BEAS: Bounded Evaluation of SQL Queries

Citation for published version:
Cao, Y, Fan, W, Wang, Y, Yuan, T, Li, Y & Chen, LY 2017, BEAS: Bounded Evaluation of SQL Queries. in *SIGMOD '17 Proceedings of the 2017 ACM International Conference on Management of Data.* ACM, pp. 1667-1670, 2017 ACM International Conference on Management of Data, Chicago, Illinois, United States, 14/05/17. https://doi.org/10.1145/3035918.3058748

Digital Object Identifier (DOI):
10.1145/3035918.3058748

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
SIGMOD '17 Proceedings of the 2017 ACM International Conference on Management of Data

BEAS: Bounded Evaluation of SQL Queries

Yang Cao[1], Wenfei Fan[1], Yanghao Wang[1], Tengfei Yuan[1] Yanchao Li[2] Laura Yu Chen[3]
[1]University of Edinburgh [2]Nanjing University of Science and Technology [3]Huawei America Research Center

Abstract

We demonstrate BEAS, a prototype system for querying relations with bounded resources. BEAS advocates an unconventional query evaluation paradigm under an access schema $\mathcal{A}$, which is a combination of cardinality constraints and associated indices. Given an SQL query Q and a dataset D, BEAS computes $Q(D)$ by accessing a bounded fraction D_Q of D, such that $Q(D_Q) = Q(D)$ and D_Q is determined by $\mathcal{A}$ and Q only, no matter how big D grows. It identifies D_Q by reasoning about the cardinality constraints of $\mathcal{A}$, and fetches D_Q using the indices of $\mathcal{A}$. We demonstrate the feasibility of bounded evaluation by walking through each functional component of BEAS. As a proof of concept, we demonstrate how BEAS conducts CDR analyses in telecommunication industry, compared with commercial database systems.

1. Introduction

Querying big relations is often beyond reach for small companies. It may take hours to join tables of millions of tuples. Given a query Q and a dataset D, it is NP-complete to decide whether a tuple is in $Q(D)$ when Q is in SPC (selection, projection, Cartesian product). It is PSPACE-complete for Q in relational algebra [1]. One might think that parallelism would solve the problem by using more processors. However, small companies can often afford only limited resources.

Is it feasible to query big data with bounded resources?

One approach to addressing this challenge is based on bounded evaluation [5–9]. The idea is to use an access schema $\mathcal{A}$ over a database schema $\mathcal{R}$, which is a combination of cardinality constraints and associated indices. A query Q is *boundedly evaluable* under $\mathcal{A}$ if for each instance D of $\mathcal{R}$ that conforms to $\mathcal{A}$, there exists a small $D_Q \subseteq D$ such that

- $Q(D_Q) = Q(D)$, and
- the time for identifying D_Q is decided by Q and $\mathcal{A}$.

Intuitively, D_Q consists of only data needed for answering Q. Its size $|D_Q|$ is determined by Q and $\mathcal{A}$ only, not by $|D|$.

BEAS. As a proof of concept of [5–9], we have developed BEAS [4], a prototype system for Bounded EvAluation of SQL. BEAS has the following unique features that differ from conventional query evaluation paradigm and DBMS.

(1) Quantified data access. BEAS identifies D_Q by reasoning about the cardinality constraints in $\mathcal{A}$, and fetches D_Q by using the indices in $\mathcal{A}$. In the process it deduces a bound M on the amount of data to be accessed, and can hence decide whether Q is boundedly evaluable before Q is executed.

(2) Reduced redundancy. Using $\mathcal{A}$, BEAS fetches only (distinct) partial tuples needed for answering Q. This reduces duplicated and unnecessary attributes in tuples fetched by traditional DBMS. It also reduces joins, in which the redundancies get inflated rapidly (see an example shortly).

(3) Scalability. Putting these together, BEAS computes $Q(D)$ by accessing a bounded fraction D_Q of D, no matter how big D grows. Hence to an extent, it makes big data analysis possible for small businesses with bounded resources.

(4) Ease of use. BEAS can be built on top of any conventional DBMS, and make seamless use of existing optimization techniques of DBMS. This makes it easy to extend DBMS with the functionality of bounded evaluation.

One of our industry collaborators has deployed and tested BEAS, and found that BEAS outperforms commercial DBMS by orders of magnitude for more than 90% of their queries.

Demo overview. We demonstrate the bounded evaluation functionality of BEAS in two parts. (1) To illustrate how bounded evaluation works, we walk through each functional component of BEAS, from access schema discovery and maintenance to bounded query plan generation and execution. (2) To demonstrate the performance of BEAS, we adopt a real-life scenario from telecommunication industry for CDR (call detail record) analyses, and visualize how different query plans perform compared with commercial DBMS.

Below we first present the foundation (Section 2) and the functional components (Section 3) of BEAS. We then propose a more detailed demonstration plan (Section 4).

2. Foundations of BEAS

We start with a review of access schema and bounded evaluability [5, 8, 9], which are the foundations of BEAS.

Access schema. Over a database schema $\mathcal{R}$, an *access constraint* ψ is of the form $R(X \to Y, N)$, where R is a relation in $\mathcal{R}$, X, Y are sets of attributes of R, and N is a natural number [5, 8, 9]. A relation instance D of R *conforms* to ψ if

- for any X-value $\bar{a}$ in D, $|D_Y(X = \bar{a})| \leqslant N$, where $D_Y(X = \bar{a}) = \{t[Y] \mid t \in D, t[X] = \bar{a}\}$; and
- there exists an *index on* X *for* Y that given an X-value $\bar{a}$, retrieves $D_Y(X = \bar{a})$ by accessing at most N tuples.

That is, for any given X-value, there exist at most N distinct corresponding Y values in D (cardinality constraint), and the Y values can be fetched by using the index for ψ (index).

An access schema $\mathcal{A}$ over $\mathcal{R}$ is a set of access constraints over $\mathcal{R}$. A database instance D of $\mathcal{R}$ *conforms* to $\mathcal{A}$, denoted by $D \models \mathcal{A}$, if D conforms to each constraint in $\mathcal{A}$.

Example 1: Consider a commercial benchmark of schema $\mathcal{R}_0$ from a telecommunication company (name withheld). It includes three (simplified) relations: (a) $\mathsf{call}(\mathsf{pnum}, \mathsf{recnum}, \mathsf{date}, \mathsf{region})$, recording that number pnum called recnum in region on date; (b) $\mathsf{package}(\mathsf{pnum}, \mathsf{pid}, \mathsf{start}, \mathsf{end}, \mathsf{year})$, saying that pnum is in service package pid from month start to end in year; and (c) $\mathsf{business}(\mathsf{pnum}, \mathsf{type}, \mathsf{region})$ says that business number pnum in region is of type, *e.g.,* bank, hospital.

An access schema $\mathcal{A}_0$ over $\mathcal{R}_0$ includes:

- ψ_1: $\mathsf{call}(\{\mathsf{pnum}, \mathsf{date}\} \to \{\mathsf{recnum}, \mathsf{region}\}, 500)$,
- ψ_2: $\mathsf{package}(\{\mathsf{pnum}, \mathsf{year}\} \to \{\mathsf{pid}, \mathsf{start}, \mathsf{end}\}, 12)$, and

◦ ψ_3: business({type, region} → pnum, 2000).

Here (1) access constraint ψ_1 states that each number calls at most 500 distinct numbers in a region per day; (2) ψ_2 says that each number can be in at most 12 distinct packages per year since it has to stay in each package for at least a month; and (3) ψ_3 states that for each type in each region, there are at most 2000 businesses of the same type. Constants 500 and 2000 are upper bounds aggregated from historical datasets.

Given a pnum and date, the index for ψ_1 retrieves all distinct (recnum, region) pairs from relation call, at most 500 (partial) call tuples; similarly for indices of ψ_2 and ψ_3. □

Bounded evaluability. Underlying BEAS is the theory of bounded evaluability [5,6,8]. Given an access schema $\mathcal{A}$ and a query Q, the key idea is to generate a *bounded* query plan that accesses data quantified by $\mathcal{A}$, as illustrated below.

Example 2: Consider a benchmark query Q to *find regions in which there are numbers that were called by some business number x on date d_0 in 2016, where x was (a) of business type t_0, (b) in region r_0, and (c) in service package c_0*:

select call.region
from call, package, business
where business.type = t_0 **and** business.region = r_0 **and** business.pnum = call.pnum **and** call.date = d_0 **and** call.pnum = package.pnum **and** package.year = 2016 **and** package.start $\leqslant d_0$ **and** package.end $\geqslant d_0$ **and** package.pid = c_0

Under access schema $\mathcal{A}_0$ given in Example 1, Q has a bounded query plan and can be answered as follows:

(1) Fetch a set T_1 of at most 2000 pnum's from relation business by using the index for ψ_3 with key (t_0, r_0).

(2) For each pnum in T_1, fetch at most 12 distinct (pid, start, end) triples using the index for ψ_2 with key (pnum, 2016), yielding a set T_2 of at most 2000×12 partial package tuples.

(3) Select pnum's from T_2 with start $\leqslant d_0 \leqslant$ end and pid $= c_0$.

(4) For each pnum selected in (3), fetch at most 500 (recnum, region) pairs from relation call using the index for ψ_1 with key (pnum, d_0). This yields a set T_4 of region's as the query answer, by accessing at most $500 \times 2000 \times 12$ call tuples.

The bounded plan accesses a set D_Q of at most 2000 partial business tuples, 24000 package tuples and 12 million call tuples in total, no matter how big the relations are.

As will be shown in the demo, on a 20GB dataset generated by the benchmark, BEAS can find exact answers to Q in 96.13ms using the bounded plan, while a commercial DBMS takes 187.8s, *i.e.,* BEAS is 1953 times faster, although it still accesses over 12 million tuples. One of the reasons is that BEAS fetches partial tuples instead of entire tuples, and avoids joining big relations, *e.g.,* it replaces the join of package and call (each may have billions of tuples) by a fetch of 24000 tuples and selection in steps (2) and (3) above. □

Query Q is *boundedly evaluable* under $\mathcal{A}$ if it has a bounded query plan in which each fetch operation is controlled by an access constraint of $\mathcal{A}$, *e.g.,* steps (1), (2), (3) in the plan of Example 2 by ψ_3, ψ_2, ψ_1, respectively (see [5,8] for details).

As illustrated in Example 2, if Q is boundedly evaluable, then (a) the amount $|D_Q|$ of data accessed can be deduced from the cardinality in $\mathcal{A}$. Moreover, D_Q is determined by Q and $\mathcal{A}$ only, no matter how big D is. This allows us to answer Q with bounded resources. (b) A bounded query plan fetches partial distinct tuples. It reduces redundancies introduced by irrelevant and duplicated attributes, and their inflation by joins. Hence bounded evaluation may substantially improve the scalability and efficiency of query evaluation.

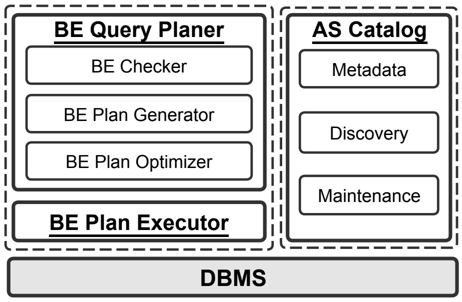

Figure 1: Architecture of BEAS

While desirable, it is undecidable to determine whether an SQL query is boundedly evaluable under an access schema [8]. Nonetheless, we can still make practical use of bounded evaluation due to the existence of an effective syntax.

Feasibility Theorem [5]: *Under an access schema $\mathcal{A}$, there exists a class of queries covered by $\mathcal{A}$ such that*

(1) an RA *query Q is boundedly evaluable under $\mathcal{A}$ if and only if there exists an* RA *query Q' that is covered by $\mathcal{A}$ such that $Q(D) = Q'(D)$ for all $D \models \mathcal{A}$; and*

(2) it is in PTIME *to decide whether Q is covered by $\mathcal{A}$.*□

That is, Q is boundedly evaluable if and only if it can be rewritten into an equivalent Q' covered by $\mathcal{A}$. In other words, covered queries make the core subclass of boundedly evaluable queries in relational algebra, without sacrificing their expressive power. This is along the same lines as the study of (undecidable) safe relational calculus queries [1].

BEAS extends Theorem 1 to SQL queries, and extends DBMS with a bounded evaluation functionality as follows:

(1) given an SQL query Q, BEAS first checks whether Q is covered by the access schema $\mathcal{A}$ available; if so

(2) it generates a bounded query plan and computes exact answers to Q within bounded resources;

(3) otherwise, it generates partially bounded plans and uses DBMS to compute exact answers (see Section 3).

Algorithms for checking the bounded evaluability and for generating bounded query plans have been reported in [5,7].

In fact, for queries that are not covered by $\mathcal{A}$ (case (3) above), BEAS supports bounded approximation under access constraints, which guarantees a deterministic accuracy lower bound on approximate answers computed, and moreover, it accesses a bounded number of tuples in the entire process. These offer a solution to querying relations with bounded resources. To simplify the presentation, we focus on bounded evaluation for computing exact query answers in the demo.

3. The Architecture of BEAS

As shown in Fig. 1, BEAS consists of three major components: (1) offline service AS_Catalog to manage access schema for different applications; and (2) online service BE_Query_Planner and BE_Plan_Executor to process SQL queries. It can be built on top of any commercial DBMS.

AS_Catalog. It consists of three modules itself.

(1) Metadata module maintains (a) access schema, and (b) statistics including the index size in a system table *as_catalog*, for query plan generation and optimization.

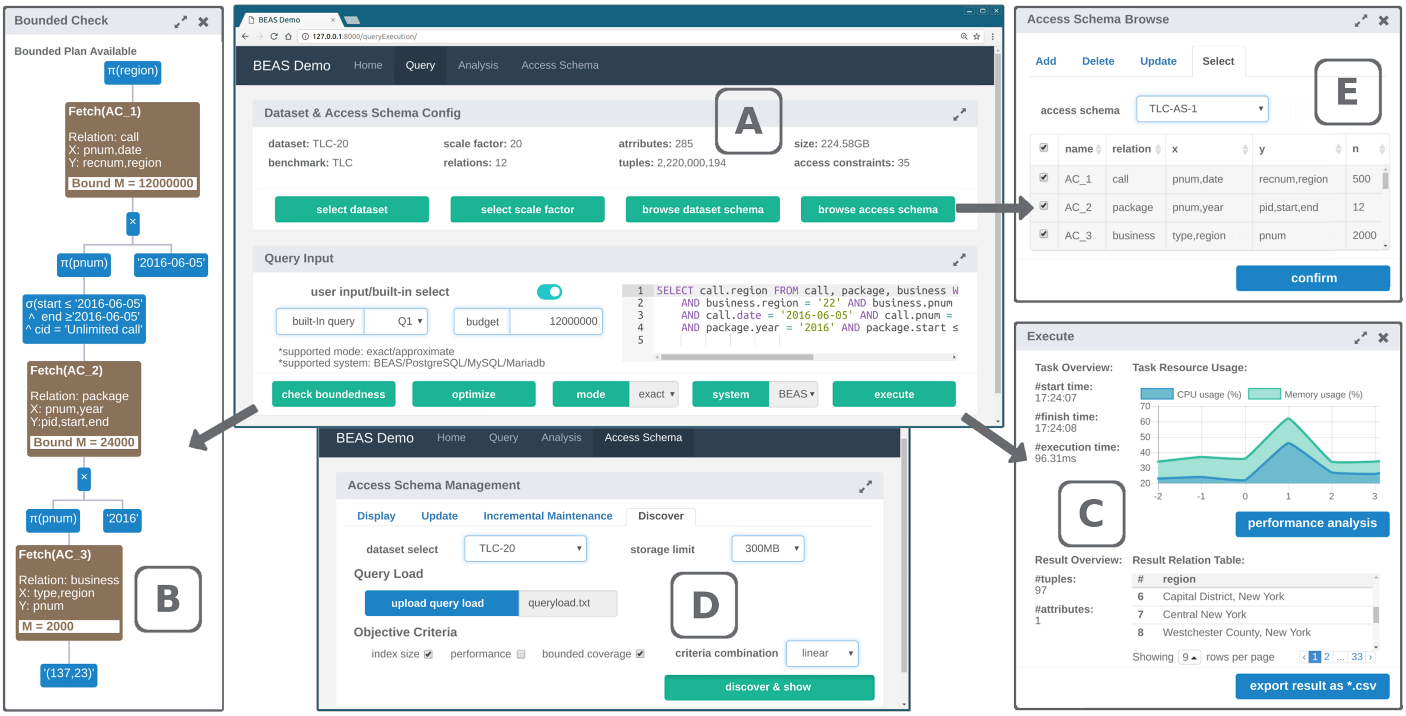

Figure 2: BEAS user interface

(2) Discovery module. Given an application, it automatically discovers an access schema from its real-life datasets. It is a multi-criteria optimization problem that covers (a) the performance of bounded evaluation of the query load, (b) storage limit for indices, (c) historical query patterns, and (d) statistics of datasets in the application. We defer the details of the discovery algorithm to a later publication.

For each access constraint $\psi = R(X \to Y, N)$ discovered, its index on a relation D of R is a modified hash index such that (a) it takes attributes X as the key; and (b) each key value $\bar{a}$ points to a bucket $D_Y(X = \bar{a})$ (see Section 2), the set of at most N distinct Y-values in D corresponding to $\bar{a}$.

(3) Maintenance module maintains access schema $\mathcal{A}$ in response to changes to query loads and datasets in each application. It (a) periodically adjusts constraints in $\mathcal{A}$ based on the changes to the historical queries, to optimize the performance of bounded evaluation; and (b) incrementally updates the indices of $\mathcal{A}$ in response to changes to the datasets, by employing an optimal incremental algorithms reported in [5].

BE Query Planner. It also has three modules.

(1) BE Checker checks whether an input SQL query Q is boundedly evaluable under the access schema discovered. A checking algorithm has been reported in [5] for RA, based on the effective syntax of the Feasibility Theorem. BEAS extends the algorithm to (possibly aggregate) SQL queries.

(2) BE Plan Generator generates (a) a bounded query plan for Q if Q is found boundedly evaluable by *BE Checker*, by extending the bounded-plan generation algorithm reported in [5] from RA to SQL; and (b) if Q is not bounded, it picks a conventional query plan for Q generated by the underlying DBMS, and applies BE Plan Optimizer to it (see below).

As shown in Example 2, a bounded plan consists of relational algebra operators [10] (*i.e.*, select, project, join, union and set difference), aggregates, group-by, and a new operator fetch($X \in T, Y, R$) with access constraint $R(X \to Y, N)$, which fetches all Y-values corresponding to the X-values in intermediate results T. It accesses data only via fetch operations, and answers Q by using a bounded amount of data.

(3) BE Plan Optimizer improves the conventional plan of the DBMS for Q when Q is not bounded, to support *partially bounded evaluation*. It identifies sub-queries of Q that are boundedly evaluable under access schema $\mathcal{A}$, and speeds up the evaluation of Q by capitalizing on the indices of $\mathcal{A}$.

Alternatively, if users can afford only bounded resources and hence opt to take approximate query answers, BEAS offers *resource bounded approximation*. We defer the details of the approximation scheme to a later publication.

BE Plan Executor. It executes bounded query plans by extending the physical plan implementation of DBMS [10] to support the fetch operator. For each fetch($X \in T, Y, R$) with access constraint $R(X \to Y, N)$ in a bounded plan, where T is an intermediate relation, it fetches all associated Y values for each X value $\bar{a}$ in T by using the modified hash index for ψ with key $\bar{a}$, and returns their union (see Section 2).

Observe the following. (1) The design of BE_Query_Planner and BE_Plan_Executor allows us to implement BEAS on top of any DBMS. It is also easy to add other modules to DBMS, *e.g.*, resource-bounded approximation. (2) There have been recent efforts to query big relations with limited resources, *e.g.*, BlinkDB [2] and PIQL [3]. These systems, however, focus on approximate query answering, by sampling [2] or by restricting the fetched data with a user specified bound [3] in the flavor of anytime algorithms [11]. In contrast, BEAS introduces access schema and aims to provide exact query answers with bounded resources as much as possible.

4. Demonstration Overview

We next present our demonstration plan, including settings and scenarios, as well as an industry application. The target audience of the demo includes anyone who is interested in query answering with bounded resources.

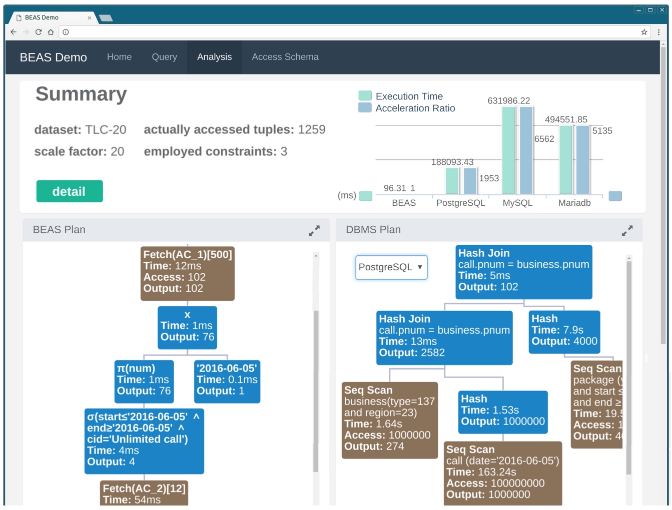

Figure 3: Performance analysis of Q in Example 2

We have implemented BEAS on top of PostgreSQL 9.4.6. We have also created a demo portal as shown in Fig. 2, via which the audience will be able to interact with BEAS. It is deployed at a workstation with Xeon E3-1535M@2.9GHz CPU, 64GB of memory and 1.5TB of disk.

We will invite the audience to (1) examine the feasibility of bounded evaluation by interacting with BEAS. The audience will also (2) experience the performance of BEAS for exact SQL query answering, compared with commercial DBMS, using a telecommunication application as a testbed.

(1) A walk through. We visualize and demonstrate each major component of bounded evaluation underlying BEAS.

(a) Bounded evaluability checking. As shown in Fig. 2(A), the audience will be invited to enter an SQL query Q, select a dataset, pick an access schema $\mathcal{A}$ discovered, and check whether Q is boundedly evaluable under $\mathcal{A}$ using BE Checker. Users can also enter a budget on the amount of data to be accessed, and use BE Checker to find whether Q can be answered within the budget under $\mathcal{A}$, without executing Q.

(b) Bounded planning and optimization. As shown in Fig. 2(B), when Q is boundedly evaluable under $\mathcal{A}$, the users will see a bounded query plan suggested by BE Plan Generator, in which each fetch operation is annotated with an upper bound on the amount of data to be fetched. The upper bound is deduced by reasoning about $\mathcal{A}$.

If Q is not bounded, BEAS picks a query plan ξ generated by PostgreSQL. BE Plan Optimizer then makes ξ partially bounded by identifying bounded sub-queries of Q under $\mathcal{A}$.

(c) Analysis. After a query plan is carried out by BE Plan Executor, a performance analysis is provided (Fig. 2(C)).

(d) Access schema management. As offline services, (i) the discovery module of BEAS takes as input a dataset D, a set $\mathcal{Q}$ of query patterns, and a choice of the objective function; it discovers an access schema $\mathcal{A}$ and register it by AS_Catalog (Fig. 2(D)). For instance, Figure 2(E) shows part of an access schema discovered. (ii) The maintenance module automatically updates $\mathcal{A}$ in response to changes to D and queries. It also allows users to add or remove access constraints.

(2) Performance. We demonstrate how BEAS works in practice using a commercial benchmark from a telecommunication company (name withheld), and compare its performance with PostgreSQL, MySQL and MariaDB.

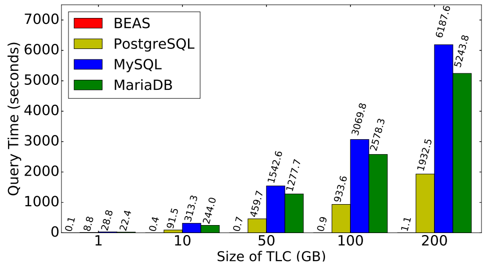

Figure 4: Scalability comparison

Telecommunication. The benchmark, denoted by TLC, has 12 relations with 285 attributes in total. It has 11 built-in queries, simulating industrial data analytical jobs in real-life mobile communication scenarios of the company, *e.g.,* query Q given in Example 2. We will see that these analytical queries are actually boundedly evaluable under a small access schema. In contrast, conventional DBMS may access almost the entire database to answer these queries.

Efficiency. The users are invited to interact with BEAS, pick built-in queries or enter their own queries, and examine the effectiveness of bounded evaluation. For instance, for query Q of Example 2 on a TLC dataset D of 20GB, a snapshot of the BEAS performance analyzer is given in Fig. 3, which shows that BEAS is 1953, 6562 and 5135 times faster than PostgreSQL, MySQL and MariaDB, respectively. It details (a) the overall execution time, acceleration ratio compared to commercial DBMS, the total number of tuples fetched and the number of access constraints employed; and (b) a breakdown of the cost to each individual operation in the query plan, compared to its counterpart in plans generated by commercial DBMS. It illustrates why BEAS works better.

Scalability. The audience will also witness the scalability of BEAS by scaling up the datasets. Figure 4 shows the evaluation time of Q of Example 2 with BEAS, PostgreSQL, MySQL and MariaDB when varying TLC from 1GB to 200GB. One can see that BEAS consistently takes about 1s when D varies, and is hence "scale-independent". In contrast, PostgreSQL, MySQL and MariaDB grow to 1932s, 6187s and 5243s, respectively, if we allow them to run to completion.

5. References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[3] M. Armbrust, S. Tu, A. Fox, M. J. Franklin, D. A. Patterson, N. Lanham, B. Trushkowsky, and J. Trutna. PIQL: a performance insightful query language. In *SIGMOD*, 2010.

[4] BEAS. *http://139.196.196.250:8000/BEAS*.

[5] Y. Cao and W. Fan. An effective syntax for bounded relational queries. In *SIGMOD*, 2016.

[6] Y. Cao, W. Fan, F. Geerts, and P. Lu. Bounded query rewriting using views. In *PODS*, 2016.

[7] Y. Cao, W. Fan, T. Wo, and W. Yu. Bounded conjunctive queries. *PVLDB*, 2014.

[8] W. Fan, F. Geerts, Y. Cao, and T. Deng. Querying big data by accessing small data. In *PODS*, 2015.

[9] W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *PODS*, 2014.

[10] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, 2000.

[11] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3), 1996.