge.

Scale Factor	Size
1	8.6M
3	18M
10	41M
30	100M
100	259M
300	656M
1 000	1.9G
3 000	5.1G
10 000	16G
30 000	47G

Table B.4: The total size of the factor tables.

## C BENCHMARK CHECKLIST

We expect LDBC benchmarks to be used in many scenarios. For most research papers, fully audited results are unrealistic and even unaudited results can provide insight into the performance of the systems under test (SUT). However, we ask authors to include the following information in their papers:

- Were the results cross-validated for at least one scale factor?
- Were the results cross-validated for all scale factors used in the benchmark?
- Does the SUT have a persistent storage?
- Does the SUT provide ACID transactions?
- Does the SUT provide any level of fault-tolerance?
- How many warm-up rounds were performed?
- How many execution rounds were performed?
- How were the execution times summarized?<sup>1</sup>
- Is the loading phase included in the query execution times?<sup>2</sup>
- If the SUT is not your own system, did you contact its developers or experts to help optimizing the queries?<sup>3</sup>

These results will help the reader to put the results in context. For example, a non-ACID compliant, non-fault-tolerant system working on read-only graphs and offering no persistent storage is expected to have significantly better results than a fully-fledged disk-based DBMS.

We also suggest the reader to take a look at the checklist presented in [69].

---

<sup>1</sup>Paper [37] provides an excellent overview on how to summarize benchmark results.

<sup>2</sup>This might be relevant for systems without persistent storage, or systems providing lazy/incremental computation.

<sup>3</sup>For a research prototype tool, the tuning knobs are usually not well documented. Hence, it is worth contacting the tool's authors, who are generally keen to help. For more mature systems (e.g. most established RDBMSs), there is a large body of knowledge available, in the form of books and online forums, which should help your optimization efforts. It is also possible to contact experienced DBAs who can assist with fine tuning the system.

## D LEGACY DATA SETS FOR THE INTERACTIVE WORKLOAD

The Interactive workload uses the legacy version of the data sets. These can be generated using the Hadoop-based Datagen hosted at <https://github.com/ldbc/ldbc-snb-datagen-hadoop/>. This chapter documents these data sets.

The SNB data sets are available in the SURF/CWI LDBC SNB data repository [79] at <https://repository.surfsara.nl/datasets/cwi/ldbc-snb-interactive-v1-datagen-v100>.

- **Serializers:** `csv_basic`, `csv_basic-longdateformatter`, `csv_composite`, `csv_composite-longdateformatter`, `csv_composite_merge_foreign`, `csv_composite_merge_foreign-longdateformatter`, `csv_merge_foreign`, `csv_merge_foreign-longdateformatter`, `ttl`
- **Partition numbers:**  $2^k$  (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024) and  $6 \times 2^k$  (12, 24, 48, 96, 192, 384, 768).

The **key differences from the latest (BI and Interactive v2) data sets** are the following:

- **DateTime values** follow the format `yyyy-mm-ddTHH:MM:ss.sss+0000`, i.e. their offset string is `0000` instead of `00:00`. This implies that they are not compatible with the recommendations of RFC-3339<sup>1</sup>.
- The `Forum-hasModerator-Person` edge type has an *exactly one* cardinality on the `Person`'s end.

### D.1 Output Data

For each scale factor, Datagen produces three different artefacts:

- **Dataset:** The dataset to be bulk loaded by the SUT. In the Interactive workload, it corresponds to roughly the 90% of the total generated network.
- **Update Streams:** A set of update streams containing update queries, which are used by the driver to generate the update queries of the workloads. This update streams correspond to the remaining 10% of the generated dataset.
- **Substitution Parameters:** A set of files containing the different parameter bindings that will be used by the driver to generate the read queries of the workloads.

#### D.1.1 Scale Factors

LDBC SNB defines a set of scale factors (SFs), targeting systems of different sizes and budgets. SFs are computed based on the ASCII size in Gibibytes of the generated output files using the `CsvMergeForeign` serializer (see Section D.1.2) and default settings, i.e. both the 90% initial data and the 10% update streams count towards the total size. For example, SF1 takes roughly 1 GiB in CSV format, SF3 weighs roughly 3 GiB and so on and so forth. It is important to note that for a given scale factor, data sets generated using different serializers contain exactly the same data, the only difference is in how they are represented.<sup>2</sup> The provided SFs are the following: 1, 3, 10, 30, 100, 300, 1 000, 3 000. Additionally, two small data sets, 0,1, and 0,3 are provided to help initial validation efforts.

The Test Sponsor may select the SF that better fits their needs, by properly configuring the Datagen, as described in Section 3.3. The size of the resulting dataset is mainly affected by the following configuration parameters: the number of persons and the number of years simulated. By default, all SFs are defined over a period of three years, starting from 2010, and SFs are computed by scaling the number of Persons in the network. Table D.1 shows some metrics of SFs 0,1, ..., 1 000 data sets.

Table D.3 show how each CSV serializer handles attributes/edges of different cardinalities. The data shows why `CsvBasic` has the most files and `CsvCompositeMergeForeign` has the least number of files.

<sup>1</sup><https://tools.ietf.org/html/rfc3339>

<sup>2</sup>Naturally, there are slight differences in the disk usage of the data sets created with different serializers. For example, for a given scale factor, the disk usage of the data set serialized with the `CsvBasic` serializer is expected to be higher, while with the `CsvMergeForeignComposite`, it is expected to be lower.

Scale Factor	SF0.1	SF0.3	SF1	SF3	SF10	SF30	SF100	SF300	SF1 000	SF3 000
# Persons	1.5K	3.5K	11K	27K	73K	182K	499K	1.25M	3.6M	8.3M
# nodes	327.6K	908K	3.2M	9.3M	30M	88.8M	282.6M	817.3M	2.7B	7.7B
# edges	1.5M	4.6M	17.3M	52.7M	176.6M	540.9M	1.8B	5.3B	17.8B	51.8B

Table D.1: Properties of data sets for each scale factor in the Interactive workload produced by the Hadoop-based generator. For detailed statistics, see Table B.1

INS	Operation	SF0.1	SF0.3	SF1	SF3	SF10	SF30	SF100	SF300	SF1 000
1	Add person	172	386	1 108	2 672	7 355	18 570	50 374	125 931	360 960
2	Add like to post	44 313	132 041	494 410	1 460 471	4 875 874	14 378 128	45 633 086	129 721 727	410 899 721
3	Add like to comment	30 395	105 061	460 487	1 450 891	5 210 730	16 114 277	54 990 638	163 624 084	539 128 029
4	Add forum	3 059	6 913	19 757	49 223	131 439	330 288	898 185	2 257 347	6 479 509
5	Add forum membership	126 615	405 441	1 566 914	4 874 316	16 647 977	51 095 793	165 881 862	478 826 826	1 543 247 540
6	Add post	32 610	78 164	229 614	592 875	1 655 168	4 304 447	12 236 177	32 109 577	96 023 955
7	Add comment	46 969	144 917	490 328	1 372 420	4 414 427	12 588 582	39 547 415	112 862 922	362 292 612
8	Add friendship	3 197	10 337	40 124	122 714	431 916	1 304 053	4 252 839	12 047 072	36 762 818
Total insert operations		287 330	883 260	3 302 742	9 925 582	33 374 886	100 134 138	323 490 576	931 575 486	2 995 195 144

Table D.2: Update stream statistics for SNB Interactive v1.0

### D.1.2 Serializers

The datasets are generated in the `social_network/` directory, split into static and dynamic parts (Figure 3.1). The filenames (without the extension) end in `_i_j` where `i` is the block id and `j` is the partition id (set by `numThreads`). The SUT has to take care only of the generated Dataset to be bulk loaded. Using `NULL` values for storing optional values is allowed.

Datagen’s CSV (Comma Separated Values) serializers produce text files which use the pipe character “|” as the primary field separator and the semicolon character “;” as a separator for multi-valued attributes (for the composite serializers). The CSV files are stored in two subdirectories: `static/` and `dynamic/`. Depending on the number of threads used for generating the dataset, the number of files varies, since there is a file generated per thread. We indicate this with “\*” in the specification.

The following CSV variants are supported:

- **CsvBasic:** Each entity, relation and attribute with a cardinality larger than one (including attributes `Person.email` and `Person.speaks`), are output in a separate file. Generated files are summarized in Table D.4.
- **CsvMergeForeign:** This serializer is similar to `CsvBasic`, but relations that have a cardinality of 1-to-N are merged in the entity files as a foreign keys. There are 13 such relations in total:
  - `comment_hasCreator_person`, `comment_isLocatedIn_place`, `comment_replyOf_comment`, `comment_replyOf_post` (merged to `Comment`);
  - `forum_hasModerator_person` (merged to `Forum`);
  - `organisation_isLocatedIn_place` (merged to `Organisation`);
  - `person_isLocatedIn_place` (merged to `Person`);
  - `place_isPartOf_place` (merged to `Place`);

Serializer	Nodes	Attributes		Edges	
		single-valued	multi-valued	one-to-many	many-to-many
CsvBasic	⊗	○	⊗	⊗	⊗
CsvComposite	⊗	○	○	⊗	⊗
CsvMergeForeign	⊗	○	⊗	○	⊗
CsvCompositeMergeForeign	⊗	○	○	○	⊗

Table D.3: Attributes and edges serialized to separate files the different CSV serializers.

C	File	Content
N	organisation_*.csv	id   type   name   url
E	organisation_isLocatedIn_place_*.csv	Organisation.id   Place.id
N	place_*.csv	id   name   url   type
E	place_isPartOf_place_*.csv	Place.id   Place.id
N	tag_*.csv	id   name   url
E	tag_hasType_tagclass_*.csv	Tag.id   TagClass.id
N	tagclass_*.csv	id   name   url
E	tagclass_isSubclassOf_tagclass_*.csv	TagClass.id   TagClass.id
N	comment_*.csv	creationDate   id   locationIP   browserUsed   content   length
E	comment_hasCreator_person_*.csv	creationDate   Comment.id   Person.id
E	comment_hasTag_tag_*.csv	creationDate   Comment.id   Tag.id
E	comment_isLocatedIn_place_*.csv	creationDate   Comment.id   Place.id
E	comment_replyOf_comment_*.csv	creationDate   Comment.id   ParentComment.id
E	comment_replyOf_post_*.csv	creationDate   Comment.id   ParentPost.id
N	forum_*.csv	creationDate   id   title   type
E	forum_containerOf_post_*.csv	creationDate   Forum.id   Post.id
E	forum_hasMember_person_*.csv	creationDate   Forum.id   Person.id   type
E	forum_hasModerator_person_*.csv	creationDate   Forum.id   Person.id
E	forum_hasTag_tag_*.csv	creationDate   Forum.id   Tag.id
N	person_*.csv	creationDate   id   firstName   lastName   gender   birthday   locationIP   browserUsed
A	person_email_emailaddress_*.csv	creationDate   Person.id   email
E	person_hasInterest_tag_*.csv	creationDate   Person.id   Tag.id
E	person_isLocatedIn_place_*.csv	creationDate   Person.id   Place.id
E	person_knows_person_*.csv	creationDate   Person1.id   Person2.id
E	person_likes_comment_*.csv	creationDate   Person.id   Comment.id
E	person_likes_post_*.csv	creationDate   Person.id   Post.id
A	person_speaks_language_*.csv	creationDate   Person.id   language
E	person_studyAt_organisation_*.csv	creationDate   Person.id   Organisation.id   classYear
E	person_workAt_organisation_*.csv	creationDate   Person.id   Organisation.id   workFrom
N	post_*.csv	creationDate   id   imageFile   locationIP   browserUsed   language   content   length   Forum.id
E	post_hasCreator_person_*.csv	creationDate   Post.id   Person.id
E	post_hasTag_tag_*.csv	creationDate   Post.id   Tag.id
E	post_isLocatedIn_place_*.csv	creationDate   Post.id   Place.id

Table D.4: Files output by the CsvBasic serializer (33 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge.

- post\_hasCreator\_person, post\_isLocatedIn\_place, forum\_containerOf\_post (merged to Post);
- tag\_hasType\_tagclass (merged to Tag);
- tagclass\_isSubclassOf\_tagclass (merged to TagClass)

Generated files are summarized in Table D.5.

- **CsvComposite:** Similar to the CsvBasic format but each entity, and relations with a cardinality larger than one, are output in a separate file. Multi-valued attributes (`Person.email` and `Person.speaks`) are stored as composite values. Generated files are summarized in Table D.6.
- **CsvCompositeMergeForeign:** Has the traits of both the CsvComposite and the CsvMergeForeign formats. Multi-valued attributes (`Person.email` and `Person.speaks`) are stored as composite values. Generated files are summarized in Table D.7.

Additionally, the Hadoop Datagen can generate files in Turtle format (`.ttl`).

**Inheritance** The inheritance hierarchies in the schema have two important characteristics (1) all subclasses use the same id space, e.g. there cannot be a Comment and a Post with id 1 at the same time, (2) they are serialized to CSVs using either the *map hierarchy to single table* or *map each concrete class to its own table* strategies<sup>3</sup>:

<sup>3</sup><http://www.agiledata.org/essays/mappingObjects.html>



C	File	Content
N	organisation_*.csv	id   type   name   url   place
N	place_*.csv	id   name   url   type   isPartOf
N	tag_*.csv	id   name   url   hasType
N	tagclass_*.csv	id   name   url   isSubclassOf
N	comment_*.csv	id   creationDate   locationIP   browserUsed   content   length   creator   place   replyOfPost   replyOfComment
E	comment_hasTag_tag_*.csv	Comment.id   Tag.id
N	forum_*.csv	id   title   creationDate   moderator
E	forum_hasMember_person_*.csv	Forum.id   Person.id   joinDate
E	forum_hasTag_tag_*.csv	Forum.id   Tag.id
N	person_*.csv	id   firstName   lastName   gender   birthday   creationDate   locationIP   browserUsed   place
A	person_email_emailaddress_*.csv	Person.id   email
E	person_hasInterest_tag_*.csv	Person.id   Tag.id
E	person_knows_person_*.csv	Person.id   Person.id   creationDate
E	person_likes_comment_*.csv	Person.id   Post.id   creationDate
E	person_likes_post_*.csv	Person.id   Post.id   creationDate
A	person_speaks_language_*.csv	Person.id   language
E	person_studyAt_organisation_*.csv	Person.id   Organisation.id   classYear
E	person_workAt_organisation_*.csv	Person.id   Organisation.id   workFrom
N	post_*.csv	id   imageFile   creationDate   locationIP   browserUsed   language   content   length   creator   Forum.id   place
E	post_hasTag_tag_*.csv	Post.id   Tag.id

Table D.5: Files output by the CsvMergeForeign serializer (20 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge.

**Message = Comment | Post** *Map each concrete class to its own table* is used i.e. separate CSV files are used for the Comment and the Post classes.

**Place = City | Country | Continent** *Map hierarchy to single table* is used, i.e. all Place node are serialized in a single file. A discriminator attribute “type” is used with the value denoting the concrete class, e.g. “Country”.

**Organisation = Company | University** *Map hierarchy to single table* is used, i.e. all Organisation nodes are serialized in a single fiel. A discriminator attribute “type” is used with the value denoting the concrete class, e.g. “Company”.

### D.1.3 Update Streams

The generic schema is given in Table D.8, while the concrete schema of each insert operation is given in Table D.9. The update stream files are generated in the `social_network/` directory and are grouped as follows:

- `updateStream_*_person.csv` files contain update operation 1: **INS 1**
- `updateStream_*_forum.csv` files contain update operations 2–8: **INS 2** **INS 3** **INS 4** **INS 5** **INS 6** **INS 7** **INS 8**

Remark: update streams in Interactive v1 only contain inserts. Delete operations are being designed and will be released later.

### D.1.4 Substitution Parameters

The substitution parameters are generated in the `substitution_parameters/` directory. Each parameter file is named `{interactive|bi}<id>_param.txt`, corresponding to an operation of Interactive complex reads ( **IC 1** – **IC 14v1** ) and BI reads ( **BI 1** – **BI 20** ). The schemas of these files are defined by the operator, e.g. the schema of **IC 1** is “personId|firstName”.

**Warning.** Note that the substitution parameter files use UNIX epoch timestamps (which should be converted to a datetime value in GMT+0).

C	File	Content
N	organisation_*.csv	id   type   name   url
E	organisation_isLocatedIn_place_*.csv	Organisation.id   Place.id
N	place_*.csv	id   name   url   type
E	place_isPartOf_place_*.csv	Place.id   Place.id
N	tag_*.csv	id   name   url
E	tag_hasType_tagclass_*.csv	Tag.id   TagClass.id
N	tagclass_*.csv	id   name   url
E	tagclass_isSubclassOf_tagclass_*.csv	TagClass.id   TagClass.id
N	comment_*.csv	id   creationDate   locationIP   browserUsed   content   length
E	comment_hasCreator_person_*.csv	Comment.id   Person.id
E	comment_hasTag_tag_*.csv	Comment.id   Tag.id
E	comment_isLocatedIn_place_*.csv	Comment.id   Place.id
E	comment_replyOf_comment_*.csv	Comment.id   Comment.id
E	comment_replyOf_post_*.csv	Comment.id   Post.id
N	forum_*.csv	id   title   creationDate
E	forum_containerOf_post_*.csv	Forum.id   Post.id
E	forum_hasMember_person_*.csv	Forum.id   Person.id   joinDate
E	forum_hasModerator_person_*.csv	Forum.id   Person.id
E	forum_hasTag_tag_*.csv	Forum.id   Tag.id
N	person_*.csv	id   firstName   lastName   gender   birthday   creationDate   locationIP   browserUsed   language   email
E	person_hasInterest_tag_*.csv	Person.id   Tag.id
E	person_isLocatedIn_place_*.csv	Person.id   Place.id
E	person_knows_person_*.csv	Person.id   Person.id   creationDate
E	person_likes_comment_*.csv	Person.id   Post.id   creationDate
E	person_likes_post_*.csv	Person.id   Post.id   creationDate
E	person_studyAt_organisation_*.csv	Person.id   Organisation.id   classYear
E	person_workAt_organisation_*.csv	Person.id   Organisation.id   workFrom
N	post_*.csv	id   imageFile   creationDate   locationIP   browserUsed   language   content   length
E	post_hasCreator_person_*.csv	Post.id   Person.id
E	post_hasTag_tag_*.csv	Post.id   Tag.id
E	post_isLocatedIn_place.csv	Post.id   Place.id

Table D.6: Files output by the CsvComposite serializer (31 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge.

C	File	Content
N	organisation_*.csv	id   type   name   url   place
N	place_*.csv	id   name   url   type   isPartOf
N	tag_*.csv	id   name   url   hasType
N	tagclass_*.csv	id   name   url   isSubclassOf
N	comment_*.csv	id   creationDate   locationIP   browserUsed   content   length   creator   place   replyOfPost   replyOfComment
E	comment_hasTag_tag_*.csv	Comment.id   Tag.id
N	forum_*.csv	id   title   creationDate   moderator
E	forum_hasMember_person_*.csv	Forum.id   Person.id   joinDate
E	forum_hasTag_tag_*.csv	Forum.id   Tag.id
N	person_*.csv	id   firstName   lastName   gender   birthday   creationDate   locationIP   browserUsed   place   language   email
E	person_hasInterest_tag_*.csv	Person.id   Tag.id
E	person_knows_person_*.csv	Person.id   Person.id   creationDate
E	person_likes_comment_*.csv	Person.id   Post.id   creationDate
E	person_likes_post_*.csv	Person.id   Post.id   creationDate
E	person_studyAt_organisation_*.csv	Person.id   Organisation.id   classYear
E	person_workAt_organisation_*.csv	Person.id   Organisation.id   workFrom
N	post_*.csv	id   imageFile   creationDate   locationIP   browserUsed   language   content   length   creator   Forum.id   place
E	post_hasTag_tag_*.csv	Post.id   Tag.id

Table D.7: Files output by the CsvCompositeMergeForeign serializer (18 in total). The first part of the table contains the static entites, the second part contains the dynamic ones. Notation – C: entity category, N: node, E: edge.

start time ( $t_s$ )	dependant time ( $t_d$ )	operation id	...
----------------------	--------------------------	--------------	-----

Table D.8: Generic schema of update (insert) stream files. The start time ( $t_s$ ) is identical to the creationDate attribute (repeated later in the row).

$t_s$   $t_d$   1   personId   personFirstName   personLastName   gender   birthday   creationDate   locationIP   browserUsed   cityId   languages   emails   tagIds   studyAt   workAt
$t_s$   $t_d$   2   personId   postId   creationDate
$t_s$   $t_d$   3   personId   commentId   creationDate
$t_s$   $t_d$   4   forumId   forumTitle   creationDate   moderatorPersonId   tagIds
$t_s$   $t_d$   5   forumId   personId   creationDate
$t_s$   $t_d$   6   postId   imageFile   creationDate   locationIP   browserUsed   language   content   length   authorPersonId   forumId   countryId   tagIds
$t_s$   $t_d$   7   commentId   creationDate   locationIP   browserUsed   content   length   authorPersonId   countryId   replyToPostId   replyToCommentId   tagIds
$t_s$   $t_d$   8   person1Id   person2Id   creationDate

Table D.9: Schemas of the lines in the update stream (insert stream) files.

## E EXAMPLE GRAPH

Figure E.1 shows a static snapshot of an example graph, while Figure E.2 shows an example graph with update operations. Insertions are denoted with a green asterisk  $\star$ . Deletions of a single element are denoted with a red cross  $\times$ , while recursive deletions are denoted with a purple cross  $\times$ .

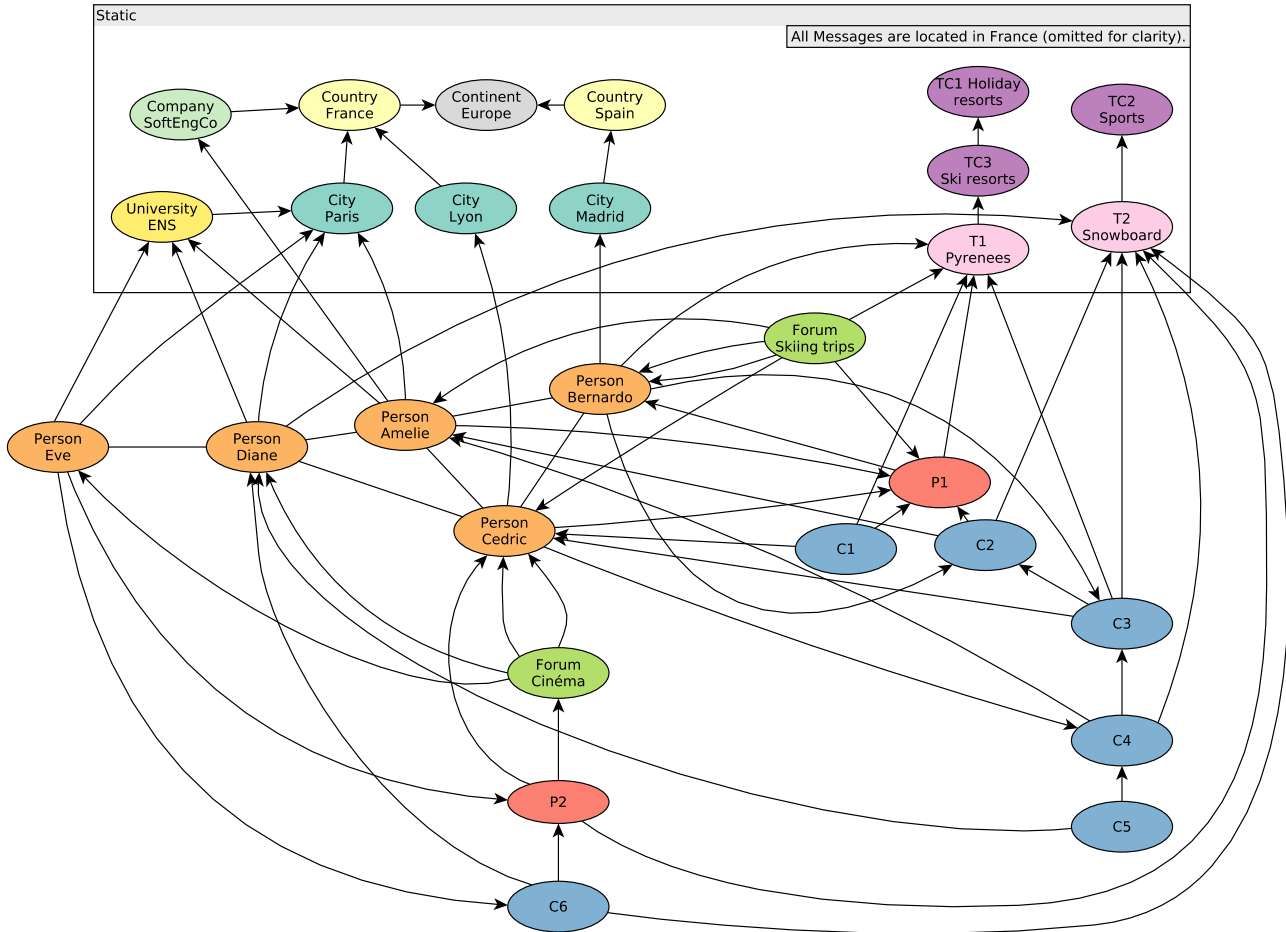


Figure E.1: Example graph snapshot (without update operations).

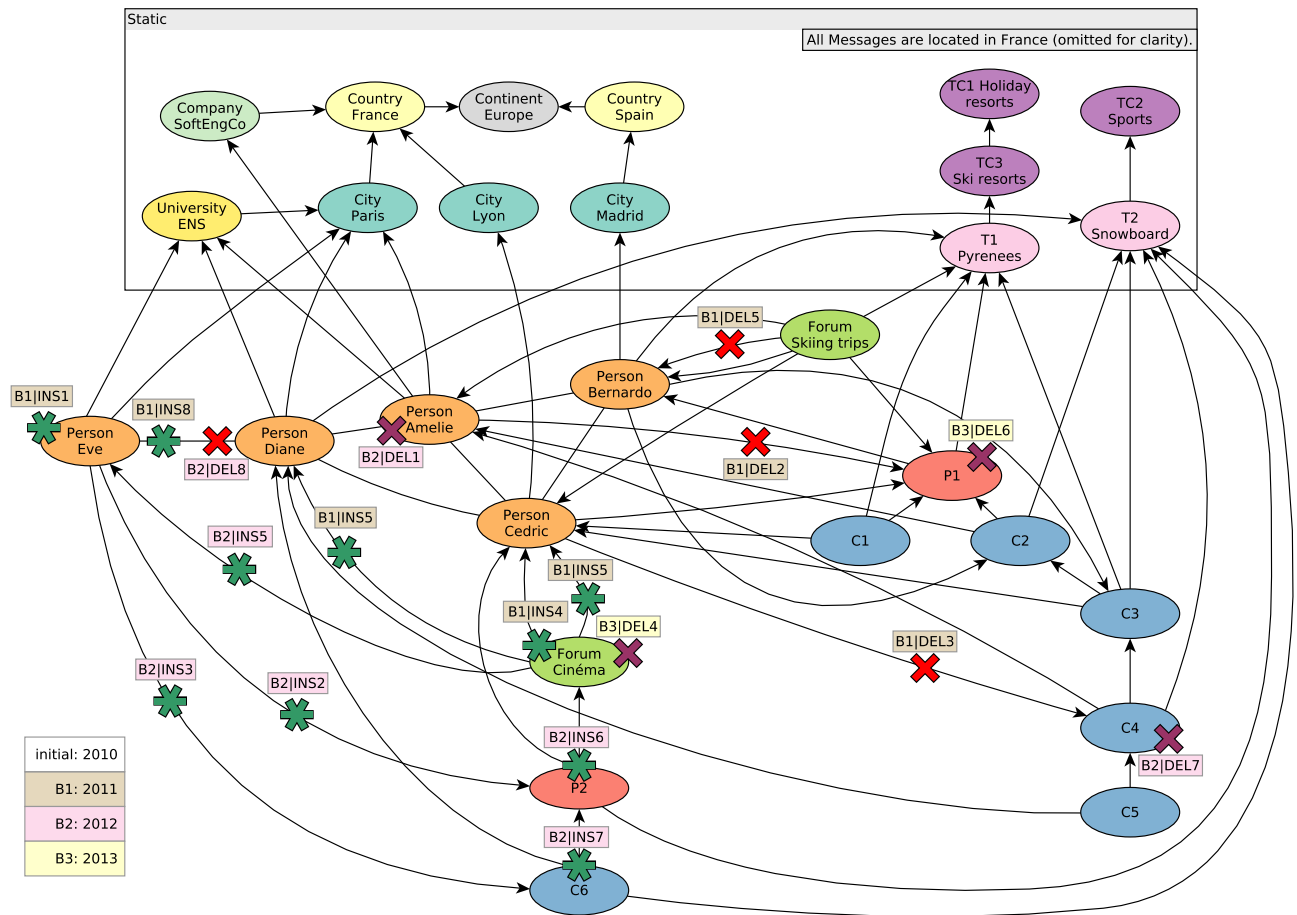


Figure E.2: Example graph with update operations.