LDBC

Cooperative Project

FP7 – 317548

D6.3.3 LDBC Benchmark Auditing Policies

Coordinator: Orri Erling

1st Quality Reviewer: Alex Averbuch
2nd Quality Reviewer: Peter Boncz

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M24
Actual delivery date:	M24
Version:	1.0
Total number of pages:	24
Keywords:	benchmark auditing, execution rules, result reporting

Abstract

This document presents general guidelines on benchmark auditing and execution rules. The first section introduces the auditable aspects of benchmark execution and expresses general guidelines on what constitutes reasonable rules. This is in large part influenced by the practices in TPC benchmarks. The second part discusses specific rules applicable to the Semantic Publishing and Social Network benchmarks of LDBC. We note that auditing, and hence proficiency and training requirements for auditors, cannot be tackled separately from the rules auditing is expected to enforce. Hence we speak of both auditor qualification and of the benchmarks themselves in the same document.

Executive Summary

This document describes the general principles of benchmark auditing in LDBC. The diverse aspects of benchmark execution are covered and examples are provided in the context of present LDBC benchmarks. The document concludes with preliminary auditing checklists for the Social Network and Semantic Publishing benchmarks.

Document Information

IST Project Number	FP7 – 317548	Acronym	LDBC
Full Title	LDBC		
Project URL	http://www.ldbc.eu/		
Document URL	provide the URI for the document		
EU Project Officer	Carola Carstens		

Deliverable	Number	D6.3.3	Title	LDBC Benchmark Auditing Policies
Work Package	Number	WP6	Title	Exploitation

Date of Delivery	Contractual	M24	Actual	M24
Status	version 1.0		final ☐	
Nature	Report (R) ☒ Prototype (P) ☐ Demonstrator (D) ☐ Other (O) ☐			
Dissemination Level	Public (PU) ☒ Restricted to group (RE) ☐ Restricted to programme (PP) ☐ Consortium (CO) ☐			

Authors (Partner)	names of main authors (with partner names in parentheses)			
Responsible Author	Name	Orri Erling	E-mail	erling@xs4all.nl
	Partner	Partner	Phone	

Abstract (for dissemination)	This document presents general guidelines on benchmark auditing and execution rules. The first section introduces the auditable aspects of benchmark execution and expresses general guidelines on what constitutes reasonable rules. This is in large part influenced by the practices in TPC benchmarks. The second part discusses specific rules applicable to the Semantic Publishing and Social Network benchmarks of LDBC. We note that auditing, and hence proficiency and training requirements for auditors, cannot be tackled separately from the rules auditing is expected to enforce. Hence we speak of both auditor qualification and of the benchmarks themselves in the same document.
Keywords	benchmark auditing, execution rules, result reporting

Version Log			
Issue Date	Rev. No.	Author	Change
18/09/2014	0.1	Orri Erling	First version
30/09/2014	0.2	Mirko Spasic	Integration of Alex's suggestions

Table of Contents

Executive Summary 3

Document Information 4

1 Introduction 6

2 Rationale and General Principles 7

3 Auditor Training and Qualification 8

4 Auditable Properties of Systems and Benchmark Implementations 9
4.1 Definitions . . . 9
4.2 ACID Compliance . . . 10
4.3 Schema Design . . . 11
4.3.1 Schema Lastness . . . 11
4.4 Qualification and Correctness of Results . . . 11
4.5 Data Access Transparency . . . 12
4.6 Query Declarativity . . . 12
4.7 Materialization . . . 12
4.8 Steady State . . . 13
4.9 Operation Mix . . . 13
4.10 System Configuration and Pricing . . . 13
4.11 Benchmark Specifics . . . 15

5 Auditing Process and Auditor Selection 16
5.1 Challenges . . . 16

6 Semantic Publishing Benchmark 17
6.1 Scaling . . . 17
6.2 Schema and Data Model . . . 17
6.3 Schema and Ontology . . . 17
6.4 Query Formulation . . . 18
6.5 ACID Compliance . . . 18
6.6 Workload Mix and Duration . . . 18
6.7 System Architecture . . . 18
6.8 Full Disclosure . . . 19

7 Social Network Interactive 20
7.1 Scaling . . . 20
7.2 Data Model . . . 20
7.3 Auxiliary Data Structures and Precomputation . . . 21
7.4 Implementation Language and Data Access Transparency . . . 21
7.5 ACID Compliance . . . 22
7.6 Benchmark execution . . . 22
7.6.1 Concerning Checkpoints . . . 23
7.7 Query Mix . . . 23
7.7.1 Minimum Measurement Window . . . 24
7.7.2 Test Driver . . . 24
7.8 Full Disclosure . . . 24

1 Introduction

This document is divided in the following parts:

- Motivation of benchmark result auditing

- General discussion of auditable aspects of benchmarks

- Specific checklists and running rules for the SNB and SPB benchmarks

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

2 Rationale and General Principles

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

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

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

3 Auditor Training and Qualification

Auditor certification is by the LDBC task force responsible for the benchmark being audited. An auditor is certified for a particular benchmark by the task force maintaining the benchmark in question.

The task force may in special cases appoint itself as auditor of a particular result. This is not however the preferred course of action but may be done if no suitable third party auditor is available

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

The auditor certification is done in the form of an examination administered by the task force. The examination may be carried out by teleconference. The task force will subsequently vote on accepting each auditor, by simple majority.

4 Auditable Properties of Systems and Benchmark Implementations

4.1 Definitions

Test sponsor: The party which publishes a benchmark This is typically the vendor of a key component of the SUT, e.g. DBMS or hardware.

Full Disclosure Report (FDR): The FDR is a document which allows reproduction of any benchmark result by a third party. This contains complete description of the SUT and the circumstances of the benchmark run, e.g. configuration of SUT, dataset and test driver etc.

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

Workload: This is the totality of the tasks a particular benchmark does against a SUT. This includes data loading as well as the query/update workload. This does not include preparatory stages such as generating benchmark data with a data generator or copying said data to the platform constituting the SUT.

Schema: A schema is the totality of the non-built-in declarations which are fed into the SUT prior to running a workload. For a relational system the schema consists or tables, indices, views, materialized views etc. The schema does not include stored procedures, triggers etc or other imperative (procedural) application specific code that may reside on the SUT. The schema does include declarative constraints, for example foreign key constraints. An ontology for an RDF system is considered as schema if it is loaded on the SUT. An RDF SUT may have no schema at all and still run the workload. However, any declaration or setting that is not on by default across all application of a SUT and is used in a benchmark run counts as schema.

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

Data Structure: This is anything that may influence the result of a database query or may be changed by an update of the database. These may be resident in RAM or durable media or both. Examples of data structures are database indices in their memory based as well as durable media based manifestations, tables in memory based or durable media based manifestations, materialized views in any manifestation, with or without materialized aggregation.

Auxiliary Data Structure - ADS: This is a data structure which materializes content from a primary data structure for providing more efficient access to all or parts of the primary data structure. A secondary index of a relational table, in its memory based and durable media based manifestations is an example. Some systems, such as many RDF stores have multiple covering indices on the primary data structure. In this case the definition is that the primary data structure consists of all the differently ordered full copies of the base table, in the RDF case a table of PSPOG. (subject predicate object graph). Auxiliary data structures in this case may be such data structures which materialize some subset of the SPOG.

In the case of relational systems, an auxiliary data structure may be an index from primary key values to a heap table, if the system in question has such concepts. A secondary index is not considered an auxiliary data structure since it must be declared, making it an explicit auxiliary data structure. An auxiliary data structure must be implicit and not created by any specific DDL statement or directive.

In the case of RDF systems, if the implementation supports user definable index schemes, as long as these are defined once and apply to all triples/quads, such structures are designated as ADS. If an RDF system selectively makes data structures which apply to some quads but not to others, then such structures are designated as EADS.

Explicit Auxiliary Data Structure - EADS: These are any application or workload specific structures that are declared in addition to the base tables or entity types managed by the SUT. Secondary indices, materialized views, with or without aggregates, are all examples of this in a relational context. In an RDF context an EADS is a data structure which holds some of the triples/quads in whole or in part but does not hold some others. An EADS is always declared by the in the schema.

4.2 ACID Compliance

This section outlines transactional behaviors of systems under test which may be verified in the course of auditing a benchmark run.

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

ACID compliance:

- Atomicity: Either all of the effects of the transaction are in effect after the transaction or none of the effects is in effect. This is by definition only verifiable after a transaction has finished.
- Consistency: Auxiliary data structures such as secondary indices will be consistent among themselves as well as with the table or other primary data structure, if any. This such consistency (compliance to all constraints, if these are declared in the schema, e.g. primary of foreign key constraints) may be verified after the commit or rollback of a transaction. If a single thread of control runs within a transaction, then subsequent operations are expected to see consistent state across all data indices pertaining to a table or similar object. Multiple threads which may share a transaction context are not required to observe a consistent state at all times during the execution of the transaction. Consistency will however always be verifiable after the commit or rollback of any transaction, regardless of the number of threads that have either implicitly or explicitly participated in the transaction. Any intra-transaction parallelism introduced by the SUT will preserve transactional semantics statement-by-statement. If explicit, application created sessions share a transaction context, then this definition of consistency does not hold: For example, if two threads insert into the same table at the same time in the same transaction context, these may or may not see a consistent image of ADS or EADS for the parts affected by the other thread. All things will be consistent after the commit or rollback, however, regardless of the number of threads, implicit or explicit that have participated in the transaction.
- Isolation: Isolation is defined as the set of phenomena operations running in within a transaction context may or may not observe:
  - Read uncommitted: No guarantees apply.
  - Read committed: A transaction will never see a state that has at no point in time been part of a committed state.

  - Repeatable read: If a transaction reads a value several times during its execution, then it will see the original state with its own modifications so far applied to it. If the transaction itself consists of multiple reading and updating threads then the ambiguities that may arise fall outside the scope of transaction isolation.
  - Serializable: The transactions see values that could have corresponded to a fully serial execution of all client transactions. This is like repeatable read except that if the transaction reads something, and repeats the read, it is guaranteed that no new values will appear for the same search condition on a subsequent read in the same transaction context. For example, a row that was seen not to exist when first checked will not be seen by a subsequent read. Likewise, counts of items will not be seen to change.

- Durability: Durability means that once the SUT has confirmed a successful commit, the committed state will survive any instantaneous failure of the SUT, for example a power failure, software crash, reboot or the like. Durability is tied to atomicity in that if one part of the changes made by a transaction survives then all parts must survive. This is a special concern in distributed systems which must coordinate durability across multiple physical systems and processes.

4.3 Schema Design

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

By default, a system may declare no schema at all, as may be the case with RDF or graph DBMS's . If ADS are declared, then these must be consistently applied to all data. The nature permitted EADS will depend on the benchmark.

4.3.1 Schema Lastness

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

A benchmark may allow use of EADS with a schema-less data model such as RDF with the proviso that while some data structures may become thereby more efficient no data structure is ipso facto prohibited. The schema-less nature may persist but some common structures may benefit from more efficient physical representation.

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

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

4.4 Qualification and Correctness of Results

A benchmark should be published with a deterministically recreatable validation dataset. Validation queries applied to the validation dataset will deterministically produce a set of correct answers. This is used as a pre-benchmark run test for the correctness of a SUT or implementation.

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

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

4.5 Data Access Transparency

A benchmark may specify that an implementation is not allowed the use of explicit access paths. For example, explicitly specifying which EADS or ADS should be used for any given operation may be prohibited. Further, in scale-out systems, it explicit references to data location (other than via values of partitioning keys) may be prohibited.

Generally, references to internal data representation of an entity, e.g. row in a table, should be prohibited. Reference should take place via column names in a schema or property URI's in RDF, not via physical offsets or the like.

4.6 Query Declarativity

Generally, OLTP benchmarks allow implementation via stored procedures, effectively amounting to explicit query plans. Generally, analytical benchmarks prohibit these. In the graph database world where there is no standard query language and API's are in common use, these distinctions are not as clear or enforceable.

Generally, if a benchmark is analytical and a query is declaratively stated, explicit control of join order, join type etc should be prohibited. These may still be allowed in a declarative query if the application has a clear OLTP nature. Consider for example TPC C, where implementations are known to specify join orders and indices.

In the current benchmark in practice, OLTP and analytical workload are distinct, and different rules are usually applied.

The LDBC situation vis a vis declarativity is not as simple as that of for example the TPC: With the TPC, declarativity with no hints is required for analytics and is not required for OLTP. With LDBC, the systems are more heterohenous and the analytics workloads more diverse, including in addition to BI queries similar to TPC-H also graph analytics which cannot be expressed as queries. Hence individual benchmarks are expected to specify their policy concerning declarativity. In general such policies, insofar they allow imperative expression of workload, should require full disclosure of all code.

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

An API based implementation may choose between semantically equivalent implementations of an operation based on parameters. This simulates the behavior of a query optimizer in the presence of literal values in the query. If an implementation does this, the code must be disclosed and the decision must be based on values extracted from the database, not on hard coded literal values in the implementation.

4.7 Materialization

The mix of read and update operations in a workload will determine to which degree precomputation of results is possible or warranted. Precomputation of aggregates or joins is generally forbidden unless explicitly allowed by the benchmark definition.

The auditor must check that materializations fall in the explicitly allowed and provide consistency at the end of each transaction.

4.8 Steady State

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

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

4.9 Operation Mix

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

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

For example, a run of the semantic publishing benchmark which had a large number of short queries and a number of long queries that did not finish during the measurement window would be non-compliant. Benchmarks shall each specify a minimum count of operations and relative frequencies of operations for a qualifying execution.

4.10 System Configuration and Pricing

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

A SUT may consist of one or more pieces of hardware. For each distinct configuration, the FDR shall disclose the number of units of the type as well as the following:

- Common name of the item, e.g. Dell PowerEdge xxxx.
- Type and number of CPU's , cores/threads per CPU, clock frequency, cache size
- Amount of memory, type of memory and memory frequency, e.g. DDR3 1333MHz.
- Disk controller or motherboard type if disk controller on motherboard
- For each distinct type of secondary storage device, the number and specification of the device, e.g. 4xSeagate Constellation 2TB SATA 6Gbit/s.
- Number and type of network controllers, e.g. 1x Mellanox QDR InfiniBand HCA, PCIE 2.0, 2x 1GbE on motherboard.
- Number and type of network switches. If multiple switches are used, the wiring between the switches should be disclosed.

- Date of availability of the system as a whole, i.e. the latest date of availability of any part.

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. The price should reflect the single quantity list price that any buyer could expect when purchasing one system with the given specification. The price may be either an item by item price or a package price if the system is sold as a package.

The SUT software must be described at least as follows:

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

The auditor must ascertain that a reported run has indeed taken place on the SUT in the disclosed configuration.

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

- Complete configuration files of the DBMS, including any general server configuration files, any configuration scripts run on the DBMS for setting up the benchmark run etc.
- Complete schema of the DBMS, including eventual specification of storage layout.
- Any OS configuration parameters if other than default, e.g. vm.swappiness, vm.max_map_count in Linux.
- Complete text of any server-side logic, e.g. stored procedures, triggers.
- Complete text of driver side benchmark implementation.
- Description of system architecture
- Use of partitioning or replication across multiple machines shall be disclosed if used. The specific partitioning keys or replication criteria, as well as the transactional behavior of said partitioning or replication shall be described. This shall not be inconsistent with the ACID behaviors specified in the benchmark.

The full disclosure shall contain any relevant parameters of the benchmark execution itself, including:

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

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

The intent of the disclosure is to make it possible for a third party to completely reconstruct any reported benchmark run.

4.11 Benchmark Specifics

The TPC benchmarks include rules prohibiting so-called benchmark specials.

5 Auditing Process and Auditor Selection

A benchmark result is to be audited by an LDBC appointed auditor or the LDBC task force managing the benchmark. An LDBC audit may be performed by remote login and does not require the auditor's physical presence on site.

The test sponsor shall grant the auditor any access necessary for validating the benchmark run. This will typically include administrator access to the SUT hardware.

Each benchmark specifies a specific checklist to be verified by the auditor.

The benchmark run may be performed by the auditor or more typically by the test sponsor while the auditor monitors the proceedings via remote login. The auditor shall take copies of pertinent configuration files and test results for future checking against the full disclosure report produced by the test sponsor.

The FDR is produced by the test sponsor. The auditor produces an attestation letter to be included into the FDR as an addendum,. The attestation letter is free form but shall state that the auditor has established compliance with a specified version of the benchmark specification.

The FDR and any benchmark specific summaries thereof shall be published at the LDBC web site.

5.1 Challenges

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

6 Semantic Publishing Benchmark

This document specifies rules for compliant executions of the LDBC Semantic Publishing Benchmark (SPB).

6.1 Scaling

SPB datasets are scaled in triples. Scales greater than 50 million triples are valid for publishing.

6.2 Schema and Data Model

The SPB benchmark is specified as an RDF workload with queries and updates formulated in SPARQL. A compliant implementation must be able to ingest the RDF data and to answer queries written in SPARQL over the SPARQL protocol. The RDF serialization format used must be one that is produced by the SPB data generator. The SPARQL queries must be formulated within the restrictions set forth below.

A compliant implementation must support a large number of named graphs and must be able to query across all these graphs so that any triple that would match by its SPO can be considered in the query evaluation regardless of the graph where it occurs.

Implementations that are based on SQL or a graph database API are specifically not compliant.

A workload similar to SPB may be implemented in any non-RDF data model but this is not a priori comparable to the RDF implementation. The benchmark relies on specific properties of RDF such as named graphs and while these may be simulated in other data models the difference is too large for comparability.

6.3 Schema and Ontology

The SPB benchmark specifies a set of subtypes and subclasses. These must be supported by an implementation according to the applicable RDFS semantics. Therefore an implementation will take any implementation specific steps to cause such compliance.

ADS and EADS are allowed. The DBMS may allow specifying diverse ways of indexing or physically representing all or part of the RDF quads making up the dataset and workload updates.

It is understood that the RDF is an attractive data model for this type of application because of its schema flexibility. The ADS or EADS or other constraints set in the DBMS shall not be so set as to prohibit the insertion of arbitrary quads which may or may not comply comply with the structure of the SPB dataset.

In specific, the above rules out implementations which are based on strict RDF to relational mapping or use of property tables which strictly enforce a single value per property.

EADS which materialize aggregates are not allowed. In specific, using an external engine for answering faceted search queries is not allowed. EADS which maintain counts or memberships in a particular bucket of values based on a property value are not allowed. An example of a forbidden EAS is the Lucene ADS for faceted search where documents are pre-classified based on a metedata field's value falling in a bucket.

EADS which combine object values from several different RDF subjects into a single physical representation are not allowed. EADS which store commonly co-occurring properties of a single subject together are allowed.

EADS which materialize a data ordering on multiple properties, e.g. time and subject matter are not allowed. An EADS which physically clusters data by a single attribute is allowed. The latter corresponds to a clustered index in SQL terms. The word clustering here means causing physical data placement to be correlated to a non-primary key property of the data in question.

Specifying guidelines for indexing object values property by property are allowed. For example, maintaining an index for a low-cardinality property (e.g. bbc:audience) usually does not make sense. DBMS specific directives controlling indexing are allowed to the extent they do not prohibit symmetric retrieval of quads from the database. It is accepted that some access paths may be notably slower than others.

Any EADS must be consistent with the primary data at the end of each transaction.

6.4 Query Formulation

The SPB workload relies on the existence of a full text index and possibly a geospatial index for good performance. Use of any such index is allowed. Since query formulations will vary from vendor to vendor as far as these features are concerned, such variations are explicitly allowed.

The graph replacing transaction may be formulated with a SPARQL delete, insert or update statement or combination thereof or an equivalent DBMS specific statement. In all cases the semantic of the operation is that no quad of the former content of the graph which is not part of the new content shall be accessible via any access path and that all quads which are part of the new content shall be accessible after the commit of the transaction. The sole exception is the allowed latency for full text index maintenance.

Explicit declaration of join order or join type in query formulations is not allowed. Query directives which give explicit hints about the cardinality of parts of the query are not allowed.

6.5 ACID Compliance

The SPB workload has ACID transactions which delete and reinsert a all triples of a named graph. The delete and insertion are submitted to the SUT as a single SPARQL protocol request. This operation must be implemented as a single transaction.

If the implementation materializes inference results, such materialization must be in effect after the successful commit of the transaction causing said materialization.

Full text indices are commonly not maintained up to date transaction by transaction. Implementations are known to use external full text indices, e.g. Lucene. These do not generally share the transaction boundaries of the host DBMS. Hence it is allowed to keep a full text index in sync with updates with a small delay, not to exceed 120 seconds for any given text literal to be retrievable via full text index.

Geospatial indices may be kept up to date transaction by transaction or with a delay. If there is a delay, the delay shall not exceed 120 seconds.

Full text and geospatial indices must be durable, i.e. must survive any instantaneous failure just as any other data structures used by the SUT for database content. Even if full text index content is maintained with a delay, an instantaneous failure during the delay, followed by recovery, must show the data in the full text index at the end of recovery.

The workload does not have transactions involving multiple message exchanges between the SUT and the test driver.

Read queries should not return uncommitted data. Read committed or equivalent semantics are sufficient for the read workload.

6.6 Workload Mix and Duration

The SPB workload exhibits high variability of query execution time depending on the query parameters. Therefore a qualifying run shall have a minimum number of executions of each query type. The minimum run consists of 3 hours of simulation time, with all the updates falling on this window successfully executed, and all the read queries accordingly.

6.7 System Architecture

A publishing workload is characterized by very high read volume and an update rate that is not necessarily proportional to the read volume. Hence implementations with master-slave replication may be favored, so that read throughput is optimized while update throughput is not, as all replicas end up doing the same updates.

A log shipping, replication architecture is explicitly allowed, within the following bounds: The latency between the commit of an update on the master and its replication to the last of the slave copies shall not be

longer than 60 seconds during the benchmark run. The test sponsor shall provide adequate demonstration of this to the auditor.

Use of partitioning across a cluster is allowed. If this is used, then transactional semantics shall not be different from what is expected of a single server.

Rationale: We recognize that in many deployments absolute data volumes do not justify a partitioned scale-out solution. If such an implementation is however used, it should have transactional behavior indistinguishable from a single server. Multiple scale-out databases may be used, so that one is a master and others are read-only slave replicas. In this case, each scale-out database will show one self-consistent state of the data but copies of the same data in other replicas may be behind as stated earlier. It is implicitly assumed that a partitioned scale out system resides in one geographical location and does not involve high latency network, e.g. wide area.

In the case of log shipping replication, the benchmark does not address issues of geographical distance. Many real world deployments do have wide area replication or eventual consistency but this benchmark does not address this. Eventual consistency is not addressed in the full sense since only one copy of the data receives updates at a time, hence no conflict resolution is needed.

6.8 Full Disclosure

The general rules on full FDR's apply. In addition, the following benchmark specific items shall be addressed:

- Implementation of inference: The means for supporting the subclass/subproperty semantics required by the workload shall be disclosed.
- All query formulations shall be disclosed.

7 Social Network Interactive

7.1 Scaling

The scale factor of a SNB dataset is the size of the dataset in GB of comma separated values. All dataset scales contain data for three years of social network activity.

The scaling is done in buckets. The size of the dataset is either 10GB or 30GB or a power of ten multiple of either. Thus scales are 10, 30 100, 300, 1000 GB and so forth. This is identical to the scaling of TPC-H and TPC-DS.

The validation scale is 10GB.

The dataset is divided in a bulk loadable initial database population and an update stream. These are generated by the SNB data generator. The data generator has options for splitting the dataset into any number of files.

The update stream contains the latest 10% of the events in the simulated social network. These events form a single serializable sequence in time. Some events will depend on preceding events, for example a post must exist before a reply to the post is posted. The data generator guarantees that these are separated by at least 2 minutes of simulation time.

The update stream may be broken into arbitrarily many substreams. The partition scheme should preserve dependencies between operations, so as, not to refer to non existent entities. For example, like should not be added to a post which is not been inserted yet.

The different versions of the driver may vary in their support for this, and benchmark implementers are required to modify the driver in order to produce different scale-out behaviors.

Rationale: The authors are aware that the prevalent practice for online benchmarks is to tie the reported throughput to the scale, e.g. max 12.5 tpmC per warehouse in TPC-C. The authors depart from this practice here because with throughput tied to scale, test systems with interesting throughputs rapidly become very expensive, raising the entry barrier for publishing a result. It is thought that scaling in buckets lowers the barrier of entry and reduces incentive to use hardware configurations that would be unusual in a production environment.

7.2 Data Model

SNB may be implemented with different data models, e.g. relational, RDF and different graph data models.

The reference schema is provided as RDFS and SQL. The data generator produces TTL syntax for RDF and comma separated values for other data models.

A single attribute has a single data type, as follows:

- Identifier: This is an integer value foreign key or a URI in RDF. If this is an integer column, the implementation data type should support at least 2^{50} distinct values.
- Date: A datetime should support a date range from 0000 to 9999 in the year field, with a resolution of no less than one second
- Short string: The string column for names may have a variable length and may have a declared maximum length, e.g. 40 characters.
- Long string: For example a post content may be a long string that is often short in the data but may not declare a maximum length and must support data sizes f up to 1MB.

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

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

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

cardinality statements therein. An RDF implementation is allowed creation of EADS subject to the EADS restrictions prescribed for other data models.

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

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

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

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

7.3 Auxiliary Data Structures and Precomputation

Q2 retrieves for a person the latest n posts or comments posted by a contact of the person. An implementation may choose to precompute this "top of the wall" query.

If doing so, inserting any new post or comment will add the item in question to the materialized top k post views of each of the contacts of the person. If after insertion, this list were to be longer than 20 items, the transaction will delete the oldest item.

If this precomputation is applied, the update of the "top of the wall" materialization of the users concerned must be implemented as a single transaction.

This precomputation may be implemented as client side logic in the test driver, as stored procedures or as triggers. In all cases the operations, whether one or many, must constitute a single transaction. A SPARQL protocol operation consisting of multiple statements may be a valid implementation of the SUT executes the statements as a single transaction.

Other precomputation of query results is explicitly prohibited.

An exception is made for RDF where an EADS materializing several properties of a single subject is allowed. Presence of such an EADS must not however prohibit insertion of arbitrary numbers of values or arbitrary new properties. Thus common cases may be optimized but rare cases may not be prohibited and the RDF quad (triple+graph) semantics must be upheld.

7.4 Implementation Language and Data Access Transparency

The queries and updates may be implemented in a declarative query language or as procedural code using an API.

If a declarative query language is used, e.g. SPARQL or SQL, then explicit query plans are prohibited in all the read-only queries. The update transactions may still consist of multiple statements, effectively amounting to explicit plans.

Explicit query plans include but are not limited to:

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

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

7.5 ACID Compliance

The interactive workload requires full ACID support from the SUT.

- Atomicity: All the updates in a transaction must either take place or be all canceled.
- Consistency: If a database object, e.g. table, has auxiliary data structures, e.g. indices, the content of these must be consistent after the commit or rollback of a transaction. If multiple client application threads share one transaction context, these may transiently see inconsistent states, e.g. there may be a time when an insert of a row is reflected in one index of a table but not in another.
- Isolation: If a transaction reads the database with intent to update, the DBMS must guarantee that repeating the same read within the same transaction will return the same data. This also means that no more and no less data rows must be returned. In other words, this corresponds to snapshot or to serializable isolation. This level of isolations applied for the operations where the transaction mix so specifies.

  If the database is accessed without transaction context or without intent too update, then the DBMS should provide read committed semantics, e.g. repeating the same read may produce different results but these results may never include effects of pending uncommitted transactions.
- Durability: The effects of a transaction must be made durable against instantaneous failure before the SUT confirms the successful commit of a transaction to the application.

  For systems using a transaction log, this implies syncing the durable media of the transaction log before confirming success to the application.. This will typically entail group commit where transactions that fall in the same short window are logged together and the logging device will typically be an SSD or battery backed RAM on a storage controller. For systems using replication for durability, this will entail receipt of a confirmation message from the replicating party before confirming successful commit to the application.

7.6 Benchmark execution

A benchmark execution is divided into the following steps:

- Data Preparation: This includes running the data generator, placing the generated files in a staging area, configuring storage, setting up the SUT configuration and preparing any data partitions in the SUT. This may include preallocating database space but may not include loading any data or defining any schema having to do with the benchmark.
- Bulk Load: This includes defining the database schema, if any, loading the initial database population, making this durably stored, gathering any optimizer statistics,. The bulk load time is reported and is equal to the amount of elapsed wall clock time between starting the schema definition and receiving the confirmation message of the end of statistics gathering.
- Benchmark Run: The run begins after the bulk load or after another benchmark run. If the run does not directly follow the bulk load, it must start at a point in the update stream that has not previously been played into the database. In other words, a run may only include update events whose timestamp is later

than the latest post creation date in the database prior to start of run. The run starts when the first of the test drivers sends its first message to the SUT. If the SUT is in-process with the driver the window starts when the driver starts.

- Measurement Window: The measurement window is the timed portion of the benchmark run. It may begin at any time during the run. The activity during the measurement window must meet the criteria set forth in Query Mix and must include enough updates to satisfy the criteria in Minimum Measurement Window. The measurement window is terminated at the discretion of the test sponsor at any time when the Minimum Measurement Window criteria are met. All the processes constituting the SUT are to be killed at the end of the window or alternatively all the hardware components of the SUT are to be powered off.

- Recovery Test: The SUT is to be restarted after the measurement window and the auditor will verify that the SUT contains the entirety of the last update recorded by the test driver(s) as successfully committed. The driver or the implementation have to make this information available.

7.6.1 Concerning Checkpoints

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

A checkpoint typically involves a synchronization barrier at which all data committed prior too the moment is required to be in durable storage that does not depend on the transaction log.

Note all DBMS's use a checkpointing mechanism for durability. For example a system may rely on redundant storage of data for durability guarantees against instantaneous failure of a single server.

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

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

The restart test consists of restarting the SUT process(es) and finishes when the SUT is back online in with all its functionality and the last successful update logged by the driver can is seen to be in effect in the database. In the case of a scale-out system, a particular partition may be recovered whereas another one is still in the process of recovering. If this is so, then checking for the last update shall not be done until all partitions are online.

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

7.7 Query Mix

The base unit of work is the insertion of one post (or comment). For each 1000 posts inserted, the following number of other operations must be completed: Q1 - 20, Q2 - 40, Q3 - 5, Q4 - 600, Q5 - 15, Q6 - 2, Q7 - 900, Q8 - 2950, Q9 - 2, Q10 - 4, Q11 - 300, Q12 - 15, Q13 - 20, Q14 - 10.

For a run to qualify the number of successfully executed operation must not deviate from the above frequencies by more than 2%.

7.7.1 Minimum Measurement Window

The measurement window starts at the point in simulation time given by the post with the highest creation date in the timestamp. The update stream must contain at least 35 days worth of events in simulation time that are in the future from the datetime of the latest pre-run post in the database.

The minimal measurement window corresponds to 30 days worth of events in simulation time, as generated by the data generator. Thus the number of events in the minimal window is linear to the scale factor. This grows the database by approximately 3%.

When the test sponsor decides to start the measurement window, the time of the latest new post event successfully completed by any of the test drivers is taken to be the start of the measurement window in simulation time. The window may be terminated by the test sponsor at any time when the timestamp of the latest new post event in the log of any of the test drivers is more than 30 days of simulation time in the future of the starting timestamp.

The test summary tool (see below) may be used for reading the logs being written by a test driver.

7.7.2 Test Driver

A qualifying run must use the SNB test driver provided with the data generator. The test driver may be modified by the test sponsor for purposes of interfacing to the SUT. The parameter generating and result recording and workload scheduling parts of the test driver should not be changed. The auditor needs to have access to the test driver source code used for producing the driver used in the reported run.

The test driver produces the following artifacts as a by product of the run: start and end time of each execution in real time, recorded with microsecond precision, including the identifier of the operation and any substitution parameters.

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

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

- Count of executions
- Minimum/average/90th percentile/maximum execution time.
- Metric in operations per second at scale.
- Number of test drivers
- Number of database sessions (threads) per test driver

7.8 Full Disclosure

In addition to the general requirements, the full disclosure shall specify the following:

- Approach to materialization - Whether the allowed Q2 precomputation was done.