CaJaDE: Explaining Query Results by Augmenting Provenance with Context

Chenjie Li¹, Juseung Lee¹, Zhengjie Miao², Boris Glavic¹, Sudeepa Roy²

¹IIT, Chicago, IL, USA ²Duke University, Durham, NC, USA
{cli112,jlee302}@hawk.iit.edu,{zjmiao,sudeepa}@cs.duke.edu,bglavic@iit.edu

ABSTRACT
In this work, we demonstrate CaJaDE (Context-Aware Join-Augmented Deep Explanations), a system that explains query results by augmenting provenance with contextual information from other related tables in the database. Given two query results whose difference the user wants to understand, we enumerate possible ways of joining the provenance (i.e., contributing input tuples) of these two query results with tuples from other relevant tables in the database that were not used in the query. We use patterns to concisely explain the difference between the augmented provenance of the two query results. CaJaDE, through a comprehensive UI, enables the user to formulate questions and explore explanations interactively.

PVLDB Reference Format:

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/IITDBGroup/CaJaDe.

1 INTRODUCTION
In today’s data-driven world, data is analyzed using complex queries to search for trends and anomalies, and subsequently to make decisions based on data. Interpreting results of such queries is a challenging task which requires the analyst to explore possible root causes for a result. Provenance [3], information about what input data was used to derive a result, provides a natural foundation for several “explanation” frameworks that have been proposed by the database community [5–8]. However, real world data exhibits intertable relationships that connect the provenance of a query with data that has not been accessed by the query. Current approaches do not take these crucial relationships into account. Thus, the explanations they produce may lack important contextual information from parts of the database that do not belong to the query’s provenance.

In this demonstration, we showcase CAJADE (Context Aware Join Augmented Deep Explanations), a novel explanation system that augments the provenance of a query using relations not accessed by this query. CaJaDE is open source and is available on github¹. Using database constraints and user-provided information² about how tables in the database are connected, CAJADE searches for ways to augment the provenance of a query (i.e., contributing input tuples) by joining it with tuples from other unused tables while obeying these constraints. The augmented provenance produced by a particular join graph, which captures one possible way to augment the provenance, is then summarized using selection patterns. Specifically, a pattern consists of conjunctions of equality and inequality predicates and represents the tuples in the augmented provenance that satisfy the pattern. In CaJaDE, we mainly focus on explaining differences between two query result tuples t₁ and t₂ selected by the user. Thus, a pattern with good quality should summarize the difference between the join-augmented provenance of t₁ and t₂. Summarizing provenance with patterns is already expensive [8]. With the expansion of the search space caused by considering context, designing the tractable search process becomes even more challenging. We leverage a collection of optimizations and heuristics that will be described briefly later. For more a detailed description of the ideas presented in this demonstration, please refer to our research paper published in SIGMOD’21 [4].

EXAMPLE 1. Consider a simplified NBA (National Basketball Association) database [1] with the following relations (the keys are underlined). We will use a full version of this dataset in the demonstration with several additional tables [4]. Some example tuples from these two relations are shown in Figure 1.

- Game(game_date, home, away, home_pts, away_pts, winner, season): information for each game such as the competing teams (home and away), scores for each team, and game date.
- PlayerGameStats(game_date, home, pts, rebs, mins): the points, rebounds, and minutes played for each player in each game.

Query Q₁ shown below returns the number of wins of team GSW (Golden State Warriors) per season.

SELECT winner as team, season, count(*) as win FROM Game WHERE winner = 'GSW' GROUP BY winner, season

As shown in Figure 1c, GSW won 73 games in the 2015-16 season, which is the greatest number of games won in a single season by any team in history (t₂). Compared with just 3 seasons ago in 2012-13 with 47 wins (t₁), GSW has drastically improved its winning record. Notice that in Q₁, only Game table (1a) was accessed. This table provides the user with information about each game. However, such information is not enough for understanding why GSW won more games in the 2015-16 season than in the other seasons, because in each season a

¹https://github.com/IITDBGroup/CaJaDe/
²Such information can also be obtained automatically using join discovery techniques, e.g., [9].
D. Green played more games in season 2015-16 because player D. Green played more minutes in 2015-16 than in 2012-13. The F-score for this explanation pattern is \( \frac{72}{81} \times \frac{94}{94} = 0.94 \).
Figure 3: CaJaDE’s Main GUI: After connecting to the database, the corresponding schema graph is shown in ①. The user runs an aggregation query in ② over the schema shown in ①, formulates a question by selecting two result tuples in ③, can browse explanations (patterns) through their descriptions shown in ④ and endorse/dislike explanations to refine them further, explore explanations produced by different join graphs shown in ⑦, visualize an explanation in terms of its join graph in ⑥, and explore distribution of attributes covered by a pattern as histograms in ⑨.

Figure 4: CaJaDE’s system architecture

2.3 Implementation and Optimizations

The architecture of CaJaDE is shown in Figure 4. CaJaDE takes provenance produced by GProM (https://github.com/IITDBGroup/gprom) and the schema graph as inputs. The join graph generator enumerates the join options based on the schema graph and the relations from the query’s provenance. The provenance augmentation component takes each join graph and PT to materialize the augmented provenance table (APT). The pattern miner mines patterns from the APTs. Finally, CaJaDE ranks the patterns based on a weighted score combining the F-score and diversity.

Note that even for a single join graph, the search space for patterns is large: polynomial in the number of distinct values per attribute, but exponential in the number of attributes. Furthermore, even for moderately-sized schema graphs, the number of join graphs (subgraphs of the schema graph) for a given query can be huge. We apply a suite of novel optimizations and heuristics to enable CaJaDE to scale to large datasets. These include: (i) clustering similar or correlated attributes to reduce the search space of patterns and avoid redundant explanations (e.g., if a pattern with birth date is produced then a pattern with age can be ignored); (ii) we train a classifier to determine which attributes are most predictive of the difference between the two query results appearing in the user question to prune additional attributes from pattern generation; (iii) Given an APT, we use a variant of the LCA (Least Common Ancestor) method from [2] that handles categorical attributes. Intuitively, this step will help identify the most frequently appearing combinations of attribute values as pattern candidates. We then refine the subset of these pattern candidates that have sufficiently high recall by adding numerical attributes. Since the search space for numerical predicates is significantly larger than the search space for categorical predicates, it is beneficial to avoid refining unpromising patterns (with low recall); (iv) we enumerate join graph candidates by size by iteratively extending previously explored join graphs one edge at a time. This enables us to not further extend a join graph if extensions are unlikely to yield good patterns. For a detailed description of our techniques and optimizations see [4].

3 DEMONSTRATION

In this demonstration, we will use real world datasets including this NBA dataset [1], which contains statistics about teams, players, games etc. from the 2009-10 season to the 2018-19 season. The main user interface using the NBA dataset is shown in Figure 3. The
system with multiple datasets to select from will be accessible to the users as a web application during the demonstration. A typical user session is described below.

1. **Run aggregate query and formulate question.** After connecting to the database, the user can familiarize themselves with the database schema (1) and the schema graph encoding allowable join paths (5). Once the user has gained an initial understanding of the schema, the user can run a group-by/aggregate query (2). After the query is executed, the query results are shown in (3). Users can inspect these results, and if they want to understand the difference between two output tuples of interest, they can choose the tuples by clicking (will be highlighted in two different colors).

2. **Join graphs and explanations.** CAJADE produces results incrementally so that the user can start exploring explanations right away without waiting. While CAJADE is running in the background, the list of join graphs (2) along with their top-k explanation patterns (4) are continuously updated. Histograms of the F-scores of patterns are plotted above each join graph to give user an overview of the pattern quality for each join graph. The overall distribution of F-scores of the patterns is shown in (6). Users can hover over an explanation to see the pattern description. Furthermore, selecting a join graph in (2) restricts the patterns shown in (4) to patterns for this join graph.

3. **Interpreting a pattern and stating preferences.** We provide additional ways to help the user interpret an explanation pattern. First, when the user selects a pattern, the join graph along with the pattern predicate is shown in (6). As shown in Figure 5a, the selected example pattern has a join graph with 3 nodes (tables): the provenance table (PT), player.game.stats, and player. The pattern predicates applying to attributes from each table R are shown as nodes in the graph connected to R (Figure 5a). Furthermore, this visualization shows the fraction of tuples satisfying the patterns for the two selected output tuples in their corresponding colors (in green and purple, saying that 9 out of 47 and 72 out of 73 tuples from the provenance satisfy the pattern predicate for the green (t₁) and purple (t₂) result tuples selected in (5)). The user can state their preference for or against patterns as shown in (4) in Figure 3 and Figure 5b by clicking thumbs up/down. Internally, CAJADE will prioritize patterns that are similar to patterns upvoted by the user and dissimilar to patterns downvoted by the user. Finally, based on the feedback we got from our previously user study [4], in order to help the user understand how the constant from a predicate in a pattern compares with other values from the attribute’s domain (e.g., how frequently do we encounter 15.48 mins playtime or 3.25 rebounds), we let the user select one of the predicates from the current pattern as shown in (7) in Figure 3 and Figure 5c. We then plot two histograms showing the distribution of the values for the attribute used in the predicate, one for each of the output tuples t₁, t₂ selected in the user question. In this example, we are looking at player Draymond Green’s minutes played per game for the two seasons involved in the user question: the selected pattern predicate is minutes>15.48. As shown in the histogram, Green played all the games over 25 minutes in the 2015-16 season, whereas in the 2012-13 season, he played for a significantly lower number of minutes in the majority of the games (and in fact played for < 15.48 minutes in many games as shown in blue). This difference may contribute towards the significantly better performance of GSW in 2015-16 compared to 2012-13 (Green is known to be an important player for GSW). CAJADE is the first system that automatically finds such interesting explanations and helps users to better understand the difference of two query results incorporating relevant contextual information from tables that are unused in the query.

ACKNOWLEDGMENTS

This work was supported by NSF Awards IIS-1552538, IIS-1703431, IIS-1956123, IIS-2008107 and IIS-2107107

REFERENCES


