Scale Independence: Using Small Data to Answer Queries on Big Data

Floris Geerts

University of Antwerp
Query answering on big data

- Queries can be **slow** on big data due to the **size of the data**:

  ![Cylinder](image)

  \( Q \)

- Current database technology tells us how to **quickly** answer queries on **“normal”-sized** data.
Query answering on big data

- It would be great if we can answer $Q$ on a **big database** using a **small database** inside it!
- One could then rely on **existing database technology** to answer queries on **big data**.

\[ Q \quad = \quad Q(\text{big data}) \]
Scale independent queries

This idea has been pursued before...

“Scale independent queries that satisfy their performance objectives on small data sizes will continue to meet those objectives as the database size grows, …”

Query scaling classes

- Armbrust et al. distinguish between different query classes.
- Depending on **how much data is needed to answer** them:
  - Class I: constant amount of data (key value);
  - Class II: bounded amount of data (5000 facebook friends);
  - Class III: (sub-)linear amount of data;
  - Class IV: superlinear (cartesian product).

- **Class I & II queries** are the focus of this talk.
Questions:

1. Which queries are scale independent?

2. How to define this notion?
Scale independent queries

A query $Q$ is **scale independent in a database $D$** if

- $Q(D) = Q(D_Q)$ for some **part** $D_Q \subseteq D$; and
- the size $|D_Q|$ of $D_Q$ is **independent** of the size $|D|$ of $D$.

A query $Q$ is **scale independent** if it is scale independent in **all databases**.

To answer scale independent queries one only needs a **bounded amount of data**.

Checking scale independence

Not surprisingly, for FO queries:

- Testing scale independence is undecidable.
- The class of scale independent FO queries is not even recursively enumerable.

Scale independence can also be uninteresting.

For (U)CQ queries:

- Only trivial CQ queries can be scale independent, e.g., queries that output a constant tuple on all databases.
- This is due to monotonicity.
Undecidable or uninteresting: End of story...

Thank you for your attention.
Scale Independence

We need some stronger assumptions on the data
**PIQL: Performance-Insightful Query Language**

Armbrust et al. propose an extension of SQL that allows to express

- **relationship cardinalities;**
- **result size** requirements

+ compiler that **bounds** the number of operations performed by query.

**Schema:**

```
facebook(member_id, friend_id)
```

**Constraint:**

```
facebook[member_id -> friend_id, 5000]
```

**Query:**

```
SELECT friend_id
FROM facebook
WHERE member_id=Trump
```

⇒ Only requires **fetching** at most 5000 tuples.

(Probably less for Trump)

---

Constraints are **crucial** in PIQL:

Relation `facebook(id1, id2)`
+ **cardinality constraint** `facebook[id1 -> id2, 5000]` (aka Facebook constraint)

Relation `person(id, name, city)`
+ **key constraint** `person[id -> {person, city},1]`

These constraints:

- **bound** the amount of data; and
- specify **access patterns** (how to access the data)

} access constraints
Access constraints

\[
\begin{align*}
\text{person}[\text{id} & \rightarrow \{\text{person}, \text{city}\}, 1] \\
\text{facebook}[\text{id1} & \rightarrow \text{id2}, 5000]
\end{align*}
\]

- **Available indexes:**
  - Given an id-value one can efficiently fetch the corresponding name and city values from the person table.
  - Similarly for the facebook relation.

- **Cardinality constraints:**
  - At most one tuple in person table for each id-value (key);
  - At most 5000 friends for each member of Facebook.

Can we make queries scale independent by using such constraints?
Example: Query evaluation using access constraints

Consider query

\[ Q(\text{Trump}, \text{name}) = \exists id \text{ facebook}(\text{Trump}, id) \land \text{person}(id, \text{name}, \text{NYC}) \]

Execution plan:

1. **Fetch all** id-values associated with Trump using **index** facebook[id1 -> id2]
   \[ \Rightarrow \text{at most 5000 tuples.} \]

2. **For each** of the fetched id-values, **fetch** unique tuple in person using **index** person[id-> \{ name, city\}]
   \[ \Rightarrow \text{at most 1 tuple per id-value.} \]

3. **Return all fetched** name-values for persons living in NYC.
   \[ \Rightarrow \text{at most 10000 tuples in total.} \]

If such execution plan exists, then we say that a query is **boundedly evaluable.**
Boundedly evaluable queries

We next define what boundedly evaluable queries are.
Databases that satisfy access constraints

- A database $D$ conforms to a set $\mathcal{A}$ of access constraints, if it satisfies these constraints.
  - For each $R(X \rightarrow Y, N)$ in $\mathcal{A}$, we have that $|\pi_Y(\sigma_{X=a}(D))| \leq N$ for any constant tuple $\bar{a}$.

- When looking at query equivalence, we mean $\mathcal{A}$-equivalence, i.e., equivalence on all databases that conform to the access constraints in $\mathcal{A}$. 
A query $Q$ is **boundedly evaluable** relative to a set $\mathcal{A}$ of access constraints if

- there exists a **bounded query plan** $\xi_Q$ such that
- for all databases $D$ conform to $\mathcal{A}$

$$Q(D) = \xi_Q(D).$$

**Bounded query plans** ensure that only a **bounded amount of data is fetched** from the underlying data.

Bounded query plans (bqplan)

- Starting from basic **fetch operators**, pipelining using constraints in $\mathcal{A}$:
  
  $$
  \text{bqplan} = \text{fetch}(X = \bar{a}, R, Y)
  $$
  
  $$
  \text{bqplan} = \text{fetch}(X \in \text{bqplan}, R, Y)
  $$

- **SPJ-plan**: Allowing **projection**, **selection**, **renaming** and **product**:
  
  $$
  \text{bqplan} = \pi_X(\text{bqplan})
  $$
  
  $$
  \text{bqplan} = \sigma_C(\text{bqplan})
  $$
  
  $$
  \text{bqplan} = \rho_{A/B}(\text{bqplan})
  $$
  
  $$
  \text{bqplan} = \text{bqplan} \times \text{bqplan}
  $$

- **SPJU-plan**: Further allowing **union**:
  
  $$
  \text{bqplan} = \text{bqplan} \cup \text{bqplan}
  $$

- **RA-plan**: Also allowing **difference**:
  
  $$
  \text{bqplan} = \text{bqplan} \setminus \text{bqplan}
  $$
Example: Bounded query plan

Consider query

\[ Q(\text{Trump, name}) = \exists id \ \text{facebook}(\text{Trump, id}) \land \text{person}(\text{id, name, NYC}) \]

Bounded query plan:

1. \( \text{bqplan}_1(\text{id}_1, \text{id}_2) = \text{fetch}(\text{id}_1 = \text{Trump, facebook, id}_2) \)
2. \( \text{bqplan}_2(\text{id}_2) = \pi_{\text{id}_2}(\text{bqplan}_1) \)
3. \( \text{bqplan}_3(\text{id, name, city}) = \text{fetch}(\text{id} \in \text{bqplan}_2, \text{person}, (\text{name, city})) \)
4. \( \text{bqplan}_4(\text{id, name, city}) = \sigma_{\text{city} = \text{NYC}}(\text{bqplan}_3) \)
5. \( \text{bqplan}(\text{id}_1, \text{name}) = \pi_{\text{id}_1}(\text{bqplan}_1) \times \pi_{\text{name}}(\text{bqplan}_4). \)

On databases \( D \) conform to \( A \), this query plan correctly evaluates \( Q \) and fetches a bounded amount of data.
Question:

- What do we gain by this definition?
Boundedly evaluable queries

Bounded evaluation makes scale independence a bit more interesting

Although it is still **undecidable** to decide whether an FO query is boundedly evaluable...

...the **presence** of access constraints allow to identify **interesting classes of queries** that are boundedly evaluable.

And this for CQ, UCQ, and FO queries.
Question:

- Which queries are boundedly evaluable?
- Déjà vu?
Querying under access patterns...

Looks similar to the work on **querying under limited access patterns** by Li, Chang, Deutsch, Nash, Ludäscher and others.

Consider query

\[
Q(\text{Trump, name}) = \exists \text{id friend}(\text{Trump}^i, \text{id}^o) \land \text{person}(\text{id}^i, \text{name}^o, \text{NYC}^o)
\]

and **access patterns** `friend(id^i, id^o)` and `person(id^i, name^o, city^o)` indicating **in-and output positions**.

Can it be answered using a **valid access pattern sequence** (query plan)?

- ...
Querying under access patterns

**Syntactic condition:**
A query is orderable when one can parse it from left to right using access patterns.

- Orderable queries have a **linear executing plan** using access patterns.

**Semantic condition:**
Query is executable/stable if it is equivalent to an orderable query.

- Characterizations and complexity results for executable queries are known.

Most of this work considers CQ and UCQ, but also fragments of FO.
Querying under access patterns vs access constraints

However,

- Access patterns cover all attributes in relations; access constraints are more flexible; and
- No cardinality constraints are embedded in access patterns.
- Standard equivalence vs. $\mathcal{A}$-equivalence.
- Query plans are not bounded.
Querying under access patterns vs access constraints

Nevertheless, it serves as inspiration:

<table>
<thead>
<tr>
<th>Access patterns</th>
<th>Access constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable/stable queries</td>
<td>Boudedly evaluable queries</td>
</tr>
<tr>
<td>Orderable queries</td>
<td>??</td>
</tr>
</tbody>
</table>

We next generalize the notion of orderable queries in the context of access constraints.
Covered queries

1. Define a **syntactic fragment** of conjunctive queries:
   \[ \Rightarrow \text{covered queries}. \]
2. Covered queries are **boundedly evaluable** (SPJ-plan).
3. Every boundedly evaluable CQ is \( \mathcal{A} \)-equivalent to a covered CQ.

<table>
<thead>
<tr>
<th>Access patterns</th>
<th>Access constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable/stable queries</td>
<td>Boundedly evaluable queries</td>
</tr>
<tr>
<td>Orderable queries</td>
<td>covered queries</td>
</tr>
</tbody>
</table>

Covered conjunctive queries

Intuitively, a conjunctive query is covered if

(i) all its relevant variables are bounded by access constraints
(ii) all its relations are properly indexed.

Precise definition uses deduction rules that propagate information on bounds and indexes based on the structure of the query.

It is in \textsc{Ptime} to check whether a CQ is covered.

If covered, the successful deductive proof generates a bounded query plan. (Hence, covered queries are indeed boundedly evaluable.)
Covered CQs: Deduction rules

(i) **Bounding the free variables** using access constraints.

Deduction rules $\mathcal{I}_{Bnd}$:

- **(Reflexivity)**
  
  If $\bar{x}' \subseteq \bar{x}$ then $\bar{x} \rightarrow \mathcal{I}_{Bnd} (\bar{x}', 1)$

- **(Actualization)**
  
  If $R(X \rightarrow Y, N) \in \mathcal{A}$ then $\bar{x} \rightarrow \mathcal{I}_{Bnd} (\bar{y}, N)$

- **(Augmentation)**
  
  If $\bar{x} \rightarrow \mathcal{I}_{Bnd} (\bar{y}, N)$ then $\bar{x} \cup \bar{z} \rightarrow \mathcal{I}_{Bnd} (\bar{y} \cup \bar{z}, N)$

- **(Transitivity)**
  
  If $\bar{x} \rightarrow \mathcal{I}_{Bnd} (\bar{y}_1, N_1)$ and $\bar{y}_1 \rightarrow \mathcal{I}_{Bnd} (\bar{z}, N_2)$ then $\bar{x} \rightarrow \mathcal{I}_{Bnd} (\bar{z}, N_1 \cdot N_2)$

A conjunctive query $Q(\bar{x})$ is **bounded** if for each $x \in \bar{x}$

$$\Sigma_Q \rightarrow \mathcal{I}_{Bnd} (x, N_x)$$

for some $N_x \in \mathbb{N}$, where $\Sigma_Q$ are the variables in $Q$ bound to a constant.
Example: Query with “bounded” variables

Consider query

\[ Q(id_1, \text{name}) = \exists \text{id, city } \text{facebook}(id_1, \text{id}) \land \text{person}(id', \text{name, city}) \land id_1 = Trump \land \text{city = NYC} \land id = id' \]

Is the variable “name” bounded?
1. \( \Sigma_Q = \{id_1, \text{city}\} \)
2. \( \Sigma_Q \rightarrow I_{Bnd} (id, 5000) \) (Actualization)
3. \( id' \rightarrow I_{Bnd} (\text{name, city, 1}) \) (Actualization)
4. \( \Sigma_Q \rightarrow I_{Bnd} (\text{name, city, 5000}) \) (Transitivity)

Similarly for variable \( id_1 \). Hence,

\( \Sigma_Q \rightarrow I_{Bnd} \{ (id_1, 1), (\text{name, 5000}) \} \).

It takes \( O(|Q|(|A| + |Q|)) \) time to check whether a CQ query is bounded.
Example: Boundedness alone does not suffice

Consider query:

\[ Q(\text{user}, \text{photo}, \text{time}, \text{location}) = \text{Instagram}(\text{user}, \text{photo}, \text{time}, \text{location}) \]
\[ \wedge \text{user} = \text{Trump} \wedge \text{location} = \text{Bordeaux}. \]

Access constraints:

\[ \text{Instagram}[(\text{user}, \text{location}) \rightarrow (\text{photo}, N)] \]
\[ \text{Instagram}[(\text{user}, \text{location}) \rightarrow (\text{time}, N')] \]

Then, using the deduction rules one can show that all variables in \( Q \) are bounded.

Nevertheless, \( Q \) is not boundedly evaluable.
Example: Boundedness alone does not suffice

Access constraints

Instagram[(user, location) \rightarrow (photo, N)] \quad \text{Instagram}[(user, location) \rightarrow (time, M)]

Indexes can only fetch parts of the relation and full relation cannot be recovered (lossy decomposition):

\[
\begin{array}{c|c|c|c}
\text{Trump} & \text{photo1} & \text{time1} & \text{Bordeaux} \\
\text{Trump} & \text{photo2} & \text{time2} & \text{Bordeaux} \\
\text{Trump} & \text{photo3} & \text{time3} & \text{Bordeaux} \\
\end{array}
\neq
\begin{array}{c|c|c|c}
\text{Trump} & \text{photo1} & \text{Bordeaux} \\
\text{Trump} & \text{photo2} & \text{Bordeaux} \\
\text{Trump} & \text{photo3} & \text{Bordeaux} \\
\end{array}
\times
\begin{array}{c|c|c|c}
\text{time1} & \text{photo1} & \text{Bordeaux} \\
\text{time2} & \text{photo2} & \text{Bordeaux} \\
\text{time3} & \text{photo3} & \text{Bordeaux} \\
\end{array}
Covered CQs: Revised deduction rules

Ensure that access constraints **suffice to correctly check existence of tuples in base relations**.

\[
\text{Instagram}[(\text{user, location}) \rightarrow (\text{photo, N})] \quad \text{Instagram}[(\text{user, location}) \rightarrow (\text{time, N}')] \\
\]

Additional access constraint is needed, e.g.,

\[
\text{Instagram}[(\text{photo, time}) \rightarrow (\text{user, photo, time, location, N''})] \\
\]

A **refinement** of the deduction rules $\mathcal{I}_{Bnd}$ can be defined such that when all relevant variables are **bounded and indexed**, then the query is boundedly evaluable.

Automatic bounded query plan generation

Access constraints:

Instagram[(user, location) \rightarrow (photo, N)]
Instagram[(user, location) \rightarrow (time, N')]
Instagram[(photo, time) \rightarrow (user, photo, time, location, N)]

Deductive proof automatically gives bounded query plan:

1. \( \text{bqplan}_1(\text{photo}) = \pi_{\text{photo}}(\text{fetch}((\text{Trump}, \text{Bordeaux}), \text{instagram}, \text{photo})) \)
2. \( \text{bqplan}_2(\text{time}) = \pi_{\text{time}}(\text{fetch}((\text{Trump}, \text{Bordeaux}), \text{instagram}, \text{time})) \)
3. \( \text{bqplan}_3(\text{photo}, \text{time}) = \text{bqplan}_1(\text{photo}) \times \text{bqplan}_2(\text{time}) \)
4. \( \text{bqplan}_4(\text{user}, \text{photo}, \text{time}, \text{location}) = \text{fetch}((\text{photo}, \text{time}) \in \text{bqplan}_3, \text{instagram}, (\text{user}, \text{photo}, \text{time}, \text{location})) \)
5. \( \text{bqplan}_5(\text{user}, \text{photo}, \text{time}, \text{location}) = \sigma_{\text{name}=\text{Trump} \land \text{location}=\text{Bordeaux}}(\text{bqplan}_4) \).
Question:

- How good are these query plans in practice?
Covered CQs: Experiments

Data:

- UK traffic accident data (19 tables, 113 attributes, 89.7 million tuples, 21.4GB)
- Ministry of Transport Test data (1 table, 36 attributes, 55 million tuples, 16.2GB)
- TPCH (restricted to 8 tables, varying sizes up to 32GB)

Access constraints:

- UK traffic accident data: 84 constraints (e.g., [date -> (aid, 610)])
- Ministry of Transport Test data: 27 constraints
- TPCH: 61 constraints

Queries: 15 queries on each dataset (varying selection conditions and # joins).
Covered CQs: Experiments - TPCH

- 70% of queries turned out to be boundedly evaluable (when all access constraints were “on”)

Comparison between generated bounded query plan vs mysql query plan:

![Graph 1: Elapsed time vs |D|](image1)

![Graph 2: Elapsed time vs |A|](image2)

Semantic characterization?

Being covered is a **syntactic condition**, i.e., it depends on how the query is written.

We also want a **semantic characterization**:

Suppose that $Q$ is not covered. Is $Q$ $\mathcal{A}$-equivalent to a query $Q'$ that is covered?

Clearly, such queries can also be evaluated in scale independent way:

- Simply execute the bounded query plan for $Q'$. 
### Decision algorithm

Given CQ $Q$, is $Q \equiv_A$-equivalent to a covered CQ?

1. **Decompose** $Q \equiv_A Q_1 \cup \cdots \cup Q_k$.
   - Each $Q_i \models A$ and no redundant $Q_i$’s.

2. **Compute** the infimum query $\inf_A(Q)$ of $\{Q_1, \ldots, Q_k\}$.
   - For any other CQ $Q'$ such that $Q_i \subseteq Q'$ for all $i$, we have that $\inf_A(Q) \subseteq Q'$.

3. **Construct** $A$-expansion $\exp_A(Q)$ of $\inf_A(Q)$.
   - All possible “covered” atoms embedded in $\inf_A(Q)$ are added.

4. **Identify** maximal covered subquery $Q_c$ in $\exp_A(Q)$.

**Theorem**

A conjunctive query $Q$ is $A$-equivalent to a covered CQ $Q$ if and only if $Q$ is $A$-equivalent to the covered CQ $Q_c$. 

Complexity

- The characterisation implies a coNEXPTIME upper bound:
  - exponential number of base queries $Q_i$;
  - infimum query is exponential in the number (and size) of base queries.
- Lower bound is open.
Beyond CQ

What can we say about other query languages?
Boundedly evaluable queries: UCQ

For unions of conjunctive queries (UCQ)

- Bounded query plans may use union (SPJU-plan).
- Notion of covered UCQ can be defined.
- Every boundedly evaluable UCQ is $\mathcal{A}$-equivalent to a covered UCQ.
- Without impact on complexity.

Similarly for conjunctive queries with (nested) unions.

Boundedly evaluable queries: First-order logic

A notion of covered FO queries has recently been proposed:

1. **Convert** FO query $Q$ to relational algebra expression $e_Q$.
2. Require in the **query tree** $T_{e_Q}$ of $e_Q$ that:
   - Every max conjunctive subtree is covered.

(I.e., difference is pushed to top levels on unions of covered sub-queries)

**Theorem**

*Every covered FO query is boundedly evaluable (RA-plan) and every boundedly evaluable FO query is $A$-equivalent to a covered FO query.*

It takes **PTIME** to check whether an FO query is covered.

---

Yang Cao, Wenfei Fan: An Effective Syntax for Bounded Relational Queries. SIGMOD 2016.
Boundedly evaluable queries: First-order logic

A RA bounded query plan can be generated for covered FO queries.

Underlying idea:

1. Encode Q and A as a hypergraph;
2. A hyperpath corresponds to a bounded query plan.

Algorithm: Find a hyperpath.

Experiments show that these query plans also outperform those used by mysql.
Generating plans from proofs

In recent work by Benedikt et al., the following approach is followed:

1. Isolate a **semantic property** that any input query $Q$ must have with respect to the class of target plans in order to have an **equivalent** plan of the desired type.

2. **Express** this property as a **proof goal**: a statement that formula $\phi_2$ follows from $\phi_1$.

3. **Search for a proof** of the entailment, within a given proof system.

4. From the **proof**, **extract a plan**.

---

Semantic property: Access determinacy

In the context of access patterns (not access constraints!):

A query $Q$ is said to access determined if
- for any $D$ and $D'$ that have the same accessible part
  \[
  \text{AccPart}(D) = \text{AccPart}(D')
  \]
- it holds that $Q(D) = Q(D')$.

Intuitively, $\text{AccPart}(D)$ are all values that can be accessed from $D$.

Clearly, $Q$ cannot be answered using access patterns if it is not access-determined.
Access determinacy: Entailment

An FO query $Q$ is access determined if and only if

$$Q \land Access^+ \models Q_{acc}$$

where $Q_{acc}$ is the inferred accessible version of $Q$

- obtained by replacing each $R$ in $Q$ by its accessible part

and $Access^+$ is an axiomatization of accessibility:

- rules that tell what is accessible and what not, based on access patterns.
A query $Q$ is said to **access monotonic determined** if

- for any $D$ and $D'$ that have **contained accessible parts**
  \[
  \text{AccPart}(D) \subseteq \text{AccPart}(D')
  \]
- it holds that $Q(D) \subseteq Q(D')$. 
Access monotonic determinacy: Entailment

A CQ query $Q$ is **access monotonic determined** if and only if

$$Q \land \text{Access} \models Q_{\text{acc}}$$

where $Q_{\text{acc}}$ is the inferred accessible version of $Q$ and **Access** is an axiomatization of accessibility:

- rules that tell what is accessible, based on access patterns.
Access determinacy: Plans from proofs

**Nice property:** Chase proofs witnessing

\[ Q \land \text{Access} \models Q_{acc} \]

or

\[ Q \land \text{Access}^+ \models Q_{acc} \]

result in **SPJ and RA-plans**, for CQ and FO queries, respectively.

Furthermore, **cost functions** can be incorporated to find **cost optimal** proofs (plans).
Future work: Bounded access (monotonic) determinacy?

The following seems a natural thing to try:

1. Define a notion of **bounded** access (monotonic) determinacy.
2. Consider **access constraints** instead of access patterns.
3. Taking into account $A$-equivalence.
4. **Extract** bounded query plans from **proofs**.

    TODO...
Recap

- Successfully identified class of **covered queries** that are **boundedly evaluable**.

- **Every boundedly evaluable query** is \( A \)-equivalent to **covered** one.

- Definition of covered queries “implies” bounded query plan **generation procedure**.

- Generated query plans **work well in practice**.
Other issues

- Scale independent query approximation.
- Incremental scale independence.
- Scale independence using views.
Scale independent query approximation

Given a query $Q$ that is not boundedly evaluable.

Find two boundedly evaluable queries $Q_{\ell}$ (lower envelope) and $Q_{u}$ (upper envelope) such that

$$Q_{\ell} \sqsubseteq A Q \sqsubseteq A Q_{u}.$$ 

and $Q_{\ell}$ is maximal and $Q_{u}$ is minimal wrt $A$-containment.

- Solved for CQ, when $Q_{u}$ and $Q_{\ell}$ are assumed to be covered CQ queries.
- Characterization and complexity results are known.
- Full treatment is required....

Query approximation: Example

Consider
\[ Q(x) = \exists y, z, w \ (R(w, x) \land R(y, w) \land R(x, z) \land w = 1) \]
and access constraint
\[ R(A \rightarrow B, N). \]
Then \( Q \) is not \textbf{boundedly evaluable}.

We can sandwich \( Q \) between two boundedly evaluable queries:

\[ Q_\ell(x) = \exists y, z \ (R(1, x) \land R(y, 1) \land R(x, z) \land R(x, y)) \]

and
\[ Q_u(x) = \exists y, z \ (R(1, x) \land R(x, z)). \]

Furthermore, \(|Q(D) \setminus Q_\ell(D)| \leq N\) and \(|Q_u(D) \setminus Q(D)| \leq N\).

Such envelopes, if they exist, can be obtained by \textbf{relaxing and generalizing} the input query.
Incremental scale independence

\[ \Delta D = (\Delta D, \nabla D): \text{List of tuples } \Delta D \text{ to be inserted into } D \text{ and a list } \nabla D \text{ of tuples to be deleted.} \]

\[ \Delta Q = (\Delta Q, \nabla Q): \text{queries such that} \]

\[
Q((D \setminus \nabla D) \cup \Delta D)) = (Q(D) - \nabla Q(\Delta D, D)) \cup \Delta Q(\Delta D, D)
\]

Then, \( Q \) is incrementally boundedly evaluable iff

- \( \Delta Q \) is boundedly evaluable; and
- \( \nabla Q \) is boundedly evaluable.

That is, to incrementally answer \( Q \) in \( D \) in response to \( \Delta D \), we need to access a bounded number of tuples from \( D \).

Scale independence using views

**Enlarge** the class of boundedly evaluable queries by using **cached views**:
  - Cached views allow **fast access** ⇒ **all view data** can be used.

**Extension of** bounded query plans:
  - Allow to fetch data from views in an **unrestricted way**.

**Complication:**
  - Ensure that views only pass a **bounded** amount of data to indexes (access constraints) on **base relations**.

Complexity results and effective syntax for CQ and FO are established, assuming a (constant) bound on the size of query plans.

Conclusion

- Boundedly evaluable queries are a **nice concept** with interesting links to
  - Safety;
  - Querying using access patterns;
  - Access determinacy and query rewriting.

- Main complications arise from the presence of **cardinality constraints**.

- Experiments show that bounded query plans can **outperform** query plans suggested by optimizer.
Conclusion

- I did not mention **complexity results** for various associated decision problems.

- These can be found here:
Future work

Looking ahead

- **Bounded access determinacy** and generation of bounded query plans from proofs.

- Enlarge class of covered FO queries.

- **Index suggestion** to make queries boundedly evaluable.

- Integration with integrity constraints.

- Scale-independence in a distributed/parallel context.

- Scale-independence on graph data and query languages.
Thank you. The End. Questions?

(Thanks to Wenfei Fan, Leonid Libkin, Cao Yang, Ting Deng, Ping Lu)

(and don’t forget to send your best work to PODS 2017, 1st deadline June 17, 2016)