



LDBC Social Network Benchmark (SNB) - v0.2.4

Coordinator: [names of main authors]

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 three sub-benchmarks, or workloads, that focus on different functionalities. In this document, we present a preliminary version of the Interactive Workload. The other workloads, still in development, are the Business Intelligence Workloads (with analytical queries), and the Graph Analytics Workload (with graph algorithms).

This document contains a detailed explanation of the data used in the whole LDBC-SNB benchmark, a detailed description for all the Interactive Workload, 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

EXECUTIVE SUMMARY	3
DOCUMENT INFORMATION	4
1 INTRODUCTION	9
1.1 Motivation for the Benchmark	9
1.2 Relevance to Industry	9
1.3 General Benchmark Overview	10
1.4 Participation of Industry and Academia	11
2 BENCHMARK SPECIFICATION	13
2.1 Requirements	13
2.2 Software and Useful links	13
2.3 Data	13
2.3.1 Data Types	13
2.3.2 Data Schema	14
2.3.3 Output Data	18
2.3.4 Scale Factors	20
3 WORKLOADS	21
3.1 Query Description Format	21
3.2 Interactive Workload	21
3.2.1 Choke Points	21
3.2.2 Complex Reads Query Descriptions	22
3.2.3 Short Reads Query Descriptions	28
3.2.4 Update Query Descriptions	30
3.2.5 Substitution parameters	32
3.2.6 Load Definition	33
3.3 Business Intelligence Workload	34
3.3.1 Choke Points	34
3.3.2 Business Intelligence Queries	34
4 AUDITING RULES	43
4.1 Preparation	43
4.1.1 Collect System Details	43
4.1.2 Setup the Benchmark Environment	44
4.1.3 Load Data	44
4.2 Running the Benchmark	44
4.3 Recovery	44
4.4 Serializability	45
A SCALE FACTOR STATISTICS	47
A.1 Scale Factor Statistics	47

SPECIAL THANKS

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

- Renzo Angles (Universidad de Talca)
- Alex Averbuch (Neo Technologies)
- Peter Boncz (Vrije Universiteit Amsterdam)
- Orri Erling (Google)
- Andrey Gubichev (Google)
- Moritz Kaufmann (Technische Universität München)
- Josep Lluís Larriba Pey (Universitat Politècnica de Catalunya)
- Minh-Duc Pham (Altran)
- Marcus Paradies (SAP)
- Arnau Prat (Sparsity Technologies)
- Mirko Spasić (Openlink Software)
- Norbert Martínez (Huawei Technologies)

LIST OF FIGURES

2.1	The LDBC-SNB data schema	15
-----	------------------------------------	----

LIST OF TABLES

2.1	Description of the data types.	13
2.2	Attributes of Forum entity.	14
2.3	Attributes of Message interface.	16
2.4	Attributes of Organization entity.	16
2.5	Attributes of Person entity.	16
2.6	Attributes of Place entity.	16
2.7	Attributes of Post entity.	17
2.8	Attributes of Tag entity.	17
2.9	Attributes of TagClass entity.	17
2.10	Description of the data relations.	18
2.11	Files output by the CSV serializer (32 in total)	19
2.12	Files output by the CSV_MERGE_FOREIGN serializer (23 in total)	20
2.13	Parameters of each scale factor.	20
3.1	Frequencies for each query type for SF1.	34
A.1	Frequencies for each query and SF.	47

DEFINITIONS

DATAGEN: Is the data generator provided by the LDBC-SNB, which is responsible for generating the data needed to run the benchmark.

DBMS: A DataBase Management System.

LDBC-SNB: Linked Data Benchmark Council Social Network Benchmark.

Query Mix: Refers to the ratio between read and update queries of a workload, and the frequency at which they are issued.

SF (Scale Factor): 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, measured in Gigabytes.

SUT: The System Under Test is defined to be the database system where the benchmark is executed.

Test Driver: A program provided by the LDBC-SNB, which is responsible for executing the different workloads and gathering the results.

Test Sponsor: The Test Sponsor is the company officially submitting the Result with the FDR and will be charged the filing fee. Although multiple companies may sponsor a Result together, for the purposes of the LDBC processes the Test Sponsor must be a single company. A Test Sponsor need not be a LDBC member. The Test Sponsor is responsible for maintaining the FDR with any necessary updates or corrections. The Test Sponsor is also the name used to identify the Result.

Workload: A workload refers to a set of queries of a given nature (i.e. interactive, analytical, business), how they are issued and at which rate.

1 INTRODUCTION

1.1 Motivation for the Benchmark

The new era of data economy, based on large, distributed and complexly structured data sets, has brought on new and complex challenges in the field of data management and analytics. These data sets, 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 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 LDBC Social Network Benchmark (LDBC-SNB) aims at being comprehensive benchmark 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 case of modern graph-like applications. LDBC-SNB is a work in progress, and initially, it only includes the Interactive Workload [1], but two more workloads will be introduced in the future: Business Intelligence and Analytics. By designing three separate workloads, LDBC-SNB targets a broader range of systems with different nature and characteristics. LDBC-SNB aims at capturing the essential features of these usage scenarios while abstracting away details of specific business deployments.

1.2 Relevance to 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 systems** (e.g. Neo4j, InfiniteGraph, Sparksee, Titan) are novel technologies aimed at storing directed and labeled graphs. They support graph traversals, typically by means of APIs, though some of them also support some sort of graph oriented query language (e.g. Neo4j's Cypher). These systems' internal structures are typically designed to store dynamic graphs that change over time. They often support transactional queries with some degree of consistency, and value-based indexes 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 programming frameworks** (e.g. Giraph, Signal/Collect, GraphLab, Green Marl) are designed to perform global graph queries computations, executed in parallel or lockstep. 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. Though these systems will likely implement the Graph Analytics workload.
- **RDF database systems** (e.g. OWLIM, Virtuoso, BigData, Jena TDB, Stardog, Allegrograph) 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 system 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. Postgres, MySQL, Oracle, DB2, SQLserver, Virtuoso, MonetDB, Vectorwise, Vertica, but also Hive and Impala) treat data as relational, 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
- **noSQL database systems** (e.g. key-value stores such as HBase, REDIS, MongoDB, CouchDB, or even MapReduce systems like Hadoop and Pig) are cluster-vbased and scalable. Key-value stores could possibly implement the Interactive Workload, though its navigational aspects would pose some problems as potentially many key-value lookups are needed. MapReduce systems could be suited for the Graph Analytics workload, but their query latency would presumably be so high that the Business Intelligence workload would not make sense, though we note that some of the key-value stores (e.g. MongoDB) provide a MapReduce query functionality on the data that it stores which could make it suited for the Business Intelligence workload.

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, on 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. 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 to 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.

In Section 2.3, LDBC-SNB defines the schema of the data used in the benchmark. The schema, represents a realistic social network, including people and their activity 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. Persons' 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 parallel paradigm, that 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 Section 3, the Interactive Workload is proposed. Two more workloads are planned: Business Intelligence Workload and Analytical workload. 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. Interactive workload reproduces the interaction between the users of the social network by including lookups and transactions that update small portions of the data base. These queries are designed to be interactive and target systems capable of responding such queries with low latency for multiple concurrent users. Business Intelligence workload, will represent those business intelligence analytics a social network company would like to perform in the social network, in order to take advantage of the data to discover new business opportunities. This workload will explore moderate portions of data from different entities, and performing more resource intensive operations. Finally, the graph analytics workload will aim at exploring the characteristics of the underlying structure of the network. Shortest paths, community detection or centrality, are representative queries of this workload, and will imply touching a vast amount of the dataset.

LDBC-SNB provides 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 all the necessary software tools to run the benchmark, and therefore to reduce the benchmark's entry point as much as possible.

1.4 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 that meets all the desired needs. The list of participants is the following:

- FOUNDATION FOR RESEARCH AND TECHNOLOGY HELLAS
- NEO4J
- ONTOTEXT
- OPENLINK

- TECHNISCHE UNIVERSITAET MUENCHEN
- UNIVERSITAET INNSBRUCK
- UNIVERSITAT POLITECNICA DE CATALUNYA
- VRIJE UNIVERSITEIT AMSTERDAM

Besides the aforementioned institutions, during the development of the benchmark several meetings with the technical and users community have been conducted, receiving an invaluable feedback that has contributed to the whole development of the benchmark in every of its aspects.

2 BENCHMARK SPECIFICATION

2.1 Requirements

LDBC-SNB is designed to be flexible and to have an affordable entry point. From small single node and in memory systems to large distributed multi-node clusters have its own place in LDBC-SNB. Therefore, the requirements to fulfill for executing LDBC-SNB are limited to pure software requirements to be able to run the tools. All the software provided by LDBC-SNB have been developed and tested under Linux-based operating 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. Consequently, any language or API capable of expressing the proposed queries can be used. Similarly, data can be stored in the most convenient manner the test sponsor may decide.

2.2 Software and Useful links

- **LDBC Driver 0.3** – https://github.com/ldbc/ldbc_driver: The driver responsible for executing the LDBC SNB workload.
- **DATAGEN 0.2.5** – https://github.com/ldbc/ldbc_snb_datagen: The data generator used to generate the datasets of the benchmark.

2.3 Data

This section 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.

2.3.1 Data Types

Table 2.1 describes the different types used in the whole benchmark.

Type	Description
ID	integer type with 64-bit precision. All IDs within a single entity are unique
32-bit Integer	integer type with 32-bit precision
64-bit Integer	integer type with 64-bit precision
String	variable length text of size 40 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: <i>yyyy-mm-dd</i> , where <i>yyyy</i> is a four-digit integer representing the year, the year, <i>mm</i> is a two-digit integer representing the month and <i>dd</i> is a two-digit integer representing the day.
DateTime	date with a precision of milliseconds, encoded as a string with the following format: <i>yyyy-mm-ddTHH:MM:ss.sss+0000</i> , where <i>yyyy</i> is a four-digit integer representing the year, the year, <i>mm</i> is a two-digit integer representing the month and <i>dd</i> is a two-digit integer representing the day, <i>HH</i> is a two-digit integer representing the hour, <i>MM</i> is a two digit integer representing the minute and <i>ss.sss</i> is a five digit fixed point real number representing the seconds up to millisecond precision. Finally, the <i>+0000</i> of the end represents the timezone, which in this case is always GMT.

Table 2.1: Description of the data types.

2.3.2 Data Schema

Figure 2.1 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, Organizations, 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.

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.

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

Textual Restrictions and Notes

- Posts have content or imageFile. They have one of them but not both. The one they do not have is an empty string.
- Posts in a forum can be created by a non-member person if and only if that person is a modeartor.

Entities

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 Organization, and represents a company where persons work.

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

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: persons' personal walls, image albums, and groups. They are distinguished by their titles. Table 2.2 shows the attributes of 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 2.2: Attributes of Forum entity.

Message: an abstract entity that represents a message created by a person. Table 2.3 shows the attributes of 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[0..1]	The content of the message.
length	32-bit Integer	The length of the content.

Table 2.3: Attributes of Message interface.

Organization: an institution of the real world. Table 2.4 shows the attributes of Organization entity.

Attribute	Type	Description
id	ID	The identifier of the organization.
name	Long String	The name of the organization.

Table 2.4: Attributes of Organization 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 2.5 shows the attributes of 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[1..*]	The set of emails the person has.
speaks	String[1..*]	The set of languages the person speaks.
browserUser	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 2.5: Attributes of Person entity.

Place: a place in the world. Table 2.6 shows the attributes of Place entity.

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

Table 2.6: Attributes of 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 2.7 shows the attributes of Post entity.

Attribute	Type	Description
language	String[0..1]	The language of the post.
imageFile	String[0..1]	The image file of the post..

Table 2.7: Attributes of 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 2.8 shows the attributes of Tag entity.

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

Table 2.8: Attributes of Tag entity.

TagClass: a class or a category used to build a hierarchy of tags. Table 2.9 shows the attributes of TagClass entity.

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

Table 2.9: Attributes of TagClass entity.

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

Relations

Relations connect entities of different types. Entities are defined by their "id" attribute.

Name	Tail	Head	Type	Description
containerOf	Forum[1]	Post[1..*]	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[0..*]	D	A Person and a Tag representing a topic the person is interested in
hasMember	Forum[0..*]	Person[1..*]	D	A Forum and a member (Person) of the forum <ul style="list-style-type: none"> • Attribute: joinDate • Type: DateTime • Description: The Date the person joined the forum
hasModerator	Forum[0..*]	Person[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[0..*]	D	A Forum and a Tag representing the forum's topic
hasType	Tag[0..*]	TagClass[0..*]	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..*]	D	A TagClass its parent TagClass
knows	Person[0..*]	Person[0..*]	U	Two Persons that know each other <ul style="list-style-type: none"> • Attribute: creationDate • Type: DateTime • Description: The date the knows relation was established
likes	Person[0..*]	Message[0..*]	D	A Person that likes a Message <ul style="list-style-type: none"> • Attribute: creationDate • Type: DateTime • Description: The date the like was issued
replyOf	Comment[0..*]	Message[1]	D	A Comment and the Message it replies
studyAt	Person[0..*]	University[0..*]	D	A Person and a University it has studied <ul style="list-style-type: none"> • Attribute: classYear • Type: 32-bit Integer • Description: The year the person graduated.
workAt	Person[0..*]	Company[0..*]	D	A Person and a Company it works <ul style="list-style-type: none"> • Attribute: workFrom • Type: 32-bit Integer • Description: The year the person started to work at that company

Table 2.10: Description of the data relations.

2.3.3 Output Data

DATAGEN produces outputs three different items:

- **Dataset:** The dataset to be bulk loaded by the SUT. 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.

The SUT have to take care only of the generated Dataset to be bulk loaded. Three different formats are supported by DATAGEN:

- **CSV:** Data output in CSV format, one file per different entity and on file per different relation. Also, there is a file por those attributes whose cardinality is larger than one (i.e. Person.email, Person.speaks, etc.).
- **CSVMergeForeign:** Similar to CSV format, but in this case, those relations of the form 1 to 1 and 1 to N, are stored in the tail entity file as a foreign keys.
- **Turtle:** Dataset in Turtle format for RDF systems.

CSV

This is a comma separated format. Each entity, relation and properties with a cardinality larger than one, are output in a separate file. Generated files are summarized at Table 2.11. Depending on the number of threads used for generating the dataset, the number of files varies, since there is a file generated per thread. The * in the file names indicates a number between 0 and $NumberOfThreads - 1$.

File	Content
comment_*.csv	id creationDate locationIP browserUsed content length
comment_hasCreator_person_*.csv	Comment.id Person.id
comment_isLocatedIn_place_*.csv	Comment.id Place.id
comment_replyOf_comment_*.csv	Comment.id Comment.id
comment_replyOf_post_*.csv	Comment.id Post.id
forum_*.csv	id title creationDate
forum_containerOf_post_*.csv	Forum.id Post.id
forum_hasMember_person_*.csv	Forum.id Person.id joinDate
forum_hasModerator_person_*.csv	Forum.id Person.id
forum_hasTag_tag_*.csv	Forum.id Tag.id
organization_*.csv	id type("university", "company") name url
organisation_isLocatedIn_place_*.csv	Organisation.id Place.id
person_*.csv	id firstName lastName gender birthday creationDate locationIP browserUsed
person_email_emailaddress_*.csv	Person.id email
person_hasInterest_tag_*.csv	Person.id Tag.id
person_isLocatedIn_place_*.csv	Person.id Place.id
person_knows_person_*.csv	Person.id Person.id creationDate
person_likes_comment_*.csv	Person.id Post.id creationDate
person_likes_post_*.csv	Person.id Post.id creationDate
person_speaks_language_*.csv	Person.id language
person_studyAt_organisation_*.csv	Person.id Organisation.id classYear
person_workAt_organisation_*.csv	Person.id Organisation.id workFrom
place_*.csv	id name url type("city", "country", "continent")
place_isPartOf_place_*.csv	Place.id Place.id
post_*.csv	id imageFile creationDate locationIP browserUsed language content length
post_hasCreator_person_*.csv	Post.id Person.id
post_hasTag_tag_*.csv	Post.id Tag.id
post_isLocatedIn_place.csv	Post.id Place.id
tag_*.csv	id name url
tag_hasType_tagclass_*.csv	Tag.id TagClass.id
tagclass_*.csv	id name url
tagclass_isSubclassOf_tagclass_*.csv	TagClass.id TagClass.id

Table 2.11: Files output by the CSV serializer (32 in total)

CSV_MERGE_FOREIGN

This is a comma separated format. It is similar to CSV, but those relations connecting two entities A and B, where an entity A has a cardinality of one, A is output as a column of entity B. Generated files are summarized

at Table 2.12. Depending on the number of threads used for generating the dataset, the number of files varies, since there is a file generated per thread. The * in the file names indicates a number between 0 and $NumberOfThreads - 1$.

File	Content
comment_*.csv	id creationDate locationIP browserUsed content length creator place replyOfPost replyOfComment
forum_*.csv	id title creationDate moderator
forum_hasMember_person_*.csv	Forum.id Person.id joinDate
forum_hasTag_tag_*.csv	Forum.id Tag.id
organization_*.csv	id type("university", "company") name url
organisation_isLocatedIn_place_*.csv	Organisation.id Place.id
person_*.csv	id firstName lastName gender birthday creationDate locationIP browserUsed place
person_email_emailaddress_*.csv	Person.id email
person_hasInterest_tag_*.csv	Person.id Tag.id
person_knows_person_*.csv	Person.id Person.id creationDate
person_likes_comment_*.csv	Person.id Post.id creationDate
person_likes_post_*.csv	Person.id Post.id creationDate
person_speaks_language_*.csv	Person.id language
person_studyAt_organisation_*.csv	Person.id Organisation.id classYear
person_workAt_organisation_*.csv	Person.id Organisation.id workFrom
place_*.csv	id name url type("city", "country", "continent")
place_isPartOf_place_*.csv	Place.id Place.id
post_*.csv	id imageFile creationDate locationIP browserUsed language content length creator Forum.id place
post_hasTag_tag_*.csv	Post.id Tag.id
tag_*.csv	id name url
tag_hasType_tagclass_*.csv	Tag.id TagClass.id
tagclass_*.csv	id name url
tagclass_isSubclassOf_tagclass_*.csv	TagClass.id TagClass.id

Table 2.12: Files output by the CSV_MERGE_FOREIGN serializer (23 in total)

Turtle

This is the standard Turtle¹ format. DATAGEN outputs two files: 0_ldbc_socialnet_static_dbp.ttl and 0_ldbc_socialnet.ttl.

2.3.4 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 Gigabytes of the generated output files using the CSV serializer. For example, SF 1 weights roughly 1 GB in CSV format, SF 3 weights roughly 3 GB and so on and so forth. The proposed SFs are the following: 1, 3, 10, 30, 100, 300, 1000. The Test Sponsor may select the SF that better fits their needs, by properly configuring the DATAGEN.

The size of the resulting dataset, is mainly affected by the following configuration parameters: the number of persons and the number of years simulated. Different SFs are computed by scaling the number of Persons in the network, while fixing the number of years simulated. Table 2.13 shows the parameters used in each of the SFs.

Scale Factor	1	3	10	30	100	300	1000
# of Persons	11K	27K	73K	182K	499K	1.25M	3.6M
# of Years	3	3	3	3	3	3	3
Start Year	2010	2010	2010	2010	2010	2010	2010

Table 2.13: Parameters of each scale factor.

For example, SF 30 consists of the activity of a social network of 182K users during a period of three years, starting from 2010.

¹Description of the Turtle RDF format <http://www.w3.org/TR/turtle/>

3 WORKLOADS

3.1 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; and *results*, a list of expected results and their types. The syntax used in *parameters* and *results* sections is as follows:

- **Entity:** entity type in the dataset.
One word, possibly constructed by appending multiple words together, starting with uppercase character and following the camel case notation, e.g. `TagClass` represents an entity of type “TagClass”.
- **Relationship:** relationship type in the dataset.
One word, possibly constructed by appending multiple words together, starting with lowercase character and following the camel case notation, and surrounded by arrow to communicate direction, e.g. `-worksAt->` represents a directed relationship of type “worksAt”.
- **Attribute:** attribute of an entity or relationship in the dataset.
One word, possibly constructed by appending multiple words together, starting with lowercase character and following the camel case notation, and prefixed by a “.” to dereference the entity/relationship, e.g. `Person.firstName` refers to “firstName” attribute on the “Person” entity, and `-studyAt->.classYear` refers to “classYear” attribute on the “studyAt” relationship.
- **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.

3.2 Interactive Workload

3.2.1 Choke Points

The design of the interactive workload queries has been conceived around two main aspects: realism and technological relevance. While realism has been assessed by looking at existing social networks and thinking about what interesting functionalities a user might desire from them, technological relevance has been achieved by identifying a set of choke points queries should stress. These choke points capture those critical operations, techniques or technologies that could significantly affect the performance of the queries. The choke points can be summarized in the following list:

- **Aggregation Performance.**

The queries generally have a top- k order by and often a group by in addition to this. These offer multiple optimization opportunities. The queries also often have distinct operators, i.e. distinct friends within two steps. Collectively these are all set operations that may be implemented with some combination of hash and sorting, possibly exploiting ordering in the data itself. The aggregates are not limited to counts and sums. For example string concatenation occurs as an aggregate, testing possible user defined aggregate support. There is a wide range of cardinalities in grouping, from low, e.g. country, to high, e.g. post.

- **Join Performance.**

Each graph traversal step is in principle a join. The join patterns are diverse, exercising both index- and hash-based operators. Queries are designed so as to reward judicious use of hash join by having patterns starting with one entity, fetching many related entities and then testing how these are related to a third entity, e.g. posts of a user with a tag of a given type.

- **Data Access Locality.**

Graph problems are notoriously non-local. However, when queries touch any non-trivial fraction of a dataset, locality will emerge and can be exploited, for example by vectored index access or by ordering data so that that a merge join is possible.

- **Expression Calculation.**

Queries often have expressions, including conditional expressions. This provides opportunities for vectoring and tests efficient management of intermediate results.

- **Correlated Subqueries.**

The workload has many correlated subqueries, for example constructs like `x` within two steps but not in one step, which would typically be a correlated subquery with `NOT EXISTS`. There are also scalar subqueries with aggregation, for example returning the count of posts satisfying a certain criteria.

- **Parallelism and Concurrency.**

All queries offer opportunities for parallelism. This tests a wide range of constructs, for example partitioned parallel variants of `group by` and `distinct`. An interactive workload will typically avoid trivially parallelizable table scans. Thus the opportunities that exist must be captured by index based, navigational query plans. The choice of whether to parallelize or not is often left to run time and will have to depend on the actual data seen in the execution, as starting a parallel thread with too little to do is counter-productive.

- **Graph Specifics.**

Graph problems are generally characterized by transitive properties and the fact that neighboring vertices often have a large overlap in their environments. This makes cardinality estimation harder. For example, a query optimizer needs to recognize whether a relationship has a tree or graph shape in order to make correct cardinality estimations. Further, there are problems aggregating properties over a set of consecutive edges. The workload contains business questions dealing with paths and aggregates across paths, as well as the easier case of determining a membership in a hierarchy with a transitive part-of relation.

3.2.2 Complex Reads Query Descriptions

Notes:

- Some queries require returning the content of a post. As stated in the schema, posts have either some content or an `imageFile`, but not both. An empty string in content represents the post not having any content, therefore, it must have a non-empty string in `imageFile` or the other way around.

1. Friends with certain name

- **Description:** Given a start Person, find Persons with a given first name that the start Person is connected to (excluding start Person) by at most 3 steps via `Knows` relationships. Return Persons, including summaries of the Persons workplaces and places of study.

- **Parameters:**

Person.id	ID
Person.firstName	String

- **Results:**

Person.id	ID
Person.lastName	String
Person.birthday	Date
Person.creationDate	DateTime
Person.gender	String
Person.browserUsed	String
Person.locationIP	String
{ Person.emails }	{ String }
{ Person.language }	{ String }
Person-isLocatedIn->Place.name	String
{ Person-studyAt->University.name,	
Person-studyAt->.class Year,	
Person-studyAt->University-isLocatedIn->City.name }	{ <String, 32-bit Integer, String> }
{ Person-workAt->Company.name,	
Person-workAt->.workFrom,	
Person-workAt->Company-isLocatedIn->Country.name }	{ <String, 32-bit Integer, String> }
- **Sort:**
 - 1st distance from person (ascending)
 - 2nd Person.lastName (ascending)
 - 3rd Person.id (ascending)
- **Limit:**
 - 20

2. Recent posts and comments by your friends

- **Description:** Given a start Person, find (most recent) Messages from all of that Person's friends, that were created before (and including) a given date.
- **Parameters:**

Person.id	ID
date	DateTime
- **Results:**

Message-hasCreator->Person.id	ID
Message-hasCreator->Person.firstName	String
Message-hasCreator->Person.lastName	String
Message.id	ID
Message.content or Post.imageFile	String
Message.creationDate	DateTime
- **Sort:**
 - 1st Message.creationDate (descending)
 - 2nd Message.id (ascending)
- **Limit:**
 - 20

3. Friends and friends of friends that have been to countries X and Y

- **Description:** Given a start Person, find Persons that are their friends and friends of friends (excluding start Person) that have made Posts/Comments in both of the given Countries, X and Y, within a given period. Only Persons that are foreign to Countries X and Y are considered, that is Persons whose Location is not Country X or Country Y.
- **Parameters:**

Person.id	ID	
CountryX.name	String	
CountryY.name	String	
startDate	Date	// beginning of requested period
duration	32-bit Integer	// duration of requested period, in days
		// the interval [startDate, startDate + Duration) is closed-open

- **Results:**

Person.id	ID	
Person.firstName	String	
Person.lastName	String	
countx	32-bit Integer	// number of Messages from Country X made by Person within the given time
county	32-bit Integer	// number of Messages from Country Y made by Person within the given time
count	32-bit Integer	// countx + county
- **Sort:**
 - 1st countx (descending)
 - 2nd Person.id (ascending)
- **Limit:**
 - 20

4. New topics

- **Description:** Given a start Person, 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, and that were never attached to friends' Posts created before this interval.
- **Parameters:**

Person.id	ID	
startDate	Date	
duration	32-bit Integer	// duration of requested period, in days // the interval [startDate, startDate + Duration) is closed-open
- **Results:**

Tag.name	String	
count	32-bit Integer	// number of Posts made within the given time interval that have this Tag
- **Sort:**
 - 1st count (descending)
 - 2nd Tag.name (ascending)
- **Limit:**
 - 10

5. New groups

- **Description:** Given a start Person, find the Forums which that Person's friends and friends of friends (excluding start Person) became Members of after a given date. For each forum find the number of Posts that were created by any of these Persons. For each Forum and consider only those Persons which joined that particular Forum after the given date.
- **Parameters:**

Person.id	ID	
date	Date	
- **Results:**

Forum.title	String	
count	32-bit Integer	// number of Posts made in Forum that were created by friends
- **Sort:**
 - 1st count (descending)
 - 2nd Forum.id (ascending)
- **Limit:**
 - 20

6. Tag co-occurrence

- **Description:** Given a start Person and some Tag, 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 For each Tag, find the count of Posts that were created by these Persons, which contain both this Tag and the given Tag.
- **Parameters:**

Person.id	ID
Tag.name	String
- **Results:**

Tag.name	String
count	32-bit Integer // number of Posts that were created by friends and friends of friends, which contain this Tag
- **Sort:**

1st	count (descending)
2nd	Tag.name (ascending)
- **Limit:**

10

7. Recent likers

- **Description:** Given a start Person, find (most recent) Likes on any of start Person's Messages. Find Persons that Liked any of start Person's Messages, the Messages they liked most recently, creation date of that Like, and the latency (in minutes) between creation of Messages and Like. Additionally, for each Person found return a flag indicating whether the liker is a friend of start Person. In the case that a Person Liked multiple Messages at the same time, return the Message with lowest identifier.
- **Parameters:**

Person.id	64-bit Integer
-----------	----------------
- **Results:**

Person.id	ID
Person.firstName	String
Person.lastName	String
Like.creationDate	DateTime
Message.id	ID
Message.content or Post.imageFile	String
latency	32-bit Integer // duration between creation of Message and Like, in minutes
isNew	Boolean // false if liker Person is friend of start Person, true otherwise
- **Sort:**

1st	Like.creationDate (descending)
2nd	Person.id (ascending)
- **Limit:**

20

8. Recent replies

- **Description:** Given a start Person, find (most recent) Comments that are replies to Messages of the start Person. Only consider immediate (1-hop) replies, not the transitive (multi-hop) case. Return the reply Comments, and the Person that created each reply Comment.
- **Parameters:**

Person.id	ID
-----------	----
- **Results:**

Person.id	ID
Person.firstName	String
Person.lastName	String
Comment.creationDate	DateTime
Comment.id	ID
Comment.content	String

- **Sort:**
 - 1st Comment.creationDate (descending)
 - 2nd Comment.id (ascending)
- **Limit:**
 - 20

9. Recent posts and comments by friends or friends of friends

- **Description:** Given a start Person, find the (most recent) Messages created by that Person's friends or friends of friends (excluding start Person). Only consider the Messages created before a given date (excluding that date).
- **Parameters:**

Person.id	ID
date	Date
- **Results:**

Message-hasCreator->Person.id	ID
Message-hasCreator->Person.firstName	String
Message-hasCreatr->Person.lastName	String
Message.id	ID
Message.content or Post.imageFile	String
Message.creationDate	DateTime
- **Sort:**
 - 1st Message.creationDate (descending)
 - 2nd Message.id (ascending)
- **Limit:**
 - 20

10. Friend recommendation

- **Description:** Given a start Person, find that Person's friends of friends (excluding start Person, and immediate friends), who were born on or after the 21st of a given month (in any year) and before the 22nd of the following month. Calculate the similarity between each of these Persons and start Person, where similarity for any Person is defined as follows:
 - common = number of Posts created by that Person, such that the Post has a Tag that start Person is Interested in
 - uncommon = number of Posts created by that Person, such that the Post has no Tag that start Person is Interested in
 - similarity = common - uncommon
- **Parameters:**

Person.id	ID
month	32-bit Integer // between 1-12
- **Results:**

Person.id	ID
Person.firstName	String
Person.lastName	String
similarity	32-bit Integer
Person.gender	String
Person-isLocatedIn->Place.name	Sting
- **Sort:**
 - 1st similarity (descending)
 - 2nd Person.id (ascending)
- **Limit:**
 - 10

11. Job referral

- **Description:** Given a start Person, find that Person's friends and friends of friends (excluding start Person) who started Working in some Company in a given Country, before a given date (year).
- **Parameters:**

Person.id	ID
Country.name	String
year	32-bit Integer
- **Results:**

Person.id	ID
Person.firstName	String
Person.lastName	String
Person-worksAt->Organization.name	String
Person-worksAt->.worksFrom	32-bit Integer
- **Sort:**
 - 1st Person-worksAt->.worksFrom (ascending)
 - 2nd Person.id (ascending)
 - 3st Person-worksAt->Organization.name (descending)
- **Limit:**
 - 10

12. Expert search

- **Description:** Given a start Person, find the Comments that this Person's friends made in reply to Posts, considering only those Comments that are immediate (1-hop) replies to Posts, not the transitive (multi-hop) case. Only consider Posts with a Tag in a given TagClass or in a descendent of that TagClass. Count the number of these reply Comments, and collect the Tags (with valid tag class) that were attached to the Posts they replied to. Return Persons with at least one reply, the reply count, and the collection of Tags.
- **Parameters:**

Person.id	ID
TagClass.name	String
- **Results:**

Person.id	ID
Person.firstName	String
Person.lastName	String
{Tag.name}	{String}
count	32-bit Integer // number of reply Comments
- **Sort:**
 - 1st count (descending)
 - 2nd Person.id (ascending)
- **Limit:**
 - 20

13. Single shortest path

- **Description:** Given two Persons, find the shortest path between these two Persons in the subgraph induced by the Knows relationships. Return the length of this path.
 - -1 : no path found
 - 0: start person = end person
 - > 0: regular case
- **Parameters:**

Person.id	ID	// person 1
Person.id	ID	// person 2
- **Results:**

length	32-bit Integer
--------	----------------

- **Sort:**

-

- **Limit:**

-

14. Weighted/unweighted paths

- **Description:** Given two Persons, find all (unweighted) shortest paths between these two Persons, in the subgraph induced by the Knows relationship. 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. The weight for a pair of Persons is calculated such that every reply (by one of the Persons) to a Post (by the other Person) contributes 1.0, and every reply (by ones of the Persons) to a Comment (by the other Person) contributes 0.5. Return all the paths with shortest length, and their weights.

- **Parameters:**

Person.id ID // person 1

Person.id ID // person 2

- **Results:**

[Person.id] [ID] // Identifiers representing an ordered sequence of the Persons in the path

weight 64-bit Float

- **Sort:**

1st weight (descending) // The order of paths with the same weight is unspecified

- **Limit:**

-

3.2.3 Short Reads Query Descriptions

1. Person Profile

- **Description:** Given a start Person, retrieve their first name, last name, birthday, IP address, browser, and city of residence.

- **Parameters:**

Person.id ID

- **Results:**

Person.firstName String

Person.lastName String

Person.birthday Date

Person.locationIP String

Person.browserUsed String

Person-isLocatedIn->Place.id 32-bit Integer

Person.gender String

Person.creationDate DateTime

- **Sort:**

-

- **Limit:**

-

2. Person Recent Messages

- **Description:** Given a start Person, retrieve the last 10 Messages created by that user. For each message, return that message, the original post in its conversation, and the author of that post. If any of the Messages is a Post, then the original Post will be the same Message, i.e. that Message will appear twice in that result.

- **Parameters:**

Person.id ID

- **Results:**

Message.id	64-bit Integer
Message.content or Post.imageFile	String
Message.creationDate	DateTime
Post.id or Comment-replyOf*->Post.id	ID
Post-hasCreator->Person.id or Comment-replyOf*->Post-hasCreator->Person.id	ID
Post-hasCreator->Person.firstName or Comment-replyOf*->Post-hasCreator->Person.firstName	String
Post-hasCreator->Person.lastName or Comment-replyOf*->Post-hasCreator->Person.lastName	String

- **Sort:**
 - 1st Message.creationDate (descending)
 - 2nd Message.id (descending)

- **Limit:**

-

3. Person Friends

- **Description:** Given a start Person, retrieve all of their friends, and the date at which they became friends.

- **Parameters:**

Person.id	ID
-----------	----

- **Results:**

Person.id	ID
Person.firstName	String
Person.lastName	String
Knows.creationDate	DateTime

- **Sort:**
 - 1st Knows.creationDate (descending)
 - 2nd Person.id (ascending)

- **Limit:**

-

4. Message Content

- **Description:** Given a Message, retrieve its content and creation date.

- **Parameters:**

Message.id	ID
------------	----

- **Results:**

Message.creationDate	ID
Message.content or Post.imageFile	String

- **Sort:**

-

- **Limit:**

-

5. Message Creator

- **Description:** Given a Message, retrieve its author.

- **Parameters:**

Message.id	ID
------------	----

- **Results:**

Message-hasCreator->Person.id	ID
Message-hasCreator->Person.firstName	String
Message-hasCreator->Person.lastName	String

- **Sort:**

-

- **Limit:**

-

6. Message Forum

- **Description:** Given a Message, 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.

- **Parameters:**

Message.id ID

- **Results:**

Message<-containerOf-Forum.id	ID
Message<-containerOf-Forum.title	String
Message<-containerOf-Forum-hasModerator->Person.id	ID
Message<-containerOf-Forum-hasModerator->Person.firstName	String
Message<-containerOf-Forum-hasModerator->Person.lastName	String

- **Sort:**

-

- **Limit:**

-

7. Message Replies

- **Description:** Given a Message, retrieve the (1-hop) Comments that reply to it. In addition, return a boolean flag indicating if the author of the reply knows the author of the original message. If author is same as original author, return false for "knows" flag.

- **Parameters:**

Message.id ID

- **Results:**

Message<-replyOf-Comment.id	ID
Message<-replyOf-Comment.content	String
Message<-replyOf-Comment.creationDate	DateTime
Message-hasCreator->Person.id	ID
Message-hasCreator->Person.firstName	String
Message-hasCreator->Person.lastName	String

- **Sort:**

1st Message<-replyOf-Comment.creationDate (descending)
2nd Message-hasCreator->Person.id (ascending)

- **Limit:**

-

3.2.4 Update Query Descriptions

1. Add Person

- **Description:** Add a Person to the social network.
- **Parameters:**

Person.id	ID
Person.firstName	String
Person.lastName	String
Person.gender	String
Person.birthDay	Date
Person.creationDate	DateTime
Person.locationIp	String
Person.browserUsed	String
Person-isLocatedIn->City.id	ID
Person.speaks	{ String }
Person.emails	{ String }
Person-hasInterest->Tag.id	{ ID }
{ Person-studyAt->University.id,	
Person-studyAt->.classYear }	{ ID, 32-bit Integer }
{ Person-workAt->Company.id,	
Person-workAt->.workFrom }	{ ID, 32-bit Integer }

2. Add Post Like

- **Description:** Add a Like to a Post of the social network.
- **Parameters:**

Person.id	ID
Post.id	ID
Person-likes->.creationDate	DateTime

3. Add Comment Like

- **Description:** Add a Like to a Comment of the social network.
- **Parameters:**

Person.id	ID
Comment.id	ID
Person-likes->.creationDate	DateTime

4. Add Forum

- **Description:** Add a Forum to the social network.
- **Parameters:**

Forum.id	ID	// person 1
Forum.title	String	// person 2
Forum.creationDate	DateTime	
Forum-hasModerator->Person.id	{ ID }	
Forum-hasTag->Tag.id	{ ID }	

5. Add Forum Membership

- **Description:** Add a Forum membership to the social network.
- **Parameters:**

Person.id	ID
Person-hasMember->Forum.id	ID
Person-hasMember->.joinDate	DateTime

6. Add Post

- **Description:** Add a Post to the social network.
- **Parameters:**

Post.id	ID
Post.imageFile	String
Post.creationDate	DateTime
Post.locationIp	String
Post.browserUsed	String
Post.language	String
Post.content	Text
Post.length	32-bit Integer
Post-hasCreator->Person.id	ID
Forum-containerOf->Post.id	ID
Post-isLocatedIn->Country.id	ID
{Post-hasTag->Tag.id}	{ID}

7. Add Comment

- **Description:** Add a Comment replying to a Post/Comment to the social network.
- **Parameters:**

Comment.id	ID	
Comment.creationDate	DateTime	
Comment.locationIp	String	
Comment.browserUsed	String	
Comment.content	Text	
Comment.length	32-bit Integer	
Comment-hasCreator->Person.id	ID	
Comment-isLocatedIn->Country.id	ID	
Comment-replyOf->Post.id	ID	// -1 if the comment is a reply of a comment.
Comment-replyOf->Comment.id	ID	// -1 if the comment is a reply of a post.
{Comment-hasTag->Tag.id}	{ID}	

8. Add Friendship

- **Description:** Add a friendship relation to the social network
- **Parameters:**

Person.id	ID	// person 1
Person.id	ID	// person 2
Person-knows->.creationDate	DateTime	

3.2.5 Substitution parameters

Together with the dataset, DBGEN 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 updates.

3.2.6 Load Definition

LDBC-SNB 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 3.1 shows the frequencies assigned to each query type for SF1. The frequencies of the different scale factors can be found in Appendix A.1.

Query Type	freq	Query Type	freq
Query 1	26	Query 8	45
Query 2	37	Query 9	157
Query 3	69	Query 10	30
Query 4	36	Query 11	16
Query 5	57	Query 12	44
Query 6	129	Query 13	19
Query 7	87	Query 14	49

Table 3.1: Frequencies for each query type for SF1.

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 complex reads and short reads. 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. 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 3.1). Therefore, different throughputs can be tested while maintaining the relative ratios between the different query types.

3.3 Business Intelligence Workload

3.3.1 Choke Points

3.3.2 Business Intelligence Queries

1. Posting summary

- **Description:** Given a date, find all Messages created before that date.
Group them by a 3-level grouping:
 - (a) Year of creation
 - (b) Each year, group into message types: Posts, Comments
 - (c) Each year-type group, split into four groups based on length of their content:
 - $0 \leq \text{length} < 40 \rightarrow \text{'short'}$
 - $40 \leq \text{length} < 80 \rightarrow \text{'one liner'}$
 - $80 \leq \text{length} < 160 \rightarrow \text{'tweet'}$
 - $160 \leq \text{length} \rightarrow \text{'long'}$
- **Parameters:**
date DateTime
- **Results:**

year	32-bit Integer	
Message.type	String	
Message.length	32-bit Integer	
category	32-bit Integer	
count	32-bit Integer	// Total number of Messages (Posts/Comments) in that group
average	32-bit Integer	// Average length of the Message content in that group
sum	32-bit Integer	// Sum of all message content lengths
percentage	32-bit Float	// Message count in group as percentage of all messages created before given date

- **Sort:**

1st	year	DESC
2nd	message type (Post 1st, Comment 2nd)	ASC
3rd	size category (short 1st, one-liner 2nd, tweet 3rd, long 4th)	ASC

2. Top tags for country, age, gender, time

- **Description:** Select all Messages (Posts & Comments) created between date1-date2 (inclusive) by persons located in country1 or country2. Select the creators (Persons) and the Tags of these Messages. Split these Persons, Tags and Messages into a 5-level grouping:

- Name of country of person
- Month message was created
- Gender of person
- Age group of person, defined as:
 - Years between person's birthday and end of simulation (2013-01-01), divided by 5, rounded down
- Name of tag attached to message

Only return groups where number of messages is greater than 100

- **Parameters:**

date1	Date
date2	Date
country1	String
country2	String

- **Results:**

Country.name	String
month (1-12)	32-bit Integer
gender	String
age group	32-bit Integer
Tag.name	String
message count	64-bit Integer

- **Sort:**

1st	message count	DESC
2nd	Tag.name	ASC
3rd	age group	ASC
4th	Person.gender	ASC
5th	month	ASC
6th	Country.name	ASC

- **Limit:** 100

3. Tag Evolution

- **Description:** Given a year and a month, find the Tags that were used in Messages during the given month of the given year, and the Tags that were used during the month after the given month of the given year. For both months, compute the count of Messages that used each of the Tags.

- **Parameters:**

year	32-bit Integer
month (1-12)	32-bit Integer

- **Results:**

For each of the tags:

Tag.name	String	
count_month1	32-bit Integer	// occurrences of Tag during year-month 1
count_month2	32-bit Integer	// occurrences of Tag during year-month 2
diff	32-bit Integer	// difference between occurrences of this Tag in month 1 and month 2

- **Sort:**

1st	difference	DESC
2nd	Tag.name	ASC

4. Popular topics in a Country

- **Description:** Given a TagClass and a Country, find all the Forums created in the given Country, containing at least one Post with Tags belonging directly to the given TagClass. The location of a Forum is identified by the location of the Forum's moderator.
- **Parameters:**

TagClass.name	String
Country.name	String
- **Results:**

For each Forum:

Forum.id	64-bit Integer
Forum.title	String
Forum.creationDate	DateTime
Forum-hasModerator->Person.id	64-bit Integer
count	32-bit Integer // Post count in Forum with ≥ 1 Tags belonging to TagClass
- **Sort:**

1st	count	DESC
2nd	Forum.id	ASC
- **Limit:** 20

5. Top posters in a Country

- **Description:** Find the most popular Forums for a given Country, where the popularity of a Forum is measured by the number of members that Forum has from the given Country. For each member of the 100 most popular Forums, count the number of Posts they made in any of those (most popular) Forums. Group persons by:
 - Person.id
 - Person.firstName
 - Person.lastName
 - Person.creationDate
- **Parameters:**

Country.name	String
--------------	--------
- **Results:**

Person.id	64-bit Integer
Person.firstName	String
Person.lastName	String
Person.creationDate	DateTime
count	32-bit Integer // Posts created by Person in 100 most popular Forums
- **Sort:**

1st	count	DESC
2nd	Person.id	ASC
- **Limit:** 100

6. Most active posters on a given topic

- **Description:** Get Persons who have created a Message (Post or Comment) with a given Tag. Each Person has a score, computed as follows: Count of Messages with the given Tag PLUS count of Likes and Comments in reply of their Messages with the given Tag The sum is weighted as follows: Messages are multiplied by 1, Comments to Messages are multiplied by 2, Likes are multiplied by 10.
- **Parameters:**

Tag.id	32-bit Integer
--------	----------------

- **Results:**

Person.id	64-bit Integer
reply_count	32-bit Integer
like_count	32-bit Integer
message_count	32-bit Integer
score	32-bit Integer
- **Sort:**

1st	score	DESC
2nd	Person.id	ASC
- **Limit:** 100

7. Most authoritative Person on a given topic

- **Description:** Given a Tag, find all Persons that ever created a Message with the given Tag. For each of these Persons compute their "authority score". The "authority score" for a given Person is defined as the sum of "popularity scores" of the Persons that liked any of that Person's Messages with the given Tag. A Person's "popularity score" is defined as the total number of likes on all of their Messages.
- **Parameters:**

Tag.name	String
----------	--------
- **Results:**

For each Person:

Person.Id	ID
authorityScore	32-bit Integer

Return the top 100 Persons sorted the authorityScore (descending) and Person.Id (ascending).

8. Related Topics

- **Description:** Find all Messages that have a given Tag. Find the Tags attached to replies of these Messages, but only those replies that do not have the given Tag. Group the Tags by name, and get the count of replies in each group.
- **Parameters:**

Tag.name	String
----------	--------
- **Results:**

For each group:

Tag.name	String	
count	32-bit Integer	// The count of replies with the Tag

Return the top-100 Tags, sorted by count (descending) and by Tag.name (ascending).

9. Forum with related Tags

- **Description:** Given two tagClasses, find those forums that contain Posts with Tags belonging tagClass1 and contain Posts with Tags belonging to tagClass2 (not transitive). Consider the forums with a number of members greater than a given threshold. For every such forum, count the number of Posts that have a Tag from tagclass t1 (count1), and the number of posts that have a tag from the tagclass2.
- **Parameters:**

TagClass1.name	String
TagClass2.name	String
threshold	32-bit Integer
- **Results:**

For each Forum:

Forum.id	ID	
count1	32-bit Integer	// The count of Posts with at least one Tag belonging to TagClass1
count2	32-bit Integer	// The count of Posts with at least one Tag belonging to TagClass2

10. Central Person for a topic

- **Description:** Given a Tag, find all Persons that are either interested on the Tag, or have written a Post with the given Tag. For each Person, compute the score as the sum of the following two aspects:
 - 100, if the Person has this tag as their interest, or 0 otherwise

- number of posts by this person with the given tag

- **Parameters:**

Tag.name String

- **Results:**

For each Person:

Person.Id ID

score 32-bit Integer

friendsScore 32-bit Integer // The sum of the score of the Person's friends

Return the top-100 Persons, sorted the sum of score and friendsScore (descending), and Person.id (ascending)

11. Unrelated replies

- **Description:** Find those Persons of a given country that replied to any Message, such that the reply does not have any Tag in common with the Message. Consider only those replies not containing any word from a given blacklist. For each Person and valid reply, retrieve the Tags associated with the reply, and retrieve the number of likes on the reply.

Group the results by:

- First, Person.id
- Second, Tag.name

- **Parameters:**

Country.name 32-bit Integer

blacklist String

- **Results:**

For each group:

Person.id ID

Tag.name String

countLikes 32-bit Integer // The count likes to replies with that Tag

countReplies 32-bit Integer // The count of replies with that Tag

Return top-100 results, sorted by count (descending) and Person.id (ascending)

12. Trending Posts

- **Description:** Find all Message created after a given date, that received more than a given number of likes.

- **Parameters:**

creationDate Date

likeCount 32-bit Integer

- **Results:**

For each Message:

Message.id ID

Message.creationDate DateTime

Person.firstName String // The first name of the Message creator

Person.lastName String // The last name of the Message creator

likeCount 32-bit Integer // The number of likes the message received

Return top-100 by count (descending) and Message.id (ascending).

13. Popular Tags per month in a Country

- **Description:** Find all Messages in a given Country, as well as their Tags. Group the results by:

- First, message creation year
- Second, message creation month

For each group, find the 5 most popular Tags, where popularity is the number of Messages (from within the same group) where the Tag appears.

- **Parameters:**

Country.name String

```

For each group:
  Year      32-bit Integer
  Month     32-bit Integer           // From 1 to 12
  popularTags {<Tag.name, popularity>} // Sorted by popularity (descending), then ascending
                                           by Tag.name (ascending).
                                           Popularity is the count of Messages the Tag appears
Return top-100, sorted by year (descending) and then by month (ascending).

```

- **Description:** Find all the threads initiated in a given time interval (closed interval). A thread consists of a Post and the reply tree to that post. The date when a thread is initiated is the creation date of the Post initiating the thread. Group the results by creator (Person) id.
- **Parameters:**
 - begin Date
 - end Date
- **Results:**

For each group:

Person.id	ID
Person.firstName	String
Person.lastName	String
threadCount	32-bit Integer // The number of threads initiated by that Person
messageCount	32-bit Integer // The number of Messages created in all the threads initiated by that Person

Return top-100 results, sorted messageCount (descending) and by Person.id (ascending).

- **Description:** Given a Country, find out all the Persons of the Country whose number of friends in the given Country equals the average number of friends Persons of the given Country have in the given Country.
- **Parameters:**
Country.name
- **Results:**
For each Person:
Person.Id ID
count 32-bit Integer // The number of friends
Return top-100 results sorted Person.id (ascending).

- **Description:** Given a Person, find all friends connected to that person by a path through the knows relation, that live in a given country. From these friends, select their Posts that containing a Tag belonging to a given tagClass (direct relation) and the Tags.
Group the results:
 - First, by Tag.name
 - Second, by Person.id
- **Parameters:**

Person.id	ID
Country.name	String
TagClass.name	String
- **Results:**

For each group:

Person.name	ID
Tag.name	String
postCount	32-bit Integer // The number of Posts created by the Person containing that Tag

Return the top-100 results, sorted postCount (descending), by Tag.name (ascending) and by Person.id (ascending).

17. Friend triangles

- **Description:** For a given Country, count all the triangles formed by knows relationship of Persons in that Country.
- **Parameters:**
Country.name String
- **Results:**
count 32-bit Integer

18. How many Persons have a given number of Posts

- **Description:** For each Person, count the number of Posts they made after a given date. Group the results by the number of Posts created
- **Parameters:**
creationDate Date
- **Results:**
For each group: count 32-bit Integer

19. Stranger's interaction

- **Description:** For all the Persons born after a certain date, find all the strangers they interacted with. We define strangers as Persons that are not friends. Consider only strangers that are members of Forums tagged with tagclass1 and are members of Forums tagged with tagclass2. We define interaction as follows: a Person replied to a Post or a Comment by the stranger. For each Person, count the number of strangers they interacted with and total number of messages exchanged with all strangers.
- **Parameters:**
date Date
TagClass.name String
TagClass.name String
- **Results:**
Person.id ID
strangersCount 32-bit Integer Return Top-100 results sorted by messageCount (descending) and
messageCount 32-bit Integer
ascending by Person.id (ascending).

20. High level topics

- **Description:** Find all the TagClasses that are a children of at least one TagClass of a given set of TagClasses. For each of these children TagClasses, count the number of Messages that have a Tag that belongs to the children TagClass or any of its sub classes.
- **Parameters:**
TagClasses.name String
- **Results:**
For each children TagClass: TagClass.name String Return top-100 results, sorted by
messageCount 32-bit Integer
messageCount (descending) and by TagClass.name (ascending)

21. Zombies in a Country

- **Description:** A zombie is a Person who produced less Messages than the number of months her profile has existed (only consider full months). A zombie score of a Person is the ratio between Likes that this Person's Messages received from zombies and the total number of Likes that the Persons's Messages received. For both cases (zombie-likers and all likers) only count likes received by profiles that existed for at least one month. For all persons in a given country, count their zombie scores.
- **Parameters:**
Country.name String
endDate Date

- **Results:**

Person.Id	ID	
zombieLikeCount	32-bit Integer	
realLikeCount	32-bit Integer	
zombieScore	32-bit Integer	

(ascending). Return top-100, sorted zombieScore (descending) and Person.id

22. International dialog

- **Description:** Given two countries X and Y, find all the Persons from country X that communicated the most with Persons from country Y.
We define the degree of communication between two Persons p1 and p2 as the sum of the score for the following interactions, where each interaction is scored as follows (0 if none of the cases apply)
 - p1 has left a Comment on p2's Comment or post, this has the score 4
 - p1 has created a post or comment that was commented by p2, this has the score 1
 - p1 and p2 are friends, score 15
 - p1 liked a post or comment by p2, score 10
 - p1 has created a post or comment that was liked by p2, score 1

For all pairs of people (p1, p2) such that p1 is in some city of country X and p2 is anywhere in country Y, find the pairs with the largest communication score (one pair per any city in country X)
- **Parameters:**

Country1.name	String
Country2.name	String
- **Results:**

Person1.id	ID
Person2.id	ID
score	32-bit Integer

Sort results score (descending), Person1.id (ascending) and by Person2.id (ascending).

23. Holiday destinations

- **Description:** Find all the destinations where Persons from the given Country are traveling to. Traveling is derived from Messages, i.e. a Person has created a Message in a country which is not their country of residence (given country) Count the amount of Messages created in foreign Countries. Group the results by:
 - First, by name of the destination Country
 - Second, by the month in which people traveled there.
- **Parameters:**

Country.name	String
--------------	--------
- **Results:**

For each group:

Country.name	String
messageCount	32-bit Integer
month	32-bit Integer // Between 0 and 12

Return top-100 results, sorted messageCount (descending) and Country.name (ascending).

24. Messages by topic and continent

- **Description:** Find all Messages, tagged with a Tag from a given TagClass. For every Message, find the continent where it was published (can be null) Group all the Messages:
 - First, by year,
 - Second, by month of their publication
 - Third, by the continent.
- **Parameters:**

TagClass.name	String
---------------	--------

- **Results:**

For each group:

messageCount	32-bit Integer
likeCount	32-bit Integer
year	32-bit Integer
month	32-bit Integer
Continent.name	String

Sort the results year (ascending), month (ascending) and Continent name (ascending).

4 AUDITING RULES

This chapter describes the rules to audit benchmark runs, that is, what techniques are allowed and what are not, what must be provided to the auditor and guidelines for the auditors to perform the audit.

4.1 Preparation

The first step when doing an audit is to determine the versions of the following items that will be used for the benchmark:

- The benchmark specification
- The data generator
- The driver

These 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 scale factor must be provided, including the update streams and substitution parameters.

4.1.1 Collect System Details

The next step is to collect the technical and pricing details of the system under test. This includes the following items:

- Common name of the system, e.g. Dell PowerEdge xxxx.
- Type and number of CPUs, cores/threads per CPU, clock frequency and cache hierarchy characteristics (levels, size per level, etc.).
- The amount of the system's memory, type and frequency.
- The disk controller or motherboard type if disk controller is on the motherboard.
- For each distinct type of secondary storage device, the number and characteristics of the device.
- The number and type of network controllers.
- The number and type of network switches. Wiring must be disclosed.
- Date of availability of the system.

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, no network need be reported. The price of the hardware in question must be disclosed and 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.

Besides hardware characteristics, also software details must be collected:

- The DBMS and operating system name and versions.
- Installation and configuration information of both the DBMS and operating system, which must be provided by the test sponsor.
- Price of the software license used, which can be tied to the number of concurrent users or size of data.
- Date of availability of the software.

Also, the test sponsor must provide all the source code relevant to the benchmark.

4.1.2 Setup the Benchmark Environment

Once all the information has been collected, the auditor will setup the environment to perform the benchmark run. This setup includes configuring the following items:

- Setup the LDBC Data generator in the test machine if datasets are not available from a trusted source.
- Setup the LDBC driver with the connectors provided by the test sponsor. The test sponsor must provide the configuration parameters to configure the driver (tcr, number of threads, etc.).
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. 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. These can be found in Appendix A.1. If the driver will be executed on a separate machine, gather the characteristics of that machine in the same way as specified above.

4.1.3 Load Data

The test sponsor must provide all the necessary documentation and scripts to load the dataset into the database to test. 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. The auditor will measure the time to load the data, which will be disclosed.

4.2 Running the Benchmark

Running the benchmark consists of three separate parts: (1) validating the query implementations, (2) warming the database and (3) performing the benchmark run. The queries are validated by means of the official validation datasets provided by LDBC consortium in their official software repositories. The auditor must load the provided dataset and run the driver in validation mode, which will test that the queries provide the official results.

The warmup can be performed either using the LDBC driver or externally, and the way it is performed must be disclosed.

A valid benchmark run must last at least 2 hours of simulation time (datagen time). Also, in order to be valid, a benchmark run needs to meet the following requirements. 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:

$$actual_start_time - scheduled_start_time < 1 \text{ second}$$

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

4.3 Recovery

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 warmup period, an execution of the benchmark will be performed under the same terms as in the previous measured run.

At an arbitrary point close to 2 hours of simulation execution time, the machine will be disconnected. 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.

4.4 Serializability

Optionally, the test sponsor can execute update queries atomically. The auditor will verify that serializability is guaranteed.

REFERENCES

- [1] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat-Pérez, M. Pham, and P. A. Boncz. The LDBC social network benchmark: Interactive workload. In T. K. Sellis, S. B. Davidson, and Z. G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 619–630. ACM, 2015.

A SCALE FACTOR STATISTICS

A.1 Scale Factor Statistics

Query Type	SF1	SF3	SF10	SF30	SF100	SF300	SF1000
Query 1	26	26	26	26	26	26	26
Query 2	37	37	37	37	37	37	37
Query 3	69	79	92	106	123	142	165
Query 4	36	36	36	36	36	36	36
Query 5	57	61	66	72	78	84	91
Query 6	129	172	236	316	434	580	796
Query 7	87	72	54	48	38	32	25
Query 8	45	27	15	9	5	3	1
Query 9	157	209	287	384	527	705	967
Query 10	30	32	35	37	40	44	47
Query 11	16	17	19	20	22	24	26
Query 12	44	44	44	44	44	44	44
Query 13	19	19	19	19	19	19	19
Query 14	49	49	49	49	49	49	49

Table A.1: Frequencies for each query and SF.