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

Chenjie Li
IIT
cli112@hawk.iit.edu

Zhengjie Miao
Duke University
zjmiao@cs.duke.edu

Qitian Zeng
IIT
qzeng3@hawk.iit.edu

Boris Glavic
Illinois Institute of Technology
bglavic@iit.edu

Sudeepa Roy
Duke University
sudeepa@cs.duke.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 used data provenance (input tuples contributing to the result(s) of interest), which is limited by the fact that relevant information for explaining an answer may not be fully 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. We develop a suite of optimization techniques, and demonstrate experimentally using real datasets and through a user study that our approach produces meaningful results by efficiently navigating the large search space of possible explanations.

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 SIGMOD ’21: ACM Symposium on Neural Gaze Detection, June 03–05, 2021, Woodstock, NY. ACM, New York, NY, USA, 32 pages. https://doi.org/10.1145/1122445.1122456

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 [14, 21], which has been studied for several decades, is an immediate form of explanations that describes how an answer is derived from input data. However, provenance is often insufficient for unearth 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 emerged in the data management literature [7, 19, 34, 41, 42, 47, 50, 52] that return insightful answers in response to questions from a user. 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 that can aid the user in developing a deeper understanding of the results. We illustrate how to use context to explain a user’s question using data extracted from the official website of the NBA [35].

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 participating 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, away_pts, away_pts, winner, season): participating teams and the winning team.
- PlayerGameScoring(player, year, month, day, home, away, home_pts, away_pts, winner, season): participating teams and the winning team.
- GameLineupGameStats(lineupid, year, month, day, home, away, home_pts, away_pts, winner, season): the minutes played by each player in each game.
- GameLineup(lineupid, player): players for a lineup.

Suppose we are interested in exploring the winning records of the team GSW (Golden State Warriors) in every season. The following query Q1 returns this information:

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

Figure 1e shows the number of games team GSW won for each season. GSW made history in the NBA in the 2015-16 season to be the team that won the most games in a single season. Observe that team GSW improved its performance significantly from season 2012-13 (t1) to season 2015-16 (t2). Such a drastic increase in a relatively short period of time naturally raises the question of what changed between these 2 seasons (denoted as the user question UQ1 in Figure 1f). Note that only the Game table (shown in Figure 1a) was accessed by Q1. This table provides the user with information about each game such as the name of the opponent team or whether GSW was the home team or not. However, such information is not enough for understanding why GSW won or lost more games than in the other seasons, since in each season a team would play 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 answer questions like UQ1 (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 (we refer to this as the provenance table, or PT for short), and nodes representing other tables that were joined with the provenance table to provide the context. Edges in a join graph represent joins between two tables. Each edge is labeled with the join condition that was used to connect the tables. (2) A pattern, which is a conjunction of predicates on attributes from the provenance or any table from the context. (3) The support of the pattern in terms of the number of tuples from the provenance of each of the two result tuples from the user question that are covered by the pattern (bold and underlined in the explanations shown below, formally defined in Section 2.5).

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 2012-13 season.

This implies that Green and Thompson’s increase of playing time together might have helped improve GSW’s record. We will discuss other example queries, user questions, and explanations returned by our approach using the NBA and the MIMIC hospital records dataset [26] in Section 6.

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 should be 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. A high $F$-score is likely to combine two desired properties for the summary distinguishing $t_1$ and $t_2$ by giving preference to patterns with (i) high recall (the pattern covers many tuples in the provenance of $t_1$) and (ii) high precision (the pattern does not cover many tuples in the provenance of $t_2$). (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 poses challenges to efficiently mining 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)

(3) Mining join graphs giving useful patterns. We also address the challenge of mining patterns over different 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 as well as detecting from the available join patterns if the join graph is unlikely to generate high quality patterns. (Section 4)

(4) Qualitative and quantitative evaluation. We quantitatively evaluated the explanations produced by our approach using a case study using two real world datasets: NBA and MIMIC. We further 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 original provenance alone (Section 6). We conducted performance experiments using the NBA and MIMIC database to evaluate scalability varying parameters of our algorithms, and demonstrate the effectiveness of our optimization. (Section 5)

2 JOIN-AUGMENTED PROVENANCE

A database $D$ comprises a set of relations $\text{rels}(D) = \{R_1, \ldots, R_k\}$. We will use $D$ and $R_1, \ldots, R_k$ both for the schema and the instance where it is clear from the context. For a relation $R$, $\text{attrs}(R)$ denotes the set of attributes in $R$; similarly, for a set of relations $S$, $\text{attrs}(S) = \bigcup_{R \in S} \text{attrs}(R)$ denotes the set of attributes in relations in $S$. Without loss of generality, we assume the attribute names are distinct and use $\text{RA}$ 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 (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 $\text{rels}_Q(D) \subseteq \text{rels}(D)$ to denote the relations accessed by $Q$.\(^2\)

2.1 Provenance Table

A large body of work has studied provenance semantics for various classes of queries (e.g., [14, 21]). Here we resort to a simple \textit{why-provenance} [14] 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 $\text{rels}_Q(D)$. For instance, Perm [20] 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].

\textbf{Definition 1 (Provenance Table).} Given a query $Q$ with $\text{rels}_Q(D) = \{R_{i_1}, \ldots, R_{i_n}\}$, we define the provenance table $\mathcal{PT}(Q, D)$ for $D$ and $Q$ to be a subset of $R_{i_1} \times \cdots \times R_{i_n}$. 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).

\textbf{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 $g_5$.

\textbf{2.2 Schema Graphs and Join Graphs}

\textbf{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 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 $\text{Cond}$ to denote the set of all predicates involving Boolean conjunctions ($\land$) and equality ($=$) of two attributes or an attribute with a constant that can be used for joining relations in $D$ (i.e., only \textit{equi-joins} are allowed, although all common attributes do not have to be used as in natural joins).

\textbf{Definition 2 (Schema Graph).} Given a database schema $D$, schema graph $G = (V_S, E_S, l_{\text{edge}})$ for $D$ is an undirected edge-labeled graph with nodes $V_S = \text{rels}(D)$, edges $E_S$, and a labeling function $l_{\text{edge}} : E_S \rightarrow 2^{\text{Cond}}$ 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 $l_{\text{edge}}(u)$ only references attributes from relations adjacent to $u$.

Note that $l_{\text{edge}}$ 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, $l_{\text{edge}}$ could be determined using join discovery techniques such as [18, 45, 53]. Figure 3 shows the simplified schema graph for the NBA dataset discussed in Example 1. Unused relations in examples above are omitted. 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, $l_{\text{edge}}(u_1)$ in Figure 3 implies that we are allowed to join PlayerGameScoring 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 form the key of a game and therefore gives players’ stats in all the games they played, and (2) with an additional condition on home = winner, which gives players’ stats in the games when the home team won. Note that there is an edge $u_4$ which suggests node LineupPlayer$(L)$ can be joined with itself on condition Lineup$\mathit{lineupid} = \mathit{lineupid}$ (renaming of $L$ is needed in the actual join) to find players in the same lineup.

\textbf{Join graphs.} A join graph $\Omega$ encodes one possible way of augmenting $\mathcal{PT}(Q, D)$ with related tables in the schema. It contains a distinguished node $\mathcal{PT}$ representing the relations in $\text{rels}_Q(D)$. The other nodes of $\Omega$ are labeled with relations, edges in $\Omega$ are labeled

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{Schema Graph for Example 1.}
\end{figure}

\footnotesize
\(^1\)Extensions are discussed in Section 8.

\footnotesize
\(^2\)This can be a subset of $\text{rels}(D)$.
with join conditions allowed by the schema graph $G$, and there can be multiple parallel edges between two nodes ($\Omega$ is a multi-graph).

**Definition 3 (Join Graph).** Given a database $D$, schema graph $G = (V_S, E_S, I_{\text{edge}})$ and query $Q$, a join graph $\Omega$ for $G$ is a node- and edge-labeled undirected multigraph $(V_J, E_J, I_{\text{node}}, I_{\text{edge}})$ with nodes $V_J$, edges $E_J$, a node labeling function $I_{\text{node}} : V_J \rightarrow \text{rels}(G) \cup \{PT\}$, and edge labeling function $I_{\text{edge}} : E_J \rightarrow \text{Cond}$. 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:

- $I_{\text{edge}}(e) \in I_{\text{edge}}(u)$ (modulo renaming relations using their aliases for disambiguation as discussed below)
- If $I_{\text{node}}(n_1) = PT$, then $R_3 \in \text{rels}(D)$, else, $I_{\text{node}}(n_1) = R_1$
- If $I_{\text{node}}(n_2) = PT$, then $R_2 \in \text{rels}(D)$, else, $I_{\text{node}}(n_2) = R_2$

The first condition above says that the join condition between two relations in the join graph $\Omega$ should be one of the allowed conditions in the schema graph $G$. The second and third conditions say 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 labelled with the same relation and also relations from $\text{rels}(D)$ may appear node labels.

**Disambiguation of relations and attributes in a join graph using aliases and multigraph.** 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 $\Omega$ may contain the same relation $R_I$ multiple times with node label $\neq PT$. We give each such occurrence of $R_I$ a fresh label $R_{I1}, R_{I2}, \cdots$ in $\Omega$. Each original attribute $A_I$ in the conditions in labels $\theta$ are now renamed as $R_{I1}.A, R_{I2}.A,$ and so on in the edges incident on $R_{I1}, R_{I2}, \cdots$ respectively in $\Omega$. (2) In addition to the join graph $\Omega$, even in the original query $Q$ and therefore in the provenance table $PT(Q, D)$, the same relation $R_I \in \text{rels}(D)$ can appear multiple times using different aliases, say, $R_{I1}, R_{I2}$. Suppose in the schema graph $G$ there is an edge between $R_{I1}, R_{I2}$. Then in a join graph $\Omega$, there can be two parallel edges between node $PT$ and $R_I$, one corresponding to the join condition between $R_{I1}$ and $R_{I2}$, and the second one corresponding to the join condition between $R_{I2}$ and $R_I$. 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 in a join graph, there can be a combination of (1) and (2).

**Example 3.** Consider the join graph $\Omega_2$ from Figure 2c. Since $\text{rels}(D) = \{\text{Game}\}$, $PT$ represents the only 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, join condition on $e_3$ from $\Omega_2$ is the first condition in the label of $u_2$ from the schema graph, i.e., $I_{\text{edge}}(e_1) \in I_{\text{edge}}(u_2).$ Similarly, $I_{\text{edge}}(e_2) \in I_{\text{edge}}(u_3).$ As discussed above, LineupPlayer appears more than once in the join graph renamed as LineupPlayer$_1(L_1)$ and LineupPlayer$_2(L_2)$.

## 2.4 Explanations with Augmented Provenance

CJA1DE’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 answer (e.g., the aggregate value is high/low, or the value of a tuple is higher/lower than another). To explain such interesting results, CJA1DE returns patterns (i.e., predicates) that each summarize the difference between the augmented provenance for two query result tuples (or the provenance of one result tuple).

**User questions.** Given a database $D$ and a query $Q$. CJA1DE 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, CJA1DE 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 $t' \in Q(D)$ 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 $t.val$ higher/lower than another value $t.val$ [34, 42, 52], the goal of CJA1DE 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 [34, 42, 52], in CJA1DE, the aggregate values $t.val, t_2.val, t.val, t.val$ do not play a role in the explanations.



---

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

The graph $\Omega$ 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 $\Omega = (V_J, E_J, I_{\text{node}}, I_{\text{edge}})$. Let $S_{j_1}, \cdots, S_{j_p} = V_J - \{PT\}$, i.e., all the relations that appear in $\Omega$ with labels $\neq PT$. Furthermore, let $t \in Q(D)$ and tuple $t' \in PT(Q, D, t)$, we define: The augmented provenance table (APT) for $D, Q,$ and $\Omega$ (and $t, t'$) is defined as

$$\text{APT}(Q, D, \Omega, t) = \sigma_{\theta_\Omega}(\text{APT}(Q, D) \times S_{j_1} \times \cdots \times S_{j_p})$$

Here $\theta_\Omega = \bigwedge_{(S_{j_1}, \cdots, S_{j_p}) \in I_{\text{edge}}(\Omega)}$ is the conjunction of join conditions in the join graph $\Omega$. The join conditions only use equality comparisons between two attributes. We assume that duplicate (renamed) columns are removed from $\text{APT}(Q, D, \Omega)$.

**Example 4.** Consider $\Omega_1$ in Figure 2a that combines provenance table $PT$ with PlayerGameScoring through an equi-join on year, month, day, and home. Figure 4 shows the join result $\text{APT}(Q_1, D, \Omega_1)$ using the tuples from Figures 1a and 1c.
Summarization patterns and explanations. In the CAJADE framework, explanations are provided as patterns or conjunctive predicates to compactly represent sets of tuples from the augmented provenance tables based on different join graphs. This type of patterns has been used widely for explanations [19, 31, 34, 42, 52].

Definition 5 (Summarization Pattern and Matching Tuples). Let \( R \) be a relation with attributes \( \{A_1, \ldots, A_m\} \), and let \( \mathcal{D}_i \) denote the active domain of attribute \( A_i \) in \( R \). A summarization pattern (or simply a pattern) \( \Phi \) is an \( m \)-ary tuple such that for every \( A_i \in R \), (i) if \( A_i \) is a numerical or ordinal attribute: \( \Phi.A_i \in \mathbb{U}_{X \in \mathcal{D}_i} \{ (X, \leq), (X, \geq), (X, =) \} \cup \{ \ast \} \), and (ii) if \( A_i \) is a categorical attribute: \( \Phi.A_i \in \mathbb{U}_{X \in \mathcal{D}_i} \{ (X, =) \} \cup \{ \ast \} \).

Here \( \ast \) denotes that the attribute is not used in the pattern and \( X \in \mathcal{D}_i \) denotes a threshold for numeric attributes. If \( \Phi.A_i \neq \ast \), then \( \Phi.A_i[0] \) denotes the threshold \( X \) and \( \Phi.A_i[1] \) denotes the comparison operator \( \leq, \geq, = \).

A tuple \( t \in R \) matches a pattern \( \Phi \), written as \( t \vdash \Phi \), if \( t \) and \( \Phi \) agree on all conditions, i.e., \( \forall i \in \{1, \ldots, m\} \), one of the following must hold: (i) \( \Phi.A_i = \ast \), or (ii) \( \Phi.A_i \geq \Phi.A_i[0] \), or (iii) \( \Phi.A_i \leq \Phi.A_i[0] \), or (iv) \( \Phi.A_i[0] = \Phi.A_i[1] \).

When presenting textual descriptions of summarization patterns, we omit attributes which are set to \( \ast \), 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 given by CAJADE consist of a join graph \( \Omega \), a pattern \( \Phi \) over \( \mathcal{AP}T(\Omega, \Omega) \), and statistics on \( \Phi \) to show how it differentiates one tuple from the others by augmenting the provenance using \( \Omega \), and thereby including 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 \( \Omega \) is a join graph for \( G; \Phi \) is a pattern over the augmented provenance table \( \mathcal{AP}T(\Omega, \Omega) \); and \( (v_1, a_1) \) and \( (v_2, a_2) \) denote the relative support of \( \Phi \) for \( t_1, t_2 \).

For simplicity, we will often drop \((v_1, a_1)\) and \((v_2, a_2)\) as the supported tuples can be easily computed with this information.

Example 5. Consider the explanation from Figure 2a. The pattern \( \Phi_1 \) is found from \( \Omega_1 \) by CAJADE as the following tuple: \((\text{player: } \text{S.Curry}, \ast, \text{pts: } 23, \geq)\). Here player is a categorical attribute and pts is a numeric attribute (the other attributes are \( \ast \)), both coming from the PlayerGameScore table. Any tuple from the \( \mathcal{AP}T(\Omega_1, \Omega_1) \) which fulfills player = \( \text{S.Curry} \) and pts \( \geq 23 \) will be included in \( \text{match}(\Phi_1, \mathcal{AP}T(\Omega_1, \Omega_1)) \). Rephrasing the text box for \( UQ_1 \) in Figure 1f one explanation is: \((\Phi_1, \Omega_1, (58, 73), (21, 47))\).

It can be noted that the 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 quality measures that we discuss next.

2.5 Quality Measure of Explanations
First, we discuss the quality measures for explanations when the join graph is given, 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 the pattern in a good explanation to match as much provenance of \( t_1 \) as possible, and not match much in the provenance of \( t_2 \). For this purpose, we adapt the standard 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 \( \Omega \), 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 \( t'' \in \mathcal{AP}T(\Omega, \Omega, t_1) \) (ref. Definition 4) such that \( t'' \vdash \Phi \). The coverage of \( E \) on \( t_1, t' \) in \( \mathcal{PT}(Q, \Omega, D) \) is: \( \text{cov}(E, \Omega, t_1, t') = | \{ \text{match}(\Phi, \mathcal{AP}T(Q, \Omega, D, t_1, t')) \cap t' \} | \). \( | \{ \} | \) is the indicator function.

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

\[
\text{TP}(E, \Omega, t_1) = \sum_{t' \in \mathcal{PT}(Q(D), t_1)} \text{cov}(E, \Omega, t_1, t')
\]

(c) The false positive 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 \( t_2 \) (not appearing on the right-hand side here):

\[
\text{FP}(E, \Omega, t_1, t_2) = \sum_{t' \in \mathcal{PT}(Q(D), t_2)} \text{cov}(E, \Omega, t_1, t')
\]

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

\[
\text{FN}(E, \Omega, t_1) = \sum_{t' \in \mathcal{PT}(Q(D), t_1)} | \{ \text{match}(\Phi, \mathcal{AP}T(Q, \Omega, D, t_1, t')) \} | - \text{cov}(E, \Omega, t_1, t')
\]

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

\[
\text{Prec}(E, \Omega, t_1, t_2) = \frac{\text{TP}(E, \Omega, t_1)}{\text{TP}(E, \Omega, t_1) + \text{FP}(E, \Omega, t_1, t_2)}
\]

\[
\text{Rec}(E, \Omega, t_1) = \frac{\text{TP}(E, \Omega, t_1)}{\text{TP}(E, \Omega, t_1) + \text{FN}(E, \Omega, t_1)}
\]

\[
\text{Fscore}(E, \Omega, t_1, t_2) = \frac{2 \cdot \text{TP}(E, \Omega, t_1) \cdot \text{Rec}(E, \Omega, t_1)}{\text{TP}(E, \Omega, t_1) \cdot \text{Rec}(E, \Omega, t_1) + \text{FP}(E, \Omega, t_1, t_2)}
\]

A high recall implies that the pattern \( \Phi \) describes the tuples contributing to \( t_1 \) well. A high precision implies that \( \Phi \) 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 \( t' \in \mathcal{PT}(Q(D)) \) 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 examples and in Definition 6, an explanation \( E = (\Omega, \Phi, (v_1, a_1), (v_2, a_2)) \) includes the relative support of the pattern \( \Phi \) for \( t_1, t_2 \in Q(D) \) to illustrate how this pattern differentiates the output tuples \( t_1, t_2 \). Here \( v_1 = \text{TP}(\Phi, \Omega, t_1) \) and \( a_1 = \text{TP}(\Phi, \Omega, t_1) + \text{FN}(\Phi, \Omega, t_1) \).
as defined in Definition 7, denoting the set of tuples in the provenance of \( t_1 \) covered by the pattern \( \Phi \), 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 illustrate the difference with \( t_1 \).

**Finding Top-\( k \) Patterns with Highest F-scores.** Given a join graph \( \Omega \), our goal is to find the top-\( k \) patterns \( \Phi \) 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 \( \Omega \) has polynomial data complexity [48] (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 bounded by \( O(p^p) \) (the number of distinct attribute values is bounded by \( n = \) total number of tuples in the database, and each attribute can appear as don’t care \( * \) and with at most three comparison operators), so even a naive approach of computing the top-\( k \) patterns with the highest F-score values is polynomial in data size. However, this naive approach is not scalable in practice and therefore we adopt a number of heuristic optimizations to solve this problem as described in Section 3.

**Explanations over All Join Graphs.** When mining multiple join graphs \( \Omega \), 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 4).

## 3 MINING PATTERNS GIVEN A JOIN GRAPH

In this section, we give an overview of our algorithm for mining patterns from an augmented provenance table (APT) \( \mathcal{AP}T(\Omega, D, \Omega) \) generated based on a given join graph \( \Omega \). 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 a preprocessing step then we cluster attributes that are highly correlated to reduce redundancy in the generated patterns. The output of this preprocessing step are the generated clusters and one representative selected for each cluster. Then, we use random forests to determine the relevance of each attribute on predicting tuples to belong to the provenance of one of the two data points from the user’s question. The purpose of this step is to remove attributes that are unlikely to yield patterns of high quality. (ii) In the next phase of pattern mining we only consider categorical attributes and mine pattern candidates using a variation of the LCA (Least Common Ancestor) method from [19] that can only handle categorical attributes. From the set of patterns returned by the LCA method, we then select the \( k_{cat} \) patterns with the highest scores for the next step. (iii) These patterns are then refined by adding conditions on numerical attributes that can improve precision at the potential cost of reducing recall. (iv) The refined patterns are filtered to remove patterns with recall below a threshold \( \lambda_{recall} \) and are ranked by their score according to Definition 7. Finally, the top-\( k \) patterns according to this ranking are returned.

Before describing the individual steps of our pattern mining algorithm, we first need to introduce additional notation used in this section. Given a pattern \( \Phi \), we call a pattern \( \Phi' \) a refinement of \( \Phi \) and \( \Phi' \) can be derived from \( \Phi \) by replacing one or more placeholders \( * \) with comparisons. For instance, pattern \( \Phi_2 = ((X, =), (Y, \leq)) \) is a refinement of \( \Phi_1 = ((X, =), * ) \). The following observation holds:

**Proposition 3.1.** Using Definition 7, given a tuple \( t \in Q(D) \) and a join graph \( \Omega \), \( \text{Rec}(E_2, \Omega, t) \leq \text{Rec}(E_1, \Omega, t) \), where \( E_1 = (\Omega, \Phi_1), E_2 = (\Omega, \Phi_2) \), and \( \Phi_2 \) is a refinement of \( \Phi_1 \).

**Proof.** Following Definition 7 (a), if a \( t' \in \mathcal{PT}(Q, D, t) \) is covered by \( E_2 \), then it is also covered by \( E_1 \). Hence, by (b) and (d), \( TP(E_1, \Omega, t) \geq TP(E_2, \Omega, t) \) and \( FN(E_1, \Omega, t) \leq FN(E_2, \Omega, t) \). Therefore, using (e), \( \text{Rec}(E_2, \Omega, t) \leq \text{Rec}(E_1, \Omega, t) \).

### Algorithm 1 MineAPT to find top-\( k \) patterns given a join graph \( \Omega \)

**Other inputs:** Database \( D \), Query \( Q \). Answer tuples in user question \( t_1, t_2 \), limit on the number of categorical attributes in the pattern \( k_{cat} \), and input parameters \( \lambda \) as described in text.

**MineAPT (D, \Omega, Q, t_1, t_2, k, k_{cat}, \lambda_{sel-attr})**

1. \( E = \emptyset \) \* min-heap: to store top-\( k \) explanations by their score
2. \( APT = \mathcal{AP}T(\Omega, D) \)
3. \( S = \text{createSample}(APT) \)
4. \( (\Lambda_{num}, \Lambda_{cat}, \text{clusters}) = \text{filterAttrs}(APT, \lambda_{sel-attr}) \)
5. \( \Phi_{cat} = \text{LCA}(S, \Lambda_{cat}) \)
6. \( \text{top} = \text{pickTop}(\Phi_{cat}, k_{cat}, APT) \)
7. \( done = \emptyset \)
8. \( \text{while } \text{top} \neq \emptyset \)
9. \( \Phi = \text{pop}(\text{top}) \)
10. \( done = done \cup \{ \Phi \} \)
11. \( \text{for } \text{c}_{\text{cat}} \in \{ t_1, t_2 \} \)
12. \( \text{if recall}(\Phi, t_{\text{cat}}) > \lambda_{\text{recall}} \)
13. \( \text{if } F(\Phi, t_{\text{cat}}) > F(\Phi(\text{peek}(E)) \)
14. \( E = \text{deleteMin}(E) \)
15. \( E = \text{insert}(\text{E}, (\Phi, t_{\text{cat}})) \)
16. \( \text{for } A \in \Lambda_{num} \)
17. \( \text{if } \Phi = \ast \)
18. \( \text{for } \text{op} \in \{ =, \leq \}, c \in \text{getFragments}(\text{Dom(APT,A)} \)
19. \( \Phi_{\text{new}} = \Phi \)
20. \( \Phi_{\text{new}} = (\text{op}, c) \)
21. \( \text{if } \Phi_{\text{new}} \notin done \)
22. \( \text{top} = \text{top} \cup \{ \Phi_{\text{new}} \} \)
23. \( \text{return } E \)

**filterAttrs(APT, \lambda_{sel-attr})**

1. \( A = APT \)
2. \( \Phi_{\text{filtered}} = \text{filterBasedOnRelevance}(APT, \lambda_{sel-attr}) \)
3. \( \text{clusters} = \text{clusterAttributes}(\Phi_{\text{filtered}}, APT) \)
4. \( \Lambda_{repr} = \text{pickClusterRepresentatives}(\text{clusters}, APT) \)
5. \( \Lambda_{num} = \{ A \ | \ A \in \Lambda_{repr} \land A \text{ is numeric} \} \)
6. \( \Lambda_{cat} = \{ A \ | \ A \in \Lambda_{repr} \land A \text{ is categorical} \} \)
7. \( \text{return } (\Lambda_{num}, \Lambda_{cat}, \text{clusters}) \)
We exploit the above fact to exclude patterns and their refinements early on in the process if their recall is below a threshold.

### 3.1 Clustering and Filtering Attributes

Before mining patterns, we analyze the attributes of an APT to (1) determine the attributes that are unlikely to contribute to top-k patterns because they are not helpful in distinguishing between the two outputs of interested provided as part of the user’s question, and (2) to identify clusters of attributes with strong mutual correlations, because such attributes can lead to redundancy in explanations as discussed below.

**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 a pattern restricting 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 [44], a clustering algorithm closely related to principal component analysis and other dimensionality reduction techniques [46]. However, any technique that can cluster correlated attributes would be applicable.

**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 one of the two outputs from the user’s question [10]. We then rank attributes based on the relevance and find the fraction $\lambda_{\text{sel-attr}}$ of attributes with the highest relevance ($\lambda_{\text{sel-attr}}$ is a threshold used by the system). The rationale for this step is to avoid generating patterns involving attributes that are irrelevant for distinguishing the two output tuples $t_1, t_2$ in the user question. This reduces the search space for patterns and additionally has the advantage of excluding attributes that are mostly constant in the rows contributing to the two outputs. Such attributes can be added to any pattern with minimal effect on the recall and precision of patterns, since they essentially do not affect the matches for the patterns. This could mislead users into thinking that the value of this attribute is a distinguishing factor for the tuples in the user question when in fact the value of this attribute has very limited or no effect.

### 3.2 Patterns over Categorical Attributes

We then generate a sample of size $\lambda_{\text{pat-samp}}$ (an input parameter) from $\mathcal{APT}(Q, D, \Omega)$ and generate a set of candidate patterns over categorical attributes (ignoring all numerical attributes at this stage) using the LCA method from [19]. The LCA method generates pattern candidates from a sample by computing the cross product of the sample with itself. 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. Thus, 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 that only works for categorical attributes (equality comparisons) generates patterns that reflect common combinations of constants in the data. Note that we ignore numeric attributes at this step, i.e., we use $*$ for all numeric attributes. The rationale of focusing on categorical attributes first is that we can (i) use the established heuristic of the LCA method to generate pattern candidates from categorical attributes, and (ii) we can significantly reduce the search space by pruning all refinements of patterns that fail to have a sufficiently high recall.

### 3.3 Filtering Categorical Pattern Candidates

Next we calculate the recall for each pattern by filtering the input table to determine the matches of the pattern. As an optimization, we can calculate the recall over a sample of the data (using a sample size parameter $\lambda_{\text{quality-sample-rate}}$). This may require using a separate sample size ($k_{\text{cat}}$) 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. Irrespective of whether the recall is estimated or calculated precisely, we then filter out patterns whose recall is below a threshold $\lambda_{\text{recall}}$. Out of these patterns, we then pick the top patterns based on their recall. In the following we use $P_{\text{cat}}$ to denote the set of patterns that are returned by this step.

### 3.4 Refinement and Numeric Attributes

In the next step, we then generate refinements of patterns from $P_{\text{cat}}$ by replacing placeholders on numerical attributes with predicates. Even though such refinements can at best have the same recall as the pattern they originate from, their precision may be higher resulting in greater F-scores. Recall that for numerical attributes we allow for both equality as well as inequality predicates. Domains of numerical attributes are typically large, resulting in large number of possible constants to use in inequality comparisons. To reduce the size of the search space, we split the domain of each numerical attribute into a fixed number $\lambda_{\text{frac}}$ of fragments (e.g., quartiles) and only use boundaries of these fragments when generating refinements. For example, for $\lambda_{\text{frac}} = 3$ we would use the minimum, median, and maximum value of an attribute’s domain. We systematically enumerate all refinements of a pattern by extending it by one predicate at a time. For each such refinement we calculate its recall as described above. Patterns whose recall is below $\lambda_{\text{recall}}$ are not further refined. We use $P_{\text{refined}}$ to denote the union of $P_{\text{LCA}}$ with the set of patterns generated in this step.

### 3.5 Computing Top-k Patterns

Finally, we calculate the F-scores for each pattern in $P_{\text{refined}}$ and return $k$ patterns. To improve the diversity of the returned patterns we rank them based on a score that combines F-scores (Definition 7) with a diversity score $D(\Phi, \Phi')$ that measure how similar two patterns are. The first pattern to be returned is always the one with the highest F-score. Then we determine the $i + 1$th pattern based on this score. Note that the diversity score $D(\Phi)$ for a pattern $\Phi$ depends on the set $R$ of patterns we have selected to far, because it measure how close the pattern $\Phi$ is to the most similar pattern in $R$. We calculate this as a score $D(\Phi, \Phi')$ that ranges between -2 and 1 (larger scores means that the two patterns are more
dissimilar). The formula for calculating the score is shown below. For each attribute \( A \) of pattern \( \Phi \) we add 1 if the attribute does not appear in \( \Phi' \), we add a penalty (-0.3) if the attribute appears in both patterns (but with different constants), and a larger penalty (-2) if it appears in both patterns with the same constant. We repeatedly add the pattern with the highest score to \( R \) until we have \( k \) patterns to return.

\[
\text{wscore}(\Phi) = \text{F score}(E, \Omega, t_1, t_2) + \min \ \{D(\Phi, \Phi') : \Phi' \in R\} + \sum_{A, \Phi' \in R} \text{matchscore}(\Phi, \Phi', A)
\]

\[
\text{matchscore}(\Phi, \Phi', A) = \begin{cases} 1 & \Phi', A = \Phi \\ -0.3 & \Phi.A[0] = \Phi.A[0] \\ -2 & \Phi.A[0] \neq \Phi.A[0] \end{cases}
\]

We denote the result of this step as \( \mathcal{P}_{\text{top-}k} \).

### 4 JOIN GRAPH ENUMERATION

In this section, we describe an algorithm present in Algorithm 2 that enumerates join graphs of iteratively increasing size. In iteration \( i \), we enumerate all join graphs with \( i \) edges by adding a single edge conforming to the schema graph to one of the join graphs of size \( i - 1 \) produced in the previous iteration. The maximum size of join graphs considered by the algorithm is determined by parameter \( \lambda_{\text{edges}} \). 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 pattern mining can be significantly more expensive than just generating a join graph, so we skip pattern mining for join graphs that are either unlikely to yield patterns of high quality or for which generating the APT and patterns are likely to be expensive. For join graphs that pass these tests we materialize the corresponding APT and apply the pattern mining algorithm from Section 3 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 \( \Omega \) 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. The main entry point is function \text{ENUMERATEJOINGRAPHS} from Algorithm 2. We first initialize the result to contain \( \Omega_0 \), the join graph consisting of a single node labeled (PT). Within each iteration of the main loop (line 4), each join graph \( \Omega \) from the set of join graphs generated in the previous iteration (\( \Omega_{\text{prev}} \)) is passed to function \text{EXTENDJOING} which computes the set of all possible extensions of \( \Omega \) (line 7). For any such join graph produced in this step, we then check whether it should be considered for pattern mining using function described below (line 9). For any join graph passing this check we call \text{MINAPT} to generate the top-k patterns and store them in \( E \).

Function \text{EXTENDJOING} enumerates all possible extension of a join graph with one additional edge. For that we consider all existing nodes \( v \) in the join graph as possible extension points. We then enumerate all edges from the schema graph \((V_S, E_S, l_{\text{edge}})\) that are adjacent to nodes \( r \) for relations that are represented by \( v \). For that we have to distinguish two cases. If \( v \) is labeled PT, then \( r \) can be any relation accessed by the query. Otherwise, there is a single node \( r \) that is determined based on \( v \)'s label (the relation represented by \( v \)). For each edge \( e \) adjacent to a node \( r \) (line 9), we
then iterate over all conditions \( c \) from the label of \( e \) and then use function AddEdge to enumerate all join graphs generated from \( \Omega \) by connecting \( e \) with an edge labeled with \( c \) to another node (either already in the join graph or new node added to the graph).

Function AddEdge first generates a join graph by adding a new node \( e_{\text{new}} \) labeled \( \text{end} \) (the relation that is the end point of the edge in the schema graph) and connects it to \( \alpha \). Afterwards, for each node in the join graph \( \Omega \) labeled \( \text{end} \) that is not connected to \( \alpha \) though an edge labeled \( \text{cond} \) already, we generate a new join graph by adding such an edge.

Checking Join Graph Connectivity and Skipping Expensive APT Computations. We filter out join graphs based on lack of connectivity or on high estimated computation costs.

We use the function isVALID shown in Algorithm 2 to filter out join graphs based on lack of connectivity (described in the following) or based on high estimated computation costs.

Note that schema graphs may contain tables with multiple primary key attributes that are connected 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 in a particular game. Assume that there exists another table player that is not part of the running example. The primary key of this table consists of a foreign key to the game table and the name of the player for which we are recording stats. For this current example also assume that the schema graph permits that PlayerGameScoring to be joined with the Player table. Consider a query that joins the Game table with LineupPerGameState, LineupPlayer, and Games tables and selects games played by team GSW. A valid join graph for this query would be to join the node \( PT \) with PlayerGameScoring on the primary key of game. Note that while the result of the query contains rows that pair a player of GSW with one line up they played in 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 that because of their redundancy often lead to large APT tables, our algorithm checks that for every node in the join graph, the primary key attributes of the relation corresponding to 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 filter join graphs that are not fully connected, some generated join graphs will result in APTs of significant size which are expensive to materialize and have significantly high cost for pattern mining. Recall that we use queries to materialize join graphs. We use the DBMS to estimate the cost of this query upfront. We skip pattern mining for join graphs where the estimated cost of this query is above a threshold \( \lambda_{\text{cost}} \). While we may lose explanations by skipping such join graphs, experimental results demonstrate that it is necessary for reasonable performance to apply this check. Further, keeping the join graph size relatively small is likely not to overload the user with too complex explanations.

Ranking Results. After we have enumerated join graphs and have computed top-k patterns for each join graph, we rank the union of all pattern sets based on their F-score. We decided to rank patterns to reduce the load on the user by increasing the likelihood that good patterns are shown early, but without having the risk of completely filtering out patterns that have lower scores.

5 EXPERIMENTAL EVALUATION

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

Datasets. We used the same NBA and MIMIC dataset as described in Section 6. We created several scaled versions of these two datasets preserving the relative sizes of most tables and join results. In detail, subsets of the dataset were created by random sampling, preserving relative sizes of most tables and ensuring that the result sizes of joins are scaled appropriately. Similarly, for scaling up the dataset size we duplicate rows appending identifiers to primary key columns and other selected columns to ensure that the constraints of the schema are not violated and the join result sizes are scaled too. We use scale factors from 0.1 (∼ 17MB) up to 10.0 (∼ 1.7GB).

Figure 5 shows the schema of the NBA dataset. Primary key attributes are marked with a key symbol and foreign key attributes with a blue arrow on the right of an attribute’s name.

Game. Table game stores information about games including the date when the game took place (game_date), the number points scored by the home and visiting team (home_points and away_points), the number of ball possessions by each team (home_possessions and away_possessions), the ids of the home, visiting, and winning team (home_id, away_id, winner_id), and the season when the team took place (season_id). Games are uniquely identified by their date and the home team.

team. Table team stores an artificial key (team_id and the name (team) of NBA teams.

player. Table player records the name (player_name) and artificial identifiers (player_id) of NBA players.

player_salary. Table player_salary stores the salary a player is earning in a particular season.

play_for. Table play_for stores which player played for which team for which time period (start_date to end_date).

line_up and lineup_player. Table lineup records lineups. Each lineup is a list of 5 players (table lineup_player) from a team that are together on the field during a game.

team_game_stats. This table stores statistics related to the performance of a team in a particular game. The following statistics are reported. The number of points scored by the team (points), offensive possessions (offposs), number of field goals made (fg_two_m), two point field goal percentage (fg_two_pct), three point field goal made (fg_three_m), three point percentage (fg_three_pct), team assists total (assists), team total rebounds (rebounds), team defensive rebounds (defrebounds), team offensive rebounds (offrebounds). We used this simplified version of team_game_stats table in user study. In experiments we used a richer number of columns for this
Figure 5: Schema Graph for NBA

table and because of the space constraint we only report the list of their names here: fg_two_a, fg_three_a, nonheavefg_three_pct, ftpoints, ptsassisted_two_s, ptsunassisted_two_s, ptsassisted_three_s, ptsunassisted_three_s, assisted_two_spct, nonputbacksassisted_two_spct, assisted_three_spct, fg_three_apct, shotqualityavg, efgpqct, tspct, ptsputbacks, fg_two_ablocked, fg_two_apctblocked, fg_three_ablocked, fg_three_apctblocked, assistpoints, two_ptassists, three_ptassists, atrimassists, shortmidrangeassists, longmidrangeassists, corner_three_assists, arc_three_assists,-ftdefrebounds, defftrebounds, def_two_ptrebounds, def_three_ptrebounds, def_two_ptrebounds, def_three_ptrebounds, defoffrebounds, offftrebounds, off_two_ptrebounds, off_two_ptrebounds, off_three_ptrrebounds, off_three_ptrrebounds, off_four_ptrrebounds, off_four_ptrrebounds, defatrimreboundpct, defshortmidrangereboundpct, deflongmidran
lineup_game_stats. This table stores statistics about the performance of a lineup in a game. Total number of minutes the lineup played in the game (mp), total number of possessions this lineup had in the game (mposs), total number of opponent lineup possessions in the game (oppo_mposs).

player_game_stats. This table records statistics about a player’s performance in a game. The most of the attributes contained in this table has same names and meanings as those in team_game_stats except it has a unique attribute usage which describes what percentage of team plays a player was involved in while he was on the floor. Similarly, we used this simplified table in user study and richer version (please refer to team_game_stats for details) in the experiments.

Figure 6 shows the schema of the MIMIC dataset.

admissions. This table stores information about hospital admissions. hadm_id is a unique identifier for an admission. admittime is the time when the patient got admitted. discharge_time is the time when the patient was discharged from the hospital. admission_type describes the type of the admission: ELECTIVE, URGENT, NEW-BORN or EMERGENCY. admission_location provides information about the previous location of the patient prior to arriving at the hospital. discharge_location provides information about the location of the patient after visiting the hospital. insurance is the type of insurance the patient has. martial_status is the patient’s marital status. edregtime is the time that the patient was registered in the emergency department. edouttime is the time that the patient was discharged from the emergency department. diagnosis provides a preliminary, free text diagnosis for the patient. dod is the date of death for the given admission. hospital_expire_flag indicates whether the patient died within the given hospitalization. 1 indicates death in the hospital, and 0 indicates survival to hospital discharge. hospital_stay is the length of the patient’s stay in the hospital in days.

procedures. Table procedures records information of medical procedures for each patient. seq_num is part of the primary key in procedure table to differentiating one procedure from another during one admission for the same patient. icd9_code is ICD-9 Codes to represent the procedure.

patients. This table records information about patients. subject_id is a unique identifier for a patient. For each patient we record their gender and date of birth (dob). dodis the date of death for the given patient. dod_hosp is the date of death as recorded in the hospital database. dod_ssn is the date of death from the social security database.

patients_admit_info. This table records additional information about patients at the time of an hospital admission. For each hospital admission the patient’s age, language of choice, religion, and ethnicity are recorded.

icustays. This table records information about intensive care unit (ICU) stays of a patient during an admission. A patient may stay multiple times in ICU during an admission. dbsource: ‘carevue’ indicates the record was sourced from CareVue, while metavision indicated the record was sourced from Metavision.

first_careunit and last_careunit contain, respectively, the first and last ICU type in which the patient was cared for. first_wardid and last_wardid contain the first and last ICU unit in which the patient stayed. intime provides the date and time the patient was transferred into the ICU. outtime provides the date and time the patient was transferred out of the ICU. los is the length of the patient’s stay in intensive care. los_group is a categorized length of stay where it is divided into 5 groups.

diagnoses. This table stores information about diagnosis for patients for a particular admission. seq_num is part of the primary key in procedure table to differentiating one diagnosis from another during one admission for the same patient. icd9_code is ICD-9 Diagnoses Codes to represent the procedure.

We first evaluate how the maximum number of join graphs affects our approach. We compare our approach without feature selection as discussed in Section 3 (Naive) against with feature selection (opt). For opt we also vary the sample rate for F-score calculation (λF1–Samp)

| Parameter          | Description                              | Default |
|--------------------|------------------------------------------|---------|
| sub-nice           | the size of the database (scale factor)   | 10      |
| λedges             | maximum number of edges per join graph   | 3       |
| #attributes        | attributes returned by feature selection  | 3       |
| opt_num            | max number of numerical attributes allowed in a pattern | 3 |
| λp–samp            | sample rate for LCA pattern generation    | 0.1     |
| optim             | sample rate for calculating F-scores of patterns | 0.1     |

Table 1: Parameters of our approach and default values

5.1 Feature Selection

In this set of experiments, we evaluate how feature selection affects our approach. We compare our approach without feature selection, and we also vary the sample rate for F-score calculation (λF1–Samp)

5.2 Join Graph Size

We first evaluate how the maximum number of join graphs affects performance by varying λedges from 1 up to 3. We compare the runtime of our algorithm with feature selection varying the sample rate for F-score calculation (λF1–Samp = {0.1, 0.3, 0.5, 1.0}). We use query Q1 and user question (NBA dataset) from the running example in the introduction. The results of this experiment are shown in

Figure 7 (Figure 7a) shows the runtime breakdown for MIMIC (NBA).
Figure 6: Schema Graph for MIMIC

Figure 7: Feature sel.: runtime in sec. (MIMIC, $\lambda_{edges} = 3$)

Figure 8: As expected runtime is increases significantly in $\lambda_{edges}$. because the number of join graphs to be considered increases dramatically when we allow for more edges per join graph. Sampling for F-score calculation improves performance by up to $\sim 50\%$ for $\lambda_{edges} > 1$ when $\lambda_{F1-samp}$ is set to $10\%$.

5.3 Scalability
To evaluate the scalability of our approach, we use scaled versions of the NBA and MIMIC datasets ranging ($\sim 10\%$ to $\sim 800\%$). We varied the F-score sample rate ($\lambda_{F1-samp}$) from 0.1 to 0.7. The results are shown in Figure 9 comparing against linear scaling (black
Figure 9: Scalability in database size.

The effect of database size on runtime is similar for both 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 60% (70%) faster than $\lambda_{F1-samp} = 0.7$ for scale factor 8 on the NBA (MIMIC) dataset. A detailed breakdown of the runtime of the individual steps of our algorithm for $\lambda_{F1-samp} = 0.7$ is shown in Figure 9c and Figure 9d. Based on the results of our study of how sampling rate for pattern generation ($\lambda_{pat-samp}$) affects performance and quality presented in Section 5.4 (see Figure 10b to Figure 10e), we capped the number of rows sampled for LCA at 1000. Recall that we measure the following steps of our approach: (i) we apply feature selection to determine which attributes to use for explanations (Feature Selection); (ii) we use the LCA method to generate candidate patterns for an APT (Gen. Pat. Cand.); (iii) we materialize APTs (Materialize APTs); (iv) we create a sample of APTs (Sampling for F1) and then calculate F-scores of patterns using this sample (F-score Calc.), (v) in the refinement step we create patterns generated over categorical attributes by adding numerical attributes (Refine Patterns). The major contributing factor for larger database sizes is F-score calculation which makes up more than 50% of the runtime. The step with the largest growth rate is sampling for F-score calculations for the NBA dataset which is $\sim$ 183 times slower for scale factor 8 compared to scale factor 0.1. Based on these results, using lower sample rates for F-score calculation is preferable for larger database sizes.

5.4 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 a database. This approach computes a cross product of a sample with itself. 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: $\Omega_2$ and $\Omega_3$ for $Q_1$, and $\Omega_3$ and $\Omega_4$ for $Q_{mimic4}$. Figure 10a shows the number of rows and attributes for the APTs, and join graph structure for each of these join graphs. The results are shown in Figure 10b to Figure 10e. We measure pattern quality as the number of patterns from the top-10 computed over the full dataset that occur in the top-10 computed...
based on a sample (see the blue lines labeled match). For each materialized join graph, we also measure the runtime of generating the top-10 patterns. As expected because of the cross product computed over the sample, runtime increases quadratically in the sample size. For Figure 10d and Figure 10e, all ground truth top-10 patterns are found even for just 3% sample rate. Whereas as shown in Figure 10c, even for 20% sample rate (13000 rows), we only find one matching pattern. The reason behind the different result observed in Figure 10d is that one of the columns in \( \Omega_2 \) has over 800 distinct values that are roughly evenly distributed and, thus, the recall-based ranking is sensitive to small variations in frequency caused by sampling. For fig. 10b, 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 \( \lambda_{\text{pat-samp}} = 0.1 \) for the rest of the experiments and set a cap number of rows in the sample as 1000.

**Sampling for F-score Calculation.** We also use sampling to reduce the cost of the quality measure calculation (parameter \( \lambda_{\text{f1-samp}} \)). 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 10 shows the running time and the quality of patterns when varying the sample rate and maximum number of edges in join graphs (\( \lambda_{\text{edges}} \)) for queries Q1 and Qmimic. We use the normalized discounted cumulative gain (NDCG) [24] as the sample quality metric, which is often used in information retrieval for evaluating the ranking results of recommendation and search engine algorithms. 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 10f shows that for both datasets for \( \lambda_{\text{edges}} = 1 \), the similarity between the sampled result and result over the full dataset is high, even for aggressive sampling. For larger join graphs (\( \lambda_{\text{edges}} = 2, 3 \)), the NDCG score fluctuates around 0.7 for the NBA dataset. For the MIMIC dataset, the NDCG converges to ~ 1.0 at a sampling rate of 0.5. Overall, even for low sample rates, the NDCG score is at least ~ 0.6 (~ 0.8) for the NBA (MIMIC) dataset. We also evaluate the number of patterns from sampling that are present in the results without sampling, for which we used recall as the metric as shown in Figure 10g.

### 5.5 Comparison with Explanation Tables

We also compared our approach against the approach from [19] (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 Q1 and the corresponding user question from the introduction. The corresponding APT has ~ 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 12 lists the runtime of CAJADE and ET after applying feature selection. As a qualitative comparison, we list the first 20 patterns returned by ET in Appendix A.1. While slower for a sample size of 16 our approach scales much better when increasing the sample size (~ 50x faster for sample size 512). That being said, we would like to point out the major contribution of our work 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.

### 5.6 Comparison with CAPE

We compared our approach against CAPE ([34]). The question we used are from NBA running example Q1, which asks question about number of wins for GSW over the years and \( Q_{\text{bab}} \) from the case study which asks for 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 \( U_{\text{Qcape}} \) for Q1: *Why was GSW number of wins high in 2015-16 season?* and \( U_{\text{Qcape}} \) for \( Q_{\text{bab}} \): *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 (\( U_{\text{Qcape}} \)) and PT – team_game_stats (\( U_{\text{Qcape}} \)). Figure 13 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 counter-balances too.

5.7 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 2. We designed these queries to access different relations and use different group-by attributes. The SQL code for these queries is shown below. All queries were run with $A_{Fi-samp} = 0.3$ and $A_{edges} = 3$. The results are shown in Figure 12. 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 12).

**SQL code for Query Workload.** Note that the queries are based on the original schema graph which are more complex than the ones presented in the running example. The real schema can be found in Figure 6 and Figure 5.

**NBA.** $Q_{w1}$. The average points change over the years for player Draymond Green.

```sql
SELECT AVG(points) as avp_pts, s.season_name
FROM player p, player_game_stats pgs, game g, season s
WHERE p.player_id=pgs.player_id AND
    g.game_date = pgs.game_date AND
    s.season_id = g.season_id
AND p.player_name="Draymond_Green"
GROUP BY s.season_name
```

$Q_{w2}$. GSW average assists over the years

```sql
SELECT AVG(tgs.assists) as avgast, s.season_name
FROM teamGameStats tgs, game g, team t, season s
WHERE s.season_id = g.season_id AND
    tgs.game_date = g.game_date AND
    tgs.home_id=g.home_id AND
    tgs.team_id = t.team_id AND
    t.team='GSW'
GROUP BY s.season_name
```

$Q_{w3}$. Average points for LeBron James over the years

```sql
SELECT AVG(points) AS avp_pts, s.season_name
FROM player p, player_game_stats pgs, game g, season s
WHERE p.player_id=pgs.player_id AND
    g.game_date = pgs.game_date AND
    g.home_id = pgs.home_id AND
    s.season_id = g.season_id
AND p.player_name='LeBron_James'
GROUP BY s.season_name
```

$Q_{w4}$. GSW wins over the years, but we used different 2 seasons

```sql
SELECT COUNT(*) AS win, s.season_name
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
```

$Q_{w5}$. Average points by Jimmy Butler over the years

```sql
SELECT AVG(points) AS avp_pts, s.season_name
FROM player p, player_game_stats pgs, game g, season s
WHERE p.player_id=pgs.player_id AND
    g.game_date = pgs.game_date AND
    g.home_id = pgs.home_id AND
    s.season_id = g.season_id
AND p.player_name='Jimmy_Butler'
GROUP BY s.season_name
```

**MIMIC.** $Q_{w6}$. Return the number of diagnosis by chapter. A chapter is a type of diagnosis.

```sql
SELECT count(*) AS cnt, chapter
FROM diagnoses
GROUP BY chapter
```

$Q_{w7}$. Returns the death rate of patients grouped by their insurance.

```sql
SELECT insurance, 1.0 * sum(hospital_expire_flag) / count(*) AS death_rate
FROM admissions
GROUP BY insurance
```

$Q_{w8}$. Number of ICU stays grouped by the length of stays (los_group).

```sql
SELECT count(*) AS cnt, los_group
FROM icustays
GROUP BY los_group
```

$Q_{w9}$. Number of procedures for a particular chapter (group of diagnosis types).

```sql
SELECT count(*) AS cnt, chapter
FROM procedures
GROUP BY chapter
```

$Q_{w10}$. Number of admits per patient ethnicity.

```sql
SELECT count(*) AS cnt, ethnicity
FROM patients_admit_info
GROUP BY ethnicity
```

6 QUALITATIVE EVALUATION

We now evaluate the quality of explanations produced by CAJADE using case studies on the same two real datasets (NBA and MIMIC) as in Section 5. We report the SQL code of queries, query results, user questions, and returned explanations for both datasets. We also report results of a user study with the NBA dataset. For both case studies, we report the top-3 explanations for each query and user question in Table 4 (NBA) and Table 6 (MIMIC). 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 "[t1]" or "[t2]" in explanations as identifier of the primary tuple for the explanation. The full sets of top-20 explanations (including join nodes and edge details) are in Appendix A.2.
Table 3: NBA queries

| Query | Top explanations | F-score |
|-------|------------------|---------|
| Q\text{\textsubscript{GSW1}} | Average points per year for Draymond Green | s.season_id = g.season_id |
| Q\text{\textsubscript{GWS1}} | GSW average assists over the years | player, player_game_stats, game, season |
| Q\text{\textsubscript{JBU1}} | Average points per year for Jimmy Butler | player, player_game_stats, game, season |

Table 4: Queries, user questions and explanations (NBA)

6.1 Case Study: NBA

Setup. For the NBA dataset, we use five queries calculating player’s and team’s stats and generated user questions based on interesting results. Table 4 shows the user questions, queries, and top-3 explanations produced by our method for these user questions. Table 3 shows a description for each these queries, which include group-by aggregations over path joins. We present the SQL code for these query below. Their results and the tuples used in the user question (highlighted rows) are shown in Figure 14.

Q\text{\textsubscript{GGSW1}}: The average points change over the years for player Draymond Green.

\begin{verbatim}
SELECT AVG(points) AS avp_pts, s.season_name FROM player p, player_game_stats pgs, game g, season s WHERE p.player_id=pgs.player_id AND g.game_date = pgs.game_date AND g.home_id = pgs.home_id AND s.season_id = g.season_id AND p.player_name='Draymond Green' GROUP BY s.season_name
\end{verbatim}

Q\text{\textsubscript{GGSW2}}: GSW average assists over the years

\begin{verbatim}
SELECT AVG(tgs.assist) AS avgast, s.season_name FROM team_game_stats tgs, game g, team t, season s WHERE s.season_id = g.season_id AND tgs.game_date = g.game_date AND tgs.home_id=g.home_id AND tgs.team_id = t.team_id AND t.team='GSW' GROUP BY s.season_name
\end{verbatim}

Explanations and Analysis. Q\text{\textsubscript{GGSW1}} Draymond Green had a big average points difference between 2 consecutive seasons. All 3 explanation contains 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 lower salary. 2\textsuperscript{nd} and 3\textsuperscript{rd} explanation successfully find key game related
factors deciding the player’s points such as minutes played and shooting percentage (e.g., in 2nd, Green had more games where he played more than 31 minutes and shooting percentage higher than 0.4 in 2015 – 16 season). Q_{nba2}. The GSW team had a sudden increase in average assists. All explanations contain assist points which have a cause-and-effect relationship with assists (more assists result in more assist points). Q_{nba3} and Q_{nba5}. Both players had some significant average point changes. For Q_{wb}, Lebron James had an average points decrease. This occurred when he switched to a new team (from CLE to MIA and had less pressure offensively in the following year. CAJADE successfully identified this fact as a potential cause (2nd and 3rd). Q_{nba5}. Jimmy Butler had a big improvement in average points. Our top explanations to this improvement include an increase of usage and minutes played. Q_{nba4}. This query is similar to our running example Q1 but with a question asking for different 2 seasons. The explanations change player names (2nd, Andre Iguodala only played for GSW in 2015 – 17 season) as well as the team’s points difference and 3-point percentage (3rd). Note that while the first explanation has a high F-score, if we look at the join graph details, the salary and player constants can have no relation with GSW at all. This highlights the importance of making join graphs part of explanations.

6.2 Case Study: MIMIC

We constructed 5 queries over the MIMIC dataset accessing different tables. The simplified descriptions of the queries, user questions, and explanations Table 6. To help the reader understand the queries and explanations, we first briefly introduce the MIMIC dataset using the example below.

Example 6. Consider the (simplified) MIMIC-III Critical Care database [26] with the following relations (the keys are underlined).

- Admissions(adid, dischargeloc, adtype, insurance, isdead, HospitalStayLength) contains information about hospital admissions.
- Diagnoses(pid, adid, did, diagnosis, category) records the diagnoses for each patient (identified by pid) during each admission, one patient could have multiple diagnoses during one admission, which are identified by id.
- PatientsAdmissionInfo(pid, admid, age, religion, ethnicity) records the information from a patient upon admission to the hospital. Note that one patient could have multiple entries in this table because one patient could have multiple admissions during their lifetime.
- ICUStays(pid, admid, iid, staylength, icutype) records the information of the ICU stays patients. iid identifies different ICU stays within one admission. Suppose a data analyst writes the following query to find out the relationship between the insurance type and the death rate:

\[
Q_2 = \text{SELECT } \text{insurance, sum(isdead) / count(*) AS death_rate, count(*) AS admit_cnt } \\
\text{FROM Admissions GROUP BY insurance}
\]

Given the result of this query in Figure 15a, the analyst may ask UQ_2: given similar numbers of admissions, why is the death rate of patients using insurance=Medicare more than 2 times larger (t1) than that of patients with insurance=Private. Table 6 lists the top-3 explanations returned by CAJADE (referred to as Q_{mimic4}), which shows that among the deaths, Medicare has a larger fraction of admissions in emergency and male older patients (age < 71) compared to Private. This is aligned with the fact that Medicare is mostly for patients who are over 65 years old. 2nd and 3rd are stating that patients using Medicare has more admission because of emergency and also facts about length of hospital stays.

Table 6 gives a description for each of these queries, which include group-by aggregations over path joins. We present the SQL code of these queries below. Their results and the tuples used in the user question (highlighted rows) are shown in Figure 16.

Table 5: MIMIC queries

| User question | Q_{mimic1} | Description |
|---------------|------------|-------------|
| User question | Q_{mimic2} | Count of diagnoses over different chapters. |
| User question | Q_{mimic3} | Death rate of patients grouped by their insurance. |
| User question | Q_{mimic4} | Length of stays in ICU group by length of stays (los_group). |
| User question | Q_{mimic5} | Death rate of patients grouped by their insurance. |

| User question | Q_{mimic6} | Number of procedures group by ethnicity. |
|---------------|------------|-------------|
| User question | Q_{mimic7} | Given close number of admissions why do patients using Medicare insurance have higher death rates (t1, 14%) compared to patients using Private insurance (t2, 6%)? |

\[
Q_{mimic1}. \text{Count of diagnoses over different chapters.}
\]

User question: chapter=2 (neoplasms) VS chapter=13 (diseases of the musculoskeletal system and connective tissue)

\[
\text{SELECT } 1.\times\text{SUM(admissions.a, diagnoses.d WHERE a.hadm_id=d.hadm_id GROUP BY d.chapter)}
\]

\[
Q_{mimic2}. \text{Death rate of patients grouped by their insurance.}
\]

User question: insurance=‘Self Pay’ VS insurance=‘Private’

\[
\text{SELECT } \text{insurance, 1.\times\text{SUM(los_group)}\times\text{COUNT(*) AS death_rate FROM admissions GROUP BY insurance}}
\]

\[
Q_{mimic3}. \text{Length of stays in ICU group by length of stays (los_group).}
\]

User question: los_group>=4 VS los_group=’0-1’

\[
\text{SELECT COUNT(*) AS cnt, los_group FROM icustays GROUP BY los_group;}
\]

\[
Q_{mimic4}. \text{Death rate of patients grouped by their insurance.}
\]

User question: insurance=‘Medicare’ VS insurance=‘Private’

\[
\text{SELECT } \text{insurance, 1.\times\text{SUM(los_group)}\times\text{COUNT(*) AS death_rate FROM admissions GROUP BY insurance}}
\]

\[
Q_{mimic5}. \text{Number of procedures group by ethnicity.}
\]

User question: ethnicity=‘ASIAN’ VS ethnicity=‘HISPANIC’

\[
\text{SELECT COUNT(*) AS cnt, pai.ethnicity FROM patients_admit_info pai, procedures p WHERE p.hadm_id=pai.hadm_id AND p.subject_id=pai.subject_id GROUP BY pai.ethnicity}
\]
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 in different areas of computer science. 13 of them have some prior experience with SQL, and 5 were NBA fans.

**Tasks.** We first presented background knowledge on the NBA to each participant, and explained the schema of the dataset. Each participant was shown the SQL query $Q_1^t$ (shown below) and the SQL query $Q_3$ (shown below) and had to answer the question: "Do the patients in the query $Q_1^t$ have a higher death rate than the patients in the query $Q_3$?" Each participant was also asked a set of questions about the SQL query $Q_1^i$ (shown below) and the SQL query $Q_3$ (shown below).

**Explanations and Analysis.** Table 6 shows the top-3 explanations for returned by CajaDE for each user question. $Q_{mimic2}$. 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 1st explanation uses $\text{expire_flag}=1$ from patient table suggesting that this patient has passed away. This flag only indicates that the patient died, but not whether inside or outside the hospital, subsuming all hospital deaths. The 2nd and 3rd explanation add additional information about the lengths of hospital stays which could help user imply the severity differences between two categories thus infer the possible reason for different death rates. $Q_{mimic2}$. This query asks about differences between the death rate of patients based on their insurance. The 1st explanation states that there are more emergency admissions with Medicare than with Medicaid which may explain the higher death rate. The 2nd explanation related death rate to the $\text{expire_flag}$. The 3rd explanation suggests that Medicare has more Male patients than Medicaid. $Q_{mimic5}$. The 1st explanation shows that most of the patients staying over 8 days in ICU will stay in hospital for more than 9 days and also have procedure that is from chapter 16 (Miscellaneous Diagnostic and Therapeutic Procedures). The 2nd and 3rd explanations are both related to the hospital stay length. The 2nd 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 3rd explanation states the same facts for patients that have more than 8 days of hospital stay 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 ICU stay lengths $Q_{mimic5}$. This question uses the same query as $Q_{mimic2}$, but compares Private insurance with Medicare. The 1st 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 from online resources). The 2nd and 3rd explanations are stating that patients using Medicare are more likely to be admitted because of an emergency and also facts about length of hospital stays. $Q_{mimic3}$. The 1st explanation states that Asian patients that had a procedure are more likely to stay less than 19 days in the hospital. The 2nd explanation explains that compared to Asian patients there were more Hispanic patients younger than 66 years old and stayed more than 5 days in the hospital. The 3rd explanation points out that more Hispanic patients are Catholic. Note that the ethnicity information appeared in explanations are not from PT, but from a different patient meta info table. Because we do not currently consider functional dependencies, results like this cannot be avoided. We plan to address this in future work.

### Table 6: Queries, user questions and explanations (MIMIC)

| Query | User question | Top-3 explanations | F-score |
|-------|---------------|---------------------|---------|
| $Q_{mimic1}$ | Patient death rate grouped by diagnoses: 0.19 for chapter 15 VS 0.09 for chapter 13 | Hospital_stay_length $< 19$ & expire_flag $= 1$ | 0.88 |
| $Q_{mimic2}$ | Death rate by insurance: Medicare=0.13 VS Medicaid=0.066 | expire_flag $= 1$ | 0.7 |
| $Q_{mimic3}$ | Number of patients grouped by ICU stays length: less than 1 day VS more than 8 days | expire_flag $= 1$, procedure.chapter $= 16$ | 0.94 |
| $Q_{mimic4}$ | Death rate by insurance: Medicare=0.14 VS Private=0.06 | expire_flag $= 1$ | 0.6 |
| $Q_{mimic5}$ | Number of patients that did a procedure grouped by ethnicity: 7821 Hispanic patients VS 6247 Asian patients | expire_flag $= 1$, hospital_stay_length $< 19$ | 0.89 |
In season 2015-16, among the games GSW won, they were the visiting team and had points > 104 in 28 games (16 games in 2012-13, resp.).

SELECT
In 2015-16 season, GSW scored more than 104 points in each of 64 games (24 games in 2012-13, resp.) GSW won.

In 2015-16 season, the number of games with GSW player Draymond Green’s usage < 38 and minutes < 9.5, and possessions > 101 in 5 games (2 games in 2012-13, resp.) GSW won.

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.

We gave each participant 20 minutes to explore the dataset and manually find explanations. For those unfamiliar with SQL, we skipped this step. Participants then were asked to rate each of the top-5 explanations with the highest F-scores produced based on provenance and by CAJADE (Table 7) using a scale from 1 to 5. We also asked them which set of explanations makes more sense and 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) [24] for the entire ranked list. As shown in Table 9, ranking by precision gives the lowest pairwise ranking error for the provenance-based method, while for our method it is ranking by F-score. If we drop the most controversial explanation, the pairwise error is reduced by more than half. The NDCG$_5$ for CAJADE reaches 0.9 for all cases and even 0.95 after dropping the most controversial explanation.

Table 8: Average ratings for each explanation by users with different expertise and the measures for each explanation by CAJADE.

| C/A.JADE | Expl1 | Expl2 | Expl3 | Expl4 | Expl5 |
|----------|-------|-------|-------|-------|-------|
| Expl1    | 0.89  | 0.56  | 0.36  | 0.9  | 0.4  |
| Expl2    | 0.18  | 1.0  | 0.23  | 0.07 | 0.4  |
| Expl3    | 0.74  | 0.61  | 0.1  | 0.73 | 0.4  |

Table 9: Ranking quality: all 5 explanations (All), dropping the explanation with the largest standard deviation (-1).

To answer (S2), we find that explanations with high user ratings (Expl 3, 4, and Expl 6, 7, 9) have a positive relation with high F-score and precision. The only exception, Expl8 of CAJADE, 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 Expl8 left GSW in 2013, and people may or may not regard this as a signal that he had been relying more on other players who play a similar position to Jack. 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) [24] for the entire ranked list. As shown in Table 9, ranking by precision gives the lowest pairwise ranking error for the provenance-based method, while for our method it is ranking by F-score. If we drop the most controversial explanation, the pairwise error is reduced by more than half. The NDCG$_1$ 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 our method, 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 are consistent with participants’ preference; (3) for both methods, there can be top explanations rated low by participants, which is as 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 our method than participants without domain knowledge.

Other findings and discussion. Finally, it is also worth noting that the participants’ feedback supports 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 explanation. 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”.

7 RELATED WORK

Provenance and summarization. Provenance [14] for relational queries refers to records of the inputs that contribute to the results of a query, which has been studied extensively. For non-aggregate queries, why-provenance [11] returns a set of input tuples responsible for a given output tuple; how-provenance [22] encodes how the query combined input tuples to generate the answers. For aggregate queries, symbolic representation of semiring [4] is used to express how aggregate results are computed. Given the significant cost of managing provenance information in practical DBMS, different provenance-management frameworks to store and retrieve relevant provenance information have been proposed in the literature [6, 13, 20, 31, 37]. Some of them support provenance for aggregate queries using simplified models and query plan optimizations [27, 31, 37]. A number of recent papers have proposed summarization techniques to represent provenance approximately [2, 32, 33, 39], or using summarization rules for better usability [3]. Factorized and summarized provenance in natural language [16, 17] has also been studied for ease of user comprehension.

Data summarization. There is a line of work on summarizing relational data where informative summarization is provided with focuses on relevance, diversity, and coverage [25, 29, 38, 51]. For a relation augmented with a binary outcome attribute, Gebaly et al. [19] developed solutions to find optimally-informative summarizations of categorical attributes affecting the outcome attribute only considering the provenance (and not other relevant relations like our work), while its extension [49] considers numeric data. We adopt one of the optimizations in [19], the use of lowest common ancestor (LCA) patterns in our algorithms to prune the search space, to generate pattern candidates from a sample to avoid the costly computation of the data cube of the sample. We discussed the potential problems in adapting the approach in [19] using a materialized augmented provenance table in our experiments.

Explanations for query answers. This line of work aims at explaining unexpected outcomes in a query result, 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, 52] by characterizing a set of tuples whose removal or modification would affect the query answer of interest; while query-based explanations, i.e., changes to queries, are investigated for “why” and “why-not” questions [9, 12]. Explanations for outliers have been studied in [8, 34]. We share with [34] the motivation of considering explanations that are not (solely) based on provenance. The difference is that in [34], the same table used in the query is considered for finding explanations that “counterbalance” an outlier by learning patterns and balancing a low (high) outlier with a high (low) outlier w.r.t. a pattern, whereas we find explanations in “augmented provenance” stemming from tables not used in the query. Therefore, [34] is orthogonal to our work.

Join path discovery. Join path discovery has been studied for finding data related to a table of interest based on inclusion dependencies or string similarity [18, 23, 45, 53, 54]. Recently, [15, 30, 46] studied the performance of machine learning models trained on join results. CAJADE can utilize join graph discovery techniques to find more augmentation opportunities.

8 DISCUSSIONS AND FUTURE WORK

Explanations for database query answers is a relatively new research topic with lots of interesting future directions. For instance, currently our approach only considers correlations. In the future, we plan to integrate it with existing observational causal analysis framework from AI and Statistics [36, 43] to find causal explanations. Another interesting direction for future work is to integrate context-based explanations with join discovery techniques (e.g., [18, 53]) 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 (with nested sub-queries, negation) if we have access to a provenance system that can compute the query provenance. Beyond having an intuitive scoring function to rank explanations that may not always produce meaningful explanations, a challenging future work is to evaluate the correctness of provided explanations without much human interventions, to evaluate whether the returned explanations consider the intent of the user in the ‘why’ question, and to have a confidence score on the explanations by deciding whether the data has enough information to explain a user question. Finally, one can explore other types of user questions than our type of comparison question (like explaining an increasing/decreasing trend or explain why two results are similar) and applicability of context/provenance in ML applications.
A APPENDIX

A.1 Top 20 patterns returned by Explanation Table

Note since ET doesn’t accept numeric attributes, we did a preprocessing step by converting numeric values into categorical value. As shown in Appendix A.1, the large number of predicates make the results hard comprehend. Plus, since ET is mainly focused on maximizing information gain, the result should be interpreted as a set, which make the result even harder to understand.

| Num | Pattern Description |
|-----|---------------------|
| Pattern 1 | fg_three_pct ∈ [0, 0.08) ∧ off_three_prereboun_point ∈ [0, 0.08) |
| Pattern 2 | fg_three_pct ∈ [0, 0.08) ∧ off_three_prereboun_point ∈ [0, 0.08) |
| Pattern 3 | rebounds ∈ [1, 4] ∧ off_three_prereboun_point ∈ [0, 0.08) |
| Pattern 4 | fg_three_pct ∈ [0, 0.08) ∧ three_plasist ∈ [0, 0.08) |
| Pattern 5 | fg_three_pct ∈ [0, 0.08) ∧ two_blocked ∈ [0, 0.08) |
| Pattern 6 | prov_game_away__points ∈ [96, 104] ∧ three_plasist ∈ [0, 0.08) |
| Pattern 7 | player_name = Raymond Green |
| Pattern 8 | minutes ∈ [31.78, 49.63] ∧ fg_three_pct ∈ [0, 0.08) |
| Pattern 9 | offreboun_point ∈ [0.07, 0.75] ∧ minutes ∈ [22.20, 31.78] ∧ assisted_two_spt ∈ [0.83, 1.00] ∧ def_three_prereboun_point ∈ [0, 0.08) ∧ offreboun_point ∈ [0, 0.08) |
| Pattern 10 | off_three_prereboun_point ∈ [0, 0.08) ∧ shotqualityavg ∈ [0.45, 0.49] |
| Pattern 11 | fg_two_ablocked ∈ [1, 5] |
| Pattern 12 | assisted_two_spt ∈ [0, 0.08) ∧ def_three_prereboun_point ∈ [0, 0.08) ∧ offreboun_point ∈ [0, 0.08) |
| Pattern 13 | assisted_two_spt ∈ [0, 0.08) ∧ def_three_prereboun_point ∈ [0, 0.08) ∧ fg_three_apct ∈ [0, 0.19] ∧ usage ∈ [0, 0.08) |
| Pattern 14 | assisted_two_spt ∈ [0, 0.08) ∧ def_three_prereboun_point ∈ [0, 0.08) ∧ shotqualityavg ∈ [0.45, 0.49] |
| Pattern 15 | rebounds ∈ [1, 4] ∧ fg_three_pct ∈ [0.4, 1.0] ∧ minutes ∈ [31.78, 49.63] ∧ prov_game_away__points ∈ [96, 104] ∧ usage ∈ [0.67, 1] ∧ defreboun_point ∈ [0, 0.08) |
| Pattern 16 | assisted_two_spt ∈ [0, 0.08) ∧ shotqualityavg ∈ [0.45, 0.49] |
| Pattern 17 | assisted_two_spt ∈ [0, 0.08) ∧ assisted_two_spt ∈ [0, 0.08) ∧ fg_two_ablocked ∈ [0, 0.08) ∧ usage ∈ [0, 0.08) |
| Pattern 18 | assisted_two_spt ∈ [0, 0.08) ∧ assisted_two_spt ∈ [0, 0.08) ∧ assisted_two_spt ∈ [0, 0.08) |
| Pattern 19 | assisted_two_spt ∈ [0, 0.08) ∧ player_name = Jarrett Jack |
| Pattern 20 | prov_game_away__points ∈ [112, 135] ∧ fg_two_ablocked ∈ [1, 5] |

Table 10: First 20 Patterns Returned from Explanation Table

A.2 Top-20 explanations for Case Study queries

We present the detailed top-20 explanations for each case study queries mentioned in Section 6.

Nodes are identifiers for relations presented in the join graph. Edges are the edge details with their joint conditions. Pattern Desc are pattern descriptions. Each line represents a predicate of the pattern. The text descriptions that contain “prov_” prefix suggest that these predicates are generated from PT node followed by the name of the relation that was part of the user query, then followed by the attribute from which the constant value was generated. For example A1, prov_player_game_stats_minutes<36 should be interpreted as “minutes coming from player_game_stats table in PT smaller than 36. Precision, Recall and F-score are calculated based
| Nodes | Edges | Pattern Desc | Precision | Recall | F-score | Primary Tuple |
|-------|-------|--------------|-----------|--------|---------|---------------|
| A₁ | FT | (A₁ team_id)=A₁ prov game away_id) | A₁ prov season season__type=regular season A₁ salary=1533045.0 | 1.0 | 1.0 | 1.0 | 2015-16 |
| A₂ | team | (A₁ prov team_id)=A₁ player_id) | A₁ salary=1533045.0 | 1.0 | 1.0 | 1.0 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game winner_id) | A₁ prov season season__type=regular season A₁ salary=1533045.0 | 1.0 | 1.0 | 1.0 | 2015-16 |
| A₂ | team | (A₁ prov team_id)=A₁ player_id) | A₁ salary=1533045.0 | 1.0 | 1.0 | 1.0 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game home_id) | A₁ prov season season__type=regular season A₁ salary=1533045.0 | 1.0 | 1.0 | 1.0 | 2015-16 |
| A₂ | team | (A₁ prov team_id)=A₁ player_id) | A₁ salary=1533045.0 | 1.0 | 1.0 | 1.0 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game away_id) | A¹ prov player player__stats_type=0.69 A¹ prov player player__stats_usage=20.51 A¹ prov season season__type=regular season A¹ salary=14268070.0 | 1.0 | 0.71 | 0.83 | 2016-17 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 | 1.0 | 0.71 | 0.83 | 2016-17 |
| A₁ | FT | (A₁ team_id)=A₁ prov game home_id) | A¹ prov player player__stats_type=0.69 A¹ prov player player__stats_usage=20.51 A¹ prov season season__type=regular season A¹ salary=14268070.0 | 1.0 | 0.71 | 0.83 | 2016-17 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 | 1.0 | 0.71 | 0.83 | 2016-17 |
| A₁ | FT | (A₁ team_id)=A₁ prov game winner_id) | A¹ prov player player__stats_type=0.69 A¹ prov player player__stats_usage=20.51 A¹ prov season season__type=regular season A¹ salary=14268070.0 | 1.0 | 0.71 | 0.83 | 2016-17 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 | 1.0 | 0.71 | 0.83 | 2016-17 |
| A₁ | FT | (A₁ team_id)=A₁ prov game away_id) | A¹ prov player player__stats_type=0.69 A¹ prov player player__stats_usage=20.51 A¹ prov season season__type=regular season A¹ salary=14268070.0 | 1.0 | 0.69 | 0.82 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 | 1.0 | 0.69 | 0.82 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game home_id) | A¹ prov player player__stats_type=0.69 A¹ prov player player__stats_usage=20.51 A¹ prov season season__type=regular season A¹ salary=14268070.0 | 1.0 | 0.69 | 0.82 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 | 1.0 | 0.69 | 0.82 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game winner_id) | A¹ prov player player__stats_type=0.69 A¹ prov player player__stats_usage=20.51 A¹ prov season season__type=regular season A¹ salary=14268070.0 | 1.0 | 0.67 | 0.8 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 | 1.0 | 0.67 | 0.8 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game away_id) | A¹ prov player player__stats_type=0.69 A¹ prov player player__stats_usage=20.51 A¹ prov season season__type=regular season A¹ salary=14268070.0 | 1.0 | 0.59 | 0.74 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 | 1.0 | 0.59 | 0.74 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game home_id) | A¹ prov player player__stats_type=36.105 A¹ prov player player__stats_usage=20.21 A¹ prov season season__type=regular season A¹ salary=14268070.0 A¹ team=GSW | 1.0 | 0.59 | 0.74 | 2016-17 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 A¹ team=GSW | 1.0 | 0.59 | 0.74 | 2016-17 |
| A₁ | FT | (A₁ team_id)=A₁ prov game away_id) | A¹ prov player player__stats_type=36.55 A¹ prov player player__stats_usage=14.06 A¹ prov season season__type=regular season A¹ salary=14268070.0 A¹ team=GSW | 1.0 | 0.59 | 0.74 | 2016-17 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 A¹ team=GSW | 1.0 | 0.59 | 0.74 | 2016-17 |
| A₁ | FT | (A₁ team_id)=A₁ prov game home_id) | A¹ prov player player__stats_fg_three__apt=0.3875 A¹ prov season season__type=regular season A¹ team=GSW | 0.65 | 0.83 | 0.73 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 A¹ team=GSW | 0.65 | 0.83 | 0.73 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game away_id) | A¹ prov player player__stats_deflongmidrange=0.3 A¹ prov season season__type=regular season A¹ team=GSW | 0.59 | 0.84 | 0.69 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 A¹ team=GSW | 0.59 | 0.84 | 0.69 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game home_id) | A¹ prov player player__stats_deflongmidrange=0.3 A¹ prov season season__type=regular season A¹ team=GSW | 0.59 | 0.84 | 0.69 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 A¹ team=GSW | 0.59 | 0.84 | 0.69 | 2015-16 |
| A₁ | FT | (A₁ team_id)=A₁ prov game away_id) | A¹ prov player player__stats_deflongmidrange=0.3 A¹ prov season season__type=regular season A¹ team=GSW | 0.59 | 0.84 | 0.69 | 2015-16 |
| A₂ | team | (A¹ prov team_id)=A¹ player_id) | A¹ salary=14268070.0 A¹ team=GSW | 0.59 | 0.84 | 0.69 | 2015-16 |

Figure 17: Q̃nbu1 Top-20 patterns
| Nodes | Edges | Pattern Descrip | Precision | Recall | F-score | Primary Tuple |
|------|-------|----------------|-----------|------|--------|---------------|
| $A_1$: PT <br>$A_2$: team <br>$A_3$: player_game_stats <br>$A_4$: player | $(A_2$.team_id)=$(A_1$.prov_game_away__id) <br> $(A_1$.prov_game_game__date)=(A_3$.game_date) <br> $(A_3$.prov_game_game_id)=(A_4$.home_id) <br> $(A_4$.player_id)=$(A_3$.player_id) | $A_1$.prov_team_game_stats_assistpoints=68.0 <br> $A_4$.player_name=Raymond Green | 0.62 | 0.9 | 0.74 | 2013-14 |
| $A_1$: PT <br>$A_2$: team <br>$A_3$: player_game_stats <br>$A_4$: player | $(A_3$.team_id)=$(A_1$.prov_game_winner__id) <br> $(A_1$.prov_game_game__date)=(A_3$.game_date) <br> $(A_3$.prov_game_home_id)=(A_4$.home_id) <br> $(A_4$.player_id)=$(A_3$.player_id) | $A_1$.prov_team_game_stats_assistpoints=68.0 <br> $A_4$.player_name=Raymond Green | 0.62 | 0.9 | 0.74 | 2013-14 |
| $A_1$: PT <br>$A_2$: team <br>$A_3$: player_game_stats <br>$A_4$: player | $(A_3$.team_id)=$(A_1$.prov_game_winner__id) <br> $(A_1$.prov_game_game__date)=(A_3$.game_date) <br> $(A_3$.prov_game_home_id)=(A_4$.home_id) <br> $(A_4$.player_id)=$(A_3$.player_id) | $A_1$.prov_team_game_stats_nonputback_assisted_two__spct<0.55 | 0.78 | 0.7 | 0.74 | 2014-15 |
| $A_1$: PT <br>$A_2$: team <br>$A_3$: player_game_stats <br>$A_4$: player | $(A_3$.team_id)=$(A_1$.prov_game_winner__id) <br> $(A_1$.prov_game_game__date)=(A_3$.game_date) <br> $(A_3$.prov_game_home_id)=(A_4$.home_id) <br> $(A_4$.player_id)=$(A_3$.player_id) | $A_1$.prov_team_game_stats_nonputback_assisted_two__spct<0.5475 | 0.78 | 0.7 | 0.74 | 2014-15 |
| $A_1$: PT <br>$A_2$: team <br>$A_3$: player_salary <br>$A_4$: player | $(A_3$.player_id)=$(A_1$.prov_game_winner__id) <br> $(A_1$.prov_season_season_id)=(A_3$.season_id) <br> $(A_4$.player_id)=$(A_3$.player_id) | $A_1$.prov_team_game_stats_assistpoints=68.0 <br> $A_2$.player_name=Gal Mekel | 0.61 | 0.9 | 0.73 | 2013-14 |

Figure 18: $Q_{nba2}$ Top-20 patterns
| Nodes                | Edges                                                                 | Pattern Desc                                           | Precision | Recall | F-score | Primary Tuple |
|---------------------|------------------------------------------------------------------------|--------------------------------------------------------|-----------|--------|---------|----------------|
| A1: player salary   | (A1: prov_game_home_id)                                                | A1: prov_season_season__type=regular season            | 1.0       | 1.0    | 1.0     | 2009-10        |
| A2: team game_stats | (A2: team_id)=A1: team                                                 | A1: prov_season_season__type=regular season            | 1.0       | 1.0    | 1.0     | 2009-10        |
| A2: team            | (A2: prov_player__game__stats_deflongmidrangereboundpct<0.29)         | A1: prov_player__game__stats_deflongmidrangereboundpct<0.29 | 1.0       | 0.98   | 0.99    | 2009-10        |
| A2: team            | (A2: prov_player__game__stats_assisted__two__spct>0.19)                | A1: prov_player__game__stats_assisted__two__spct>0.19   | 1.0       | 0.98   | 0.99    | 2010-11        |
| A2: team            | (A2: prov_player__game__stats_usage>28.025)                            | A1: prov_player__game__stats_usage>28.025               | 1.0       | 0.98   | 0.99    | 2010-11        |
| A2: team            | (A2: salary<15779912.0)                                               | A1: salary<15779912.0                                  | 1.0       | 0.98   | 0.99    | 2010-11        |
| A2: team            | (A2: salary>14500000.0)                                               | A1: salary>14500000.0                                  | 1.0       | 0.98   | 0.99    | 2010-11        |
| A2: team            | (A2: team=CLE)                                                         | A1: team=CLE                                           | 1.0       | 0.98   | 0.99    | 2010-11        |

Figure 19: \( \rho_{\text{Naive}} \) Top-20 patterns
| Nodes | Edges | Pattern Desc | Precision | Recall | F-score | Primary Tuple |
|-------|-------|--------------|-----------|--------|---------|--------------|
| A1:FT | A2:player | (A1.prov_season_season__id)=A2:season_id | A1:prov_season_season__type=regular season A3:player_name=Monti Ellis A2:salary=11090000.0 | 1.0 | 1.0 | 1.0 | 2016-17 |
| A1:FT | A2:player | (A1.prov_season_season__id)=A2:season_id | A1:prov_season_season__type=regular season A3:player_name=Monte Ellis A2:salary=11090000.0 | 1.0 | 1.0 | 1.0 | 2016-17 |
| A1:FT | A2:player | (A1.prov_season_season__id)=A2:season_id | A1:prov_season_season__type=regular season A3:player_name=Monte Ellis A2:salary=11090000.0 | 1.0 | 1.0 | 1.0 | 2016-17 |
| A1:FT | A2:player | (A1.prov_season_season__id)=A2:season_id | A1:prov_season_season__type=regular season A3:player_name=Monte Ellis A2:salary=11090000.0 | 1.0 | 1.0 | 1.0 | 2016-17 |
| A1:FT | A2:team | (A1.prov_game_game__id)=A2:game_id | A2:team_id=A1.player_id A3:team_game_stats A2:game_date | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | 0.84 | 0.97 | 0.97 | 2016-17 |
| A1:FT | A2:team | (A1.prov_game_game__id)=A2:game_id | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | 0.84 | 0.97 | 0.97 | 2016-17 |
| A1:FT | A2:team | (A1.prov_game_game__id)=A2:game_id | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | 0.84 | 0.97 | 0.97 | 2016-17 |
| A1:FT | A2:team | (A1.prov_game_game__id)=A2:game_id | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | 0.84 | 0.97 | 0.97 | 2016-17 |
| A1:FT | A2:team | (A1.prov_game_game__id)=A2:game_id | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | A2:team_id=A1.team_id A3:team_game_stats A2:game_date | 0.84 | 0.97 | 0.97 | 2016-17 |

**Figure 20: Q_nb4 Top-20 patterns**
| Nodes | Edges | Pattern Desc | Precision | Recall | F-score | Primary Tuple |
|-------|-------|-------------|-----------|--------|---------|---------------|
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_prov_season_season_type=regular season | A3_salary>1112880.0 | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_game_winner_id | A1_salary | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |
| A1: FT  | A2: team  | (A2_team_id) | A1_prov_winner_id | A1_season_id | A1_team_id | 1.0 | 1.0 | 1.0 | 2014-15 |

Figure 21: Q_{nhu5} Top-20 patterns
| Nodes                                      | Edges                                                                 | Pattern Desc | Precision | Recall | F-score | Primary Tuple |
|--------------------------------------------|-----------------------------------------------------------------------|--------------|-----------|--------|---------|---------------|
| A1: PT                                     | (A2.subject_id)=A1.prov_diagnoses_subject__id                           | A2 expire_flag=1 | 0.65      | 0.72   | 0.68    | 2             |
| A2: patients                               | (A2.subject_id)=A2.prov_diagnoses_subject__id                           | A2 expire_flag=1 | 0.65      | 0.72   | 0.68    | 2             |
| A3: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)=(A2.hadm_id)              | A2 expire_flag=1 | 0.65      | 0.72   | 0.68    | 2             |
| A1: PT                                     | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A3 expire_flag=1 | 0.65      | 0.72   | 0.68    | 2             |
| A2: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A2 expire_flag=1 | 0.66      | 0.71   | 0.68    | 2             |
| A1: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A3 expire_flag=1 | 0.65      | 0.72   | 0.68    | 2             |
| A2: procedures                             | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.65      | 0.66   | 0.65    | 2             |
| A3: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A3 expire_flag=1 | 0.65      | 0.63   | 0.65    | 2             |
| A1: procedures                             | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A3 expire_flag=1 | 0.65      | 0.61   | 0.63    | 2             |
| A2: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.65      | 0.81   | 0.62    | 13            |
| A1: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A3 expire_flag=1 | 0.65      | 0.81   | 0.62    | 13            |
| A2: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.65      | 0.81   | 0.61    | 13            |
| A1: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.63      | 0.59   | 0.61    | 2             |
| A2: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.54      | 0.67   | 0.6    | 13            |
| A1: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.54      | 0.66   | 0.59    | 13            |
| A2: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.54      | 0.63   | 0.58    | 13            |
| A1: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.59      | 0.57   | 0.58    | 2             |
| A2: procedures                             | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.59      | 0.57   | 0.58    | 2             |
| A3: patients                               | (A2.subject_id)+(A1.prov_diagnoses_hadm__id)                           | A4 expire_flag=1 | 0.65      | 0.52   | 0.58    | 13            |

**Figure 22:** Qmimic: Top-20 patterns
| Nodes | Edges | Pattern Desc | Precision | Recall | F-score | Primary Triple |
|-------|-------|--------------|-----------|--------|---------|----------------|
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₁ expire flag=1 | Medicare | 0.84 | 0.85 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁ or A₁₁ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |
| A₁₁  | A₁₂ A₂₂ procedures A₁₃ patients | (A₁ prov admissions hadm id)=(A₂ hadm id) \(\land\) (A₃ subject id)=(A₂ subject id) \(\land\) A₁₂ expire flag=1 | Medicare | 0.85 | 0.91 | Medicare |

**Figure 23:** Qmimic² Top-20 patterns
| Nodes | Edges | Pattern Desc | Precision | Recall | F-score | Primary Tuple |
|-------|-------|--------------|-----------|--------|---------|---------------|
| A₁: FT | A₂: patients | (A₂.subject_id=A₁.prov_icustays_subject__id) | A₄ hospital stay length>9.0 | 0.8 | 0.94 | x:8 |
| A₁: FT | A₂: patients | (A₂.subject_id=A₁.prov_icustays_subject__id) | A₃ chapter=16 | | | |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | | | | |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₄ hospital stay length>6.0 | 0.97 | 0.77 | 0.86 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ los_group=0-1 | | | |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | 0.99 | 0.72 | 0.83 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ prov_icustays_dbsource=carevue | | | |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ hospital stay length>8.0 | 0.87 | 0.71 | 0.78 | x:8 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | 0.86 | 0.7 | 0.77 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ hospital stay length=16.0 | 0.81 | 0.7 | 0.76 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ prov_icustays_dbsource=carevue | | | |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ hospital stay length=9.0 | 0.83 | 0.69 | 0.75 | x:8 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ hospital stay length=7.0 | 0.96 | 0.47 | 0.63 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ hospital stay length=0.0 | 0.7 | 0.67 | 0.68 | x:8 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | 0.65 | 0.72 | 0.68 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | 0.65 | 0.72 | 0.68 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ admission_type=EMERGENCY | 0.7 | 0.67 | 0.68 | x:8 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | | | |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ gender=M | 0.99 | 0.51 | 0.67 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ los=0.9754 | | | |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ gender=M | 0.85 | 0.51 | 0.64 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ gender=M | 0.85 | 0.51 | 0.64 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | 0.96 | 0.47 | 0.63 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ prov_icustays_dbsource=carevue | 0.87 | 0.68 | 0.62 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ prov_icustays_dbsource=carevue | 0.87 | 0.68 | 0.62 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ gender=M | 0.97 | 0.52 | 0.62 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ admission_type=EMERGENCY | 0.7 | 0.67 | 0.68 | x:8 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | 0.69 | 0.54 | 0.6 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ expire_flag=0 | 0.69 | 0.54 | 0.6 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ los=0.9754 | 0.7 | 0.53 | 0.6 | 0:1 |
| A₁: admissions | A₂: patients | (A₂.hadm_id=A₁.hadm_id) | A₃ gender=F | 0.99 | 0.42 | 0.58 | 0:1 |

Figure 24: $Q_{mimic3}$ Top-20 patterns
| Nodes                          | Edges                                                                 | Pattern Desc                                                                 | Precision | Recall | F-score | Primary Triple |
|-------------------------------|------------------------------------------------------------------------|------------------------------------------------------------------------------|-----------|--------|---------|----------------|
| A₁, PT                        | A₂ prov_admissions_hadm__idchersh_admission__type=EMERGENCY            | A₃ prov_admissions_hospital__stay__length<22.0                               | 0.65      | 0.84   | 0.73    | Medicare       |
| A₁, patients add_info         | (A₃, subject_id)                                                       | A₃ expire_flag=0                                                             | 0.77      | 0.77   | 0.77    | Private        |
| A₁, patients                  | (A₃, subject_id)                                                       | A₃ age=70.86                                                                 |           |        |         |                |
| A₁, procedures                | (A₃, subject_id)                                                       | A₃ expire_flag=0                                                             | 0.6       | 0.79   | 0.68    | Private        |
| A₁, icustays                  | (A₃, subject_id)                                                       | A₃ expire_flag=0                                                             | 0.6       | 0.8    | 0.68    | Private        |
| A₁, procedures                | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.78      | 0.5    | 0.61    | Medicare       |
| A₁, icustays                  | (A₃, subject_id)                                                       | A₃ gender=M                                                                 | 0.48      | 0.61   | 0.54    | Private        |
| A₁, patients add_info         | (A₃, subject_id)                                                       | A₃ gender=F                                                                  | 0.6       | 0.48   | 0.53    | Medicare       |
| A₁, patients                  | (A₃, subject_id)                                                       | A₃ gender=M                                                                 | 0.48      | 0.6    | 0.53    | Private        |
| A₁, patients add_info         | (A₃, subject_id)                                                       | A₃ gender=M, A₃ age=64.72, A₃ ethnicity=WHITE                                | 0.83      | 0.39   | 0.53    | Private        |
| A₁, patients                  | (A₃, subject_id)                                                       | A₃ expire_flag=0                                                             | 0.36      | 0.43   | 0.5     | Medicare       |
| A₂ patients add_info          | (A₃, subject_id)                                                       | A₃ ethnicity=WHITE                                                           | 0.59      | 0.42   | 0.49    | Private        |
| A₂, procedures                | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.6       | 0.41   | 0.49    | Medicare       |
| A₂, icustays                  | (A₃, subject_id)                                                       | A₃ expire_flag=0                                                             | 0.6       | 0.42   | 0.48    | Medicare       |
| A₂, procedures                | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.8       | 0.34   | 0.47    | Medicare       |
| A₂, icustays                  | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.76      | 0.34   | 0.47    | Medicare       |
| A₂, procedures                | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.61      | 0.38   | 0.46    | Medicare       |
| A₂, icustays                  | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.58      | 0.36   | 0.45    | Medicare       |
| A₂, patients add_info         | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.66      | 0.35   | 0.44    | Medicare       |
| A₂, icustays                  | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.58      | 0.35   | 0.42    | Medicare       |
| A₂, patients                  | (A₃, subject_id)                                                       | A₃ expire_flag=1                                                             | 0.65      | 0.31   | 0.42    | Medicare       |

Figure 25: Qmimic Top-20 patterns
| Nodes          | Edges                                                                 | Pattern Desc                                                                 | Precision | Recall | F-score | Primary Tuple |
|----------------|----------------------------------------------------------------------|--------------------------------------------------------------------------------|-----------|--------|---------|----------------|
| A1: FF         | A1: patients                                                       | A4_hospital_length < 19.0                                                      | 1.0       | 0.82   | 0.9     | ASIAN          |
| A1: patients   | A1: patients_admit_info                                            | A4_hospital_length > 19.0                                                      | 1.0       | 0.67   | 0.8     | HISPANIC       |
| A1: admissions |                                                                      | A4_hospital_length > 5.0                                                       | 1.0       | 0.67   | 0.8     | HISPANIC       |
| A1: FF         |                                                                      | A4_admission_type = EMERGENCY                                                     | 1.0       | 0.67   | 0.8     | HISPANIC       |
| A1: patients   |                                                                      | A4_admission_type = EMERGENCY                                                     | 0.6       | 0.8    | 0.68   | HISPANIC       |
| A1: admissions |                                                                      | A4_admission_type = EMERGENCY                                                     | 0.6       | 0.8    | 0.68   | HISPANIC       |
| A1: patients   |                                                                      | A4_admission_type = EMERGENCY                                                     | 0.59      | 0.8    | 0.68   | HISPANIC       |
| A1: admissions |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: patients   |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: admissions |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: patients   |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: admissions |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: patients   |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: admissions |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: patients   |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |
| A1: admissions |                                                                      | A4_expire_flag = 0                                                             | 0.57      | 0.71   | 0.64   | HISPANIC       |

Figure 26: Qmimcs Top-20 patterns