Consistent Query Answering (CQA)

While data cleaning is widely adopted to repair the inconsistent/dirty data, finding the 'right repair' remains a challenge. Alternatively, the idea of CQA is to compute the answers that are guaranteed to be returned in all repairs.

![Diagram](image)

\[ CQA(\text{db}) = \bigcap_{db'} CQA(db') \]

We study inconsistent databases that could violate the primary key constraint: every primary key could correspond to multiple distinct tuples in the database.

**Problem:** CQA(\(Q\)), where \(Q\) is a SPJ query

**Input:** a database \(db\) that violates the primary key constraint

**Output:** the answers guaranteed to be returned by \(Q\) on all repairs of \(db\)

```
Course
course_id faculty_id
CS 703 2
CS 703 3
MATH 770 9
CS 787 9

Faculty
faculty_id name area
2 Adam DB
2 Alice ML
3 John CS
6 Xiating UC
9 Carol DB
```

Executing the following blue SQL query on the database returns the inconsistent answers CS 703 and CS 787. The rewritten query \(Q'\) by adding the red segments would find the consistent answers (i.e., CS 703) that are returned in every repair of the database.

**Definition:** A join tree rooted at some atom is a PPJT if the root of every subtree is unattacked in the subtree.

For example, \(Q\) has a PPJT:

![Diagram](image)

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

Acyclic Queries in Linear Time

Evaluating an acyclic query is a well-studied problem by using hash joins on the join tree. For example, the Boolean blue SQL query is acyclic:

\[ q: \text{Faculty}(y, z, w) \rightarrow \text{Course}(x, y) \]

Yannakakis [VLDB’81] Our result

consistent answer

The answer to every Boolean acyclic query can be computed in \(O(|db|)\).

with a pair-pruning join tree (PPJT)

Queries with projection can be reduced to Boolean queries

Pair-pruning Join Tree (PPJT)

We consider acyclic self-join-free SPJ queries (each table name occurs once).

**Definition:** A join tree rooted at some atom is a PPJT if the root of every subtree is unattacked in the subtree.

For example, \(q\) has a PPJT:

![Diagram](image)

\( Q(\text{db}) = \bigcap_{db'} CQA(db') \)

LinCQA and the Rewriting

LinCQA is a query rewriter that takes as input a SQL query, outputs its first-order rewriting.

```
python3 lincqa.py -i <input.sql> -o <output.sql>
```

Using PPJT, the consistent answers can be computed in a bottom-up fashion.

```
SELECT DISTINCT U, W FROM T WHERE A3 = 42 AND A1 = a AND A2 = b
```

For queries with projection:

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

we use PPJT to check for each potential answer (a, b)... in one program:

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

if yes, then (a, b) is a consistent answer.

Experiments

We used a 400GB StackOverflow dataset [among others] on SQL Server.

<table>
<thead>
<tr>
<th>Table</th>
<th># of rows</th>
<th>inconsistencyRatio</th>
<th>blockSize</th>
<th># of Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>14M</td>
<td>0%</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>Posts</td>
<td>53M</td>
<td>0%</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>PostHistory</td>
<td>14M</td>
<td>0.001%</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Badges</td>
<td>40M</td>
<td>0.58%</td>
<td>941</td>
<td>4</td>
</tr>
<tr>
<td>Votes</td>
<td>213M</td>
<td>30.9%</td>
<td>1441</td>
<td>6</td>
</tr>
</tbody>
</table>

![Graph](image)

**Proposition:** If a Boolean query \(q\) has a pair-pruning join tree (PPJT), then \(CQA(q)\) has a first-order rewriting that runs in linear time.