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               |

\[
\text{inconsistencyRatio} = \frac{\# \text{ facts violating PK constraint}}{\# \text{ of rows}}
\]

\[
\text{blockSize} = \max. \# \text{ facts with the same PK}
\]
Consistent Query Answering for Primary Keys

Results

Techniques
Consistent Query Answering for Primary Keys

Results

Techniques
Finding consistent answers

| Course | 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 | 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

\[ Q(db) = \{\text{CS 703, CS 787}\} \]
Finding consistent answers

| Course | 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

\[ Q(db) = \{CS 703, CS 787\} \ldots \]

Data cleaning

Q(rep)
Finding consistent answers

| Course | 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(db) = {CS 703, CS 787} ...

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

Q(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

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

Data cleaning

2 × 2 × 2 × 1 repairs

Which answers are guaranteed to be returned on all repairs?

$Q(rep)$
Finding consistent answers

| Course | CS 703 | 2 |
|--------|--------|---|
| Course | CS 703 | 5 |
| Course | CS 787 | 3 |
| Course | 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

\[ Q(\text{db}) = \{\text{CS 703, CS 787}\} \ldots \]

Data cleaning \[ 2 \times 2 \times 2 \times 1 \text{ repairs} \]

Which answers are guaranteed to be returned on all repairs?

\[ \bigcap_{\text{rep is a repair of db}} Q(\text{rep}) \]
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(db) = {CS 703, CS 787} ...

Data cleaning

2 x 2 x 2 x 1 repairs

Which answers are guaranteed to be returned on all repairs?

\[ Q(rep) = \{CS 703\} \]

\[ rep \text{ is a repair of } db \]
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

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

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

Which answers are guaranteed to be returned on all repairs?

\[
\bigcap_{\text{rep}} Q(\text{rep}) = \{\text{CS 703}\}
\]

(rep is a repair of db)

Consistent Answer
Finding consistent answers without enumeration

| Course | CS 703 | CS 703 | CS 787 | CS 787 |
|--------|--------|--------|--------|--------|
| c_id   | 2      | 5      | 3      | 5      |
| f_id   |        |        |        |        |

| 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)

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

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)

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
(aall f_id’s for the same c_id
appear in CS_Faculty)

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

The original query \( Q \) has a first-order rewriting \( Q' \).
Finding consistent answers without enumeration

| Course | 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)

\[ Q'(db) = \bigcap_{rep} Q(rep) \]

rep is a repair of db

The original query Q has a first-order rewriting Q'
Finding consistent answers without enumeration

| Course | 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)
```

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

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?
Consistent Query Answering for Primary Keys

Results

Techniques
Acyclic query evaluation

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

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

Course\((x, y)\) \quad \text{CS\_Faculty}\((y, z)\)
Acyclic query evaluation

$q() \leftarrow \text{Course}(x, y), \text{CS_Faculty}(y, z)$
The answer to every \textbf{Boolean} acyclic query can be computed in $O(|db|)$. 
Yannakakis [VLDB’81] Our result

consistent answer

The answer to every \textbf{Boolean} acyclic query can be computed in $O(|db|)$. with a pair-pruning join tree (PPJT)
consistent answer

The answer to every **Boolean** acyclic query can be computed in $O(|db|)$. with a pair-pruning join tree (PPJT)

non-Boolean $\leq_{PT}^{P}$ **Boolean**
PPJT is a wide class

- ⊂ Selection, Projection, Join queries
- star/snowflake schema (e.g. TPC-H, TPC-DS)
- two distinct table join
- Every acyclic query in $C_{forest}$ [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

\[ q() \leftarrow R(x, y), S(y, x) \]

- no non-key to non-key joins

\[ q() \leftarrow R(x, z), S(y, z) \]
PPJT is a wide class

+ $\subset$ Selection, Projection, Join queries
+ star/snowflake schema (e.g. TPC-H, TPC-DS)
+ two distinct table join
+ Every acyclic query in $C_{\text{forest}}$ [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

$$q() : \neg R(x, y), S(y, x)$$

- no non-key to non-key joins

$$q() : \neg R(x, z), S(y, z)$$
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.Id, C.Date, P.Id, P.Title
    FROM Posts P, PostHistory PH, Votes V, Comments C
    WHERE P.Tags LIKE '%SQL%'
    AND P.Id = PH.PostId
    AND C.Id = PH.PostId
    AND C.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-posts-bad_key
    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.Date, 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.Date = PostHistory-bad_key.Date AND
        PH.UserId = PostHistory-bad_key.UserId AND
        PH.PostHistoryTypeId = PostHistory-bad_key.PostHistoryTypeId)
),
Votes_bad_key AS (SELECT V.PostId, V.UserId, V.Date
    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.Date = Votes-bad_key.Date)
),
Comments_bad_key AS (SELECT C.Date, C.UserId, candidates.Title
    FROM Comments C
    JOIN candidates ON (C.Date = candidates.Date AND C.UserId = candidates.UserId)
    WHERE C.Score > 0
    UNION ALL
    SELECT C.Date, C.UserId, candidates.Title
    FROM Comments C
    JOIN candidates ON (C.Date = candidates.Date AND C.UserId = candidates.UserId)
    LEFT OUTER JOIN Posts_good_join ON (C.PostId = Posts_good_join.PostId 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

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 = V.PostId  
    AND V.VoteTypeId = 1  
    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.PostHistoryTypeId = PostHistory_bad_key.PostHistoryTypeId  
    )  
  ),  
  Votes_bad_key AS (  
    SELECT V.PostId, V.UserId, V.CreationDate  
    FROM Votes V  
    WHERE V.VoteTypeId = 1 OR V.VoteTypeId 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
## Setup & Baselines

| System                  | Target class | Interm. output | Backend            |
|-------------------------|--------------|----------------|--------------------|
| CAvSAT                  | *            | SAT formula    | SQL Server & MaxHS|
| Conquer                 | $C_{\text{forest}}$ | 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

\[ Q_1 : \text{Posts} \bowtie \text{Votes} \quad Q_2 : \text{Users} \bowtie \text{Badges} \quad Q_3 : \text{Users} \bowtie \text{Posts} \]

\[ Q_4 : \text{Users} \bowtie \text{Posts} \bowtie \text{Comments} \quad Q_5 : \text{Posts} \bowtie \text{PostHistory} \bowtie \text{Votes} \bowtie \text{Comments} \]

| Q1 | Q2 | Q3 | Q4 | Q5 |
|----|----|----|----|----|
| # poss. | 27578 | 145 | 38320 | 3925 | 1250 |
| # cons. | 27578 | 145 | 38320 | 3925 | 1245 |
Experiments on Stackoverflow

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

$Q_4$: Users $\bowtie$ Posts $\bowtie$ Comments $\quad Q_5$: Posts $\bowtie$ PostHistory $\bowtie$ Votes $\bowtie$ Comments

| Query | LinCQA | Conquer | FastFO | CAvSAT |
|-------|--------|---------|--------|--------|
| Q1    | N/A    | 100     | 101    | 102    | N/A    |
| Q2    | 100    | 101     | 102    | 103    | 1250   |
| Q3    | 100    | 103     | 103    | 103    | 1245   |
| Q4    | 100    | 103     | 103    | 103    | 1245   |
| Q5    | 100    | 103     | 103    | 103    | 1245   |

# poss. 27578 145 38320 3925 1250
# cons. 27578 145 38320 3925 1245
1. Consistent Query Answering for Primary Keys

2. Results

3. Techniques
From PPJT to FO-rewriting

Remove a primary key if some tuple with this primary key is “bad”

Course\((x, y)\)

\(\Downarrow\)

Faculty\((y, z, \text{“DB”})\)

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

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

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

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

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

Faculty\(_{fkey}\)(y) :- 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($x, y$)\[\rightarrow\]
Faculty($y, z, "DB")$\[\rightarrow\]

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

Course join() :- Course($x, y$), ¬Course$fkey(x)$
Course$fkey(x)$ :- Course($x, y$), ¬Faculty$join(y)$

Child$join(\vec{\alpha})$ :- Child($\vec{u}, \vec{v}$), ¬Child$fkey(\vec{u})$
Faculty$join(y)$ :- Faculty($y, z, w$), ¬Faculty$fkey(y)$
Faculty$fkey(y)$ :- Faculty($y, z, w$), $w \neq "DB"$

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”*

\[
\text{Course}(x, y) \quad \text{Faculty}(y, z, "DB")
\]

---

**Also expressible in SQL!**

```
Course_join() :- Course(x, y), ¬Course_fkey(x)
```

```
Course_fkey(x) :- Course(x, y), ¬Faculty_join(y)
```

```
\forall \text{Child} : Root_fkey(x) :- Root(x, y), ¬Child_join(\vec{\alpha})
```

```
Child_join(\vec{\alpha}) :- Child(\vec{u}, \vec{v}), ¬Child_fkey(\vec{u})
```

```
Faculty_join(y) :- Faculty(y, z, w), ¬Faculty_fkey(y)
```

```
Faculty_fkey(y) :- Faculty(y, z, w), w \neq "DB"
```

**Runs in \(O(N)\)**
Remove a primary key if some tuple with this primary key is “bad”
From PPJT to FO-rewriting

Remove a primary key if some tuple with this primary key is “bad”

Course\((x, y)\)

Faculty\((y, z, "DB")\)

\[\text{Course} \dashv \text{Course}_{fkey}(x) \quad \text{Course}_{fkey}(x) \dashv \text{Course}(x, y), \neg \text{Faculty}_{join}(y)\]

\[\forall \text{Child} : \text{Root}_{fkey}(\bar{x}) \dashv \text{Root}(\bar{x}, y), \neg \text{Child}_{join}(\bar{\alpha})\]

\[\text{Child}_{join}(\bar{\alpha}) \dashv \text{Child}(\bar{u}, \bar{v}), \neg \text{Child}_{fkey}(\bar{u})\]

\[\text{Faculty}_{join}(y) \dashv \text{Faculty}(y, z, w), \neg \text{Faculty}_{fkey}(y)\]

\[\text{Faculty}_{fkey}(y) \dashv \text{Faculty}(y, z, w), w \neq "DB"\]

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($x, y$)

Faculty($y, z, “DB”$)

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\((x, y)\)  
\[\text{Faculty}(y, z, \text{“DB”})\]  

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

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

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

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

\[\text{Faculty}_\text{fkey}(y) :\text{ Faculty}(y, z, w), w \neq \text{“DB”}\]

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\((x, y)\)

Faculty\((y, z, "DB")\)

\(\forall\text{Child} : \text{Root}\_fkey(\vec{x}) : \text{Root}(\vec{x}, \vec{y}), \neg\text{Child\_join}(\vec{\alpha})\)

\(\text{Course\_join}() \leftarrow \text{Course}(x, y), \neg\text{Course}\_fkey(x)\)

\(\text{Course}\_fkey(x) \leftarrow \text{Course}(x, y), \neg\text{Faculty\_join}(y)\)

\(\neg\text{Course}\_fkey(x) \leftarrow \text{Faculty}(y, z, w), w \neq "DB"\)

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(x, y)

Faculty(y, z, “DB”)  
```

```
Course\_join() :- Course(x, y), \neg Course\_fkey(x)

Course\_fkey(x) :- Course(x, y), \neg Faculty\_join(y)

\forall Child : Root\_fkey(\vec{x}) :- Root(\vec{x}, \vec{y}), \neg Child\_join(\vec{\alpha})

Child\_join(\vec{\alpha}) :- Child(\vec{u}, \vec{v}), \neg Child\_fkey(\vec{u})

Faculty\_join(y) :- Faculty(y, z, w), \neg Faculty\_fkey(y)

Faculty\_fkey(y) :- Faculty(y, z, w), w \neq “DB”
```

also expressible in SQL!
runs in $O(N)$
Remove a primary key if some tuple with this primary key is “bad”

Course\((x, y)\)

Faculty\((y, z, \text{“DB”})\)

\[
\text{Course}_{\text{join}}() \leftarrow \text{Course}(x, y), \neg\text{Course}_{\text{fkey}}(x)
\]

\[
\text{Course}_{\text{fkey}}(x) \leftarrow \text{Course}(x, y), \neg\text{Faculty}_{\text{join}}(y)
\]

\[
\forall \text{Child} : \text{Root}_{\text{fkey}}(\vec{x}) \leftarrow \text{Root}(\vec{x}, \vec{y}), \neg\text{Child}_{\text{join}}(\vec{\alpha})
\]

\[
\text{Child}_{\text{join}}(\vec{\alpha}) \leftarrow \text{Child}(\vec{u}, \vec{v}), \neg\text{Child}_{\text{fkey}}(\vec{u})
\]

\[
\text{Faculty}_{\text{join}}(y) \leftarrow \text{Faculty}(y, z, w), \neg\text{Faculty}_{\text{fkey}}(y)
\]

\[
\text{Faculty}_{\text{fkey}}(y) \leftarrow \text{Faculty}(y, z, w), w \neq \text{“DB”}
\]

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\((x, y)\)

Faculty\((y, z, “DB”)\)

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!

also expressible in SQL!
From PPJT to **FO**-rewriting

*Remove a primary key if some tuple with this primary key is “bad”*

\[
\text{Course}(x, y) \quad \text{Faculty}(y, z, “DB”) \quad \text{Course}fkey(x) \quad \text{Faculty}fkey(y) \quad \text{Child}fkey(\vec{\alpha}) \quad \text{Course}join() \quad \text{Faculty}join(y) \quad \text{Child}join(\vec{\alpha})
\]

\[
\begin{align*}
\forall \text{Child} & : \text{Root}fkey(\vec{x}) \leftarrow \text{Root}(\vec{x}, \vec{y}), \neg \text{Child}join(\vec{\alpha}) \\
\text{Course}join() & \leftarrow \text{Course}(x, y), \neg \text{Course}fkey(x) \\
\text{Course}fkey(x) & \leftarrow \text{Course}(x, y), \neg \text{Faculty}join(y) \\
\text{Child}join(\vec{\alpha}) & \leftarrow \text{Child}(\vec{u}, \vec{v}), \neg \text{Child}fkey(\vec{u}) \\
\text{Faculty}join(y) & \leftarrow \text{Faculty}(y, z, w), \neg \text{Faculty}fkey(y) \\
\text{Faculty}fkey(y) & \leftarrow \text{Faculty}(y, z, w), w \neq “DB”
\end{align*}
\]

also expressible in SQL!
runs in \(O(N)\)
From Boolean to non-Boolean

\[
\text{SELECT DISTINCT } A_1, A_2 \text{ FROM } T \text{ WHERE } A_3 = 42
\]

Step 1 Evaluate directly

| A1 | A2 |
|----|----|
| a  | b  |
| x  | y  |
| ...| ...|

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

\[
\text{SELECT DISTINCT 1 FROM } T \text{ WHERE } A_3 = 42 \text{ AND } A_1 = a \text{ AND } A_2 = b
\]

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

\[
\text{SELECT DISTINCT 1 FROM } T \text{ WHERE } A_3 = 42 \text{ AND } A_1 = x \text{ AND } A_2 = y
\]

...
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

...  

LinCQA $\rightarrow$ a single SQL/Datalog query
## Concluding remarks

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

Thank you!

Xiating Ouyang

xouyang@cs.wisc.edu

Z. Fan, P. Koutris, X. Ouyang, J. Wijsen
Concluding remarks

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

Original Query

|   | LinCQA | Conquer | FastFO | CAvSAT |
|---|--------|---------|--------|--------|
| Q1| 10     | N/A     | N/A    | N/A    |
| Q2| 100    | N/A     | N/A    | N/A    |
| Q3| 101    | N/A     | N/A    | N/A    |
| Q4| 102    | N/A     | N/A    | N/A    |
| Q5| 103    | N/A     | N/A    | N/A    |

Thank you!

Xiating Ouyang
xouyang@cs.wisc.edu
### Concluding remarks

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

---

**Thank you!**

Xiating Ouyang

xouyang@cs.wisc.edu