# An Effective Syntax for Bounded Relational Queries

Yang Cao Wenfei Fan

University of Edinburgh
Beihang University

SIGMOD 2016
June 29, 2016

## Background and Motivation

- Motivation: querying big data is cost-prohibitive
  - old challenge: complexity of query evaluation
  - new challenge: big data as the input

## Background and Motivation

- Motivation: querying big data is cost-prohibitive
  - old challenge: complexity of query evaluation
  - new challenge: big data as the input
- A recent approach: bounded evaluatility

> *querying big data by accessing small data of bounded size*

## Background and Motivation

- Motivation: querying big data is cost-prohibitive
  - old challenge: complexity of query evaluation
  - new challenge: big data as the input
- A recent approach: bounded evaluatility

*querying big data by accessing small data of bounded size*

- Previous works
  - Formalization
    ① *On Scale Independence for Querying Big Data*, W. Fan, F. Geerts and L. Libkin, PODS'14
    ② *Querying big data by accessing small data*, W. Fan, F. Geerts, Y. Cao, T. Deng, P. Lu, PODS'15
  - Incorporating views
    ③ *Bounded Query Rewriting Using Views*, Y. Cao, W. Fan, F. Geerts and P. Lu, PODS'16
  - Extending to graph data
    ④ *Making Pattern Queries Bounded in Big Graphs*, Y. Cao, W. Fan and R. Huang, ICDE'15
  - Restrictions and validation
    ⑤ *Bounded Conjunctive Queries*, Y. Cao, W. Fan. T. Wo and W. Yu, VLDB'14

# Background and Motivation

- Motivation: querying big data is cost-prohibitive
  - old challenge: complexity of query evaluation
  - new challenge: big data as the input
- A recent approach: bounded evaluatility

*querying big data by accessing small data of bounded size*

- Previous works
  - Formalization

    ① *On Scale Independence for Querying Big Data*, W. Fan, F. Geerts and L. Libkin, PODS'14

    ② *Querying big data by accessing small data*, W. Fan, F. Geerts, Y. Cao, T. Deng, P. Lu, PODS'15
  - Incorporating views

    ③ *Bounded Query Rewriting Using Views*, Y. Cao, W. Fan, F. Geerts and P. Lu, PODS'16
  - Extending to graph data

    ④ *Making Pattern Queries Bounded in Big Graphs*, Y. Cao, W. Fan and R. Huang, ICDE'15
  - Restrictions and validation

    ⑤ *Bounded Conjunctive Queries*, Y. Cao, W. Fan. T. Wo and W. Yu, VLDB'14
- The method was recently tested by a giant company: 3 to 1000+ times faster than conventional approach.

# Background and Motivation

- Motivation: querying big data is cost-prohibitive
  - old challenge: complexity of query evaluation
  - new challenge: big data as the input
- A recent approach: bounded evaluatility

*querying big data by accessing small data of bounded size*

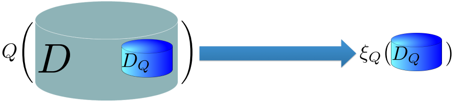
- Previous works
  - Formalization
    ① *On Scale Independence for Querying Big Data*, W. Fan, F. Geerts and L. Libkin, PODS'14
    ② *Querying big data by accessing small data*, W. Fan, F. Geerts, Y. Cao, T. Deng, P. Lu, PODS'15
  - Incorporating views
    ③ *Bounded Query Rewriting Using Views*, Y. Cao, W. Fan, F. Geerts and P. Lu, PODS'16
  - Extending to graph data
    ④ *Making Pattern Queries Bounded in Big Graphs*, Y. Cao, W. Fan and R. Huang, ICDE'15
  - Restrictions and validation
    ⑤ *Bounded Conjunctive Queries*, Y. Cao, W. Fan. T. Wo and W. Yu, VLDB'14
- The method was recently tested by a giant company: 3 to 1000+ times faster than conventional approach.

*Bounded evaluability is effective for querying big data*

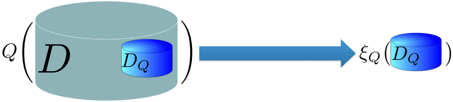
# Bounded Evaluability: Review

## Answering queries with bounded data access

Basic idea: compute $Q(D)$ via query plans that access only a small subset $D_Q$ of $D$, via indices built *w.r.t.* a set $\mathcal{A}$ of *access constraints*.



Basic idea: compute $Q(D)$ via query plans that access only a small subset $D_Q$ of $D$, via indices built *w.r.t.* a set $\mathcal{A}$ of *access constraints*.

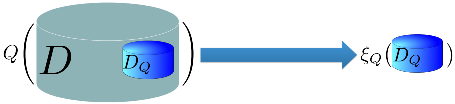



- Access constraints: a combination of cardinality constraints $R(X \to Y, N)$ with indices;

# Bounded Evaluability: Review

## Answering queries with bounded data access

Basic idea: compute $Q(D)$ via query plans that access only a small subset $D_Q$ of $D$, via indices built *w.r.t.* a set $\mathcal{A}$ of *access constraints*.

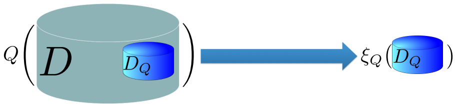



- Access constraints: a combination of cardinality constraints $R(X \to Y, N)$ with indices;
- Boundedly evaluable query plan $\xi_Q$: extended relational algebra query plans with a fetch operation fetch$(X \in T, R, Y)$ *w.r.t.* $\mathcal{A}$;

# Bounded Evaluability: Review

## Answering queries with bounded data access

Basic idea: compute $Q(D)$ via query plans that access only a small subset $D_Q$ of $D$, via indices built *w.r.t.* a set $\mathcal{A}$ of *access constraints*.

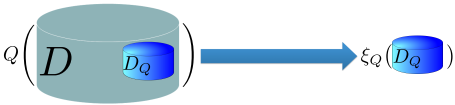



- Access constraints: a combination of cardinality constraints $R(X \to Y, N)$ with indices;
- Boundedly evaluable query plan $\xi_Q$: extended relational algebra query plans with a fetch operation fetch$(X \in T, R, Y)$ *w.r.t.* $\mathcal{A}$;
- Boundedly evaluable queries: queries with boundedly evaluable plans under $\mathcal{A}$.

# Bounded Evaluability: Review

## Answering queries with bounded data access

Basic idea: compute $Q(D)$ via query plans that access only a small subset $D_Q$ of $D$, via indices built *w.r.t.* a set $\mathcal{A}$ of *access constraints*.




- Access constraints: a combination of cardinality constraints $R(X \to Y, N)$ with indices;
- Boundedly evaluable query plan $\xi_Q$: extended relational algebra query plans with a fetch operation fetch$(X \in T, R, Y)$ *w.r.t.* $\mathcal{A}$;
- Boundedly evaluable queries: queries with boundedly evaluable plans under $\mathcal{A}$.

*Making the cost of computing $Q(D)$ independent of $|D|$!*

# Bounded Evaluability: An Example

## Graph search (Facebook)

Find me restaurants in San Francisco my Facebook friends have been to in 2015

**select** rid
**from** friend($\text{pid}_1$, $\text{pid}_2$), person(pid, name, city),
dine(pid, rid, dd, mm, yy)
**where** $\text{pid}_1 = p_0$ **and** $\text{pid}_2$ = person.pid **and**
$\text{pid}_2$ = dine.pid **and** city = SF **and** yy = 2015

# Bounded Evaluability: An Example

## Graph search (Facebook)

Find me restaurants in San Francisco my Facebook friends have been to in 2015

$$Q(\mathsf{rid}) = \exists p, p_1, n, c, d, m, y\,(\mathsf{friend}(p_0, p) \wedge \mathsf{person}(p, n, \mathrm{SF}) \wedge \mathsf{dine}(p, \mathsf{rid}, d, m, 2015))$$

Access constraints (cardinality + indices) from data semantics

# Bounded Evaluability: An Example

## Graph search (Facebook)

Find me restaurants in San Francisco my Facebook friends have been to in 2015

$$Q(\mathsf{rid}) = \exists p, p_1, n, c, d, m, y\,(\mathsf{friend}(p_0, p) \wedge \mathsf{person}(p, n, \mathrm{SF}) \wedge \mathsf{dine}(p, \mathsf{rid}, d, m, 2015))$$

Access constraints (cardinality + indices) from data semantics

- $\mathsf{friend}(\mathsf{pid}_1, \mathsf{pid}_2)$: $\mathsf{pid}_1 \rightarrow (\mathsf{pid}_2, 5000)$ 5000 friends per person
- $\mathsf{dine}(\mathsf{pid}, \mathsf{rid}, \mathsf{dd}, \mathsf{mm}, \mathsf{yy})$: $\mathsf{pid}, \mathsf{yy} \rightarrow (\mathsf{rid}, 366)$ each year has at most 366 days and each person dines at most once per day
- $\mathsf{person}(\mathsf{pid}, \mathsf{name}, \mathsf{city})$: $\mathsf{pid} \rightarrow (\mathsf{city}, 1)$ pid is a key for person

# Bounded Evaluability: An Example

## Graph search (Facebook)

Find me restaurants in San Francisco my Facebook friends have been to in 2015

$$Q(\mathsf{rid}) = \exists p, p_1, n, c, d, m, y\,(\mathsf{friend}(p_0, p) \wedge \mathsf{person}(p, n, \mathrm{SF}) \wedge \mathsf{dine}(p, \mathsf{rid}, d, m, 2015))$$

Access constraints (cardinality + indices) from data semantics

A boundedly evaluable query plan:

- Fetch 5000 pid's $(p)$ for friends of $p_0$ – 5000 friends per person
- For each $p$, check whether she lives in $\mathrm{SF}$ – 5000 person tuples
- For each $p$ living in $\mathrm{SF}$, find restaurants (rid) where she dined in 2015 – $5000 \times 366$ tuples at most

## Bounded Evaluability: An Example

### Graph search (Facebook)

Find me restaurants in San Francisco my Facebook friends have been to in 2015

$$Q(\text{rid}) = \exists p, p_1, n, c, d, m, y\,(\text{friend}(p_0, p) \wedge \text{person}(p, n, \text{SF}) \wedge \text{dine}(p, \text{rid}, d, m, 2015))$$

Access constraints (cardinality + indices) from data semantics

A boundedly evaluable query plan:

- Fetch 5000 pid's $(p)$ for friends of $p_0$ – 5000 friends per person
- For each $p$, check whether she lives in SF – 5000 person tuples
- For each $p$ living in SF, find restaurants (rid) where she dined in 2015 – $5000 \times 366$ tuples at most

Accessing $5000 + 5000 + 5000 \times 366$ tuples in total

## Bounded Evaluability: An Example

### Graph search (Facebook)

Find me restaurants in San Francisco my Facebook friends have been to in 2015

$$Q(\mathsf{rid}) = \exists p, p_1, n, c, d, m, y\, (\mathsf{friend}(p_0, p) \wedge \mathsf{person}(p, n, \mathrm{SF}) \wedge \mathsf{dine}(p, \mathsf{rid}, d, m, 2015))$$

Access constraints (cardinality + indices) from data semantics

Price to use bounded evaluability:

- **EXPSPACE-hard** to decide whether an SPC query (CQ; basic SELECT-FROM-WHERE clause) is boundedly evaluable
- **undecidable for RA (FO; SQL) queries**

# Bounded Evaluability: An Example

## Graph search (Facebook)

Find me restaurants in San Francisco my Facebook friends have been to in 2015

$$Q(\mathsf{rid}) = \exists p, p_1, n, c, d, m, y\, (\mathsf{friend}(p_0, p) \wedge \mathsf{person}(p, n, \mathrm{SF}) \wedge \mathsf{dine}(p, \mathsf{rid}, d, m, 2015))$$

Access constraints (cardinality + indices) from data semantics

Price to use bounded evaluability:

- **EXPSPACE-hard** to decide whether an SPC query (CQ; basic SELECT-FROM-WHERE clause) is boundedly evaluable
- **undecidable for RA (**FO; SQL**) queries**

How to make *practical* use of bounded evaluability *without sacrificing* its express power?

# An Effective Syntax for Bounded Evaluable RA Queries

## Covered queries

Covered RA queries $\mathcal{L}^{c}_{\mathsf{RA}}$: RA queries whose relation atoms are all "syntactically" covered by access constraints.

## An Effective Syntax for Bounded Evaluable RA Queries

### Covered queries

Covered RA queries $\mathcal{L}^c_{\mathsf{RA}}$: RA queries whose relation atoms are all "syntactically" covered by access constraints.

**Effective Syntax**

Under a set $\mathcal{A}$ of access constraints,

1. **any** boundedly evaluable RA query is $\mathcal{A}$-**equivalent** to a query covered by $\mathcal{A}$;
2. **every** covered query is **also** boundedly evaluable;
3. it takes **PTIME** to check whether $Q$ is covered by $\mathcal{A}$.

Covered RA queries $\mathcal{L}^c_{\mathsf{RA}}$: RA queries whose relation atoms are all "syntactically" covered by access constraints.

**Effective Syntax**

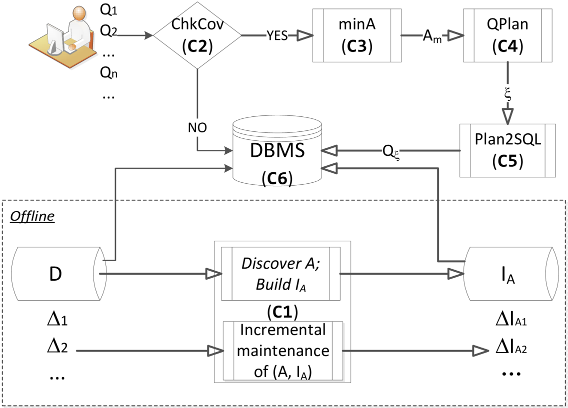
Under a set $\mathcal{A}$ of access constraints,

1. **any** boundedly evaluable RA query is $\mathcal{A}$**-equivalent** to a query covered by $\mathcal{A}$;
2. **every** covered query is **also** boundedly evaluable;
3. it takes **PTIME** to check whether $Q$ is covered by $\mathcal{A}$.

$\mathcal{L}^c_{\mathsf{RA}}$ *identifies the* core *subclass of boundedly evaluable RA queries,* **without sacrificing their expressive power**

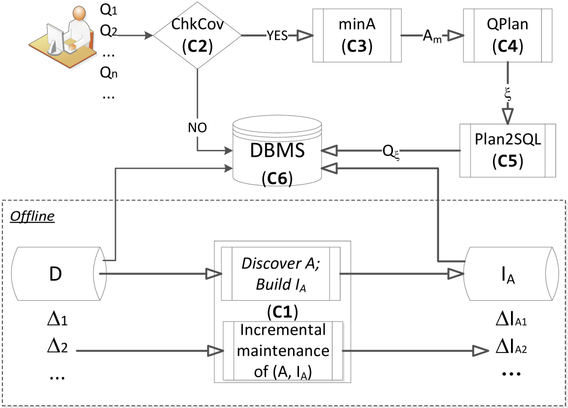
# A Bounded Evaluation Framework

## A constructive proof of the effective syntax (2-3)



# A Bounded Evaluation Framework

## A constructive proof of the effective syntax (2-3)




- **C2**: a proof of property 3
- **C4**: a proof of property 2
- **C3**: optimizing bounded plans
- **C5**: ensure DBMS-independence

# Experimental results

We evaluated the bounded evaluation framework:

- easy-to-use
  - easy to find 100+ access constraints in real-life data by extending constraints discovery algorithms
  - half of queries over the attributes in the constraints are covered
  - DBMS-independence:
    - access constraints index is easy to build via DBMS
    - bounded plans can be directly executed on DBMS engines
- speedup of boundedly evaluable plans vs conventional
  - 5.9 seconds by accessing at most 0.00017% of the data (60GB) VS. 3000+ seconds (3.75GB) for many-join queries
  - guaranteed scale independence once covered

## Experimental results

We evaluated the bounded evaluation framework:

- easy-to-use
  - easy to find 100+ access constraints in real-life data by extending constraints discovery algorithms
  - half of queries over the attributes in the constraints are covered
  - DBMS-independence:
    - access constraints index is easy to build via DBMS
    - bounded plans can be directly executed on DBMS engines
- speedup of boundedly evaluable plans vs conventional
  - 5.9 seconds by accessing at most 0.00017% of the data (60GB) VS. 3000+ seconds (3.75GB) for many-join queries
  - guaranteed scale independence once covered

*With the effective syntax, we can use bounded evaluability to query big data by accessing bounded small data.*

# The End

THANK YOU!

Q&A