CaJaDE: Explaining Query Results by Augmenting Provenance with Context

Chenjie Li[1], Juseung Lee[1], Zhengjie Miao[2], Boris Glavic[1], Sudeepa Roy[2]
[1]IIT, Chicago, IL, USA [2]Duke 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:
Chenjie Li, Juseung Lee, Zhengjie Miao, Boris Glavic, and Sudeepa Roy. CaJaDE: Explaining Query Results by Augmenting Provenance with Context. PVLDB, 15(12): 3594-3597, 2022.
doi:10.14778/3554821.3554852

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 inter-table 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 github[1]. Using database constraints and user-provided information[2] 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 t_1 and t_2 selected by the user. Thus, a pattern with good quality should summarize the difference between the join-augmented provenance of t_1 and t_2. 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 Q_1 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 (t_2). Compared with just 3 seasons ago in 2012-13 with 47 wins (t_1), GSW has drastically improved its winning record. Notice that in Q_1, 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*

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554852

[1]https://github.com/IITDBGroup/CaJaDe/
[2]Such information can also be obtained automatically using join discovery techniques, e.g., [9].

	game_date	home	away	home_pts	away_pts	winner	season
g_1	2016-04-13	GSW	MEM	125	104	GSW	2015-16
g_2	2012-12-01	GSW	IND	103	92	GSW	2012-13
g_3	2016-03-18	DAL	GSW	112	130	GSW	2015-16
g_4	2017-02-25	GSW	BKN	112	95	GSW	2016-17

(a) Game Table

	player	game_date	home	pts	rebs	mins
p_1	S.Curry	2016-04-13	GSW	46	4	29.78
p_2	D.Green	2016-04-13	GSW	11	9	29.78
p_3	D.Green	2012-12-01	GSW	2	1	11.10
p_4	D.Green	2016-03-18	DAL	15	4	36.13

(b) PlayerGameStats table

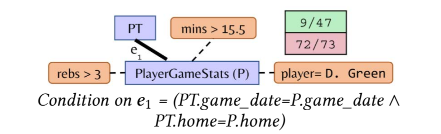
	team	season	win
t_1	GSW	2012-13	47
	GSW	2013-14	51
	GSW	2014-15	67
t_2	GSW	2015-16	73
	GSW	2016-17	67

(c) Result of Q_1

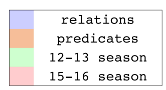
Figure 1: Simplified example NBA dataset.

UQ_1: Why did *GSW* win 73 games in 2015-16 (t_2) compared to 47 games in 2012-13 (t_1).

(a) User question UQ_1



(b) Example join graph with pattern for UQ_1



(c) Legend

Figure 2: Example user question and explanation.

team plays the same number of games, and roughly the same number of times against each opponent. That is, data provenance, which for this query only contains tuples from the Game *table, is insufficient for explaining the difference between the outcome for the two seasons.*

The scenario from Example 1 demonstrates the need to consider contextual information that is not contained in the provenance to generate meaningful explanations. This is what CaJaDE is built for. The explanation shown below gives a flavor of the top explanations produced by CaJaDE to differentiate t_1, t_2 in Figure 1c.

> *GSW* won more games in season 2015-16 because player D.Green played >15.5 minutes and had >3 rebounds in **72 out of 73 games** in 2015-16 compared to **9 out of 47 games** in 2012-13.

Given this explanation, the user can infer that D.Green was one of the key contributors for the improvement of GSW's winning record since his playing minutes and rebounds significantly improved in the 2015-16 season compared with the 2012-13 season. In CaJaDE, this explanation is represented using a join-graph as shown in Figure 2b, which augments the provenance table (PT in Figure 2b, contains only the Game table) with the PlayerGameStats table not accessed by the query. The join graph shows the pattern's predicates that apply to different tables (player=D. Green, mins > 15.5, rebs > 3) alongside the difference in statistics (72 out of 73 tuples in 2015-16 vs. 9 out of 47 tuples in 2012-13 in the augmented provenance of t_2 and t_1 satisfy the pattern, respectively).

2 CAJADE OVERVIEW

2.1 Augmented Provenance using Join Graphs

User questions. Given two tuples t_1 and t_2 in the result of a query Q evaluated over a database D, we find explanations that concisely summarize the difference between the provenance of t_1 and t_2 augmented with additional contextual information.

Provenance Tables. We define the *provenance table* (PT) for a SELECT-FROM-WHERE-GROUP BY query as a subset of the join result of the relation(s) accessed by the query, i.e., the joined relations that contribute to the query result. We use $\mathcal{PT}(Q, D)$ to denote the provenance table for query Q and database D. In Example 1, query Q_1 accesses a single table: Game, therefore, $\mathcal{PT}(Q_1, D)$ contains all the tuples from Figure 1a where *GSW* is the winner. For a result tuple $t \in Q(D)$, we use $\mathcal{PT}(Q, D, t) \subseteq \mathcal{PT}(Q_1, D)$ to denote the provenance of t. In Example 1, $\mathcal{PT}(Q_1, D, t_1)$ contains all tuples from table Game where GSW won in the 2012-13 season.

Schema Graphs. To *augment* the provenance of a query with contextual information, we need to explore plausible options of joining the tables from the query's provenance with other tables in the database providing context. CaJaDE expects as input a *schema graph*, a graph whose vertices represent the relations in the database and whose edges encode what join conditions can be used. CaJaDE can automatically generate a schema graph from the foreign key constraints of a database. Additionally, the user can specify the graph manually, e.g., using existing data discovery tools [9] to determine what tables can be joined. The problem of discovering join-ability of tables is orthogonal to the problem we address in CaJaDE: how to efficiently compute explanations with context.

Join Graphs. While the schema graph encodes *all possible ways* the provenance table can be augmented by joining with other tables in the database, a join graph encodes a single augmentation using a subset of tables as permitted by the schema graph. A join graph contains a distinguished node *PT* representing the relations from Q, i.e., the Game table from Figure 2. The other nodes of a join graph are labeled with relations, e.g., PlayerGameStats from Figure 2. Edges in a join graph are labeled with join conditions allowed by the schema graph. Each join graph encodes one of the possible ways of how to augment $\mathcal{PT}(Q, D)$.

Augmented Provenance. Given a provenance table $\mathcal{PT}(Q, D)$ and a join graph, we derive an *augmented provenance table* (APT) by joining $\mathcal{PT}(Q, D)$ with other relations in the join graph using the join conditions encoded by the edges of the join graph. The APT produced by augmenting $\mathcal{PT}(Q_1, D)$ (i.e., the Game table in Figure 1a) using the join graph in Figure 2b includes each game paired with the players participating in this game, i.e., the pairs (g_1, p_1), (g_1, p_2), (g_2, p_3), (g_3, p_4).

2.2 Patterns as Explanations

In CaJaDE, an explanation includes (a) a join graph and (b) a pattern (conjunction of predicates) that summarizes sets of tuples from the APT produced by the join graph. Intuitively, a pattern with good quality matches as many tuples as possible from the augmented provenance $\mathcal{PT}(Q, D, t_1)$ of one result tuple t_1 appearing in the user question, and as few tuples as possible from the augmented provenance $\mathcal{PT}(Q, D, t_2)$ of the other result tuple t_2. We adapt the notion of *F-score* as the scoring metric. A tuple t from the APT *matches* a pattern if t satisfies all predicates of the pattern. For example, in Example 1, the pattern (player=D.Green, mins > 15.5, rebs > 3), matches tuples (g_1, p_2) and (g_3, p_4) in the APT. Over the full NBA dataset, this pattern matches 72 out of 73 tuples from the 2015-16 season and 9 out of 47 tuples in the 2012-13 season. Thus, for t_2 in comparison with t_1, the *"recall"* is $\frac{72}{73}$ and the *"precision"* is $\frac{72}{72+9} = \frac{72}{81}$. The F-score for this explanation pattern distinguishing t_2 from t_1 is $2 \times \frac{(72/73)\times(72/81)}{(72/73)+(72/81)} \approx 0.94$.

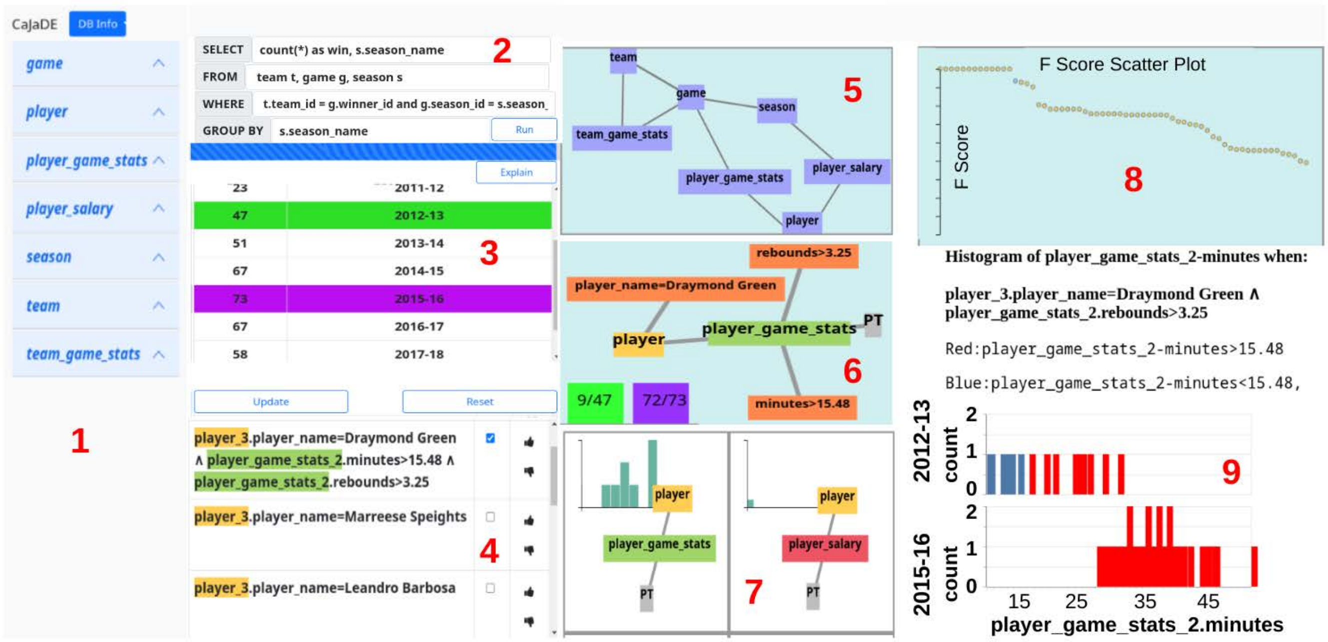


Figure 3: CaJaDE's Main GUI: After connecting to the database, the corresponding schema graph is shown in ⑤. The user runs an aggregation query in ② over the schema shown in ①, formulates a question by selecting two result tuples in ③, can browse explanations (patterns) through their descriptions shown 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.

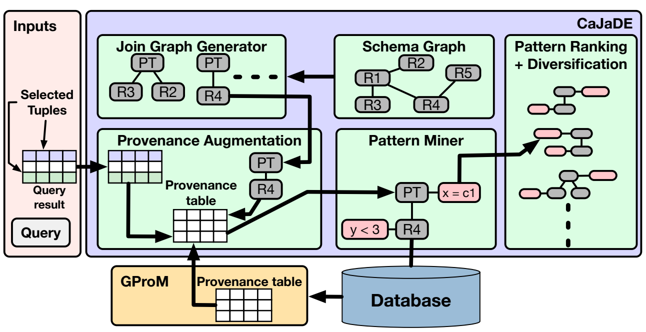


Figure 4: CaJaDE's system architecture

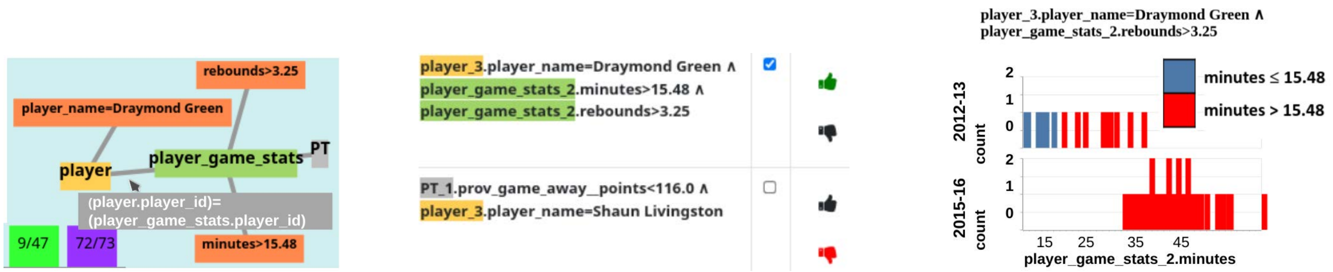
2.3 Implementation and Optimizations

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



(a) (b) (c)

Figure 5: (a) An explanation (pattern and its join graph). Predicates of the pattern (orange background) are connected to the table they apply to. (b) The user can up- and downvote patterns. (c) Distribution of the values of one attribute from a pattern predicate (values matching the predicate are shown in red and values not matching the predicates are shown in blue).

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 (⑦) 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 ⑧. Users can hover over an explanation to see the pattern description. Furthermore, selecting a join graph in ⑦ 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 explanation pattern for the green (t_1) and purple (t_2) result tuples selected in ③). 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 ⑨ 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 t_1, t_2 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

[1] National Basketball Association. 2020. The official site of the NBA. [Online; last accessed 10-September-2020].

[2] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and informative explanations of outcomes. *PVLDB* 8, 1 (2014).

[3] Boris Glavic. 2021. Data Provenance - Origins, Applications, Algorithms, and Models. *Foundations and Trends in Databases* 9, 3-4 (2021), 209–441.

[4] Chenjie Li, Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2021. Putting Things into Context: Rich Explanations for Query Answers using Join Graphs. In *SIGMOD*. 1051–1063. https://arxiv.org/pdf/2103.15797.

[5] Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2019. Going Beyond Provenance: Explaining Query Answers with Pattern-based Counterbalances. In *SIGMOD*. 485–502.

[6] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries.

[7] Xiaolan Wang and Alexandra Meliou. 2019. Explain3D: Explaining Disagreements in Disjoint Datasets. *PVLDB* 12, 7 (2019), 779–792.

[8] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB* 6, 8 (2013), 553–564.

[9] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*. 847–864.