Putting Things into Context: Rich Explanations for Query Answers using Join Graphs

Chenjie Li[1], Zhengjie Miao[2], Qitian Zeng[1], Boris Glavic[1], Sudeepa Roy[2]
[1]IIT, Chicago, IL, USA [2]Duke University, Durham, NC, USA
{cli112,qzeng3}@hawk.iit.edu,{zjmiao,sudeepa}@cs.duke.edu,bglavic@iit.edu

ABSTRACT

In many data analysis applications there is a need to explain why a surprising or interesting result was produced by a query. Previous approaches to explaining results have directly or indirectly relied on data provenance, i.e., input tuples contributing to the result(s) of interest. However, some information that is relevant for explaining an answer may not be contained in the provenance. We propose a new approach for explaining query results by augmenting provenance with information from other related tables in the database. Using a suite of optimization techniques, we demonstrate experimentally using real datasets and through a user study that our approach produces meaningful results and is efficient.

ACM Reference Format:
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 *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3459246

1 INTRODUCTION

Today's world is dominated by data. Recent advances in complex analytics enable businesses, governments, and scientists to extract value from their data. However, results of such operations are often hard to interpret and debugging such applications is challenging, motivating the need to develop approaches that can automatically interpret and explain results to data analysts in a meaningful way. Data provenance [15, 22], which has been studied for several decades, is an immediate form of explanation that describes how an answer is derived from input data. However, provenance is often insufficient for unearthing interesting insights from the data that led to a surprising result, especially for aggregate query answers. In the last few years, several *"explanation"* methods have been proposed by the database community [8, 20, 32, 35, 41, 42, 48, 51, 53]. However, real world data often exhibits complex correlations and inter-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 inter-relationships into account. Thus, the explanations they produce may lack important contextual information. We illustrate how to use context to explain a user's question using data extracted from the official website of the NBA [7].

SIGMOD '21, June 20–25, 2021, Virtual Event, China.

ACM ISBN 978-1-4503-8343-1/21/06.
https://doi.org/10.1145/3448016.3459246

Example 1. *Consider a simplified* ***NBA database*** *with the following relations (the keys are underlined, the full schema has 11 relations). Some tuples from each relation are shown in Figure 1. Each team playing in a game can use multiple* lineups *consisting of five players.* Home *refers to the home team in a game.*

- `Game(year, month, day, home, away, home_pts, away_pts, winner, season)`*: participating teams and the winning team.*
- `PlayerGameScoring(player, year, month, day, home, pts)`*: the points each player scored in each game he played in.*
- `LineupPerGameStats(lineupid, year, month, day, home, mp)`*: the minutes each lineup played in a game.*
- `LineupPlayer(lineupid, player)`*: the players in a lineup.*

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 g  WHERE winner = 'GSW' GROUP BY winner, season
```

Figure 1e shows the result of Q_1. GSW made history in the 2015-16 season by being the team that won the most games in a single season. Observe that team GSW improved its performance significantly from season 2012-13 (t_1) to season 2015-16 (t_2). Such a drastic increase in a relatively short period of time naturally raises the question of what changed between these 2 seasons (expressed as the user question UQ_1 in Figure 1f*). Note that Q_1 only accesses the* Game *table (shown in Figure 1a). This table provides the user with information about each game including the name of the opponent team and whether GSW was the home team or not. 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 team plays the same number of games and home games, and roughly the same number of times against each opponent.*

In this paper, we present an approach that answers questions like UQ_1 (Figure 1f). Our approach produces insightful explanations that are based on contextual information mined from tables that are related to the tables accessed by a user's query. To give a flavor of the explanations produced by our approach, we present two of the top explanations for UQ_1 in Figures 2a and 2c (the formal definitions and scoring function are presented in the next section). Each explanation consists of three elements: (1) A *join graph* consisting of a node labeled PT representing the table(s) accessed by the user's query (referred to as the provenance table, or PT for short), and nodes representing other tables that were joined with the provenance table to provide context. Edges in a join graph represent joins between two tables and are labeled with join conditions. (2) A *pattern*, which is a conjunction of predicates over attributes from the provenance or any table from the context. (3) The *support* of the pattern, i.e., the number of tuples from the provenance of each

	year	mon	day	home	away	home_pts	away_pts	winner	season
g_1 →	2013	01	02	MIA	DAL	119	109	MIA	2012-13
g_2 →	2012	12	05	DET	GSW	97	104	GSW	2012-13
g_3 →	2015	10	27	GSW	NOP	111	95	GSW	2015-16
g_4 →	2014	01	05	WAS	GSW	96	112	GSW	2013-14
g_5 →	2016	01	22	GSW	IND	122	110	GSW	2015-16

(a) Game Table

lineupid	player
58420	K. Thompson
58420	D. Green
13507	S. Battier
13507	L. James
67949	D. Green

(b) LineupPlayer Table

	player	year	mon	day	home	pts
p_1 →	S. Curry	2012	12	05	DET	22
p_2 →	S. Curry	2015	10	27	GSW	40
p_3 →	S. Curry	2016	01	22	GSW	39
p_4 →	K. Thompson	2012	12	05	DET	27
p_5 →	K. Thompson	2016	01	22	DET	18
p_6 →	D. Green	2012	12	05	DET	2

(c) PlayerGameScoring table

lineupid	year	mon	day	home	mp
13507	2013	11	09	MIA	4.30
77727	2012	12	12	MIA	14.70
58420	2015	11	07	SAC	10.30
58482	2015	11	07	SAC	11.10
58420	2014	12	08	MIN	11.70

(d) LineupPerGameStats Table

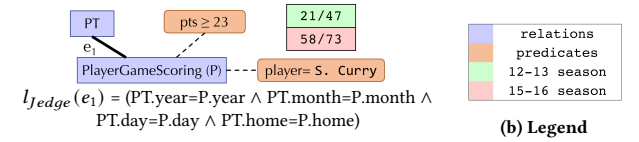
	team	season	win
t_1 →	GSW	2012-13	47
	GSW	2013-14	51
	GSW	2014-15	67
t_2 →	GSW	2015-16	73
	GSW	2016-17	67

(e) Result of Q_1

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

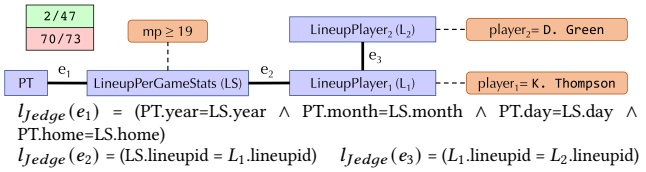
(f) User question UQ_1

Figure 1: Input/output tables, and the user question for Example 1.



(a) Join graph Ω_1 with pattern Φ_1 for UQ_1: Star Player

(b) Legend



(c) Join graph Ω_2 with pattern Φ_2 for UQ_1: Pair of players

Figure 2: Explanations for UQ_1

of the two result tuples from the user question that are covered by the pattern (underlined in the explanations shown below).

Intuitively, the explanation from Figure 2a can be interpreted as:

> *GSW* won more games in season 2015-16 because Player `S. Curry` scored ≥ 23 points in **58 out of 73 games** in 2015-16 compared to **21 out of 47 games** in 2012-13.

Given this explanation, the user can infer that `S. Curry` was one of the key contributors for the improvement of GSW's winning record since his points significantly improved in the 2015-16 season. Similarly, the explanation in Figure 2c can be interpreted as:

> *GSW* won more games in season 2015-16 because Player `D. Green` and Player `K. Thompson`'s on-court minutes together were ≥ 19 minutes in **70 out of 73 games** in the 2015-16 season compared to only **2 out of 47 games** in the 2012-13 season.

This implies that Green and Thompson's increase of playing time together might have helped improve *GSW*'s record. We discuss additional example queries, user questions, and explanations produced by our approach in Section 4.

Our Contributions. In this paper, we develop CaJaDE (Context-Aware Join-Augmented Deep Explanations), the first system that automatically augments provenance data with related contextual information from other tables. Our system produces informative summaries of the difference between the values of two tuples in the answer of an aggregate query, or, the high/low value of a single outlier tuple. We make the following contributions.

(1) Join-augmented provenance summaries as explanations. We propose the notion of join-augmented provenance and use summaries of augmented provenance as explanations. The *join-augmented provenance* is generated based on a *join graph* that encodes how the provenance is joined with tables that provide context. We use patterns, i.e., conjunctions of equality and inequality predicates, to summarize the difference between the *join-augmented provenance* of two tuples t_1, t_2 from a query's output selected by the user's question. We adapt the notion of *F-score* to evaluate the quality of patterns. That is, to explain the difference between t_1 and t_2, we search for patterns with (i) high *recall* (the pattern covers many tuples from t_1's provenance) and (ii) high *precision* (the pattern does not cover many tuples from t_2's provenance) **(Section 2)**.

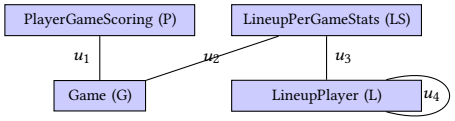
(2) Mining patterns over augmented provenance. We present algorithms for mining patterns for a given join graph and discuss a number of optimizations. Even if we fix a single join graph to compute the augmented provenance, the large number of possible patterns makes it challenging to efficiently mine patterns with high F-score values. Our optimizations include clustering and filtering attributes using machine learning methods, using a monotonicity property for the recall of patterns to prune *refinements* of patterns (patterns are refined by adding additional predicates), and finding useful patterns on categorical attributes before considering numeric attributes to reduce the search space **(Section 3.1)**.

(3) Mining join graphs giving useful patterns. We also address the challenge of mining patterns over join graphs that are based on a *schema graph* which encodes which joins are permissible in a schema. We prune the search space by estimating the cost of pattern mining and determining based on the available join options if a join graph is unlikely to generate high quality patterns **(Section 3.2)**.

(4) Qualitative and quantitative evaluation. We quantitatively evaluated the explanations produced by our approach through a case study on two real-world datasets: NBA and MIMIC. Furthermore, we conducted a user study to evaluate how useful the explanations generated by our approach are and how they compare with explanations generated based on the provenance alone **(Section 4)**. We also evaluated the scalability of our algorithms and the effectiveness of our optimizations **(Section 5)**.

2 JOIN-AUGMENTED PROVENANCE

A database D consists of a set of relations $\mathbf{rels}(D) = \{R_1, \ldots, R_k\}$. Abusing notation, we will use D and $R_1, \ldots, R_k$ to both denote a schema and an instance when clear from the context. For a relation R, $\mathbf{attrs}(R)$ denotes the set of attributes in R; Similarly, for a set of relations $\mathbf{S}$, $\mathbf{attrs}(\mathbf{S}) = \cup_{R \in \mathbf{S}} \mathbf{attrs}(R)$ denotes the set of attributes in relations in $\mathbf{S}$. Without loss of generality, we typically will assume that attribute names are distinct and use $R.A$ for disambiguation if an attribute A appears in multiple relations. In this work we focus on simple single-block SQL queries with a single aggregate function



$l_{Sedge}(u_1) = \{(P.year = G.year \wedge P.month = G.month \wedge P.day = G.day \wedge P.home = G.home), (P.year = G.year \wedge P.month = G.month \wedge P.day = G.day \wedge P.home = G.home \wedge P.home = G.winner)\}$
$l_{Sedge}(u_2) = \{(G.year = LS.year \wedge G.month = LS.month \wedge G.day = LS.day \wedge G.home = LS.home)\}$
$l_{Sedge}(u_3) = \{(LS.lineupid = L.lineupid)\}$
$l_{Sedge}(u_4) = \{(L.lienupid = L.lineupid)\}$

Figure 3: Schema Graph for Example 1.

(select-from-where-group by), or, equivalently extended relational algebra queries with the same restriction.[1]. Given a query Q, $Q(D)$ denotes the result of evaluating the query over a database D. We use $\mathbf{rels}_Q(D) \subseteq \mathbf{rels}(D)$ to denote the relations accessed by Q.

2.1 Provenance Table

A large body of work has studied provenance semantics for various classes of queries (e.g., [15, 22]). Here we resort to a simple *why-provenance* [15] model sufficient for our purpose. We define the provenance of an output tuple $t \in Q(D)$ of a query Q as a subset of the cross product ($\times$) of all relations in $\mathbf{rels}_Q(D)$. For instance, Perm [21] can produce this type of provenance for queries in relational algebra plus aggregation and nested subqueries. In our implementation we use the GProM system [5].

Definition 1 (Provenance Table). *Given a query Q having $\mathbf{rels}_Q(D) = \{R_{j_1}, \cdots, R_{j_p}\}$, we define the provenance table $\mathcal{PT}(Q, D)$ for D and Q to be a subset of $R_{j_1} \times \cdots \times R_{j_p}$. We assume the existence of a provenance model that determines which tuples from the cross product belong to $\mathcal{PT}(Q, D)$. For a tuple $t \in Q(D)$, we define the provenance table $\mathcal{PT}(Q, D, t)$ to be the subset of the provenance that contributes to t (decided by the provenance model).*

Example 2. *In Example 1, $\mathcal{PT}(Q_1, D)$ contains all the tuples from Figure 1a which has GSW as the winner, i.e., g_2, g_3, g_4 and g_5. For $t_1, t_2 \in Q_1(D)$ as shown in Table 1e, $\mathcal{PT}(Q_1, D, t_1)$ includes all the tuples where GSW won in the* 2012-13 *season, i.e., g_2, and $\mathcal{PT}(Q_1, D, t_2)$ contains g_3 and g_5 (GSW's wins in the* 2015-16 *season).*

2.2 Schema Graph and Join Graph

Schema graphs. As mentioned in the introduction, we create explanations by summarizing provenance augmented with additional information produced by joining the provenance with related tables. We assume that a *schema graph* is given as input that models which joins are allowed. The vertices of a schema graph correspond to the relations in the database. Each edge in this graph encodes a possible join between the connected relations, and is labeled with a set of possible join conditions between the two connected relations. We use Cond to denote the set of all predicates involving Boolean conjunctions ($\wedge$) and equality ($=$) of two attributes or an attribute with a constant that can be used for joining relations in D (i.e., only *equi-joins* are allowed).

[1]Extensions are discussed in Section 7.

Definition 2 (Schema Graph). *Given a database schema D, a schema graph $G = (V_S, E_S, l_{Sedge})$ for D is an undirected edge-labeled graph with nodes $V_S = \mathbf{rels}(D)$, edges E_S, and a labeling function $l_{Sedge} : E_S \rightarrow 2^{Cond}$ that associates a set of conditions with every edge from E_S. We require that for each edge $u \in E_S$, each condition in $l_{Sedge}(u)$ only references attributes from relations incident to u.*

Note that G is an input for our method. To create schema graphs, our system can extract join conditions from the foreign key constraints of a database and also allows the user to provide additional join conditions. Furthermore, l_{Sedge} could also be determined using join discovery techniques such as [19, 46, 54]. Figure 3 shows the simplified schema graph for the NBA dataset discussed in Example 1. We only show relations that are used in the examples above. In the schema graph, relations are represented by nodes and are connected through edges $(u_1, u_2, \ldots, u_4)$ with conditions as labels. For example, $l_{Sedge}(u_1)$ in Figure 3 implies that we are allowed to join `PlayerGameScoring(P)` with `Game(G)` in two different ways: (1) through an equi-join on *year, month, day*, and *home* (i.e., the home-team of a game), which returns a player's stats for all the games they played, and (2) with an additional condition on *home = winner*, which returns a player's stats for games for which the home team won. Note that there is an edge u_4 which suggests node `LineupPlayer(L)` can be joined with itself on condition *L.lineupid = L.lineupid* (renaming of L is needed in the actual join) to find players in the same lineup.

Join graphs. A *join graph* Ω encodes *one possible way* of augmenting $\mathcal{PT}(Q, D)$ with related tables. It contains a distinguished node PT representing the relations in $\mathbf{rels}_Q(D)$. The other nodes of Ω are labeled with relations. Edges in Ω are labeled with join conditions allowed by the schema graph G. There can be multiple parallel edges between two nodes, i.e., Ω is a multi-graph.

Definition 3 (join graph). *Given a database D, schema graph $G = (V_S, E_S, l_{Sedge})$ and query Q, a join graph Ω for G is a node- and edge-labeled undirected multigraph $(V_J, E_J, l_{Jnode}, l_{Jedge})$ with nodes V_J, edges E_J, a node labeling function $l_{Jnode} : V_J \rightarrow \mathbf{rels}(D) \cup \{PT\}$, and edge labeling function $l_{Jedge} : E_J \rightarrow$ Cond. For any join graph we require that it contains exactly one node labeled with PT and there are no edges with PT as both end-points. For every edge $e = (n_1, n_2) \in E_J$ we require that there exists a corresponding edge $u = (R_1, R_2) \in E_S$ such that all of the following conditions hold:*

- *$l_{Jedge}(e) \in l_{Sedge}(u)$ (modulo renaming relations using their aliases for disambiguation as discussed below)*
- *If $l_{Jnode}(n_1) = PT$, then $R_1 \in \mathbf{rels}_Q(D)$, else, $l_{Jnode}(n_1) = R_1$*
- *If $l_{Jnode}(n_2) = PT$, then $R_2 \in \mathbf{rels}_Q(D)$, else, $l_{Jnode}(n_2) = R_2$*

The first condition above says that the join condition between two relations in the join graph Ω should be one of the allowed conditions from the schema graph G. The second and third conditions state that edges adjacent to node PT should correspond to an edge adjacent to a relation accessed by query Q. Note that multiple nodes in V_J may be labeled with the same relation and also relations from $\mathbf{rels}_Q(D)$ may appear in node labels.

Disambiguation of relation and attribute names in join graphs. In join graphs corresponding to a schema graph, we may need to address some ambiguity in attribute names and relation names. (1) Unlike the schema graph G, the join graph Ω may contain the

year	mon	day	home	away	home_pts	away_pts	winner	season	player	pts
2012	12	05	DET	GSW	97	104	GSW	2012-13	S. Curry	22
2012	12	05	DET	GSW	97	104	GSW	2012-13	K. Thompson	27
2012	12	05	DET	GSW	97	104	GSW	2012-13	D. Green	2
2015	10	27	GSW	NOP	111	95	GSW	2015-16	S. Curry	40
2016	01	22	GSW	IND	122	110	GSW	2015-16	S. Curry	39
2016	01	22	GSW	IND	122	110	GSW	2015-16	K. Thompson	18

Figure 4: $\mathcal{APT}(Q_1, D, \Omega_1)$ result using example tuples

same relation R_i multiple times with node label $\neq PT$. We assign each such occurrence of R_i a fresh label $R_{i1}, R_{i2}, \cdots$ in Ω. For edges incident on $R_{i1}, R_{i2}, \ldots$ in Ω we use these labels ($R_{i1}.A, R_{i2}.A, \ldots$) instead of the original attribute name $R_i.A$. (2) In addition to the join graph Ω, even in the original query Q and therefore in the provenance table $\mathcal{PT}(Q, D)$, the same relation $R_i \in \mathbf{rels}_Q(D)$ can appear multiple times using different aliases, say, R_{i1} and R_{i2}. Suppose in the schema graph G there is an edge between R_i and R_j. Then in a join graph Ω, there can be two parallel edges between node PT and R_j, one corresponding to a join between R_{i1} and R_j, and the second one corresponding to the join between R_{i2} and R_j. The labels of these edges will use the corresponding aliases ($R_{i1.A}$ on one edge and $R_{i2.A}$ on the other) for disambiguation. Note that both (1) and (2) may occur in the same join graph.

Example 3. *Consider the join graph Ω_2 from Figure 2c. Since $\mathbf{rels}_{Q_1}(D)$ = {*`Game`*}, PT represents the one relation accessed by Q. Nodes from this join graph are connected through edges (e_1, e_2, e_3), where each edge has a corresponding condition in the schema graph shown in Figure 3. For example, the join condition on e_1 from Ω_2 is the first condition in the label of u_2 from the schema graph, i.e., $l_{Jedge}(e_1) \in l_{Sedge}(u_2)$. Similarly, $l_{Jedge}(e_2) \in l_{Sedge}(u_3)$. As discussed above,* `LineupPlayer` *appears more than once in the join graph renamed as* $\texttt{LineupPlayer}_1$ *(L_1) and* $\texttt{LineupPlayer}_2$ *(L_2).*

2.3 Augmented Provenance Table

We now describe the process of generating the relation corresponding to a given join graph Ω — the result of joining the relations in the graph Ω based on the encoded join conditions (after renaming relations and attributes as described in the previous section).

Definition 4 (Augmented Provenance Table). *Consider a database D, a query Q, and a join graph $\Omega = (V_J, E_J, l_{Jnode}, l_{Jedge})$. Let $S_{j_1}, \cdots, S_{j_p} = V_J - \{PT\}$, i.e., all the relations that appear in Ω with labels $\neq PT$. Furthermore, let $t \in Q(D)$ and tuple $t' \in \mathcal{PT}(Q, D, t)$. We define the* augmented provenance table *(*APT*) for D, Q, and Ω (and t, t') as*

$$\mathcal{APT}(Q, D, \Omega) = \sigma_{\theta_\Omega}(\mathcal{PT}(Q, D) \times S_{j_1} \times \cdots \times S_{j_p})$$
$$\mathcal{APT}(Q, D, \Omega, t) = \sigma_{\theta_\Omega}(\mathcal{PT}(Q, D, t) \times S_{j_1} \times \cdots \times S_{j_p})$$
$$\mathcal{APT}(Q, D, \Omega, t, t') = \sigma_{\theta_\Omega}(\{t'\} \times S_{j_1} \times \cdots \times S_{j_p})$$

Here $\theta_\Omega = \bigwedge_{(S_a, S_b) \in E_J} l_{Jedge}((S_a, S_b))$ is the conjunction of join conditions in the join graph Ω. The join conditions only use equality comparisons between two attributes. We assume that duplicate (renamed) columns are removed from $\mathcal{APT}(Q, D, \Omega)$.

Example 4. *Consider Ω_1 from Figure 2a that combines the provenance table PT with* `PlayerGameScoring` *through an equi-join on* year, month, day, *and* home. *Figure 4 shows the join result $\mathcal{APT}(Q_1, D, \Omega_1)$ using the tuples from Figures 1a and 1c.*

2.4 Explanations with Augmented Provenance

CaJaDE's approach for generating explanations is based on summarizing augmented provenance tables. In particular, given a database and a query, the user identifies interesting or surprising tuples in the query's result (e.g., the aggregate value is high/low, or the value of a tuple is higher/lower than that of another tuple). To explain such interesting results, CaJaDE returns *patterns* (i.e., predicates) that each summarize the difference between the augmented provenance for the two query result tuples (or the provenance of one result tuple and all other result tuples).

User questions. Given a database D and a query Q, CaJaDE supports **two-point questions** or comparisons, which we will discuss by default: *Given $t_1, t_2 \in Q(D)$, summarize input tuples in D that differentiate t_1 from t_2.* However, CaJaDE also works for **single-point questions:** *Given a single tuple $t \in Q(D)$, summarize input tuples in D that differentiate t from the rest of the tuples.* Here the intuitive idea is to treat t as t_1, and all tuples $t' \neq t \in Q(D)$ as t_2.

Explaining aggregates vs. summarizing provenance vs. non-provenance. Instead of directly explaining why an aggregate value t.`val` is high/low or t_1.`val` higher/lower than another value t_2.`val` [35, 42, 53], the goal of CaJaDE is to use "patterns" (discussed below) to summarize the input tuples that contributed the most to an output tuple as well as distinguish it from the other outputs. Therefore, unlike the approaches in [35, 42, 53], in CaJaDE, the aggregate values t_1.`val`, t_2.`val`, t.`val` do not play a role in the explanations[2].

Summarization patterns and explanations. In the CaJaDE framework, explanations are *patterns* (*conjunctive predicates*) that compactly encode sets of tuples from the augmented provenance tables based on a join graph. This type of patterns has been used widely for explanations [20, 31, 35, 42, 53].

Definition 5 (Summarization Pattern and Matching Tuples). *Let R be a relation with attributes $(A_1, \ldots, A_m)$, and let $\mathbb{D}_i$ denote the* active domain *of attribute A_i in R. A* summarization pattern *(or simply a* pattern*) Φ is an $m-$ary tuple such that for every $A_i \in R$, (i) if A_i is a numerical or ordinal attribute: $\Phi.A_i \in \bigcup_{X \in \mathbb{D}_i}\{(X, \leq), (X, \geq), (X, =)\} \cup \{*\}$, (ii) if A_i is a categorical attribute: $\Phi.A_i \in \bigcup_{X \in \mathbb{D}_i}\{(X, =)\} \cup \{*\}$. Here $*$ denotes that the attribute is not being used in the pattern and $X \in \mathbb{D}_i$ denotes a threshold for numeric attributes. If $\Phi.A_i \neq *$, then $\Phi.A_i[0]$ denotes the threshold X and $\Phi.A_i[1]$ denotes the comparison operator $\leq, \geq$, or $=$.*

A tuple $t \in R$ matches *a pattern Φ, written as $t \vDash \Phi$, if t and Φ agree on all conditions, i.e., $\forall i \in \{1, \ldots, m\}$, one of the following must hold: (i) $\Phi.A_i = *$, or (ii) ($t.A_i \geq \Phi.A_i[0]) \wedge (\Phi.A_i[1] = \text{`} \geq \text{'})$, or (iii) ($t.A_i \leq \Phi.A_i[0]) \wedge (\Phi.A_i[1] = \text{`} \leq \text{'})$, or (iv) ($t.A_i = \Phi.A_i[0]) \wedge (\Phi.A_i[1] = \text{`} = \text{'})$. We use* match$(\Phi, R)$ *to denote $\{t \in R \mid t \vDash \Phi\}$.*

When presenting textual descriptions of summarization patterns, we omit attributes which are set to $*$, and instead include the attribute name as $(A_i : \Phi.A_i[0], \Phi.A_i[1])$ to avoid ambiguity. Also, since the group-by attributes exactly capture the answer tuples t_1, t_2, and do not provide any additional information, patterns are not allowed to include attributes used in grouping in the query Q.

As discussed in the introduction, the explanations computed by CaJaDE consist of a join graph Ω, a pattern Φ over $\mathcal{APT}(\Omega, D)$,

[2]Taking the amount of contribution (responsibility/sensitivity) of input tuples into account as in [35, 42, 53] is an interesting direction for future work.

and the *support* of Φ to differentiate one tuple from the others by *augmenting* the provenance using Ω, and thereby providing additional contextual information from other tables in D.

Definition 6 (Explanations from Augmented Provenance). *Given a database D, schema graph G, query Q, and a two-point question with $t_1, t_2 \in Q(D)$, an explanation is a tuple $E = (\Omega, \Phi, (v_1, a_1), (v_2, a_2))$, where Ω is a join graph for G; Φ is a pattern over the augmented provenance table $\mathcal{APT}(Q, D, \Omega)$; and (v_1, a_1) and (v_2, a_2) denote the relative support of Φ for t_1 and t_2, respectively.*[3]

For simplicity, we will often drop (v_1, a_1) and (v_2, a_2).

Example 5. *Consider the explanation from Figure 2a. The pattern $\Phi_1 = \{(\text{player} : \textit{S.Curry}, =), (\text{pts} : 23, \geq)\}$ is based on Ω_1. Here player is a categorical attribute and pts is a numeric attribute (the other attributes are $*$), both coming from the* `PlayerGameScoring` *table. Any tuple from $\mathcal{APT}(Q_1, D, \Omega_1)$ which fulfills* player = 'S.Curry' *and $pts \leq 23$ is in $\text{match}(\Phi_1, \mathcal{APT}(Q_1, D, \Omega_1))$. One possible explanation for UQ_1 from Figure 1f is: $(\Phi_1, \Omega_1, (58, 73), (21, 47))$.*

Note that explanations for two-point questions are asymmetric, as one of the tuples is chosen as the *primary tuple* whose relative support is given by (v_1, a_1), and the second one is chosen as the *secondary tuple*, whose relative support is given by (v_2, a_2). Switching these two tuples may result in a different set of top explanations using the quality measure that we discuss next.

2.5 Quality Measure of Explanations

First, we discuss the quality measure for explanations within the context of one join graph, and then discuss how to find top explanations across all join graphs mined by our algorithms.

2.5.1 Quality Measures Given a Join Graph. For a two-point user question focusing on the difference between $t_1, t_2 \in Q(D)$, we would like an explanation's pattern to match as many tuples from the provenance of t_1 as possible, and the least amount of tuples from the provenance of t_2 as possible. For this purpose, we adapt the notion of *F-score*. Recall that $\text{match}(\Phi, R)$ denotes $\{t \in R \mid t \vDash \Phi\}$.

Definition 7 (Quality Metrics of a Pattern). *Consider a database D, a query Q, a join graph Ω, two output tuples in the user question $t_1, t_2 \in Q(D)$, and an explanation pattern $E = (\Omega, \Phi)$.*

(a) A tuple $t' \in \mathcal{PT}(Q, D, t_1)$ (similarly for t_2) is said to be **covered** *by E if there exists $t'' \in \mathcal{APT}(Q, D, \Omega, t_1, t')$ (ref. Definition 4) such that $t'' \vDash \Phi$. The* coverage *of E on t_1, t' in $\mathcal{APT}(Q, \Omega, D)$ is:*

$$\mathsf{Cov}(E, \Omega, t_1, t') = \mathbb{1}[\textit{match}(\Phi, \mathcal{APT}(Q, \Omega, D, t_1, t')) \neq \emptyset]$$

where $\mathbb{1}[]$ is the indicator function.

(b) The **coverage** *(or,* **true positives***) of E for t_1 is defined as the sum of its coverage on all tuples in the provenance table:*

$$\mathsf{TP}(E, \Omega, t_1) = \sum_{t' \in \mathcal{PT}(Q, D, t_1)} \mathsf{Cov}(E, \Omega, t_1, t')$$

(c) The **false positives** *of E for t_1 in comparison to t_2 is the sum of its coverage on all tuples in $\mathcal{PT}(Q, D)$ that are in the provenance of t_2 (t_1 does not appear on the right-hand side here):*

$$\mathsf{FP}(E, \Omega, t_1, t_2) = \sum_{t' \in \mathcal{PT}(Q, D, t_2)} \mathsf{Cov}(E, \Omega, t_2, t')$$

(d) The **false negatives** *of E for t_1 is defined as the sum of the uncovered tuples in the provenance of t_1:*

$$\mathsf{FN}(E, \Omega, t_1) = \sum_{t' \in \mathcal{PT}(Q, D, t_1)} 1 - \mathsf{Cov}(E, \Omega, t_1, t')$$

(e) Using (b)-(d), we define **precision, recall**, *and* **F-score** *for t_1 in comparison to t_2 as usual:*

$$\mathsf{Prec}(E, \Omega, t_1, t_2) = \frac{\mathsf{TP}(E, \Omega, t_1)}{\mathsf{TP}(E, \Omega, t_1) + \mathsf{FP}(E, \Omega, t_1, t_2)}$$

$$\mathsf{Rec}(E, \Omega, t_1) = \frac{\mathsf{TP}(E, \Omega, t_1)}{\mathsf{TP}(E, \Omega, t_1) + \mathsf{FN}(E, \Omega, t_1)}$$

$$\mathsf{Fscore}(E, \Omega, t_1, t_2) = \frac{2}{\frac{1}{\mathsf{Prec}(E, \Omega, t_1, t_2)} + \frac{1}{\mathsf{Rec}(E, \Omega, t_1)}}$$

A pattern Φ with high recall provides a good description of the tuples contributing to t_1. A high precision implies that Φ covers few tuples in the provenance of t_2. A high F-score indicates both. This definition can be easily adapted to single-point questions involving a single output tuple $t \in Q(D)$ by summing over $t' \in \mathcal{PT}(Q, D) \setminus \mathcal{PT}(Q, D, t)$ instead of summing over $t' \in \mathcal{PT}(Q, D, t_2)$ in the false positives definition above. The other definitions remain the same.

Support of explanation patterns. As described in the running example and in Definition 6, an explanation $E = (\Omega, \Phi, (v_1, a_1), (v_2, a_2))$ includes the relative support of the pattern Φ for $t_1, t_2 \in Q(D)$ to record how this pattern differentiates the output tuples t_1 and t_2. Here $v_1 = \mathsf{TP}(\Phi, \Omega, t_1)$ and $a_1 = \mathsf{TP}(\Phi, \Omega, t_1) + \mathsf{FN}(\Phi, \Omega, t_1) = |\mathcal{PT}(Q, D, t_1)|$ as defined in Definition 7, denoting the set of tuples in the provenance of t_1 covered by the pattern Φ, and the set of all tuples in the provenance of t_1 respectively. Similarly, we define v_2, a_2 for the output tuple t_2 to demonstrate the difference with t_1.

Finding Top-k Patterns with Highest F-scores. Given a join graph Ω, our goal is to find the top-k patterns Φ in terms of their individual F-scores according to Definition 7. However, in practice there are additional considerations that we should take into account, e.g., the maximum number of attributes appearing in a pattern.

Complexity. Finding top-k patterns given a join graph Ω has polynomial *data complexity* [49] (fixed size schema and query). The provenance table and APTs can be computed in PTIME in the size of the data. Given a pattern, its matches can be determined in PTIME and therefore, all metrics in Definition 7 can be computed in PTIME. If there are p attributes in the augmented provenance table, the number of possible patterns is in $O(n^p)$ (the number of distinct attribute values is bound by n = total number of tuples in the database, and each attribute can appear as don't-care $*$ and with one of the three comparison operators). Thus, even a naive approach for computing the top-k patterns with the highest F-score values is polynomial in data size. However, this naive approach does not scale in practice and therefore we adopt a number of heuristic optimizations to solve this problem as described in Section 3.1.

Explanations over All Join Graphs. When mining multiple join graphs Ω, there are several options for finding top patterns across all join graphs, e.g., penalizing patterns from complex join graphs. However, for simplicity, and for an interactive user experience, we find top-k patterns for each individual join graph and present a global ranking of all patterns. Thus, the user can explore explanations generated from more than one join graph (see Section 3.2).

[3] We will discuss the relative support in the next section.

3 ALGORITHMS FOR MINING PATTERNS

In this section we discuss our algorithms for mining patterns for a given join graph (Section 3.1), and for enumerating all join graphs over which patterns should be mined (Section 3.2). Pseudocode for these algorithms and additional details are presented in [34].

3.1 Mining Patterns given a Join Graph

We first give an overview of our algorithm for mining patterns from an augmented provenance table (APT) generated based on a given join graph Ω. Recall that we are dealing with patterns that may contain equality comparisons (for categorical attributes) and/or inequality comparisons (for numeric attributes). We mine patterns in multiple phases. (i) In the first *preprocessing step*, we *cluster attributes* that are highly correlated to reduce redundancy in the patterns. The output of this step are clusters each with one selected representative. (ii) In the second preprocessing step, we use random forests to remove clusters that have low relevance for predicting membership of input tuples in the provenance of only one of the two output tuples from the user question. Such attributes are unlikely to yield patterns of high quality. (iii) Next we only consider categorical attributes and mine pattern candidates using a variation of the *LCA (Least Common Ancestor)* method from [20] that can only handle categorical attributes. (iv) From the set of patterns returned by the LCA method, we then select the top-k_{cat} patterns with the highest recall and frequency for the next step. (v) These patterns are then *refined* (see below) by adding conditions on numerical attributes that can improve precision at the potential cost of reducing recall. (vi) Finally, the top-k patterns based on score according to Definition 7 are returned.

A pattern Φ' is a **refinement** of a pattern Φ if Φ' can be derived from Φ by replacing one or more $*$ with comparisons, e.g., pattern $\Phi_2 = ((X, =), (Y, \leq))$ is a refinement of $\Phi_1 = ((X, =), *)$. We use the following observation that is implied by Definition 7 (a), (b), (d), and (e) to prune patterns if their recall is below a threshold.

Proposition 3.1. *Given a tuple $t \in Q(D)$ and a join graph Ω, $\texttt{Rec}(E_2, \Omega, t) \leq \texttt{Rec}(E_1, \Omega, t)$, where $E_1 = (\Omega, \Phi_1)$, $E_2 = (\Omega, \Phi_2)$, and Φ_2 is a refinement of Φ_1.*

Next we discuss the above six steps in more detail.

3.1.1 Clustering Attributes based on Correlations. Redundancy in patterns can be caused by attributes that are highly correlated. As an extreme example, consider an APT containing both the birth date and age of a person. For any pattern containing a predicate on birth date there will be an (almost) equivalent pattern using age instead, and also a pattern using both age and birth date. To reduce the prevalence of such redundant patterns, we cluster attributes based on their mutual correlation and pick a single representative for each cluster. We use VARCLUS [45], a clustering algorithm closely related to principal component analysis and other dimensionality reduction techniques [40]. However, any other technique for clustering correlated attributes can be used instead.

3.1.2 Filtering Attributes based on Relevance. Random forests have been successfully used in machine learning applications to determine the relevance of a feature (attribute) to the outcome of a classification task. We train a random forest classifier that predicts whether a row belongs to the augmented provenance of only one of the two outputs from the user's question [11]. We then find the fraction $\lambda_{\#sel-attr}$ of attributes with the highest relevance ($\lambda_{\#sel-attr}$ is an input parameter, a summary of all parameters discussed in this section is shown in Table 4). The rationale for this step is to prune patterns involving attributes that are irrelevant for distinguishing the tuples t_1 and t_2 from the user question. Such attributes can be added to any pattern with minimal effect on the recall and precision of patterns, but could mislead users into thinking that the value of this attribute is a distinguishing factor for t_1, t_2.

3.1.3 Patterns over Categorical Attributes. We then generate a sample of size $\lambda_{pat-samp}$ (an input parameter) from $\mathcal{APT}(Q, D, \Omega)$ and a set of candidate patterns over only categorical attributes using the LCA method from [20]. We ignore all numerical attributes at this stage by using $*$. Our implementation of the LCA method generates pattern candidates from the cross product of two samples of the same size. A candidate pattern is generated for each pair (t, t') of tuples from the sample by replacing values of attributes A where $t.A \neq t'.A$ with a placeholder $*$ and by keeping constants that t and t' agree upon ($t.A = t'.A$). Note that in our case each element of a pattern is a predicate, therefore, using a constant as done in the LCA method corresponds to using an equality predicate (i.e., $A = c$ for $t.A = t'.A = c$). By keeping constants that frequently co-occur, the LCA method generates patterns that reflect common combinations of constants in the data. Focusing on categorical attributes first enables us to (i) use the established heuristic of the LCA method to generate categorical-only pattern candidates, and (ii) to significantly reduce the search space by pruning all refinements of patterns without a sufficiently high recall.

3.1.4 Filtering Categorical Pattern Candidates. Next, we calculate the recall for each pattern finding the matches for the pattern in the APT. In order to quickly find the most common patterns, we reduce the number of queries needed to be run to check recall by choosing the most frequent patterns in the LCA results. We then filter out patterns whose recall is below a threshold λ_{recall}, and pick at most k_{cat} patterns, which we denote by $\mathcal{P}_{cat}$.

3.1.5 Refinement and Numeric Attributes. We then generate refined patterns from $\mathcal{P}_{cat}$ by replacing placeholders $*$ on numerical attributes with predicates. Even though such refinements can at best have the same recall as the original pattern, their precision may be higher. Recall that for numerical attributes we allow for both equality as well as inequality predicates, for which there is a large number of possible constants to choose from, because numerical attributes tend to have large domains. To reduce the size of the search space, we split the domain of each numerical attribute into a fixed number of fragments (e.g., quartiles) and only use the boundaries of these fragments when generating refinements. We systematically enumerate all refinements of a pattern by extending it with one predicate at a time. We use $\mathcal{P}_{refined}$ to denote the union of $\mathcal{P}_{cat}$ with the set of patterns generated in this step.

3.1.6 Computing Top-k Patterns. Finally, we calculate the F-score for each pattern in $\mathcal{P}_{refined}$ and return k patterns by taking into account both F-scores and a measure for diversity to give the user different types of interesting explanations (more details are presented in [34]). As an optimization, we calculate the F-score over a

sample of the data (with sample size $\lambda_{F1-samp}$). Note that $\lambda_{F1-samp}$ can be different from the sample size $\lambda_{pat-samp}$ in the previous step since we found that a small sample is sufficient for generating a meaningful set of patterns, but may not be sufficient for estimating recall with high enough accuracy. We iteratively build the final pattern set of size k starting with the pattern with the highest F-score. When deciding what pattern to add next to the result, we penalize patterns that share attributes and values with a pattern that is already in the result.

3.2 Join Graph Enumeration

In this section, we describe an algorithm that enumerates join graphs of increasing size iteratively. The maximum size of join graphs considered by the algorithm is determined by a parameter $\lambda_{\#edges}$. We employ several heuristic tests to determine whether a join graph generated by the algorithm should be considered for pattern mining. The rationale for not considering all join graphs for pattern mining is that for some join graphs, pattern mining may not yield patterns of good quality, or it may be much more computationally expensive than generating the join graph. Hence we skip pattern mining for join graphs that are unlikely to be worth the cost. For join graphs that pass these tests we materialize the corresponding APT and apply the pattern mining algorithm from Section 3.1 to compute the top-k patterns for the APT.

Generating join graphs. We enumerate join graphs in iteration i by extending every join graph produced in iteration $i-1$ with all possible edges. For each join graph Ω produced in iteration $i-1$, we consider two types of extensions: (i) we add an additional edge between two existing nodes of the graph and (ii) we add a new node and connect it via a new edge to an existing node.

Checking join graph connectivity and skipping expensive APT computations. We filter out join graphs based on *lack of connectivity* and on *high estimated costs*. Note that schema graphs may contain tables with multiple primary key attributes incident to edges which join on part of the key. This is typical in "mapping" tables that represent relationships. For instance, consider the `PlayerGameScoring` table from our running example that stores the number of points a player scored per game. Assume that there exists another table `Player` that is not part of the running example. The primary key of `PlayerGameScoring` consists of a foreign key to the `Game` table and a foreign key to the `Player` for which we are recording stats. Consider a query that joins the `Game` table with `LineupPerGameStats`, `LineupPlayer` and selects games played by team GSW. A valid join graph for this query would be to join the node *PT* with `PlayerGameScoring` on game's primary key.

Note that while the result of this query contains rows that pair a player of GSW with lineups they played in during a particular game, the APT would pair each row with any player that played in that game irrespective of their team. Join graphs like this can lead to redundancy and large APT tables. One reason for this redundancy is that not all primary key attributes of the `PlayerGameScoring` table are joined with another table. To prevent such join graphs with redundancy, our algorithm checks that for every node in the join graph, the primary key attributes of that node are joined with at least one other node from the join graph. For instance, in the example above the join graph could be modified to pass this check by also joining the `PlayerGameScoring` table with the `Player` table.

Even though we only consider fully connected join graphs, some generated join graphs will result in APTs of significant size, which are expensive to materialize and mine patterns from. We use the DBMS to estimate the cost of the materialization query upfront. We skip pattern mining for join graphs where the estimated cost of this query is above a threshold λ_{qCost}. While we may lose explanations by skipping join graphs, experimental results demonstrate that this check is necessary for achieving reasonable performance. Furthermore, smaller join graphs result in less complex explanations.

Ranking results. After enumerating join graphs and computing top-k patterns for each join graph, we rank the union of all pattern sets on their F-scores. Ranking all patterns (instead of filtering out some patterns) reduces the load on the user by increasing the likelihood that good patterns are shown early on without having the risk of not showing patterns that have lower scores.

4 QUALITATIVE EVALUATION

We now evaluate the quality of explanations produced by CaJaDE using case studies on two real datasets (NBA and MIMIC). We also report results of a user study with the NBA dataset. Due to the space limit, we simplified some of these descriptions. Note that the same pattern may be returned for several join graphs (same attributes, but different join path). In the interest of diversity, we removed duplicates and explanations that only differ slightly in terms of constants. We show the top-3 explanations after this step. we use "$[t_1]$" or "$[t_2]$" in explanations as identifier of the *primary tuple* for the explanation. For the sets of top-20 explanations (including join graphs), SQL code of the queries, and detailed descriptions of the datasets, please see [34].

4.1 Case Study: NBA

Dataset. *NBA* is a dataset we extracted from the NBA website (https://www.nba.com/) and PBP stats website (https://www.pbpstats.com/). It includes 10 seasons' worth of data (seasons `2009-10` to `2018-19`). The dataset contains 11 relations and is ~ 170MB large.

Setup. For the NBA dataset, we use five queries calculating player's and team's stats and generated user questions based on interesting results. Figure 5 shows the queries, user questions, and top-3 explanations produced by our method for these user questions.

Explanations and Analysis. Q_{nba1}: *Draymond Green* had a big average points difference between 2 consecutive seasons. All 3 explanations contain salary change information. In reality, from 2015-16 season to 2016-17 season, *Green*'s salary increased, which could result in losing incentive to play as hard as when he earns a lower salary. The 2^{nd} and 3^{rd} explanation identify factors affecting the player's points such as minutes played and shooting percentage per game, e.g., *Green* had more games where he played more than 31 minutes and had more than 0.4 shooting percentage in $2015-16$ season (2^{nd} explanation). Q_{nba2}: The *GSW* team had a sudden increase in average assists. All explanations contain `assistpoints` which has a cause-and-effect relationship with assists (more assists result in more assistpoints). Q_{nba3} and Q_{nba5}: Both players had some significant average point changes. For Q_{nba3}, *Lebron James*

Query	User question	Top explanations	F-score
Q_{nba1}	***Draymond Green*'s average points per year**: 14 points in season 2015-16 (t_1) VS 10 points in season 2016-17 (t_2)	player_salary< 15330000 [t_1]	1
		prov.tspct< 0.69 ∧ prov.usage< 20.5 ∧ salary> 14260000 [t_2]	0.71
		prov.minutes> 31 ∧ prov.tspct> 0.4 ∧ salary< 15330000 [t_1]	0.66
Q_{nba2}	***GSW*'s average assists per year**: 23 in season 2013-14 (t_1) VS 27 in season 2014-15 (t_2)	prov.assistpoints< 68 ∧ player=Draymond Green [t_1]	0.74
		prov.assistpoints> 57 ∧ prov.nonputbackast_2_pct> 0.55 ∧ player.player=Harrison Barnes [t_2]	0.73
		prov.assistpoints< 68 ∧ offreboundpct> 0.25 [t_1]	0.72
Q_{nba3}	***LeBron James*'s average points per year**: 29.7 in season 2009-10 (t_1) VS 26.7 in season 2010-11(t_2)	player_salary> 14500000 [t_1]	1
		team=MIA [t_2]	0.98
		team=CLE [t_1]	0.93
Q_{nba4}	***GSW*'s number of wins per year**: 47 in 2012-13 (t_1) season VS 67 in 2016 − 17 season (t_2)	player_name=Pau Gasol ∧ player_salary< 19285850 [t_2]	1
		player_name=Andre Iguodala [t_2]	0.97
		fg_3_apct< 0.31∧ team_points< 121 [t_1]	0.92
Q_{nba5}	***Jimmy Butler*'s average points per year**: 13 points in season 2013-14 (t_1) VS 20 points in season 2014-15 (t_2)	player_salary> 1112880 [t_2]	1
		prov.away_points> 87 ∧ prov.efgpct> 0.38 [t_2]	0.84
		prov.usage< 23 ∧ team=CHI∧ team_assisted_2_spct> 0.5 [t_1]	0.77

Figure 5: Queries, user questions and explanations (NBA)

Query	User question	Top-3 explanations	F-score
Q_{mimic1}	**Patient death rate grouped by diagnoses**: 0.19 for chapter= 2 (t_1) VS 0.09 for chapter= 13 (t_2)	expire_flag=1 [t_1]	0.68
		hospital_stay_length< 23∧expire_flag= 1 [t_1]	0.65
		hospital_stay_length< 16,expire_flag= 1 [t_1]	0.63
Q_{mimic2}	**Death rate by insurance**: Medicare=0.138 (t_1) VS Medicaid=0.066 (t_2)	prov.admission_type=emergency [t_1]	0.85
		expire_flag= 1 [t_1]	0.70
		gender=Male [t_1]	0.65
Q_{mimic3}	**Number of patients grouped by ICU stays length**: less than 1 day (t_1) VS more than 8 days (t_2)	hospital_stay_length> 9 ∧ procedure.chapter=16 [t_2]	0.86
		hospital_stay_length< 6 ∧ los_group=0-1 [t_1]	0.86
		prov.dbsource=carevue ∧ hospital_stay_length> 8 [t_2]	0.78
Q_{mimic4}	**Death rate by insurance**: Medicare=0.14 (t_1) VS Private=0.06 (t_2)	expire_flag=0∧age< 71 [t_2]	0.77
		prov.admission_type=*emergency* [t_1]	0.73
		prov.hospital_stay_length< 22.0 ∧ expire_flag=1 [t_1]	0.61
Q_{mimic5}	**Number of patients that did a procedure grouped by ethnicity**: 7821 Hispanic patients [t_1] VS 6247 Asian patients [$t2$]	hospital_stay_length< 19 ∧ ethnicity=Asian [t_2]	0.90
		admission_type=emergency ∧ hospital_stay_length> 5 ∧ age< 66 ∧ ethnicity=Hispanic [t_1]	0.80
		prov.religion=Catholic [t_1]	0.63

Figure 6: Queries, user questions and explanations (MIMIC)

had an average points decrease. This occurred when he switched to a new team (from *CLE* in *2009-10* season to *MIA* in *2010-11* season). In *MIA* he had less pressure offensively. CaJaDE successfully identified this fact as a potential cause (2^{nd} and 3^{rd} explanation). Q_{nba5}: *Jimmy Butler* had a big improvement in average points. Our top explanations for this improvement include an increase of usage and minutes played. Q_{nba4}: This query is similar to our running example Q_1 but with a question asking for different 2 seasons. The 2^{nd} explanation identifies a player change: *Andre Iguodala* only played for *GSW* in 2015 − 17 season. The 3^{rd} explanation is about the team's points difference and 3-point percentage. While the first explanation has a high F-score, the join graph details reveal that the salary and player constants can have no relation with GSW. This highlights the importance of including join graphs in explanations.

4.2 Case Study: MIMIC

Dataset. *MIMIC* (https://mimic.physionet.org/) is a deidentified dataset of intensive care unit hospital admissions. The dataset consists of 6 relations and is ∼ 120MB large. We constructed 5 queries over this dataset accessing different tables. The simplified descriptions of the queries, user questions, and explanations are shown in Figure 6. We first briefly introduce the MIMIC dataset to help the reader understand the queries and explanations. The main table of the dataset is the `Admissions` table that records hospital admissions. The `Diagnosis` table records diagnosis for patients for each admission (a patient may be admitted more than once during their lifetime). `PatientsAdmissionInfo` contains information like age and religion for individual admissions of patients (e.g., age may change over time). `ICUStays` records intensive care unit stays of patients. There may be multiple ICU stays per admission.

Explanations and Analysis. Figure 6 shows the top-3 explanations returned by CaJaDE for each user question. Q_{mimic1}: This question asks for the difference in death rates between two diagnosis categories (chapter 2: neoplasms vs chapter 13: musculoskeletal system and connective tissue). The death rate is the fraction of patients that died during their hospital stay. The 1^{st} explanation uses *expire_flag*= 1 from the patient table suggesting that this patient has passed away. This flag only indicates that the patient died, but not whether during their hospital stay or not, subsuming all hospital deaths. The 2^{nd} and 3^{rd} explanations add additional information about the lengths of hospital stays, which indicates a difference between the severity of these two categories which explains the different death rates. Q_{mimic2}: This query asks about the difference between the death rates of two groups of patients based on their insurance types. The 1^{st} explanation states that there are more *emergency* admissions with *Medicare* than with *Medicaid*, which may explain the higher death rate. The 2^{nd} explanation relates death rate to `expire_flag`. The 3^{rd} explanation suggests that *Medicare* has more *Male* patients than *Medicaid*. Q_{mimic3}: The 1^{st} explanation shows that most of the patients staying over 8 days in the ICU will stay in hospital for more than 9 days and also have procedures from chapter 16 (*Miscellaneous Diagnostic and Therapeutic Procedures*). The 2^{nd} explanation suggests that most patients will be released from hospital in less than 6 days when their ICU stay is less than 1 day. The 3^{rd} explanation states the same fact that patients will stay more than 8 days in hospital when they stay more than 8 days in the ICU. These explanations regarding hospital stay length can help users identify that ICU stay length may be a good indicator for hospital stay length. Q_{mimic4}: This question uses the same query as Q_{mimic2}, but compares *Private* insurance with *Medicare*. The 1^{st} explanation states that for patients who have *Private* insurance, more patients are alive and less than 71 years old. This is aligned with the fact that *Medicare* is mostly for patients over 65 years old (this is a fact extracted from online resources). The 2^{nd} and 3^{rd} explanations are stating that patients using *Medicare* are more likely to be admitted because of an *emergency* and also the facts about length of hospital stays. Q_{mimic5}: The 1^{st} explanation states that *Asian* patients who had a procedure are more likely to stay less than 19 days in the hospital. The 2^{nd} explanation says that compared to *Asian* patients there were more *Hispanic* patients younger than 66 years old and that stayed more than 5 days in the hospital. The 3^{rd} explanation points out that more *Hispanic* patients are *Catholic*. Note that the ethnicity information in the explanations

	CaJaDE-PT-Only
$Expl_1$	In season 2015-16, among the games GSW won, they were the visiting team and had points > 104 in 28 games (10 games in 2012-13, resp.)
$Expl_2$	In 2015-16 season, 73 games (47 games in 2012-13, resp.) GSW won are regular season games.
$Expl_3$	In 2015-16 season, among the games GSW won, they were the visiting team, had points > 98 and possessions > 101 in 17 games (0 games in 2012-13, resp.)
$Expl_4$	In 2015-16 season, GSW scored more than 104 points in each of 64 games (24 games in 2012-13, resp.) GSW won.
$Expl_5$	In 2015-16 season, the home teams had points < 106 and possessions < 101 in each of 29 games (40 games in 2012-13, resp.) GSW won.
	CaJaDE
$Expl_6$	In 2015-16 season, the number of games with GSW player Stephen Curry's minutes < 38 and usage > 25 is 59 games (12 games in 2012-13, resp.) GSW won.
$Expl_7$	In 2015-16 season, the number of games with GSW player Draymond Green's minutes>15 is 73 games (15 games in 2012-13, resp.) GSW won.
$Expl_8$	In 2015-16 season, Jarrett Jack played in 0 games (47 games in 2012-13, resp.) GSW won.
$Expl_9$	In 2015-16 season, GSW had three_pct > 35% and points > 112 in each of 39 games (9 games in 2012-13, resp.) GSW won.
$Expl_{10}$	In 2015-16 season, GSW had fg_three_pct > 48% and points > 112 and rebounds > 51 in 5 games (2 games in 2012-13, resp.) GSW won.

Table 1: Explanations for UQ_1 used in the user study

are not from PT, but from a different `patient_admit_info` table. Because we do not consider functional dependencies, results like this cannot be avoided. We plan to address this in future work.

4.3 User Study

We conducted a user study for the NBA dataset to evaluate: (S1) whether CaJaDE provides meaningful explanations in addition to explanations that only come from the provenance, and (S2) whether the CaJaDE's quality metric is consistent with user preference.

Participants. We recruited 20 participants — all of them are graduate students studying computer science, 13 of them have some prior experience with SQL, and 5 were NBA fans.

Tasks. We first presented background knowledge of the NBA to each participant, and explained the schema of the dataset. Each participant was shown the SQL query Q_1' (shown below) and the results of this query, and then was asked to find and evaluate explanations for the user question UQ_1 from Example 1: "*Why did GSW win 73 games in season* 2015-16 *compared to 47 games in* 2012-13*?*".

```
Q1' = SELECT s.season_name, count(*) AS win
FROM team t, game g, season s WHERE t.team_id=g.winner_id
   AND g.season_id=s.season_id AND t.team='GSW'
GROUP BY s.season_name
```

We gave each participant familiar with SQL 20 minutes to explore the dataset and manually find explanations. Participants then were asked to rate each of the top-5 explanations with the highest F-scores in two groups using a scale from 1 to 5. The first group (**CaJaDE-PT-only**) of explanations is produced by CaJaDE using only the *provenance table*, while for the second group (**CaJaDE**) we use *join graphs to extend the provenance table* (see Table 1). We also asked participants which group of explanations makes more sense and whether they provided new insights. Because the top explanations in the CaJaDE group have higher F-scores, we added one with a low F-score ($Expl_{10}$) as a control. By covering a wider range of F-score values, we can test S2: (1) can participants distinguish between low and high score explanations, and (2) do participants agree with the ranking based on our quality measure.

	CaJaDE-PT-only					**CaJaDE**				
	Expl1	Expl2	Expl3	Expl4	Expl5	Expl6	Expl7	Expl8	Expl9	Expl10
All users	3.150	1.450	3.950	3.600	2.750	3.600	3.800	2.350	3.950	2.300
Stdev	1.040	0.999	0.759	1.095	1.410	0.883	1.196	1.424	0.999	1.174
NBA: Yes	3.400	1.800	3.800	3.600	2.800	3.800	3.800	2.800	4.200	2.600
NBA: No	3.067	1.333	4.000	3.600	2.733	3.533	3.800	2.200	3.867	2.200
F-score	0.69	0.56	0.38	0.8	0.4	0.82	0.91	1	0.64	0.13
recall	0.38	1	0.23	0.87	0.4	0.81	1	0.99	0.53	0.07
precision	0.74	0.61	1	0.73	0.4	0.83	0.83	0.99	0.81	0.7

Table 2: Average ratings for each explanation by users with different expertise and the measures for each explanation by CaJaDE

			CaJaDE-PT-only (All / -1)	**CaJaDE** (All / -1)
Avg. Kendall tau rank distance	All users	F-score	3.95 / 2.2	3.9 / 1.4
		recall	5.9 / 3.85	3.3 / 1.4
		precision	2.2 / 0.95	3.9 / 1.4
	Users with domain knowledge	F-score	3.6 / 2.0	3.2 / 1.8
		recall	5.2 / 3.2	3.8 / 1.8
		precision	2.2 / 1.2	4.2 / 1.8
Avg. $NDCG_n$	All users	F-score	0.875 / 0.882	0.901 / 0.955
		recall	0.844 / 0.852	0.901 / 0.955
		precision	0.933 / 0.965	0.901 / 0.955
	Users with domain knowledge	F-score	0.897 / 0.901	0.903 / 0.954
		recall	0.862 / 0.878	0.903 / 0.954
		precision	0.953 / 0.977	0.903 / 0.954

Table 3: Ranking quality: all 5 explanations (*All*), dropping the explanation with the largest standard deviation (*-1*).

Results and Analysis. Overall, the responses were positive: 16 out of 20 participants agreed that the explanations by CaJaDE make more sense to them and seeing these explanations in advance will help them find explanations that they did not think about before.

Table 2 shows the average user ratings and quality measures for each explanation. Regarding (S1), the average ratings of the top-1 explanation are the same for both groups ($Expl_9$: 3.95 vs $Expl_3$: 3.95, both explanations summarize the team statistics of GSW while $Expl_9$ refers to the table *team_game_stats* not in the provenance). Explanations $Expl_7$ and $Expl_6$ (CaJaDE) summarize the statistics of two GSW's key players and have higher average ratings ($Expl_7$: 3.8 vs $Expl_5$: 3.6, $Expl_6$: 3.6 vs $Expl_1$: 3.15). The margin is larger for participants who are familiar with the NBA (4.2 vs 3.8, 3.8 vs 3.6, 3.8 vs 3.4).

Regarding (S2), we find that explanations with high user ratings (Expl 3, 4, and Expl 6, 7, 9) have a positive correlation with high F-score and precision. The only exception, $Expl_8$ in the CaJaDE group, is also the most controversial one, indicated by the largest standard deviation. Evaluating these explanations is subjective and requires domain knowledge: the player Jack in $Expl_8$ left GSW in 2013, and participants may or may not regard this as a signal that the team had begun relying more on other players who play a similar position. Next, we evaluate the ranking results of our quality measures by regarding each participant's ratings as the ground truth. We use Kendall-Tau rank distance [28] for measuring pairwise ranking error and normalized discounted cumulative gain (NDCG) [26] for the entire ranked list. As shown in Table 3, ranking by precision gives the lowest pairwise ranking error for CaJaDE-PT-only, while for CaJaDE it is ranking by F-score. If we drop the most controversial explanation, the pairwise error is reduced by more than half. The $NDCG_n$ for CaJaDE reaches 0.9 for all cases and even 0.95 after dropping the most controversial explanation.

Takeaways. The main findings are: (1) the majority (16/20) of participants preferred CaJaDE, thanks to the new information provided by tables not used in the query, which complements the explanations only based on provenance; (2) our quality measures

Parameter	Description	Default
$\lambda_{db-size}$	the size of the database (scale factor)	1.0
$\lambda_{\#edges}$	maximum number of edges per join graph (Section 3.2)	3
$\lambda_{\#sel-attr}$	#attributes returned by feature selection (Section 3.1.2)	3
$\lambda_{attrNum}$	max number of numerical attributes allowed in a pattern	3
$\lambda_{pat-samp}$	sample rate for LCA pattern candidate generation (Section 3.1.3)	0.05
$\lambda_{F1-samp}$	sample rate for calculating F-scores of patterns (Section 3.1.6)	0.3

Table 4: Parameters of our approach and default values

are consistent with participants' preferences; (3) for both groups, there can be top explanations rated low by participants, which is expected because we did not do causal analysis, and validating such explanations may be subjective and depend on domain expertise; and (4) participants with domain knowledge had a stronger preference for CaJaDE than participants without domain knowledge.

Other findings and discussion. Finally, it is also worth noting that the participants' feedback support the motivation of CaJaDE. For example, participants found that *"The usage of Stephen Curry increases in 2015-16."*, *"'Players play both season (12-13 and 15-16) have higher point per game and assist per game"* before they saw the explanations by CaJaDE. One suggested to use health information of the players in explanations. Another participant remarked that *"the use of other tables in the database to explore how the contributions of individual players can have an outcome on the team's performance produced explanations that were more novel or interesting"*.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate the implementation of our algorithms and optimizations in CaJaDE. We evaluate both the performance in terms of runtime and the quality of results with respect to different parameters and compare against systems from related work.

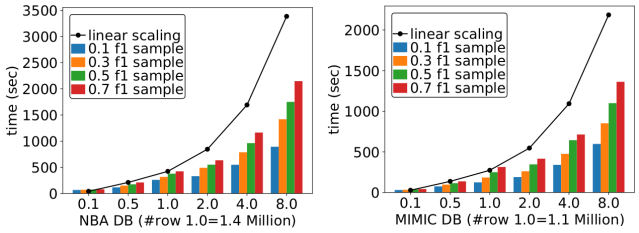
Datasets. We use the *NBA* and *MIMIC* datasets described in Section 4. We created several scaled versions of these two datasets preserving the relative sizes of most tables and join results.

Experimental setup. CaJaDE is implemented in Python (version 3.6) and runs on top of PostgreSQL (version 10.14). All experiments were run on a machine with 2 x AMD Opteron 4238 CPUs, 128GB RAM, and 4 x 1 TB 7.2K RPM HDDs in hardware RAID 5.

Parameters and Optimizations. Table 4 shows the parameters used in our experiments and their default values. We vary the following: (1) the size of the database; (2) the maximum number of join graph edges $\lambda_{\#edges}$; (3) the sample rate for F-score $\lambda_{F1-samp}$; and (4) the sample rate for pattern candidate generation ($\lambda_{pat-samp}$). In [34] we also compare our approach with and without feature selection and evaluate how the maximum allowed number of edges in join graphs ($\lambda_{\#edges}$) affects performance. Based on these results we activated feature selection and set $\lambda_{\#edges} = 3$ for all experiments. Unless stated otherwise we use queries Q_1 from Section 1 (for NBA experiments) and Q_{mimic4} from Section 4.2 (for MIMIC experiments) with their respective user questions and use the default values for all other parameters.

5.1 Scalability

To evaluate the scalability of our approach, we used scaled versions of the NBA and MIMIC datasets ranging from ~ 10% to ~ 800%. We varied the F-score sample rate ($\lambda_{F1-samp}$) from 0.1 to 0.7. The results are shown in Figure 7 comparing against linear scaling (black line). The effect of database size on runtime is similar for both

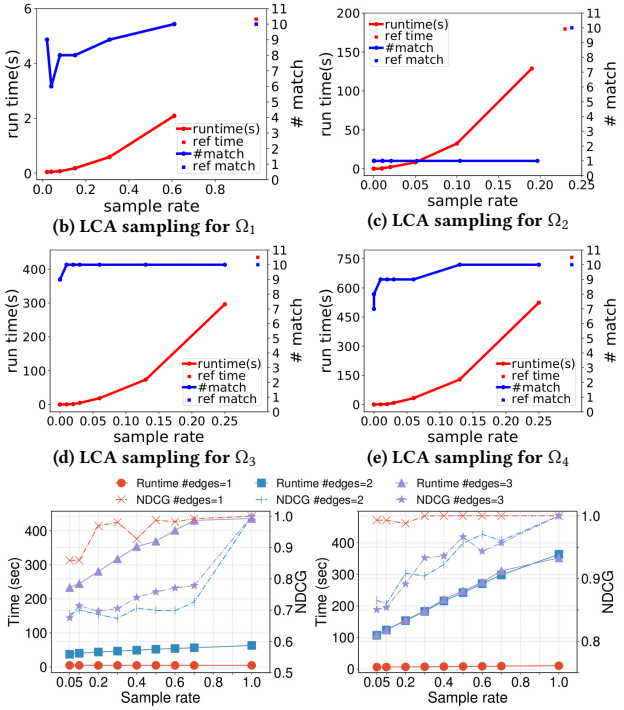


(a) NBA, varying $\lambda_{F1-samp}$ (b) MIMIC, varying $\lambda_{F1-samp}$

Figure 7: Scalability in database size

join graph	join graph structure	APT (#rows)	# attributes
Ω_1	PT	2621	2
Ω_2	PT - player_salary - player	66282	2
Ω_3	PT	50797	10
Ω_4	PT - patient_admt_info - patients	50797	19

(a) Join graph APTs size (LCA sampling)



(b) LCA sampling for Ω_1 (c) LCA sampling for Ω_2

(d) LCA sampling for Ω_3 (e) LCA sampling for Ω_4

(f) Left: NBA and right: MIMIC, varying $\lambda_{F1-samp}$

Figure 8: Effect of sampling on runtime and pattern quality

datasets. Our approach shows sublinear scaling for both datasets (note the log-scale x-axis). The benefits of sampling are more pronounced for larger database sizes: $\lambda_{F1-samp} = 0.1$ is more than 55% faster than $\lambda_{F1-samp} = 0.7$ for scale factor 8 on both datasets. We present a detailed breakdown on where time is spent in [34]. F-score calculation turned out to be the most significant factor.

5.2 Sample Size

We now study the impact of sampling for F-score calculation ($\lambda_{F1-samp}$) and for pattern candidate generation ($\lambda_{pat-samp}$) on performance and pattern quality. We treat the result produced without sampling as ground truth and measure the difference between this result and the result produced by sampling.

Sampling for Pattern Generation. Recall that we use the *LCA* approach to generate candidate patterns over the categorical attributes of an augmented provenance table. This approach computes a cross product between two samples of a fixed size. Our implementation of LCA ranks the pattern candidates generated by LCA by their recall, and then selects the top-k ranked patterns as input for the next step. In this experiment, we want to determine a robust choice for the *LCA* sample size parameter and, thus, compare the results produced by this step. We selected 4 join graphs and their APTs: Ω_1 and Ω_2 for Q_1, and Ω_3 and Ω_4 for Q_{mimic4}. Figure 8a shows the number of rows and attributes for the APTs, and join graph structure for each of these join graphs. The pattern quality and the runtime of generating top-10 patterns for these four join graphs are shown in Figure 8b to Figure 8e. We measure pattern quality as the number of patterns from the top-10 patterns computed over a significant portion of the full dataset (ground truth) that occur in the top-10 computed based on a sample (see the blue lines labeled *match*). As expected because of the cross product computed over the samples, runtime increases quadratically in the sample size. For Figure 8d and Figure 8e, all ground truth top-10 patterns are found even for just 3% sample rate. Whereas as shown in Figure 8c, even for 20% sample rate (13000 rows), we only find one matching pattern. The reason behind the different result observed in Figure 8d is that one of the columns in Ω_2 has over 800 distinct values that are roughly evenly distributed and, thus, the ranking is sensitive to small variations in frequency caused by sampling. For Figure 8b, even though this join graph also contains this attribute (over 500 distinct values), for this APT, the column's distribution in the APT is skewed leading to a more stable set of high frequency values that are used in the top-10 patterns. Based on these observations we determine the sample size $\lambda_{pat-samp}$ =0.05 for the rest of the experiments and cap the number of rows in the sample at 1000.

Sampling for F-score Calculation. We also use sampling to reduce the cost of the quality measure calculation (parameter $\lambda_{F1-samp}$). Instead of scanning all tuples in the augmented provenance table (APT), we compute the number of matching tuples over a sample of the APT for a given pattern. Figure 8f shows the running time and the quality of patterns when varying the sample rate and maximum number of edges in join graphs ($\lambda_{\#edges}$) for queries Q_1 and Q_{mimic4}. We use the normalized discounted cumulative gain (NDCG) [26] as the sample quality metric. A high NDCG score (between 0 and 1) indicates that the ranking of the top patterns returned by the sampled result is close to the top patterns produced without sampling. Figure 8f shows that for both datasets for $\lambda_{\#edges} = 1$, the similarity between the sampled result and result over the full dataset is high, even for aggressive sampling (we take the average of five runs). For larger join graphs ($\lambda_{\#edges} = 2, 3$), the NDCG score fluctuates around 0.7 for the NBA dataset. For the MIMIC dataset, the NDCG converges to $\sim$ 0.95 or even 1.0 at a sampling rate of 0.5. Overall, even for low sample rates, the NDCG score is at least $\sim$ 0.67 ($\sim$ 0.8) for the NBA (MIMIC) dataset.

5.3 Comparison with Explanation Tables

We also compared our approach against the approach from [20] (referred to as ET from now on). We compared on one join graph with structure `PT - player_game_stats - player` for the NBA dataset using query Q_1 and the corresponding user question from the introduction. The corresponding APT has $\sim$ 2600 rows and 84 columns. To be fair, we did apply our feature selection technique to filter columns for ET too, reducing the number of columns to 20. Without that step, ET took 30 seconds even for the smallest sample size (16 tuples). Figure 9 lists the runtime of CaJaDE and ET after applying feature selection. While slower for a sample size of 16, our approach scales much better when increasing the sample size ($\sim 35x$ faster for sample size 512). That being said, we would like to point out that the major contribution of our work is the efficient exploration of a schema graph for finding explanations. However, as this experiment demonstrates this would not be possible without our optimizations for mining patterns over a single APT.

Sample Size	Runtime (sec)	
	CaJaDE	ET
16	12.77	3.21
64	24.21	11.65
256	23.85	176.76
512	24.51	855.13

Figure 9: Comparison with Explanation Tables

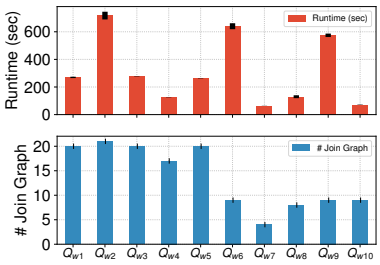


Figure 10: Varying Queries

Rank	Query	
	UQ_{cape1}	UQ_{cape2}
1	(*LeBron James*,2009-10,29.7)	(*GSW*,2013-14,51)
2	(*LeBron James*,2011-12,27.1)	(*GSW*,2014-2015,67)
3	(*LeBron James*,2013-14,27.1)	(*GSW*,2015-16,73)

Figure 11: CAPE's explanations for the NBA questions

Query	Description	Tables used
Q_{w1}	Average points for player *Draymond Green* over the years	player, game, season, player_game_stats
Q_{w2}	Team *GSW* average assists over the years	team_game_stats, game, team, season
Q_{w3}	Average points for player *Lebron James* over the years	player, game, season , player_game_stats
Q_{w4}	Team *GSW* wins over the years	team, game, season
Q_{w5}	Average points for player *Jimmy Butler* over the years	player, game, season, player_game_stats
Q_{w6}	Return the number of diagnosis group by chapter(group of procedure type)	diagnoses
Q_{w7}	Returns the death rate of patients grouped by their insurance.	admissions
Q_{w8}	Number of ICU stays grouped by the length of stays (`los_group`).	icustays
Q_{w9}	Number of procedures for a particular chapter (group of diagnosis types).	procedures
Q_{w10}	Number admissions of different ethnicities.	patients_admit_info

Table 5: NBA and MIMIC Queries

5.4 Comparison with CAPE

We also compared our approach against CAPE [35]. The questions we used are from NBA running example query Q_1 (number of wins for *GSW* over the years) and Q_{nba3} from the case study (player *LeBron James*'s average points over the seasons). CAPE expects as input one data point plus a direction *high* or *low*. We select the following question UQ_{cape_1} for Q_1: *"Why was GSW number of wins high in 2015-16 season?"* and UQ_{cape_2} for Q_{nba3}: *"Why was LeBron*

James' average points low in 2010-11 season?". Since CAPE does not explore related tables, we constructed 2 join graphs as input to CAPE, which are `PT` (UQ_{cape_1}) and `PT - team_game_stats` (UQ_{cape_2}). Figure 11 shows the top-3 explanations produced by CAPE. The system identifies a trend in the data (using regression) according to which the user question is an outlier in the user-provided direction and then returns a similar outlier in the other direction. For our experiment, this means that CAPE returns seasons with low wins for GSW and high averages points for LeBron James. This experiment demonstrates that CAPE is orthogonal to our technique. The system identifies counter-balances while we find features that are related to the difference between two query results. Nonetheless, our techniques for exploring schema graphs may be of use for finding counterbalances too.

5.5 Varying Queries

To evaluate how the runtime of our approach is affected by the choice of query, we measured the runtime for 10 different queries (5 for NBA and 5 for MIMIC) shown in Table 5. We designed these queries to access different relations and use different group-by attributes. The SQL code for these queries is shown in [34]. All queries were run with $\lambda_{F1-samp} = 0.3$ and $\lambda_{\#edges} = 3$. The results are shown in Figure 10. We observe that the runtime is relatively stable for different queries and is to some degree correlated to the number of join graphs for the query (shown on top of Figure 10).

6 RELATED WORK

Provenance and summarization. Provenance [15] for relational queries identifies inputs that contribute to the results of a query. For non-aggregate queries, why-provenance [12] returns a set of input tuples responsible for a given output tuple; how-provenance [23] encodes how the query combined input tuples to generate the answers. For aggregate queries, symbolic expressions based on an extension of semirings [4] are used to express how aggregate results are computed. Given the significant cost of managing provenance information in practical DBMS, several provenance-management frameworks that store and retrieve relevant provenance information have been proposed in the literature [6, 14, 21, 31, 38]. Some of these support provenance for aggregate queries using simplified models and query plan optimizations [27, 36, 38]. A number of recent papers have proposed *summarization* techniques to represent provenance approximately [2, 31–33, 44], or to use summarization rules for better usability [3]. Recent work has also studied natural language representations of factorized and summarized provenance [17, 18].

Data summarization. Another line of work has focused on producing summaries of relational data that are relevant, diverse, and comprehensive [25, 29, 39, 52]. For a relation augmented with a binary outcome attribute, Gebaly et al. [20] developed solutions to find informative summaries of categorical attributes affecting the outcome attribute only considering the provenance (and not other relevant relations like our work). An extension of this idea for numerical data was presented in [50]. We adopt the *lowest common ancestor* (LCA) optimization from [20] in our algorithms to prune the search space of candidate patterns. Note that we have discussed the potential problems of adapting the approach from [20] using a materialized augmented provenance table in Section 5.

Explanations for query answers. This line of work aims at explaining unexpected outcomes of a query, including outlier values, missing tuples, or existing tuples that should not exist. Provenance and provenance summaries provide a straightforward form of explanations [1, 41, 42, 53] by characterizing a set of tuples whose removal or modification would affect the query answer of interest. Query-based explanations, i.e., changes to queries, have been investigated for both "why" and "why-not" questions [10, 13]. Explanations for outliers have been studied in [9, 35]. We share with [35] the motivation of considering explanations that are not (solely) based on provenance. The difference is that in [35], only the table accessed by the query is considered for finding explanations that "counterbalance" an outlier by learning patterns that can balance a low (high) outlier with a high (low) outlier, whereas we find explanations in "augmented provenance" stemming from tables not used in the query. Therefore, [35] is orthogonal to our work.

Join path discovery. Join path discovery approaches find data related to a table of interest based on inclusion dependencies or string similarity [19, 24, 46, 54, 55]. Recently, [16, 30, 47] studied the performance of machine learning models trained on join results. CaJaDE can utilize join graph discovery techniques to find more augmentation opportunities.

7 DISCUSSIONS AND FUTURE WORK

Explanations for database query answers is a relatively new research topic with many interesting future directions. For instance, currently our approach only considers correlations. In the future, we plan to integrate it with the existing *observational causal analysis* framework from AI and Statistics [37, 43] to find causal explanations. Another interesting direction for future work is to integrate context-based explanations with join discovery techniques (e.g., [19, 54]) to automatically find datasets to be used as context. Finally, our approach is not suited well for textual and sparse data because such data cannot be summarized well using the type of patterns we support since values are rarely repeated. Different summarization techniques (e.g., using taxonomies) or preprocessing techniques (e.g., information extraction techniques) would have to be incorporated with our approach. While we discussed simple SQL aggregate queries, extensions of our model can be studied for more general queries (e.g., nested sub-queries or negation) if we have access to a provenance system that can compute the provenance of such queries. Beyond having an intuitive scoring function for ranking explanations that may not always produce meaningful explanations, a challenging direction for future work is to evaluate the correctness of the generated explanations without much human intervention, to evaluate whether the returned explanations match the user's intent, and to have a confidence score for explanations to decide whether the data contains sufficient information to explain a user question. We plan to explore other types of user questions, e.g, explaining an increasing/decreasing trend or explaining why two results are similar. Furthermore, we will investigate the applicability of context/provenance in ML applications.

Acknowledgments. This work is supported in part by NSF awards IIS-1552538, IIS-1703431, IIS-1956123, IIS-2008107, OAC-1541450, and OAC-1640864, and by NIH award R01EB025021.

REFERENCES

[1] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, et al. 2018. DIFF: a relational interface for large-scale data explanation. *PVLDB* 12, 4 (2018), 419–432.

[2] Eleanor Ainy, Pierre Bourhis, Susan B Davidson, Daniel Deutch, and Tova Milo. 2015. Approximated summarization of data provenance. In *CIKM*. 483–492.

[3] Omar AlOmeir, Eugenie Yujing Lai, Mostafa Milani, and Rachel Pottinger. 2021. Summarizing Provenance of Aggregation Query Results in Relational Databases. In *ICDE*.

[4] Yael Amsterdamer, Daniel Deutch, and Val Tannen. 2011. Provenance for aggregate queries. In *PODS*. 153–164.

[5] Bahareh Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. 2018. GProM - A Swiss Army Knife for Your Provenance Needs. *IEEE Data Eng. Bull.* 41, 1 (2018), 51–62.

[6] Bahareh Arab, Dieter Gawlick, Venkatesh Radhakrishnan, Hao Guo, and Boris Glavic. 2014. A generic provenance middleware for database queries, updates, and transactions. In *TaPP*.

[7] National Basketball Association. 2020. The official site of the NBA. https://www.nba.com/ [Online; accessed 10-September-2020].

[8] Dhiman Barman, Flip Korn, Divesh Srivastava, Dimitrios Gunopulos, Neal E. Young, and Deepak Agarwal. 2007. Parsimonious Explanations of Change in Hierarchical Data. In *ICDE*, Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (Eds.). 1273–1275.

[9] Aline Bessa, Juliana Freire, Tamraparni Dasu, and Divesh Srivastava. 2020. Effective Discovery of Meaningful Outlier Relationships. *ACM Transactions on Data Science* 1, 2 (2020), 1–33.

[10] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. 2014. Query-Based Why-Not Provenance with NedExplain. In *EDBT*, Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy (Eds.). 145–156.

[11] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[12] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *ICDT*. 316–330.

[13] Adriane Chapman and HV Jagadish. 2009. Why not?. In *SIGMOD*. 523–534.

[14] Adriane P Chapman, Hosagrahar V Jagadish, and Prakash Ramanan. 2008. Efficient provenance storage. In *SIGMOD*. 993–1006.

[15] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Found. Trends Databases* 1, 4 (2009), 379–474.

[16] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020), 1373–1387.

[17] Daniel Deutch, Nave Frost, and Amir Gilad. 2016. Nlprov: Natural language provenance. *PVLDB* 9, 13 (2016), 1537–1540.

[18] Daniel Deutch, Nave Frost, and Amir Gilad. 2017. Provenance for natural language queries. *PVLDB* 10, 5 (2017), 577–588.

[19] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *ICDE*. 1001–1012.

[20] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and Informative Explanations of Outcomes. *PVLDB* 8, 1 (2014), 61–72.

[21] Boris Glavic and Gustavo Alonso. 2009. Perm: Processing provenance and data on the same data model through query rewriting. In *ICDE*. 174–185.

[22] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *PODS*. 31–40.

[23] Todd J Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *PODS*. 31–40.

[24] Yeye He, Kris Ganjam, and Xu Chu. 2015. SEMA-JOIN: Joining Semantically-Related Tables Using Big Table Corpora. *PVLDB* 8, 12 (2015), 1358–1369.

[25] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. 2017. Interactive data exploration with smart drill-down. *TKDE* 31, 1 (2017), 46–60.

[26] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[27] Grigoris Karvounarakis, Zachary G Ives, and Val Tannen. 2010. Querying data provenance. In *SIGMOD*. 951–962.

[28] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.

[29] Alexandra Kim, Laks VS Lakshmanan, and Divesh Srivastava. 2020. Summarizing Hierarchical Multidimensional Data. In *ICDE*. 877–888.

[30] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, and Xiaojin Zhu. 2016. To Join or Not to Join?: Thinking Twice about Joins before Feature Selection. In *SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). 19–34.

[31] Seokki Lee, Bertram Ludäscher, and Boris Glavic. 2019. PUG: a framework and practical implementation for why and why-not provenance. *VLDBJ* 28, 1 (2019), 47–71.

[32] Seokki Lee, Bertram Ludäscher, and Boris Glavic. 2020. Approximate Summaries for Why and Why-not Provenance. *PVLDB* 13, 6 (2020), 912–924.

[33] Seokki Lee, Xing Niu, Bertram Ludäscher, and Boris Glavic. 2017. Integrating approximate summarization with provenance capture. In *TaPP*.

[34] Chenjie Li, Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2021. Putting Things into Context: Rich Explanations for Query Answers using Join Graphs - supplementary document. (2021). arXiv:2103.15797 [cs.DB]

[35] Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2019. Going Beyond Provenance: Explaining Query Answers with Pattern-based Counterbalances. In *SIGMOD*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). 485–502.

[36] Xing Niu, Raghav Kapoor, Boris Glavic, Dieter Gawlick, Zhen Hua Liu, Vasudha Krishnaswamy, and Venkatesh Radhakrishnan. 2018. Heuristic and Cost-based Optimization for Diverse Provenance Tasks. *TKDE* (2018).

[37] Judea Pearl. 2000. *Causality: models, reasoning, and inference*. Cambridge University Press.

[38] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-grained lineage at interactive speed. *PVLDB* 11, 6 (2018), 719–732.

[39] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2012. Diversifying Top-K Results. *PVLDB* 5, 11 (2012), 1124–1135.

[40] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.

[41] Sudeepa Roy, Laurel Orr, and Dan Suciu. 2015. Explaining query answers with explanation-ready databases. *PVLDB* 9, 4 (2015), 348–359.

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

[43] Donald B Rubin. 2005. Causal inference using potential outcomes: Design, modeling, decisions. *J. Amer. Statist. Assoc.* 100, 469 (2005), 322–331.

[44] Christopher Ré and Dan Suciu. 2008. Approximate lineage for probabilistic databases. *PVLDB* 1, 1 (2008), 797–808.

[45] WS Sarle. 1990. SAS/STAT User's Guide: The VARCLUS Procedure. *SAS Institute, Inc., Cary, NC, USA*, (1990), 134.

[46] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *SIGMOD*. 817–828.

[47] Vraj Shah, Arun Kumar, and Xiaojin Zhu. 2017. Are Key-Foreign Key Joins Safe to Avoid when Learning High-Capacity Classifiers? *PVLDB* 11, 3 (2017), 366–379.

[48] Balder ten Cate, Cristina Civili, Evgeny Sherkhonov, and Wang-Chiew Tan. 2015. High-Level Why-Not Explanations using Ontologies. In *PODS*, Tova Milo and Diego Calvanese (Eds.). 31–43.

[49] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *STOC*. 137–146.

[50] Michael Vollmer, Lukasz Golab, Klemens Böhm, and Divesh Srivastava. 2019. Informative Summarization of Numeric Data. In *SSDBM*. 97–108.

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

[52] Yuhao Wen, Xiaodan Zhu, Sudeepa Roy, and Jun Yang. 2018. Interactive summarization and exploration of top aggregate query answers. In *PVLDB*, Vol. 11. 2196.

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

[54] 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.

[55] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-Join: Joining Tables by Leveraging Transformations. *PVLDB* 10, 10 (2017), 1034–1045.