Supporting Dynamic Graphs and Temporal Entity Deletions in the LDBC Social Network Benchmark's Data Generator

Jack Waudby
Newcastle University, School of Computing
j.waudby2@newcastle.ac.uk

Benjamin A. Steer
Queen Mary University London
b.a.steer@qmul.ac.uk

Arnau Prat-Pérez
Sparsity Technologies
DAMA UPC
aprat@ac.upc.edu

Gábor Szárnyas*
Budapest University of Technology and Economics
Dept. of Measurement and Information Systems
szarnyas@mit.bme.hu

ABSTRACT

Many data processing pipelines operate on highly-connected data sets that can be efficiently modelled as graphs. These graphs are rarely static, but rather change rapidly and often exhibit dynamic, temporal, or streaming behaviour. During the last decade, numerous graph benchmarks have been proposed, which cover a significant portion of the features required in practical use cases. However, whilst these benchmarks often contain some update operations, none of them include complex deletions, which makes it challenging to test the performance of graph processing systems under such operations. To address this limitation, we have extended the LDBC Social Network Benchmark (SNB) by introducing lifespan attributes for the creation and deletion dates of its entities. We have defined constraints for selecting these dates from intervals that ensure that the graph always satisfies the cardinality constraints prescribed by the schema and other semantic constraints of the social network domain. We have implemented the proposed lifespans in the SNB generator.

ACM Reference Format:
Jack Waudby, Benjamin A. Steer, Arnau Prat-Pérez, and Gábor Szárnyas. 2020. Supporting Dynamic Graphs and Temporal Entity Deletions in the LDBC Social Network Benchmark's Data Generator. In *3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA'20), June 14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3398682.3399165

*Also with the MTA-BME Lendület Cyber-Physical Systems Research Group.

GRADES-NDA'20, June 14, 2020, Portland, OR, USA

1 INTRODUCTION

Recognizing the potential value that can be extracted from rich, highly-connected data sets, organizations often complement their existing data processing pipelines with dedicated graph processing systems such as *graph databases* [7], *graph analytical frameworks* [4], and *graph streaming engines* [6]. While this trend has led to an abundance of graph processing systems in the last decade, the maturity and performance of these system is often questionable. To stimulate competition between vendors and allow fair comparison of their systems, several benchmarks have been proposed to capture realistic graph processing workloads, including those of the Linked Data Benchmark Council (LDBC). LDBC's Social Network Benchmark (SNB) suite [2] defines two workloads: *Interactive* [14] targets transactional systems with queries accessing a small portion of the database and insert operations, while the current version of the *Business Intelligence* [29] workload features complex aggregation-heavy operations for graph-based decision support but no updates. These workloads require a significant portion of features in graph systems that are relevant in popular applications (such as social networking, fraud detection, and recommendation). However, they still lack a number of features, most crucially *transactional delete operations* necessary for handling fully dynamic graphs are common [25] and legally required by the EU's *General Data Protection Regulation* (GDPR) regulation [26].

Challenge for systems. To support deletions, graph processing systems need to solve numerous technical challenges:

(1) Users should be able to *express deletion operations* using the database API, preferably using a high-level declarative query language with clear semantics [15].
(2) Deletion operations *limit the algorithms and data structures* that can be used by a system. Certain dynamic

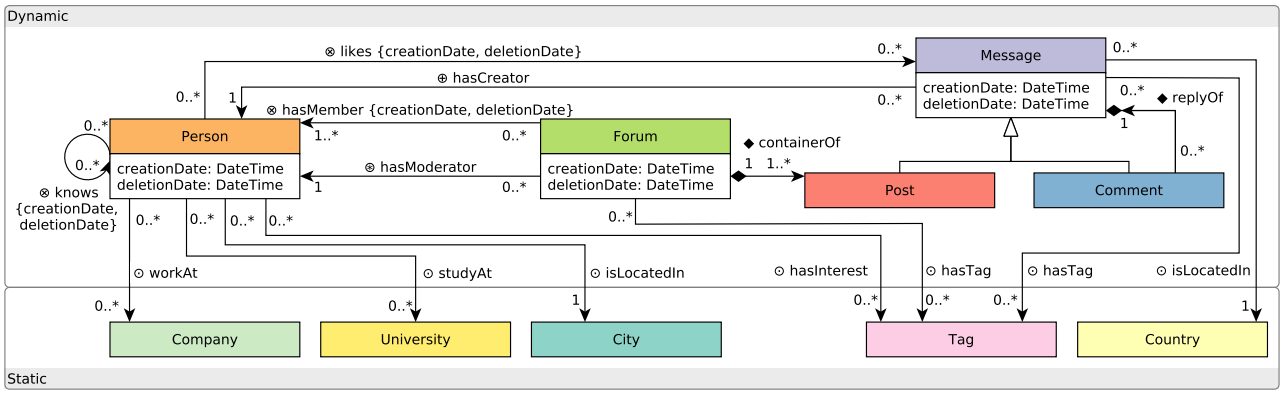


Figure 1: Partial LDBC social network schema depicted using a UML-like notation, focusing on the dynamic part of the graph (non-lifespan attributes and irrelevant entity types in the static part are omitted). In the dynamic part of the graph, all *node types* and all ⊗ *edge types with many-to-many cardinality between two dynamic node types* have individual lifespan attributes. The ◆ *containment edges* get their lifespan attributes from the contained endpoint (whose lifespan attributes are constrained by the container endpoint). ⊕ `hasCreator` follows the semantics of containment edges (even though it does not express an explicit containment relation). ⊛ `hasModerator` is a one-to-many edge that is treated separately depending on the `Forum` category (see Section A.1). The ⊙ *edges between a dynamic and a static node* get their lifespan attributes from their dynamic endpoint.

graph algorithms are significantly more expensive to recompute in the presence of deletes [23] or only support either insertions or deletions but not both [24]. A number of updatable matrix storage formats only support efficient insertions but not deletions [12]. Meanwhile some graph databases might be able to exploit indexes to speed up deletions [7].

(3) *Distributed graph databases* need to employ specialized protocols to enforce consistency of deletions [30].

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 paper focuses on the challenge of generating deletion operations within LDBC's synthetic data generator, specifically, to extend the generator with support for 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. Supporting the generation of dynamic entities poses numerous challenges:

(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 [19]. 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 [18]. 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) **Scalability.** A graph with dynamic entities should be generated at scale (up to billions of edges).

The main contribution of this paper is to address validity, with realism, scalability, and the extraction of deletion operations left for future work – we briefly discuss our intended approach in Section 4. To ensure validity, we have defined constraints that describe the permissible intervals for selecting *creation* and *deletion dates* of the graph entities in the LDBC SNB's social network graph (Section 2).

2 LIFESPAN MANAGEMENT

In this section, we define the 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 2 and Figure 3.

Graph schema. The LDBC Datagen component [20, 21] 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 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. P_1, 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 [14] (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 Δ = 10s.

2.1 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:

- $*n_1 \in [\mathbf{SS},\ \mathbf{SE})$, the node must be created between the simulation start and the simulation end.
- $\dagger n_1 \in [*n_1 + \Delta,\ \mathbf{NC})$, the node must exist for at least Δ 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:

- $*e \in [\max(*n_1, *n_2),\ \min(\dagger n_1, \dagger n_2, \mathbf{SE}))$, 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 Δ 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 its endpoints already exist and deleted no later than them:

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

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.

2.2 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:

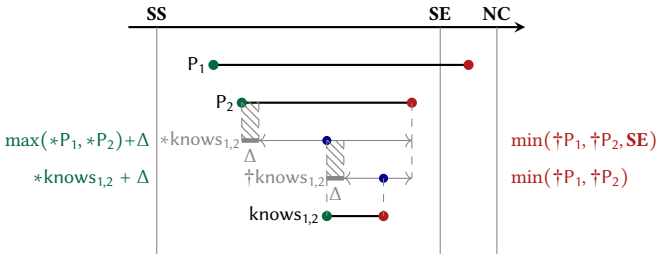
- $*p \in [\mathbf{SS},\ \mathbf{SE})$
- $\dagger p \in [*p + \Delta,\ \mathbf{NC})$

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.

2.2.1 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 2b and defined below:

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

(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* • and *deletion* • dates 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 Δ time. In case of deletion dates, it is used to ensure that the entity exists for at least Δ time.

Figure 2: Example graph and its intervals.

2.3 Forum and Message

The rules for `Forum` and `Message` nodes along with their edges are given in Section A.1 and Section A.2, respectively, and illustrated in Figure 3.

3 RELATED WORK

Graph generators. A recent survey [11] 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* [3] defines primitive delete operations targeting a single node or a single edge. *XGDBench* [13] defines an operation that deletes a single node. The *Social Network Intelligence BenchMark* (SIB) [9] (a precursor to LDBC SNB) requires the deletion of individual nodes (posts/photos). Finally, the *LDBC SNB's Interactive workload* [14] only uses insert operations and does not define any deletions.

Graph benchmarks. Several benchmarks have been proposed for semantic and graph databases [10]. Some of these perform deletions as part of their workflow but the complexity of these deletions is limited. The *Berlin SPARQL Benchmark* [8] defines deletions for a single table (Offers). The *LDBC Semantic Publishing Benchmark* [17] defines all its update operations (including deletions) on a single entity type, creative work. The *Train Benchmark* [28] uses the result set of each query to perform its "transformation" phase, which includes delete operations targeting an entity and its neighbourhood. The workload of the *BG Benchmark* [1] uses simple deletions targeting comment nodes and friendship edges. Recently, the authors of [22] extended the LDBC generator with support for lifespans encoded as integer intervals, applied to nodes, edges, and also properties. However, all nodes and edges were assigned a *deletion date* of ∞ and only some properties were assigned a finite deletion date.

4 CONCLUSION AND FUTURE WORK

In this paper, we have defined the constraints for producing valid lifespans for dynamic entities in the graph schema of the LDBC Social Network Benchmark [2]. These lifespans provide a fundamental blocking block in achieving our long term goal of incorporating delete operations into the LDBC Social Network Benchmark suite's workload specifications. We have incorporated the lifespans presented in this paper as part of the SNB data generator so that our dynamic graphs can be generated at scale.[1]

Our next step is to address the issue of realism in the generation process using studies conducted on real social network such as Facebook [5, 16], Twitter [19], and iWiW, a now-defunct social network that operated in Hungary between 2002 and 2014 [18]. We plan to ensure realistic correlations and distribution of entities over time using temporal graph analysis tools [27]. Turning the temporal graph into insert and delete operations also poses further challenges, including the distinction between *explicit deletions* and *implicit deletions* resulting from the cascading effect of other explicit deletions.

ACKNOWLEDGMENTS

The authors would like to thank Peter Boncz for his suggestions. J. Waudby was supported by the Engineering and Physical Sciences Research Council, Centre for Doctoral Training in Cloud Computing for Big Data [grant number EP/L015358/1]. B. Steer was supported by the Engineering and Physical Sciences Research Council and Alan Turing Institute [grant number EP/T001569/1]. G. Szárnyas was partially supported by the MTA-BME Lendület Cyber-Physical Systems Research Group. The parameterization of the data generator was determined with input from the the Development and Innovation Office of Hungary (NKFIH), National Research, grant number FK-128981.

[1] https://github.com/ldbc/ldbc_snb_datagen

REFERENCES

[1] Yazeed Alabdulkarim, Sumita Barahmand, and Shahram Ghandeharizadeh. 2018. BG: A scalable benchmark for interactive social networking actions. *Future Gener. Comput. Syst.* 85 (2018), 29–38. https://doi.org/10.1016/j.future.2018.02.031

[2] Renzo Angles, János Benjamin Antal, Alex Averbuch, Peter A. Boncz, Orri Erling, Andrey Gubichev, Vlad Haprian, Moritz Kaufmann, Josep-Lluís Larriba-Pey, Norbert Martínez-Bazan, József Marton, Marcus Paradies, Minh-Duc Pham, Arnau Prat-Pérez, Mirko Spasic, Benjamin A. Steer, Gábor Szárnyas, and Jack Waudby. 2020. The LDBC Social Network Benchmark. *CoRR* abs/2001.02299 (2020). http://arxiv.org/abs/2001.02299

[3] Timothy G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. 2013. LinkBench: A database benchmark based on the Facebook social graph. In *SIGMOD*. 1185–1196. https://doi.org/10.1145/2463676.2465296

[4] Omar Batarfi, Radwa El Shawi, Ayman G. Fayoumi, Reza Nouri, Seyed-Mehdi-Reza Beheshti, Ahmed Barnawi, and Sherif Sakr. 2015. Large scale graph processing systems: survey and an experimental evaluation. *Cluster Computing* 18, 3 (2015), 1189–1213. https://doi.org/10.1007/s10586-015-0472-6

[5] Eric P. S. Baumer, Phil Adams, Vera D. Khovanskaya, Tony C. Liao, Madeline E. Smith, Victoria Schwanda Sosik, and Kaiton Williams. 2013. Limiting, leaving, and (re)lapsing: An exploration of Facebook non-use practices and experiences. In *CHI*. ACM, 3257–3266. https://doi.org/10.1145/2470654.2466446

[6] Maciej Besta, Marc Fischer, Vasiliki Kalavri, Michael Kapralov, and Torsten Hoefler. 2019. Practice of Streaming and Dynamic Graphs: Concepts, Models, Systems, and Parallelism. *CoRR* abs/1912.12740 (2019). http://arxiv.org/abs/1912.12740

[7] Maciej Besta, Emanuel Peter, Robert Gerstenberger, Marc Fischer, Michal Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. 2019. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *CoRR* abs/1910.09017 (2019). http://arxiv.org/abs/1910.09017

[8] Christian Bizer and Andreas Schultz. 2009. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.* 5, 2 (2009), 1–24. https://doi.org/10.4018/jswis.2009040101

[9] Peter Boncz, Minh-Duc Pham, Orri Erling, Ivan Mikhailov, and Yrjana Rankka. 2013. Social Network Intelligence BenchMark. (2013). https://www.w3.org/wiki/Social_Network_Intelligence_BenchMark

[10] Angela Bonifati, George H. L. Fletcher, Jan Hidders, and Alexandru Iosup. 2018. A Survey of Benchmarks for Graph-Processing Systems. In *Graph Data Management, Fundamental Issues and Recent Developments*. Springer, 163–186. https://doi.org/10.1007/978-3-319-96193-4_6

[11] Angela Bonifati, Irena Holubová, Arnau Prat-Pérez, and Sherif Sakr. 2020. Graph Generators: State of the Art and Open Challenges. *CoRR* abs/2001.07906 (2020). https://arxiv.org/abs/2001.07906

[12] Federico Busato, Oded Green, Nicola Bombieri, and David A. Bader. 2018. Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices on GPUs. In *HPEC*. IEEE, 1–7. https://doi.org/10.1109/HPEC.2018.8547541

[13] Miyuru Dayarathna and Toyotaro Suzumura. 2014. Graph database benchmarking on cloud environments with XGDBench. *Autom. Softw. Eng.* 21, 4 (2014), 509–533. https://doi.org/10.1007/s10515-013-0138-7

[14] Orri Erling, Alex Averbuch, Josep-Lluís Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat-Pérez, Minh-Duc Pham, and Peter A. Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *SIGMOD*. ACM, 619–630. https://doi.org/10.1145/2723372.2742786

[15] Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Martin Schuster, Petra Selmer, and Hannes Voigt. 2019. Updating Graph Databases with Cypher. *PVLDB* 12, 12 (2019), 2242–2253. https://doi.org/10.14778/3352063.3352139

[16] Zoltán Kmetty and Renáta Németh. 2020. Which is your favorite music genre? A validity comparison of Facebook data and survey data. *CoRR* abs/2002.00501 (2020). https://arxiv.org/abs/2002.00501

[17] Venelin Kotsev, Nikos Minadakis, Vassilis Papakonstantinou, Orri Erling, Irini Fundulaki, and Atanas Kiryakov. 2016. Benchmarking RDF Query Engines: The LDBC Semantic Publishing Benchmark. In *BLINK at ISWC (CEUR Workshop Proceedings)*, Vol. 1700. CEUR-WS.org. http://ceur-ws.org/Vol-1700/paper-01.pdf

[18] László Lőrincz, Júlia Koltai, Anna Fruzsina Győr, and Károly Takács. 2019. Collapse of an online social network: Burning social capital to create it? *Soc. Networks* 57 (2019), 43–53. https://doi.org/10.1016/j.socnet.2018.11.004

[19] Seth A. Myers and Jure Leskovec. 2014. The bursty dynamics of the Twitter information network. In *WWW*. ACM, 913–924. https://doi.org/10.1145/2566486.2568043

[20] Minh-Duc Pham, Peter A. Boncz, and Orri Erling. 2012. S3G2: A Scalable Structure-Correlated Social Graph Generator. In *TPCTC*, Vol. 7755. Springer, 156–172. https://doi.org/10.1007/978-3-642-36727-4_11

[21] Arnau Prat-Pérez. 2017. LDBC SNB Datagen: Under the hood. In *9th LDBC TUC Meeting*. http://wiki.ldbcouncil.org/pages/viewpage.action?pageId=59277315&preview=/59277315/75431942/datagen_in_depth.pdf

[22] Shriram Ramesh, Animesh Baranawal, and Yogesh Simmhan. 2020. A Distributed Path Query Engine for Temporal Property Graphs. In *CCGRID*. IEEE/ACM.

[23] Liam Roditty. 2013. Decremental maintenance of strongly connected components. In *SODA*. SIAM, 1143–1150. https://doi.org/10.1137/1.9781611973105.82

[24] Liam Roditty and Uri Zwick. 2004. On Dynamic Shortest Paths Problems. In *ESA*, Vol. 3221. Springer, 580–591. https://doi.org/10.1007/978-3-540-30140-0_52

[25] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2020. The ubiquity of large graphs and surprising challenges of graph processing: Extended survey. *VLDB J.* 29, 2 (2020), 595–618. https://doi.org/10.1007/s00778-019-00548-x

[26] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. 2019. Understanding and Benchmarking the Impact of GDPR on Database Systems. *CoRR* abs/1910.00728 (2019). http://arxiv.org/abs/1910.00728

[27] Benjamin A. Steer, Félix Cuadrado, and Richard G. Clegg. 2020. Raphtory: Streaming analysis of distributed temporal graphs. *Future Gener. Comput. Syst.* 102 (2020), 453–464. https://doi.org/10.1016/j.future.2019.08.022

[28] Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. 2018. The Train Benchmark: Cross-technology performance evaluation of continuous model queries. *Softw. Syst. Model.* 17, 4 (2018), 1365–1393. https://doi.org/10.1007/s10270-016-0571-8

[29] Gábor Szárnyas, Arnau Prat-Pérez, Alex Averbuch, József Marton, Marcus Paradies, Moritz Kaufmann, Orri Erling, Peter A. Boncz, Vlad Haprian, and János Benjamin Antal. 2018. An early look at the LDBC Social Network Benchmark's Business Intelligence workload. In *GRADES-NDA at SIGMOD/PODS*. ACM, 9:1–9:11. https://doi.org/10.1145/3210259.3210268

[30] Jack Waudby, Paul Ezhilchelvan, Jim Webber, and Isi Mitrani. 2020. Preserving Reciprocal Consistency in Distributed Graph Databases. In *PaPoC at EuroSys*. ACM. https://doi.org/10.1145/3380787.3393675

A LIFESPAN MANAGEMENT FOR FORUM AND MESSAGE NODES

A.1 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 A.2). For all Forums the outgoing hasTag edges get their creation date and deletion date from $*forum$ and $\dagger forum$, respectively.

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.

- $*forum_g \in \left[*p + \Delta,\ \min(\dagger p, \mathbf{SE})\right)$
- $\dagger forum_g \in \left[*forum_g + \Delta,\ \mathbf{NC}\right)$

Group Moderator. The initial hasModerator hmd_g is the group creator. If the moderator leaves the group before it is deleted a new hasModerator edge for Person p_j is generated by replacing $*forum_g$ with $\dagger hmd_g$ in the intervals below.

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

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 Δ) and the minimum of the Person and Forum deletion dates.

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

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 Δ) 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 Δ. The deletion date is derived from the minimum of the Forum deletion date and knows deletion date.

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

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.

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

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 Δ) 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.

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

A.2 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.

A.2.1 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 A.2.3). The creation date for these is determined as follows:

$$*post \in \left[*hm + \Delta,\ \min(\dagger hm, \mathbf{SE})\right)$$

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 2ϕ-hour time window centered around a specific event time, flashmob event *fme*, in the middle of the window; there are ϕ hours each side of the specific event time.

$$*post \in \left[\max(*hm + \Delta, fme - \phi\,\mathrm{h}),\ \min(\dagger hm, fme + \phi\,\mathrm{h}, \mathbf{SE}) \right)$$

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

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

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

A.2.2 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 γ days of their parent message.

- $*comm \in \left[\max(*m, *hm) + \Delta,\ \min(*m + \gamma\,\mathrm{d}, \dagger hm, \mathbf{SE}) \right)$
- $\dagger comm \in \left[*comm + \Delta,\ \min(\dagger m, \dagger hm) \right)$

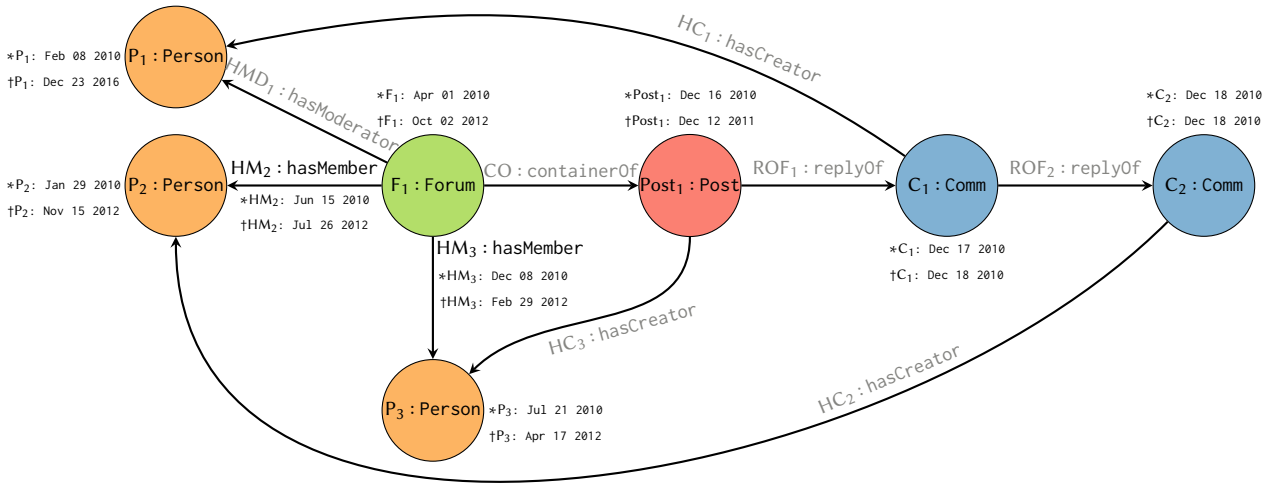
replyOf edge. `Comments` always have an outgoing `replyOf` edge with containment semantics, i.e. the target `Message` contains the `Comment`. These edges get the same lifespan as their source `Comment`.

A.2.3 likes. A `likes` edge *likes* can exist between `Person` p and `Message` m. Messages can only receive likes during a μ-day window after their creation at which point no more activity is generated.

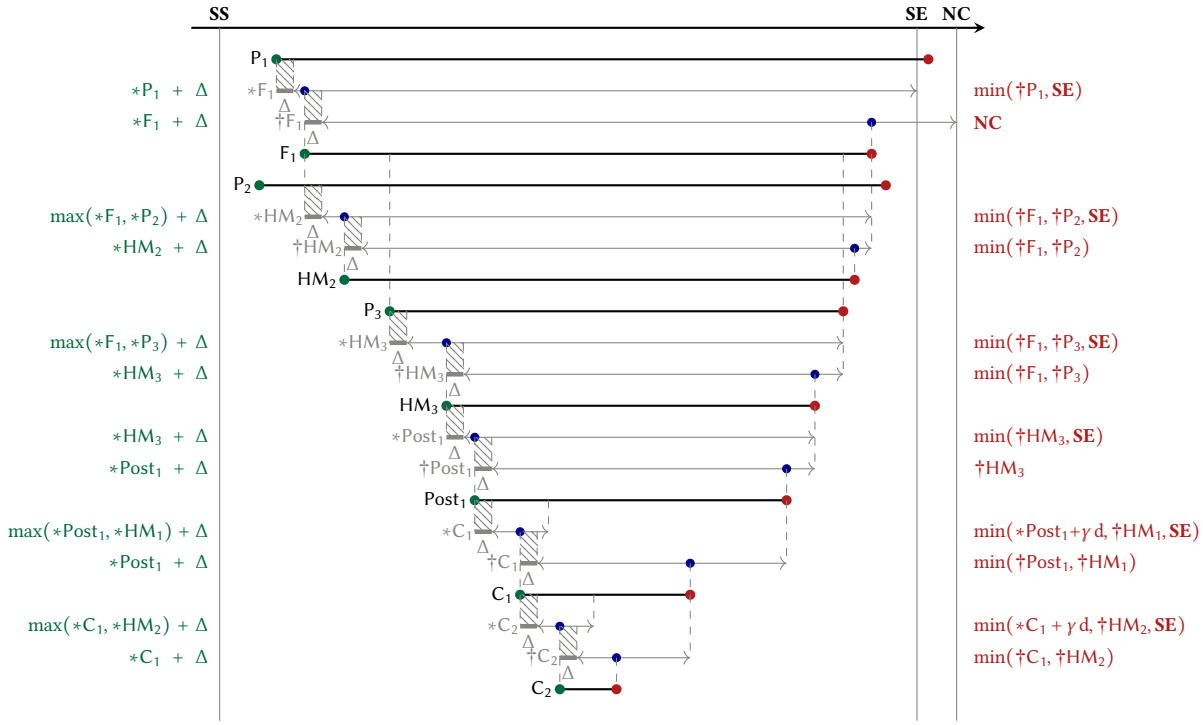
- $*likes \in \left[\max(*p, *m) + \Delta,\ \min(\dagger p, \dagger m, *m + \mu\,\mathrm{d}, \mathbf{SE}) \right)$
- $\dagger likes \in \left[*likes + \Delta,\ \min(\dagger p, \dagger m) \right)$

B COMPLEX EXAMPLE

In Figure 3, 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.



(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 1 and Section A.1.



(b) Illustration of the intervals in which the *creation* • and *deletion* • dates 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 Δ time. In case of deletion dates, it is used to ensure that the entity exists for at least Δ time.

Figure 3: Example graph and time intervals for selecting lifespan attributes, *creation* and *deletion dates*.