Data Provenance for Recursive SQL Queries

Benjamin Dietrich, Tobias Müller, Torsten Grust

College of Computing
Illinois Institute of Technology

CS 520 & Data Integration, Warehousing, and Provenance
Dr. Boris Glavic

Abstract:

The paper shows how recursion is done in sql , mainly using recursive common table expressions(CTEs) and user defined functions(UDFs).These techniques make SQl powerful but also make them more complicated. The main goal is to understand where and why a specific data comes in query results which is known as data provenance.The author suggests a method which involves rewriting the recursive queries into a two phase evaluation strategy which does not mess with the database system .

TABLE OF CONTENTS:

1)Introduction:

Recursion in SQL (CTEs and UDFs) is tough to understand and fix queries if there is an error.The author highlights the difficulty of understanding how recursive calculation is done and how they are approaching to the final answer.They suggest by using data provenance we can figure out where and why certain things happen which makes us easy to understand recursive queries especially at the level of individual table cells.By doing this we can see which steps affect the recursive math and the final answer.

2)Problem:

When we are dealing with recursive operations in SQL like Recursive CTEs or recursive UDFs.It is tough to trace intermediate steps and ensure base cases are reached which also makes debugging complex if there is an error in the code.

3)Proposed Solution:

Proposed solution uses data provenance to understand recursive queries which helps to figure out from “where” the data comes and “why” some values are included in the result. Where-provenance finds which cells from the table were used to calculate the next step in the recursive query.In simple terms,It tells from

where does the data come from.Why-Provenance locates the cells in the tables which were examined to decide whether a specific value should be a part of final result or not.In simple terms, it explains why certain data is included in the output

Two-Phase Evaluation Process:

Phase 1:

Variant Query ($q1$) :During phase 1 , it processes the data also keeping the track of decisions which were made during the execution of the query. It creates a protocol mapping that connects to the rows and columns which influenced it

Phase 2:

Variant Query ($q2$): This works with sets of dependencies between table cells. It reads the protocol from Phase 1 and evaluates based on the recorded dependencies

4)Example for recursive CTE :

4.1)Task:

The task is to calculate the number of parts required to build an humanoid robot where some parts are made up of other parts.

4.2)Initial Query:

```
1  WITH RECURSIVE bom(part, sub_part, qty) AS (
2      SELECT p.part, p.sub_part, p.qty
3      FROM   parts AS p
4      WHERE  p.part = 'humanoid'
5    UNION ALL
6      SELECT p.part, p.sub_part, p.qty -- BUG (should read p.qty * b.qty)
7      FROM   bom AS b,
8             parts AS p
9      WHERE  p.part = b.sub_part
10 )
11 SELECT b.sub_part, b.qty
12 FROM   bom AS b
```

The initial query is used to calculate the bill of materials(BOM) for assembling a humanoid robot but there is a mistake in the query that repeats(recursive part).It fails to correctly calculate the quantity of each part needed to build an robot.The error occurred because it does not multiply the quantity needed of a sub part with the quantity of its parent part.

4.3)Initial output:

parts

part	sub_part	qty
humanoid	head	1
humanoid	body	1
body	arm	2
body	leg	2
arm	finger	5
leg	foot	1
chassis	wheel	4

output (buggy)

sub_part	qty
head	1
body	1
arm	2
leg	2
finger	5
foot	1

4.4)Provenance Derivation Approach:

4.4.1)Phase 1: Value-Based Evaluation with Protocol Writing

In this phase, we rewrite the query to not only do its job but also keep track(writing a protocol) of how each part of the query is calculated . By keeping track of these details it is easy to understand where each piece of data came from and why it is there..

```
1  WITH RECURSIVE bom^𝕀(ϱ, part, sub_part, qty) AS (
2      SELECT write_FILTER(①, p.ϱ) AS ϱ,
3             p.part, p.sub_part, p.qty
4      FROM   parts^𝕀 AS p
5      WHERE  p.part = 'humanoid'
6
7    UNION ALL
8      SELECT write_JOIN(②, b.ϱ, p.ϱ) AS ϱ,
9             p.part, p.sub_part, p.qty * b.qty
10     FROM   bom^𝕀 AS b, parts^𝕀 AS p
11     WHERE  p.part = b.sub_part
12
13 )
14 SELECT  b.ϱ, b.sub_part, b.qty
15 FROM    bom^𝕀 AS b
```

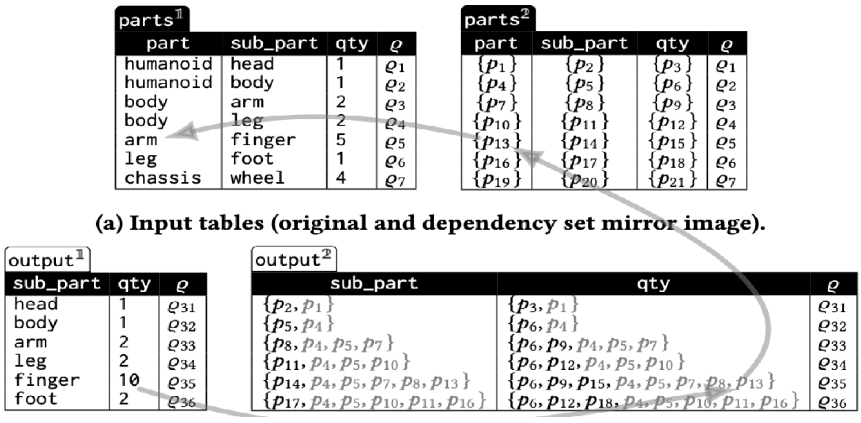
writeFILTER and writeJOIN functions are used to record the decision process.They keep track of where the data is coming from and why certain data are included in the result

4.4.2)Phase 2: Dependency Set Evaluation

In this phase, we write a new query that uses the information that we gathered in phase 1 to figure out the dependencies between different parts.This helps in understanding how change in one element of the data affect the other elements.It's like tracing the history of each piece of data to see how it arrived at its current state.

```
1  WITH RECURSIVE bom^2(ϱ, part, sub_part, qty) AS (
2      SELECT log.ϱ,
3             p.part ∪ wh.y, p.sub_part ∪ wh.y, p.qty ∪ wh.y
4      FROM   parts^2 AS p,
5             read_FILTER(①, p.ϱ) AS log(ϱ),
6             Y(p.part) AS wh(y)
7    UNION ALL
8      SELECT log.ϱ, p.part  ∪ wh.y,
9             p.sub_part ∪ wh.y, p.qty ∪ b.qty ∪ wh.y
10     FROM   bom^2 AS b, parts^2 AS p,
11            read_JOIN(②, b.ϱ, p.ϱ) log(ϱ),
12            Y(p.part ∪ b.sub_part) AS wh(y)
13 )
14 SELECT  b.ϱ, b.sub_part, b.qty
15 FROM    bom^2 AS p
```

The query operates over sets of dependencies (like a chain of influences) established in Phase 1.



parts[1]

part	sub_part	qty	ϱ
humanoid	head	1	ϱ_1
humanoid	body	1	ϱ_2
body	arm	2	ϱ_3
body	leg	2	ϱ_4
arm	finger	5	ϱ_5
leg	foot	1	ϱ_6
chassis	wheel	4	ϱ_7

parts[2]

part	sub_part	qty	ϱ
$\{p_1\}$	$\{p_2\}$	$\{p_3\}$	ϱ_1
$\{p_4\}$	$\{p_5\}$	$\{p_6\}$	ϱ_2
$\{p_7\}$	$\{p_8\}$	$\{p_9\}$	ϱ_3
$\{p_{10}\}$	$\{p_{11}\}$	$\{p_{12}\}$	ϱ_4
$\{p_{13}\}$	$\{p_{14}\}$	$\{p_{15}\}$	ϱ_5
$\{p_{16}\}$	$\{p_{17}\}$	$\{p_{18}\}$	ϱ_6
$\{p_{19}\}$	$\{p_{20}\}$	$\{p_{21}\}$	ϱ_7

(a) Input tables (original and dependency set mirror image).

output[1]

sub_part	qty	ϱ
head	1	ϱ_{31}
body	1	ϱ_{32}
arm	2	ϱ_{33}
leg	2	ϱ_{34}
finger	10	ϱ_{35}
foot	2	ϱ_{36}

output[2]

sub_part	qty	ϱ
$\{p_2, p_1\}$	$\{p_3, p_1\}$	ϱ_{31}
$\{p_5, p_4\}$	$\{p_6, p_4\}$	ϱ_{32}
$\{p_8, p_4, p_5, p_7\}$	$\{p_6, p_9, p_4, p_5, p_7\}$	ϱ_{33}
$\{p_{11}, p_4, p_5, p_{10}\}$	$\{p_6, p_{12}, p_4, p_5, p_{10}\}$	ϱ_{34}
$\{p_{14}, p_4, p_5, p_7, p_8, p_{13}\}$	$\{p_6, p_9, p_{15}, p_4, p_5, p_7, p_8, p_{13}\}$	ϱ_{35}
$\{p_{17}, p_4, p_5, p_{10}, p_{11}, p_{16}\}$	$\{p_6, p_{12}, p_{18}, p_4, p_5, p_{10}, p_{11}, p_{16}\}$	ϱ_{36}

4.5)Output:

output

part	sub_part	qty	ϱ
humanoid	head	1	ϱ_{31}
humanoid	body	1	ϱ_{32}
body	arm	2	ϱ_{33}
body	leg	2	ϱ_{34}
arm	finger	10	ϱ_{35}
leg	foot	2	ϱ_{36}

5)Example for recursive UDF :

In this example ,recursive UDF was used to calculate the similarity between two time series which is done by DTW algorithm.DTW algorithm is used to measure the similarity between two time series.It allows for stretching and compressing the time series to minimize the distance between them.The function's recursive nature is seen in the way it calls itself with different parameters to explore various alignments of the time series

5.1)Initial query:

```
1  CREATE FUNCTION dtw(i int, j int, w int) RETURNS float AS $$
2  SELECT CASE
3          WHEN dtw.i=0 AND dtw.j=0 THEN 0.0
4          WHEN dtw.i=0 OR  dtw.j=0 THEN ∞
5          WHEN abs(dtw.i-dtw.j)>dtw.w THEN ∞
6          ELSE (SELECT d.δ
7                       + LEAST(
8                           dtw(dtw.i-1, dtw.j-1, dtw.w),
9                           dtw(dtw.i-1, dtw.j  , dtw.w),
10                          dtw(dtw.i  , dtw.j-1, dtw.w))
11                FROM   distances AS d
12                WHERE  (d.i,d.j)=(dtw.i,dtw.j))
13       END
14 $$ LANGUAGE SQL STABLE;
```

In this query if both i and j are 0 then it return 0.0 which represents no distance.If either of i or j is 0 but not both it returns infinity ,which signifies a large distance.The function recursively calls itself three times for different pairs of indices (i-1, j-1, i-1, j, i, j-1). This is to explore different alignments of the time series. In the query the LEAST function is used to select the select the minimum value among these recursive calls, which is then added to the current distance d.δ

The warp parameter w defines the maximum allowed index difference. If abs(i-j) exceeds w then it returns infinity.To understand the DTW UDF we rewrite it into two phases for provenance derivation.

5.2)Provenance Derivation

5.2.1)Phase 1:

This version (dtw1) of the UDF is instrumented for provenance derivation. It includes additional operations to log the execution flow, arguments, and intermediate results.

```
1  CREATE FUNCTION dtw𝟙(ϱ rowID, i int, j int, w int) RETURNS float AS $$
2  SELECT
3    CASE writeCASE(⑥, dtw𝟙.ϱ, CASE
4         WHEN dtw𝟙.i=0 AND dtw𝟙.j=0 THEN 1
5         WHEN dtw𝟙.i=0 OR  dtw𝟙.j=0 THEN 2
6         WHEN abs(dtw𝟙.i-dtw𝟙.j)>dtw𝟙.w THEN 3
7         ELSE 0 END)
8    WHEN 1 THEN 0.0
9    WHEN 2 THEN '∞'
10   WHEN 3 THEN '∞'
11   WHEN 0 THEN
12    (SELECT t.score FROM (
13     SELECT writeJOIN(⑤,dtw𝟙.ϱ, d.ϱ) AS ϱ,
14           d.δ + LEAST𝟙(
15             writeCALL(④, dtw𝟙.ϱ),
16             dtw𝟙(writeCALL(①, dtw𝟙.ϱ), dtw𝟙.i-1, dtw𝟙.j-1, dtw𝟙.w),
17             dtw𝟙(writeCALL(②, dtw𝟙.ϱ), dtw𝟙.i-1, dtw𝟙.j  , dtw𝟙.w),
18             dtw𝟙(writeCALL(③, dtw𝟙.ϱ), dtw𝟙.i  , dtw𝟙.j-1, dtw𝟙.w)
19           ) AS score
20     FROM  distances𝟙 AS d
21     WHERE (d.i,d.j)=(dtw𝟙.i,dtw𝟙.j)
22    ) AS t(ϱ, score))
23   END
24 $$ LANGUAGE SQL VOLATILE;
```

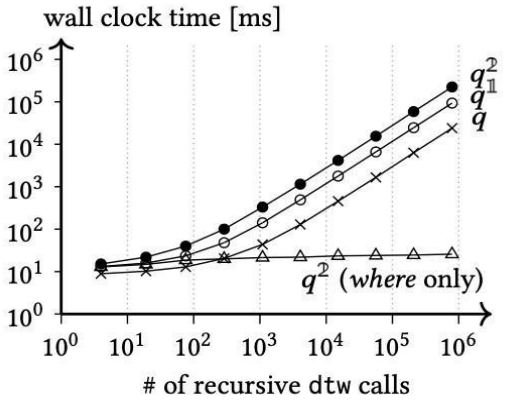
it logs execution details, including recursive calls and decisions made. The writeCASE, writeJOIN, and writeCALL functions are used for this purpose.

5.2.2)Phase 2:

The second version (dtw2) interprets the protocol data generated by dtw1. It operates using dependency sets that represent the provenance of the data.

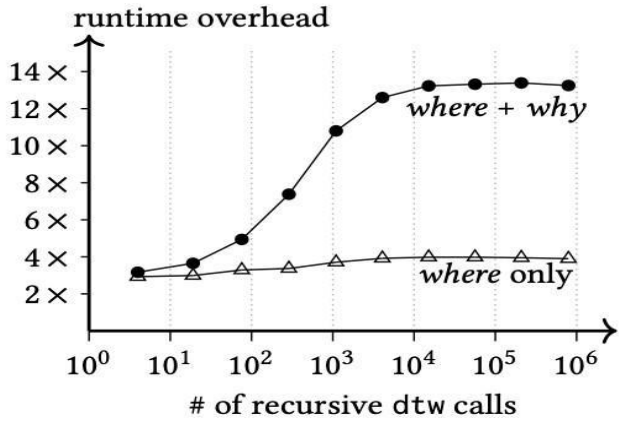
```
1  CREATE FUNCTION dtw²(ϱ rowID, i ℙ, j ℙ, w ℙ) RETURNS ℙ AS $$
2  SELECT
3    CASE read_CASE(⑥, dtw².ϱ)
4    WHEN 1 THEN ∅ ∪ Y(dtw².i ∪ dtw².j)
5    WHEN 2 THEN ∅ ∪ Y(dtw².i ∪ dtw².j)
6    WHEN 3 THEN ∅ ∪ Y(dtw².i ∪ dtw².j ∪ dtw².w)
7    WHEN 0 THEN
8      (SELECT t.score FROM (
9         SELECT log.ϱ AS ϱ,
10              d.δ ∪ LEAST²(
11                read_CALL(④, dtw².ϱ),
12                dtw²(read_CALL(①, dtw².ϱ),
13                     dtw².i ∪ ∅, dtw².j ∪ ∅, dtw².w),
14                dtw²(read_CALL(②, dtw².ϱ),
15                     dtw².i ∪ ∅, dtw².j, dtw².w),
16                dtw²(read_CALL(③, dtw².ϱ),
17                     dtw².i, dtw².j ∪ ∅, dtw².w)
18              ) ∪ Y(d.i ∪ d.j ∪ dtw².i ∪ dtw².j) AS score
19        FROM  distances² AS d,
20              read_JOIN(⑤, dtw².ϱ, d.ϱ) AS log(ϱ)
21     ) AS t(ϱ, score)) ∪ Y(dtw².i ∪ dtw².j ∪ dtw².w)
22   ELSE ∅
23   END
24 $$ LANGUAGE SQL STABLE;
```

5.3)Results:



Comparison of Query Execution Times :The original query(q) is running the fastest because it involves straightforward execution without any additional overhead.Query(q1) includes the time for accessing the protocol which adds the overhead to the execution time so it is slow compared to original query.Query(q2)

manages large dependency sets and it is slowest compared to q and q1 becuase of added complexity and overhead.The execution times of q, q1, and q2 are somewhat parallel, which means PostgreSQL is efficient for these queries, even with the extra operations involved in q1 and q2.



The graph above shows how runtime overhead increases with the number of recursive calls.It compares the overhead for capturing "where" and "why" provenance with "where" only provenance . The overhead for "where" and "why" is higher comapred to "where" only , but both the lines appear to be plateau after sometime, which shows overhead does not increase indefinitely with the complexity.

The protocol acts intermediate between different phases of query execution.The size of the protocol data depends on number recursive DTW calls, it may range

from 120 KB to 160MB .Despite handling the large amount of data ,the functions used in the queries manage to keep the performance within limits , which shows the efficiency of the implementation

6)Conclusion:

The paper presents a new and vital approach to help people deal with complex SQL queries. It provides a method that hasn't been used much before, which is quite useful for understanding tough queries. However, there are certain limitations to this method ,they may be difficult to learn and may cause queries to run slower, especially if they are large and complex. However, the paper is quite valuable because it provides an excellent system for sorting out the complex queries.