LinCQA: Faster Consistent Query Answering with Linear Time Guarantees

Xiating Ouyang [1]

joint work with Zhiwei Fan [1,2] Paris Koutris [1] Jef Wijsen [3]

University of Wisconsin–Madison [1]

Meta [2]

University of Mons [3]

SIGMOD, Seattle WA, June 18–23 2023

Should I accept the CS PhD offer from UW?

Yes!

Can I go skiing when studying at UW?

Yes!

What about the 9 months of rain?

Which UW?

Should I accept the CS PhD offer from UW?

Yes!

Can I go skiing when studying at UW?

Yes!

What about the 9 months of rain?

Which UW?

Should I accept the CS PhD offer from UW?

Yes!

Can I go skiing when studying at UW?

Yes!

What about the 9 months of rain?

Which UW?

Should I accept the CS PhD offer from UW?

Yes!

Can I go skiing when studying at UW?

Yes!

What about the 9 months of rain?

Which UW?

Should I accept the CS PhD offer from UW?

Yes!

Can I go skiing when studying at UW?

Yes!

What about the 9 months of rain?

Which UW?

Should I accept the CS PhD offer from UW?

Yes!

Can I go skiing when studying at UW?

Yes!

What about the 9 months of rain?

Which UW?

Should I accept the CS PhD offer from UW?

Yes!

Can I go skiing when studying at UW?

Yes!

What about the 9 months of rain?

Which UW?

Acronym	School	Great CS	Skiing	Rain
UW	U of WA	Yes	Yes	Yes
UW	U of WI	Yes	Yes	No

Should I accept the CS PhD offer from UW? **Yes!**

Acronym	School	Great CS	Skiing	Rain
UW	U of WA	Yes	Yes	Yes
UW	U of WI	Yes	Yes	No

Should I accept the CS PhD offer from UW? Yes!

Acronym	School	Great CS	Skiing	Rain
UW	U of WA	Yes	Yes	Yes
UW	U of WI	Yes	Yes	No

Should I accept the CS PhD offer from UW? **Yes!**

Primary key constraint (violated)

- Metadata of `stackoverflow.com` as of 02/2021 from Stack Exchange Data Dump
- 551M rows, ∼400 GB

Table	# of rows	inconsistencyRatio	blockSize	# of Attributes
Users	14M	0%	1	14
Posts	53M	0%	1	20
PostHistory	141M	0.001%	4	9
Badges	40M	0.58%	941	4
Votes	213M	30.9%	1441	6

inconsistencyRatio = # facts violating PK constraint / # of rows

blockSize = max. # facts with the same PK

1. Consistent Query Answering for Primary Keys
2. Results
3. Techniques

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

$Q(\mathbf{db}) = \{\text{CS 703, CS 787}\} \ldots$

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

$Q(\mathbf{db}) = \{\text{CS 703, CS 787}\}$ ...

Data cleaning

$Q(\mathbf{rep})$

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

$Q(\mathbf{db}) = \{\text{CS 703, CS 787}\}$ ...

Data cleaning $2 \times 2 \times 2 \times 1$ repairs

$Q(\mathbf{rep})$

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

$Q(\mathbf{db}) = \{\text{CS 703, CS 787}\}$ ...

Data cleaning $2 \times 2 \times 2 \times 1$ repairs

Which answers are guaranteed to be returned on all repairs?

$Q(\mathbf{rep})$

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

$Q(\mathbf{db}) = \{\text{CS 703, CS 787}\}$ ...

Data cleaning $2 \times 2 \times 2 \times 1$ repairs

Which answers are guaranteed to be returned on all repairs?

$$\bigcap_{\mathbf{rep} \text{ is a repair of } \mathbf{db}} Q(\mathbf{rep})$$

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

$Q(\mathbf{db}) = \{\text{CS 703, CS 787}\}$ ...

Data cleaning $2 \times 2 \times 2 \times 1$ repairs

Which answers are guaranteed to be returned on all repairs?

$$\bigcap_{\mathbf{rep} \text{ is a repair of } \mathbf{db}} Q(\mathbf{rep}) = \{\mathbf{CS\ 703}\}$$

Finding consistent answers

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id
```

$Q(\mathbf{db}) = \{\text{CS 703, CS 787}\}$ ...

Data cleaning $2 \times 2 \times 2 \times 1$ repairs

Which answers are guaranteed to be returned on all repairs?

$\bigcap$ $Q(\mathbf{rep}) = \{\mathbf{CS\ 703}\}$

rep is a repair of **db**

Consistent Answer

Finding consistent answers without enumeration

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id  AND
  (all f_id's for the same c_id
  appear in CS_Faculty)
```

$$Q'(\mathbf{db}) = \bigcap_{\mathbf{rep} \text{ is a repair of } \mathbf{db}} Q(\mathbf{rep})$$

The original query Q has a first-order rewriting Q'

Finding consistent answers without enumeration

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id  AND
  (all f_id's for the same c_id
  appear in CS_Faculty)
```

$$Q'(\mathbf{db}) = \bigcap_{\mathbf{rep} \text{ is a repair of } \mathbf{db}} Q(\mathbf{rep})$$

The original query Q has a first-order rewriting Q'

Finding consistent answers without enumeration

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id  AND
  (all f_id's for the same c_id
  appear in CS_Faculty)
```

$$Q'(\mathbf{db}) = \bigcap_{\mathbf{rep} \text{ is a repair of } \mathbf{db}} Q(\mathbf{rep})$$

The original query Q has a first-order rewriting Q'

Finding consistent answers without enumeration

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id  AND
  (all f_id's for the same c_id
  appear in CS_Faculty)
```

$$Q'(\mathbf{db}) = \bigcap_{\mathbf{rep} \text{ is a repair of } \mathbf{db}} Q(\mathbf{rep})$$

The original query Q has a first-order rewriting Q'

Finding consistent answers without enumeration

Course

c_id	f_id
CS 703	2
CS 703	5
CS 787	3
CS 787	5

CS_Faculty

f_id	f_name
2	Adam
2	Alice
5	Bob

```
SELECT DISTINCT c_id
FROM Course, CS_Faculty
WHERE Course.f_id
      = CS_Faculty.f_id  AND
  (all f_id's for the same c_id
  appear in CS_Faculty)
```

$$Q'(\mathbf{db}) = \bigcap_{\mathbf{rep} \text{ is a repair of } \mathbf{db}} Q(\mathbf{rep})$$

The original query Q has a first-order rewriting Q'

For which Q can the consistent answers be found efficiently?

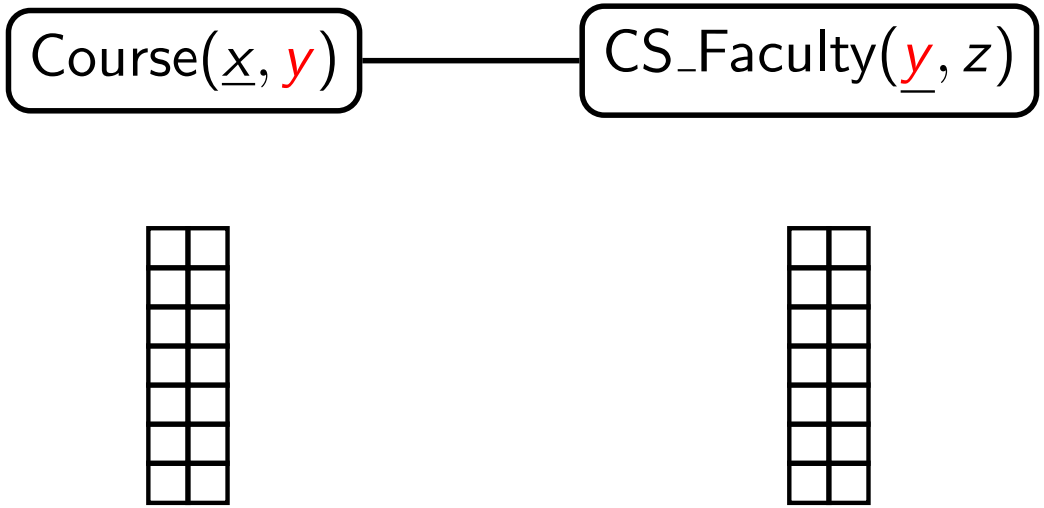
Can we build a system to find the consistent answers?

For which Q can the consistent answers be found efficiently?

Can we build a system to find the consistent answers?

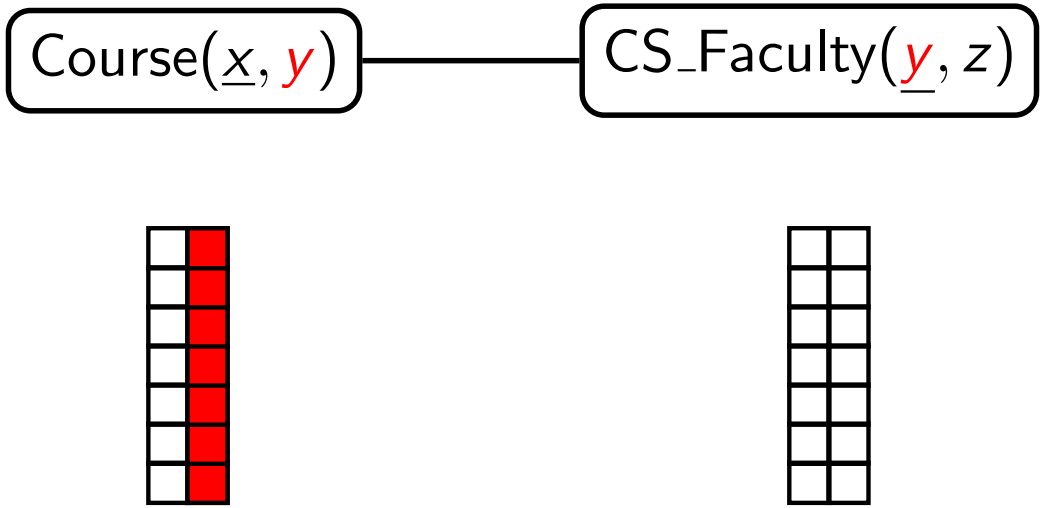
Acyclic query evaluation

$q() \text{ :- } \text{Course}(\underline{x}, y), \text{CS\_Faculty}(\underline{y}, z)$



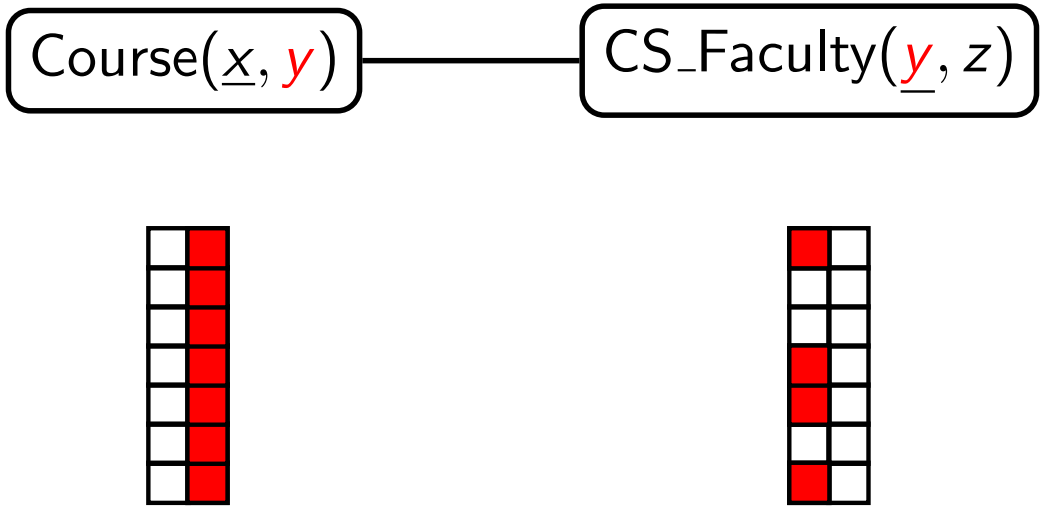
Acyclic query evaluation

$q() \text{ :- } \text{Course}(\underline{x}, y), \text{CS\_Faculty}(\underline{y}, z)$



Acyclic query evaluation

$q() \text{ :- } \text{Course}(\underline{x}, y), \text{CS\_Faculty}(\underline{y}, z)$



Yannakakis [VLDB'81]

The answer to every **Boolean** acyclic query can be computed in $O(|\mathbf{db}|)$.

Yannakakis [VLDB'81] Our result

consistent answer

The ~~answer~~ to every **Boolean** acyclic query can be computed in $O(|\mathbf{db}|)$.

$\wedge$

with a pair-pruning join tree (PPJT)

Yannakakis [VLDB'81] Our result

consistent answer

The ~~answer~~ to every **Boolean** acyclic query can be computed in $O(|\mathbf{db}|)$.

$\wedge$

with a pair-pruning join tree (PPJT)

non-Boolean $\leq_T^P$ **Boolean**

PPJT is a wide class

+ $\subset$ **S**election, **P**rojection, **J**oin queries
+ star/snowflake schema (e.g. TPC-H, TPC-DS)
+ two distinct table join
+ Every acyclic query in $\mathcal{C}_{\text{forest}}$ [ICDT'05, SIGMOD'05] has a PPJT

– no self-joins! [PODS'18,20,22] [PODS'21]

– no aggregation (yet) [ICDE, 2022] [ICDT, 2022]

– no cyclic primary keys joins

$$q() \text{ :- } R(\underline{x}, y), S(\underline{y}, x)$$

– no non-key to non-key joins

$$q() \text{ :- } R(\underline{x}, z), S(\underline{y}, z)$$

PPJT is a wide class

+ $\subset$ **S**election, **P**rojection, **J**oin queries
+ star/snowflake schema (e.g. TPC-H, TPC-DS)
+ two distinct table join
+ Every acyclic query in $\mathcal{C}_{\text{forest}}$ [ICDT'05, SIGMOD'05] has a PPJT

– no self-joins! [PODS'18,20,22] [PODS'21]

– no aggregation (yet) [ICDE, 2022] [ICDT, 2022]

– no cyclic primary keys joins

$$q() \text{ :- } R(\underline{x}, y), S(\underline{y}, x)$$

– no non-key to non-key joins

$$q() \text{ :- } R(\underline{x}, z), S(\underline{y}, z)$$

```
SELECT
  DISTINCT Posts.Id, Posts.Title
FROM
  Posts, PostHistory, Votes, Comments
WHERE
  Posts.Tags LIKE "%SQL%"
  AND Posts.id = PostHistory.PostId
  AND Posts.id = Comments.PostId
  AND Posts.id = Votes.PostId
  AND Votes.BountyAmount > 100
  AND PostHistory.PostHistoryTypeId = 2
  AND Comments.score = 0
```

```
WITH candidates AS (
  SELECT
    DISTINCT C.UserId, C.CreationDate, P.Id, P.Title
  FROM
    Posts P, PostHistory PH, Votes V, Comments C
  WHERE
    P.Tags LIKE "%SQL%"
    AND P.id = PH.PostId
    AND P.id = C.PostId
    AND P.id = V.PostId
    AND V.BountyAmount > 100
    AND PH.PostHistoryTypeId = 2
    AND C.score = 0
),

Posts_bad_key AS (
  SELECT P.Id
  FROM Posts P
  WHERE P.Tags not LIKE "%SQL%" OR P.Tags IS NULL

  UNION ALL

  SELECT Id
  FROM (
    SELECT distinct Id, Title
    FROM Posts
  )t
  GROUP BY Id
  HAVING count(*) > 1
),

Posts_good_join AS (
  SELECT P.Id, P.Title
  FROM Posts P
  WHERE NOT EXISTS (
    SELECT *
    FROM Posts_bad_key
    WHERE P.Id = Posts_bad_key.Id
  )
),

PostHistory_bad_key AS (
  SELECT PH.PostId, PH.CreationDate, PH.UserId,
      PH.PostHistoryTypeId
  FROM PostHistory PH
  WHERE PH.PostHistoryTypeId <> 2
),

PostHistory_good_join AS (
  SELECT PH.PostId
  FROM PostHistory PH
  WHERE NOT EXISTS (
    SELECT *
    FROM PostHistory_bad_key
    WHERE PH.PostId = PostHistory_bad_key.PostId AND
      PH.CreationDate = PostHistory_bad_key.CreationDate
        AND
      PH.UserId = PostHistory_bad_key.UserId AND
      PH.PostHistoryTypeId
        = PostHistory_bad_key.PostHistoryTypeId
  )
),

Votes_bad_key AS (
  SELECT V.PostId, V.UserId, V.CreationDate
  FROM Votes V
  WHERE V.BountyAmount <= 100 or V.BountyAmount IS null
),

Votes_good_join AS (
  SELECT V.PostId
  FROM Votes V
  WHERE NOT EXISTS (
    SELECT *
    FROM Votes_bad_key
    WHERE
    V.PostId = Votes_bad_key.PostId AND
    V.UserId = Votes_bad_key.UserId AND
    V.CreationDate = Votes_bad_key.CreationDate
  )
),

Comments_bad_key AS (
  SELECT C.CreationDate, C.UserId, candidates.Title
  FROM Comments C
  JOIN candidates ON (
    C.CreationDate = candidates.CreationDate
    AND C.UserId = candidates.UserId)
  WHERE C.score <> 0

  UNION ALL

  SELECT C.CreationDate, C.UserId, candidates.Title
  FROM Comments C
  JOIN candidates ON (
    C.CreationDate = candidates.CreationDate
    AND C.UserId = candidates.UserId)
  LEFT OUTER JOIN Posts_good_join ON (
    C.PostId = Posts_good_join.Id
    AND candidates.Title = Posts_good_join.Title)
  LEFT OUTER JOIN PostHistory_good_join ON (
    C.PostId = PostHistory_good_join.PostId)
  LEFT OUTER JOIN Votes_good_join ON (
    C.PostId = Votes_good_join.PostId)
  WHERE (
    Posts_good_join.Id IS NULL
    OR PostHistory_good_join.PostId IS NULL
    OR Votes_good_join.PostId IS NULL
    OR Posts_good_join.Title IS NULL
  )
),

Comments_good_join AS (
  SELECT candidates.Id, candidates.Title
```

Original query + primary key info $\xrightarrow{\text{LinCQA}}$ Query rewriting

```
WITH candidates AS (
  SELECT
    DISTINCT C.UserId, C.CreationDate, P.Id, P.Title
  FROM
    Posts P, PostHistory PH, Votes V, Comments C
  WHERE
    P.Tags LIKE "%SQL%"
    AND P.id = PH.PostId
    AND P.id = C.PostId
    AND P.id = V.PostId
    AND V.BountyAmount > 100
    AND PH.PostHistoryTypeId = 2
    AND C.score = 0
),

Posts_bad_key AS (
  SELECT P.Id
  FROM Posts P
  WHERE P.Tags not LIKE "%SQL%" OR P.Tags IS NULL

  UNION ALL

  SELECT Id
  FROM (
    SELECT distinct Id, Title
    FROM Posts
  )t
  GROUP BY Id
  HAVING count(*) > 1
),

Posts_good_join AS (
  SELECT P.Id, P.Title
  FROM Posts P
  WHERE NOT EXISTS (
    SELECT *
    FROM Posts_bad_key
    WHERE P.Id = Posts_bad_key.Id
  )
),

PostHistory_bad_key AS (
  SELECT PH.PostId, PH.CreationDate, PH.UserId,
      PH.PostHistoryTypeId
  FROM PostHistory PH
  WHERE PH.PostHistoryTypeId <> 2
),

PostHistory_good_join AS (
  SELECT PH.PostId
  FROM PostHistory PH
  WHERE NOT EXISTS (
    SELECT *
    FROM PostHistory_bad_key
    WHERE PH.PostId = PostHistory_bad_key.PostId AND
      PH.CreationDate = PostHistory_bad_key.CreationDate
        AND
      PH.UserId = PostHistory_bad_key.UserId AND
      PH.PostHistoryTypeId
        = PostHistory_bad_key.PostHistoryTypeId
  )
),

Votes_bad_key AS (
  SELECT V.PostId, V.UserId, V.CreationDate
  FROM Votes V
  WHERE V.BountyAmount <= 100 or V.BountyAmount IS null
),

Votes_good_join AS (
  SELECT V.PostId
  FROM Votes V
  WHERE NOT EXISTS (
    SELECT *
    FROM Votes_bad_key
    WHERE
    V.PostId = Votes_bad_key.PostId AND
    V.UserId = Votes_bad_key.UserId AND
    V.CreationDate = Votes_bad_key.CreationDate
  )
),

Comments_bad_key AS (
  SELECT C.CreationDate, C.UserId, candidates.Title
  FROM Comments C
  JOIN candidates ON (
    C.CreationDate = candidates.CreationDate
    AND C.UserId = candidates.UserId)
  WHERE C.score <> 0

  UNION ALL

  SELECT C.CreationDate, C.UserId, candidates.Title
  FROM Comments C
  JOIN candidates ON (
    C.CreationDate = candidates.CreationDate
    AND C.UserId = candidates.UserId)
  LEFT OUTER JOIN Posts_good_join ON (
    C.PostId = Posts_good_join.Id
    AND candidates.Title = Posts_good_join.Title)
  LEFT OUTER JOIN PostHistory_good_join ON (
    C.PostId = PostHistory_good_join.PostId)
  LEFT OUTER JOIN Votes_good_join ON (
    C.PostId = Votes_good_join.PostId)
  WHERE (
    Posts_good_join.Id IS NULL
    OR PostHistory_good_join.PostId IS NULL
    OR Votes_good_join.PostId IS NULL
    OR Posts_good_join.Title IS NULL
  )
),

Comments_good_join AS (
  SELECT candidates.Id, candidates.Title
```

Original query + primary key info $\xrightarrow{\text{LinCQA}}$ Query rewriting

Setup & Baselines

System	Target class	Interm. output	Backend
CAvSAT	*	SAT formula	SQL Server & MaxHS
Conquer	$\mathcal{C}_{\text{forest}}$	SQL	SQL Server
Improved Conquesto	SJF **FO**	SQL	SQL Server
LinCQA	PPJT	SQL	SQL Server

Stackoverflow data

- Metadata of `stackoverflow.com` as of 02/2021 from Stack Exchange Data Dump
- 551M rows, 400 GB

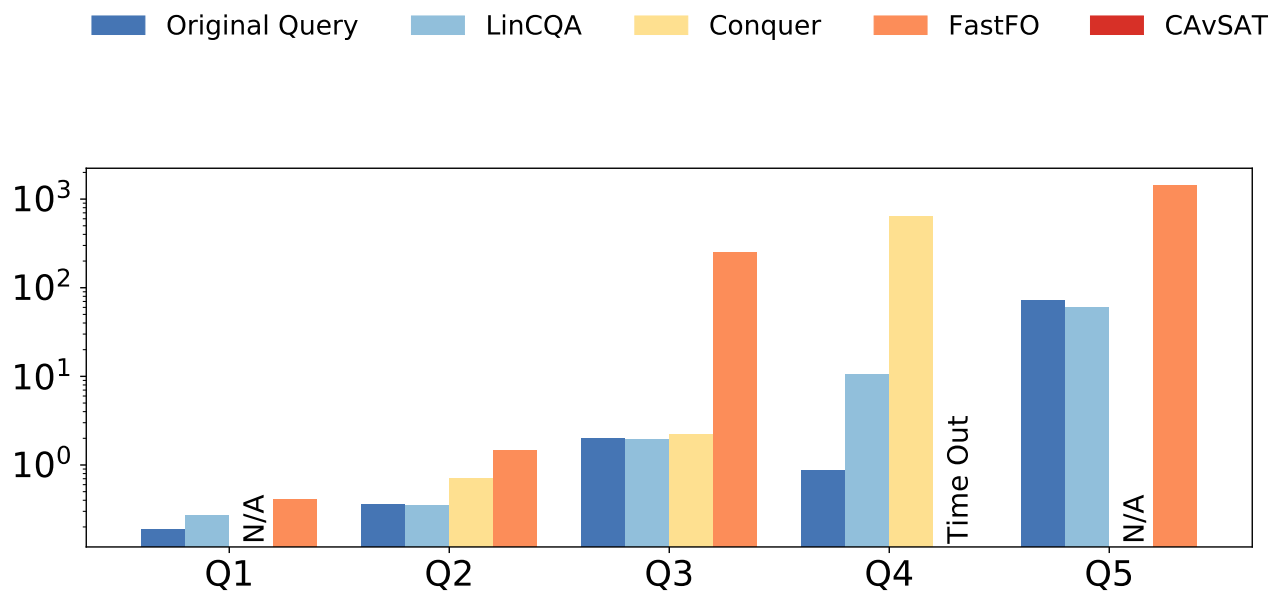
Table	# of rows	inconsistencyRatio	blockSize	# of Attributes
Users	14M	0%	1	14
Posts	53M	0%	1	20
PostHistory	141M	0.001%	4	9
Badges	40M	0.58%	941	4
Votes	213M	30.9%	1441	6

Experiments on Stackoverflow

Q_1 : Posts $\bowtie$ Votes Q_2 : Users $\bowtie$ Badges Q_3 : Users $\bowtie$ Posts

Q_4 : Users $\bowtie$ Posts $\bowtie$ Comments

Q_5 : Posts $\bowtie$ PostHistory $\bowtie$ Votes $\bowtie$ Comments



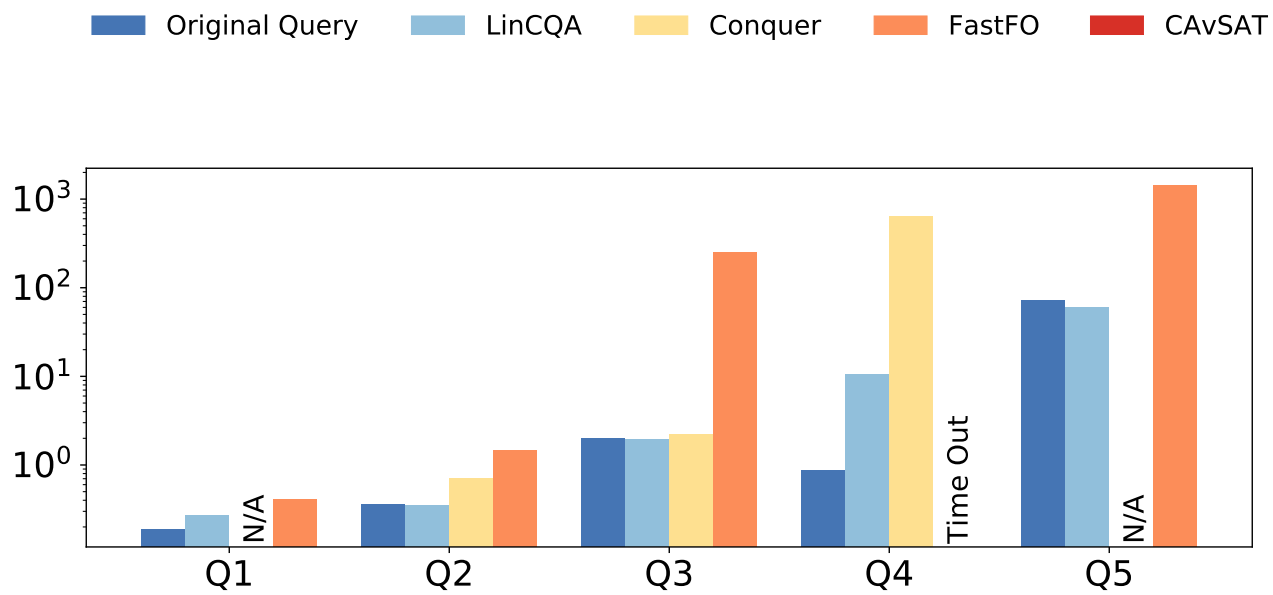
# poss.	27578	145	38320	3925	1250
# cons.	27578	145	38320	3925	1245

Experiments on Stackoverflow

Q_1 : Posts $\bowtie$ Votes Q_2 : Users $\bowtie$ Badges Q_3 : Users $\bowtie$ Posts

Q_4 : Users $\bowtie$ Posts $\bowtie$ Comments

Q_5 : Posts $\bowtie$ PostHistory $\bowtie$ Votes $\bowtie$ Comments



# poss.	27578	145	38320	3925	1250
# cons.	27578	145	38320	3925	1245

From PPJT to **FO**-rewriting

Remove a primary key if some tuple with this primary key is "bad"

$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\forall \text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})$

$\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!
runs in $O(N)$

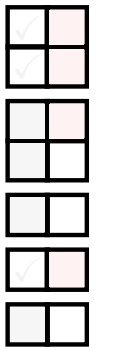
From PPJT to **FO**-rewriting

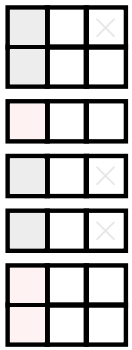
Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

Faculty($\underline{y}$, z, "DB")





$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})$

$\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!
runs in $O(N)$

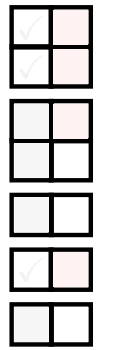
From PPJT to **FO**-rewriting

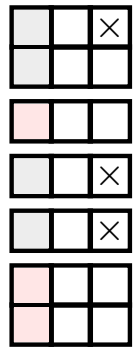
Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

Faculty($\underline{y}$, z, "DB")





$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})$

$\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

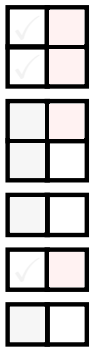
also expressible in SQL!

runs in $O(N)$

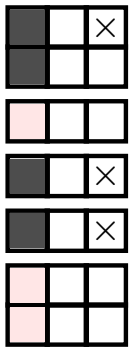
From PPJT to **FO**-rewriting

Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)



Faculty($\underline{y}$, z, "DB")



$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})$

$\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!

runs in $O(N)$

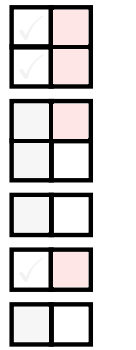
From PPJT to **FO**-rewriting

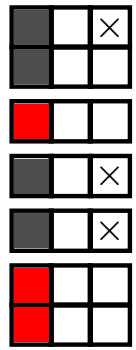
Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

Faculty($\underline{y}$, z, "DB")





$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})$

$\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!

runs in $O(N)$

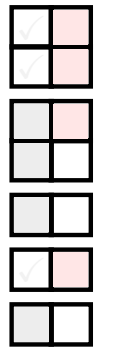
From PPJT to **FO**-rewriting

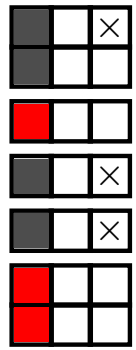
Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

Faculty($\underline{y}$, z, "DB")





$\text{Child}_{join}(\vec{\alpha})$:- $\text{Child}(\underline{\vec{u}}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})$

$\text{Faculty}_{join}(y)$:- $\text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y)$:- $\text{Faculty}(y, z, w), w \neq$ "DB"

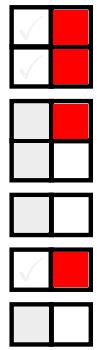
From PPJT to **FO**-rewriting

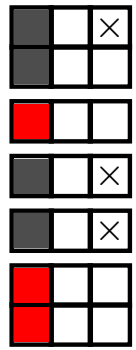
Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

Faculty($\underline{y}$, z, "DB")





$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})$

$\boxed{\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})}$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!

runs in $O(N)$

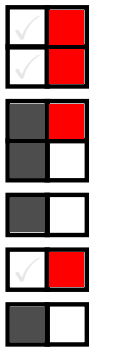
From PPJT to **FO**-rewriting

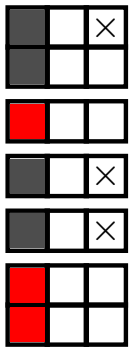
Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

Faculty($\underline{y}$, z, "DB")





$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})$

$\boxed{\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})}$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!

runs in $O(N)$

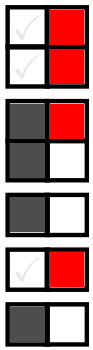
From PPJT to **FO**-rewriting

Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

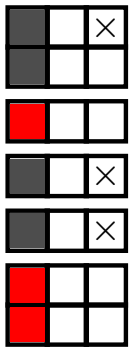
Faculty($\underline{y}$, z, "DB")



$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\boxed{\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\underline{\vec{x}}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})}$



$\boxed{\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\underline{\vec{u}}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})}$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!

runs in $O(N)$

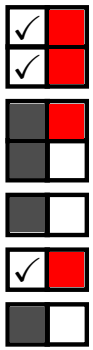
From PPJT to **FO**-rewriting

Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)

↓

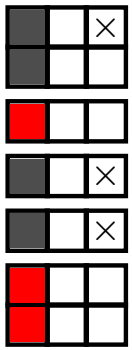
Faculty($\underline{y}$, z, "DB")



$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\boxed{\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\underline{\vec{x}}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})}$



$\boxed{\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\underline{\vec{u}}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})}$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

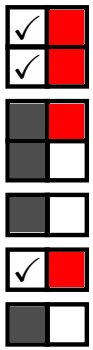
also expressible in SQL!

runs in $O(N)$

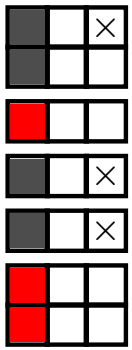
From PPJT to **FO**-rewriting

Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)



Faculty($\underline{y}$, z, "DB")



$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\boxed{\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\underline{\vec{x}}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})}$

$\boxed{\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\underline{\vec{u}}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})}$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

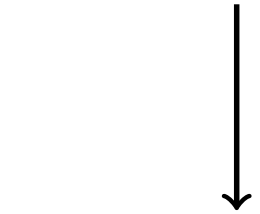
also expressible in SQL!

runs in $O(N)$

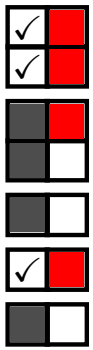
From PPJT to **FO**-rewriting

Remove a primary key if some tuple with this primary key is "bad"

Course($\underline{x}$, y)



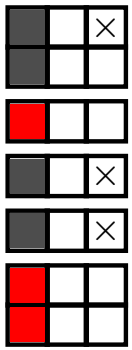
Faculty($\underline{y}$, z, "DB")



$\text{Course}_{join}() \text{ :- } \text{Course}(x, y), \neg\text{Course}_{fkey}(x)$

$\text{Course}_{fkey}(x) \text{ :- } \text{Course}(x, y), \neg\text{Faculty}_{join}(y)$

$\boxed{\forall\text{Child} : \text{Root}_{fkey}(\vec{x}) \text{ :- } \text{Root}(\underline{\vec{x}}, \vec{y}), \neg\text{Child}_{join}(\vec{\alpha})}$



$\boxed{\text{Child}_{join}(\vec{\alpha}) \text{ :- } \text{Child}(\underline{\vec{u}}, \vec{v}), \neg\text{Child}_{fkey}(\vec{u})}$

$\text{Faculty}_{join}(y) \text{ :- } \text{Faculty}(y, z, w), \neg\text{Faculty}_{fkey}(y)$

$\text{Faculty}_{fkey}(y) \text{ :- } \text{Faculty}(y, z, w), w \neq \text{"DB"}$

also expressible in SQL!
runs in $O(N)$

From Boolean to non-Boolean

```
SELECT DISTINCT A1, A2 FROM T WHERE A3 = 42
```

Step 1 Evaluate directly

A1	A2
a	b
x	y
...	...

Step 2 Reduce to **Boolean** (using PPJT)

```
SELECT DISTINCT 1 FROM T WHERE A3 = 42 AND A1 = a AND A2 = b
```

if **yes**, then output (a, b), otherwise continue

```
SELECT DISTINCT 1 FROM T WHERE A3 = 42 AND A1 = x AND A2 = y
```

...

$\xrightarrow{\text{LinCQA}}$ a single SQL/Datalog query

From Boolean to non-Boolean

```
SELECT DISTINCT A1, A2 FROM T WHERE A3 = 42
```

Step 1 Evaluate directly

A1	A2
a	b
x	y
...	...

Step 2 Reduce to **Boolean** (using PPJT)

```
SELECT DISTINCT 1 FROM T WHERE A3 = 42 AND A1 = a AND A2 = b
```

if **yes**, then output (a, b), otherwise continue

```
SELECT DISTINCT 1 FROM T WHERE A3 = 42 AND A1 = x AND A2 = y
```

...

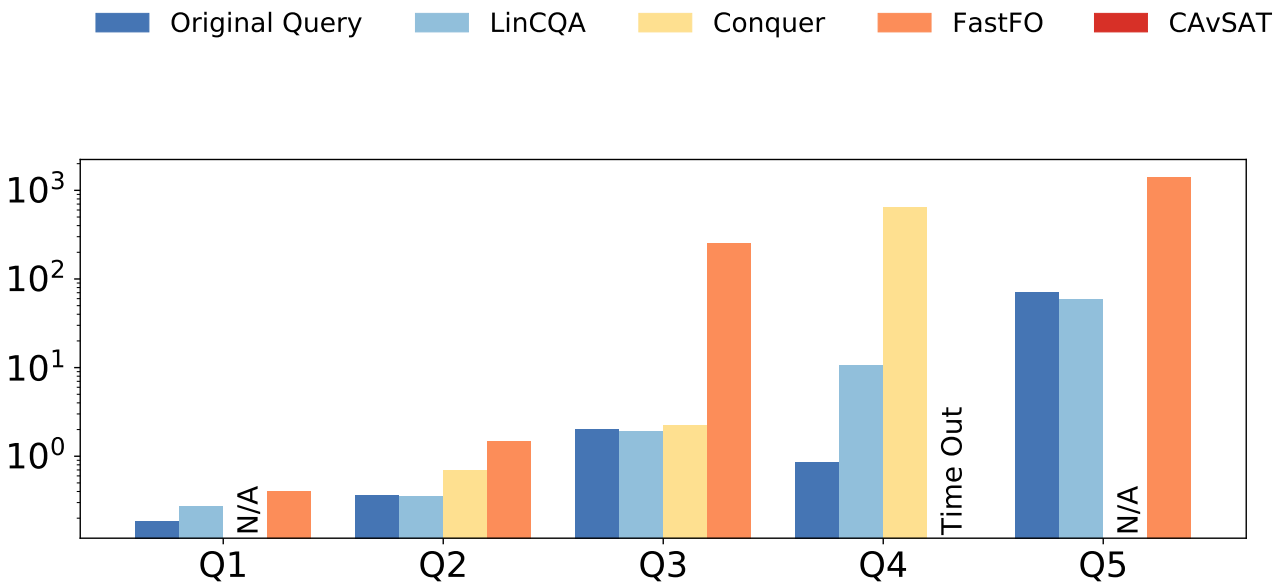
$\xrightarrow{\text{LinCQA}}$ a single SQL/Datalog query

Concluding remarks

Acyclic q	PPJT	Yannakakis [VLDB'81]
Boolean q	$O(N)$	$O(N)$
non-Boolean q	$O(N \cdot \lvert\mathsf{OUT}_{\mathsf{inconsistent}}\rvert)$	$O(N \cdot \lvert\mathsf{OUT}\rvert)$
full q (`SELECT *`)	$O(N + \lvert\mathsf{OUT}_{\mathsf{consistent}}\rvert)$	$O(N + \lvert\mathsf{OUT}\rvert)$

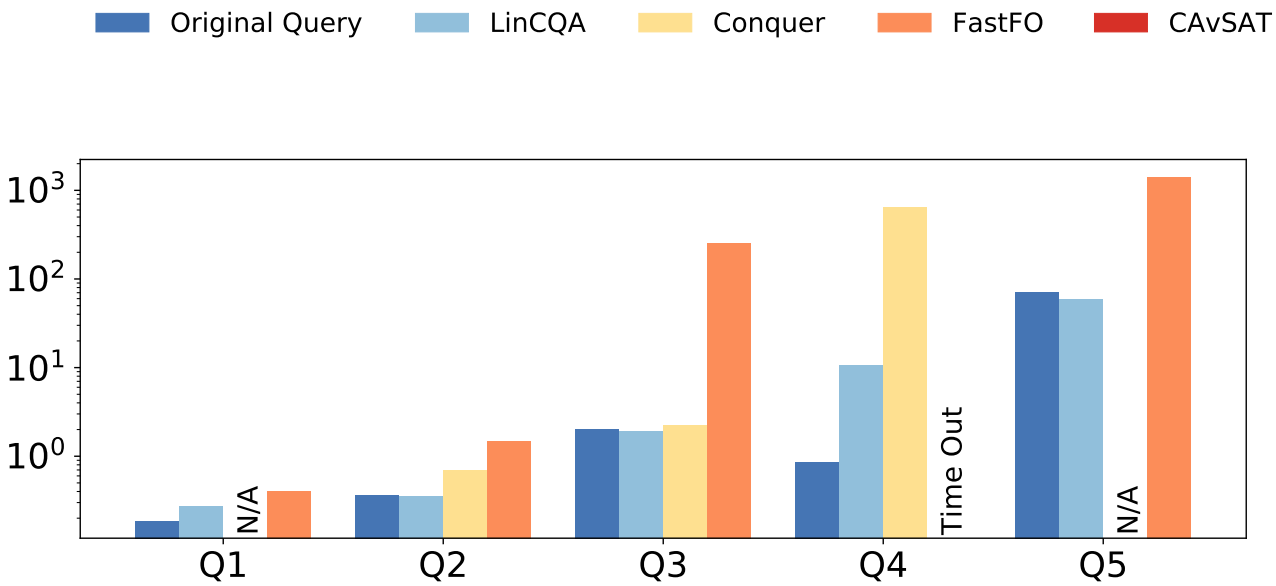
Concluding remarks

Acyclic q	PPJT	Yannakakis [VLDB'81]
Boolean q	$O(N)$	$O(N)$
non-Boolean q	$O(N \cdot \lvert \text{OUT}_{\text{inconsistent}} \rvert)$	$O(N \cdot \lvert \text{OUT} \rvert)$
full q (`SELECT *`)	$O(N + \lvert \text{OUT}_{\text{consistent}} \rvert)$	$O(N + \lvert \text{OUT} \rvert)$



Concluding remarks

Acyclic q	PPJT	Yannakakis [VLDB'81]
Boolean q	$O(N)$	$O(N)$
non-Boolean q	$O(N \cdot \lvert\text{OUT}_{\text{inconsistent}}\rvert)$	$O(N \cdot \lvert\text{OUT}\rvert)$
full q (`SELECT *`)	$O(N + \lvert\text{OUT}_{\text{consistent}}\rvert)$	$O(N + \lvert\text{OUT}\rvert)$



Thank you!

Xiating Ouyang `xouyang@cs.wisc.edu`