

# Query Reverse Engineering

Quoc Trung Tran · Chee-Yong Chan · Srinivasan Parthasarathy

Received: date / Accepted: date

**Abstract** In this paper, we introduce a new problem termed *Query Reverse Engineering* (QRE). Given a database  $D$  and a result table  $T$  – the output of some *known* or *unknown* query  $Q$  on  $D$  – the goal of QRE is to reverse-engineer a query  $Q'$  such that the output of query  $Q'$  on database  $D$  (denoted by  $Q'(D)$ ) is equal to  $T$  (i.e.,  $Q(D)$ ). The QRE problem has useful applications in database usability, data analysis, and data security. In this work, we propose a data-driven approach, TALOS for **T**ree-based classifier with **A**t **L**east **O**ne **S**emantics, that is based on a novel dynamic data classification formulation and extend the approach to efficiently support the three key dimensions of the QRE problem: whether the input query is known/unknown, supporting different query fragments, and supporting multiple database versions.

## 1 Introduction

This paper introduces a new problem, termed *Query Reverse Engineering* (QRE), that aims to reverse-engineer a query given its output from a database. Formally, given a database  $D$  and a result table  $T$ , which is the output of some *known* or *unknown* query  $Q$  on  $D$ ; the goal of QRE is to reverse-engineer a query  $Q'$  such that the output of query  $Q'$  on database  $D$  (denoted by  $Q'(D)$ ) is equal to  $T$  (i.e.,  $Q(D)$ ). We say that two queries  $Q$  and  $Q'$  are *instance-equivalent*

*queries* (IEQs) w.r.t. a database  $D$  if their results (w.r.t.  $D$ ) are equal; i.e.,  $Q \equiv_D Q'$ .

In the following, we highlight several use cases of QRE.

**Database Usability.** The most obvious application of QRE is in database usability. Consider the situation when a user wants to evaluate a query  $Q$  on a database  $D$ . Instead of simply returning the query result  $Q(D)$  to the user, the database system can also apply QRE to derive instance-equivalent queries  $Q'$  of  $Q$  that describe alternative characterizations of tuples in the query result of  $Q$ .

*Example 1* Suppose a user issues a query  $Q_1$  to a movie database to search for movies that are directed by “James Cameron” since 1997. The query result  $Q_1(D)$  consists of two movies: “Avatar” and “Titanic”. By applying QRE, the database system could also have augmented the result the following instance-equivalent query  $Q'_1$  (w.r.t.  $D$ ) “select movies that have gross-revenue greater than \$2 billion”. The query  $Q'_1$  provides a different and insightful characterization of the query result to the user.  $\square$

As the above example illustrates, the ability to return instance-equivalent queries for a given query  $Q$  can reveal interesting properties of the query result  $Q(D)$ . In addition, unusual or surprising IEQs can be useful for uncovering hidden relationships among the data. In several instances, simpler or easier to understand relationships may be uncovered, which can again aid in the understanding of the data contained within the complex database. As another example, consider a skyline query  $Q_2$  to search for people with maximal capital gain and minimal age in the U.S. Census Income dataset.<sup>1</sup> An instance-equivalent query  $Q'_2$  of  $Q_2$  provides a simpler characterization of  $Q_2$  as “find people who are very young ( $age \leq 17$ ) and have low capital gain ( $< 5000$ ), or

---

Quoc Trung Tran  
National University of Singapore  
Tel.: +6584624078  
E-mail: tqtrung@comp.nus.edu.sg

Chee-Yong Chan  
National University of Singapore E-mail: chancy@comp.nus.edu.sg

Srinivasan Parthasarathy  
The Ohio State University E-mail: srini@cse.ohio-state.edu

---

<sup>1</sup> <http://archive.ics.uci.edu/ml/datasets/Adult>

who have very high capital gain ( $> 27828$ ), work in the protective service and whose race is not classified as others”.

Besides providing alternative characterizations of the query results, IEQs can also help users to better understand the database schema. Since many enterprise data schema are very complex and large, the part of the database schema that is referenced by the user’s query may be quite different from that referenced by an IEQ. The discovery of this alternative “part” in the schema to generate an instance-equivalent query can aid the user’s understanding of the database schema, or potentially help refine the user’s initial query.

*Example 2* Consider the baseball data set<sup>2</sup> and a query  $Q_3$  that finds managers of “Cincinnati Reds” team during the years from 1982 to 1988. This query involves the join between two relations *Manager* and *Team*. An instance-equivalent query  $Q'_3$  of  $Q_3$  reveals that some of these managers were also the *players* of “Cincinnati Reds” team at the same time they managed the team. The IEQ  $Q'_3$  has revealed the alternative schema part that involves the joins among *Manager*, *Team*, *Master*, and *Batting* relations, and provided users useful information about these player-managers.  $\square$

**Explaining Why-not Questions.** QRE can also be applied to provide an explanation capability that allows users to seek clarifications on unexpected query results. For example, consider the scenario where a user has issued a query and is surprised to find that the returned query result does not contain certain expected tuples. It would be very useful to have some functionality that enables the user to ask a follow-up query on why those expected tuples are missing from the query result. The idea of QRE can be applied to automatically generate one or more *refined query*, whose result includes both the original query’s result as well as the missing tuple(s). The refined queries would serve as a form of explanation for the missing tuples: they not only could help identify the original query predicates that are responsible for the missing tuples, but also suggest alternative means to “fix” the missing tuple(s). Essentially, QRE treats the query’s result  $Q(D)$  together with the missing tuples as the output result of some query  $Q'$  and derives IEQs for  $Q'$  as refined queries [28].

**Data Exploration & Analysis.** Another important class of QRE applications is in scenarios where the input query  $Q$  is unknown.

Consider a view  $V$  (defined on a database  $D$ ) which may have been derived manually (e.g., a user selected some tuples of the database of interest to her), or by an application program that is no longer available (e.g., the program is no longer maintained or is lost). Therefore, given only the view result  $V$  on the database  $D$ , it will be very useful to be able to derive instance-equivalent queries  $Q'$  of the unknown query for  $V$  (i.e.,  $V = Q'(D)$ ) that describe the characteristics of

such tuples in  $V$ . In data exploration, such scenarios arise when the documentation and meta-data for the data sets being analyzed are incomplete, inaccurate, or missing. As an example, in the AT&T’s Bellman project [12], the data set made available to data analysts is often in the form of a delimited ASCII text file representing the output result of some query, where the query is not available for various reasons. Clearly, it will be useful to reverse engineer the queries to help make sense of the data before performing further analysis tasks (e.g., data mining).

**Database Security.** QRE may also have interesting applications in database security where attackers who have some prior domain knowledge of the data may attempt to derive sensitive information. For example, if an attacker is aware of the existing correlation structure in the data, they can easily use this information to formulate two or more separate queries that on paper look very different (e.g., using different selection criteria), but in reality may be targeting the same set of tuples in the database. Such sets or groups of queries can potentially be used to reverse-engineer the privacy preserving protocol in use. Subsequently, sensitive information can be gleaned. As a specific example, consider a protocol such as  $\epsilon$ -diversity [32], which relies on detecting how similar the current query is with a previous set of queries answered by the database, to determine if the current query can be answered without violating the privacy constraints. By generating IEQs for the historical queries, the collection of historical queries is enlarged thereby strengthening the protection against such kinds of attacks.

## 1.1 Contributions

Despite its usefulness, there is very little work on the QRE problem. An early version of this paper [29] introduces the *Query by Output (QBO)* problem where the focus is on deriving select-project-join instance-equivalent queries (SPJ IEQs) given an input query  $Q$ , database  $D$ , and the query’s result  $Q(D)$ . This paper is a significant extension that generalizes QBO along three key dimensions of the problem space: (1) the original query  $Q$  is unknown; (2) the derived query  $Q'$  belongs to a more expressive query fragment beyond the simple select-project-join (SPJ) query fragment; and (3) the database  $D$  has multiple versions.

**Unknown Query.** First, unlike the QBO problem where the original query  $Q$  is given as a part of the input (along with the database  $D$ ),  $Q$  is unknown in the QRE problem, and the absence of  $Q$  makes it more challenging to identify the relations to be included in the reverse-engineered query  $Q'$  to generate the given result table  $T$  (i.e.,  $T \equiv Q'(D)$ ).

**Supporting More Expressive IEQs.** Second, for QRE to be more practical, QRE should support more expressive IEQs beyond the SPJ fragment (referred to as *SPJ-IEQs*) such as

<sup>2</sup> <http://baseball1.com/statistics/>

SPJ-IEQs with union operators (referred to as *SPJU-IEQs*), and SPJ-IEQs with group-by aggregation operators (referred to as *SPJA-IEQs*). With this enhanced expressiveness, QRE becomes useful in more application domains such as data integration, where SPJU-IEQs are predominant. In data integration systems, the goal is to combine data residing at different sources to provide users with a unified view of these data [13]. The *global-as-view* integration approach requires that the global schema be expressed in terms of the data sources, which necessitates a query over the global schema to be reformulated in terms of a set of queries over the data sources. Thus, the QRE problem in this context requires deriving an IEQ that is a union of sub-queries over the data sources given a result table  $T$  that is generated by the integration system. Another application domain that is supported by the enhanced expressiveness of IEQs is in data analysis, where SPJA-IEQs are very common due to aggregation computations (e.g. group-by aggregation queries in OLAP applications).

**Multiple Database Versions.** Third, in contrast to the QBO problem where the specific database  $D$  is known and given as part of the input, the setting of the QRE problem is often more general where there could be multiple database versions. We consider two specific scenarios of this generalization and the additional challenges introduced by them.

The first scenarios occur in exploratory/analysis applications: it is typically the case that the time when a user needs to reverse-engineer a query  $Q'$  for a result table  $T$  occurs much later than the time when  $T$  was actually produced. Thus, it may not be meaningful or possible to derive  $Q'$  from the current version of the database, as this could be very different from the version that  $T$  was generated from. Specifically, given a result table  $T$  and a sequence of database versions  $\langle D_1, D_2, \dots, D_\ell \rangle$ , a specific goal may be to determine the most recent database version  $D_i$ ,  $i \in [1, \ell]$ , and query  $Q'$  such that  $Q'(D_i) = T$ . Depending on the applications, other variations of the problem (e.g., finding the earliest database version or all versions) are also possible. The performance challenge is how to efficiently determine both  $D_i$  as well as  $Q'$  for a given result  $T$ .

In the second scenario, the input to the QRE problem is a sequence of database versions and result pairs  $(D_1, T_1)$ ,  $(D_2, T_2)$ ,  $\dots$ ,  $(D_\ell, T_\ell)$ ; where each  $T_i$  is the result of the *same* unknown query  $Q$  on database version  $D_i$  (i.e.,  $T_i = Q(D_i)$ ). For example, the  $T_i$ 's could correspond to weekly reports generated by the same query on weekly versions of the database, or  $Q$  could be a continuous long standing query that is run periodically on different snapshots of the database. In this more general setting with multiple database and result versions, the challenge is how to efficiently reverse-engineer a query  $Q'$  such that  $Q'(D_i) = T_i$  for each  $i \in [1, \ell]$ .

In summary, this paper makes the following contributions. First, we introduce the novel problem of QRE and propose a solution, TALOS, that models the QRE problem as a data classification task with a unique property that we term *at-least-one semantics*. Second, to handle data classification with this new semantics, we develop a new dynamic class labeling technique and propose effective optimization techniques to efficiently compute IEQs. Third, to the best of our knowledge, TALOS is the first comprehensive solution for the query reverse engineering problem that covers three dimensions of the problem space: the original query is known/unknown, the type of IEQ fragmentation (i.e., SPJ with possibly union/aggregation), and the presence of multiple database versions. Finally, our experimental evaluation of TALOS demonstrates its efficiency and effectiveness in generating interesting IEQs.

**Organization.** In Section 2, we provide some basic background and describe an exemplar scenario for QRE that will serve as a running example throughout this article. Section 3 presents the concept of instance-equivalent queries and establishes some complexity results for the QRE problem. Sections 4 to 6 present our solution, TALOS, to derive IEQs. Section 7 discusses criteria for ranking IEQs. Section 8 presents an extension to handle unknown input query, and Section 9 presents an extension to support multiple database versions. Section 10 presents an experimental study to evaluate the effectiveness of TALOS. Section 11 presents related work. Finally, we conclude in Section 12. Proofs of theoretical results and other details are given in the Appendix.

## 2 Preliminaries

The QRE problem takes as inputs a database  $D$ , an optional query  $Q$ , and the query's output  $Q(D)$  (w.r.t.  $D$ ) to compute one or more IEQs  $Q'$ , where  $Q$  and  $Q'$  are IEQs if  $Q \equiv_D Q'$ . We refer to  $Q$  as the *input query*,  $Q(D)$  as the *given result table*, and  $Q'$  as the *output query*.

In this paper, we consider three fragments of SPJ relational queries for IEQs: (1) SPJ queries, which are the basic Select-Project-Join queries, (2) SPJU queries, which are of the form  $Q_1 \text{ union } Q_2 \dots \text{ union } Q_n$ , where each  $Q_i$  is an SPJ query, and (3) SPJA queries, which correspond to simple SPJ SQL queries with aggregation operators in the select-clause and an optional group-by clause.

Given a query  $Q$ , we use  $rel(Q)$  to denote the collection of relations involved in  $Q$  (i.e., relations in SQL's from-clause);  $proj(Q)$  to denote the set of projected attributes in  $Q$  (i.e., attributes in SQL's select-clause); and  $sel(Q)$  to denote the set of selection predicates in  $Q$  (i.e., conditions in SQL's where-clause).

<b>pID</b>	<b>name</b>	<b>country</b>	<b>weight</b>	<b>bats</b>	<b>throws</b>
P1	A	USA	85	L	R
P2	B	USA	72	R	R
P3	C	USA	80	R	L
P4	D	Germany	72	L	R
P5	E	Japan	72	R	R

(a) *Master*

<b>pID</b>	<b>year</b>	<b>stint</b>	<b>team</b>	<b>HR</b>
P1	2001	2	PIT	40
P1	2003	2	ML1	50
P2	2001	1	PIT	73
P2	2002	1	PIT	40
P3	2004	2	CHA	35
P4	2001	3	PIT	30
P5	2004	3	CHA	60

(b) *Batting*

<b>pID</b>	<b>year</b>	<b>salary</b>	<b>team</b>	<b>year</b>	<b>rank</b>
P1	2003	80	PIT	2001	7
P3	2002	35	PIT	2002	4
P5	2004	60	CHA	2004	3

(c) *Salaries*(d) *Team***Fig. 1** Running Example - Baseball Data Set  $D$ 

Consider two attributes: attribute  $A$  in relation  $R$  and attribute  $B$  in relation  $S$ . If  $\pi_B(S) \subseteq \pi_A(R)$ , we say that  $A$  covers  $B$  and refer to  $A$  as a *covering attribute* of  $B$ . If  $\pi_B(S) \cap \pi_A(R) \neq \emptyset$ , we say that  $A$  *partially covers*  $B$  and refer to  $A$  as a *partially covering attribute* of  $B$ . Clearly,  $A$  partially covers  $B$  if and only if  $B$  partially covers  $A$ .

For ease of presentation and without loss of generality, we express each  $Q'$  as a relational algebra expression. To keep our definitions and notations simple and without loss of generality, we shall assume that there are no multiple instances of a relation in  $Q$  and  $Q'$ .

Given a database, we use  $\mathcal{SG}$  to denote its *schema graph* where each node in  $\mathcal{SG}$  represents a relation and each edge between two nodes represents a primary-key-foreign-key relationship between the relations corresponding to the nodes.

**Running example.** We use a database housing baseball statistics as our running example as well as in our experiments. Part of the schema is illustrated in Figure 1, where the key attribute names are shown in bold. The *Master* relation describes information about each player (identified by *pID*): the attributes *name*, *country*, *weight*, *bats*, and *throws* refer to his name, birth country, weight (in pounds), batting hand (left, right, or both), and throwing hand (left or right) respectively. The *Batting* relation provides the number of home runs (*HR*) of a player when he was playing for a team in a specific year and season (*stint*). The *Salaries* relation specifies the salary obtained by a player in a specific year. The *Team* relation specifies the rank obtained by a team for a specified year.

### 3 Instance-Equivalent Queries (IEQs)

#### 3.1 Strong vs Weak IEQs

Our basic definition of *instance-equivalent queries (IEQs)* requires that the IEQs  $Q$  and  $Q'$  produce the same output (w.r.t. some database  $D$ ); i.e.,  $Q \equiv_D Q'$ . The advantage of this simple definition is that it does not require the knowledge of  $Q$  to derive  $Q'$ , which is particularly useful for QRE applications where  $Q$  is either missing or not provided. However, there is a potential “accuracy” tradeoff that arises from the simplicity of this weak form of equivalence: an IEQ may be “semantically” quite different from the input query that produced  $Q(D)$  as the following example illustrates.

*Example 3* Consider the following three queries on the baseball database  $D$  in Figure 1:

$$Q_1 = \pi_{\text{country}}(\sigma_{\text{bats}=\text{“R”} \wedge \text{throws}=\text{“R”}}(\text{Master})),$$

$$Q_2 = \pi_{\text{country}}(\sigma_{\text{bats}=\text{“R”} \wedge \text{weight} \leq 72}(\text{Master})), \text{ and}$$

$$Q_3 = \pi_{\text{country}}(\sigma_{\text{bats}=\text{“R”}}(\text{Master})).$$

Observe that although all three queries produce the same output after projection ( $\{\text{USA}, \text{Japan}\}$ ), only  $Q_1$  and  $Q_2$  select the same set of tuples  $\{P2, P5\}$  from *Master*. Specifically, if we modify the queries by replacing the projection attribute “country” with the key attribute “pID”, we have  $Q_1(D) = \{P2, P5\}$ ,  $Q_2(D) = \{P2, P5\}$ , and  $Q_3(D) = \{P2, P3, P5\}$ . Thus, while all three queries are IEQs, we see that the equivalence between  $Q_1$  and  $Q_2$  is actually “stronger” (compared to that between  $Q_1$  and  $Q_3$ ) in that both queries actually select the same set of relation tuples.  $\square$

If  $Q$  is provided as part of the input, then we can define a stronger form of instance equivalence as suggested by the above example. Intuitively, the stricter form of instance equivalence not only ensures that the instance-equivalent queries produce the same output (w.r.t. some database  $D$ ), but it also requires that their outputs be projected from the same set of “core” tuples. We now formally characterize weak and strong IEQs based on the concepts of *core relations* and *core queries*.

**Core relations.** Given a query  $Q$ , we say that  $S \subseteq \text{rel}(Q)$  is a set of *core relations* of  $Q$  if  $S$  is a minimal set of relations such that for every attribute  $R_i.A \in \text{proj}(Q)$ , (1)  $R_i \in S$ , or (2)  $Q$  contains a chain of equality join predicates “ $R_i.A = \dots = R_j.B$ ” such that  $R_j \in S$ .

Intuitively, a set of core relations of  $Q$  is a minimal set of relations in  $Q$  that “cover” all the projected attributes in  $Q$ . As an example, consider a query  $Q = \pi_{R_1.X} \sigma_p(R_1 \times R_2 \times R_3)$  where  $p = (R_1.X = R_3.Y) \wedge (R_2.Z = R_3.Z)$ . Here,  $Q$  has two sets of core relations,  $\{R_1\}$  and  $\{R_3\}$ :  $\{R_1\}$  is clearly a set of core relations since the only projected attribute in  $Q$  is from  $R_1$ , while  $\{R_3\}$  is another set of core relations due to the join predicate “ $R_1.X = R_3.Y$ ”.

**Core queries.** Given a query  $Q$  and a set of relations  $S \subseteq \text{rel}(Q)$ , we use  $Q_S$  to denote the query that is derived from  $Q$  by replacing  $\text{proj}(Q)$  with the key attribute(s) of each relation in  $S$ . If  $S$  is a set of core relations of  $Q$ , we refer to  $Q_S$  as a *core query* of  $Q$ .

Consider two IEQs  $Q$  and  $Q'$  (w.r.t. a database  $D$ ); i.e.,  $Q \equiv_D Q'$ . We say that  $Q$  and  $Q'$  are *strong IEQs* if  $Q$  has a set of core relations  $S$  such that: (1)  $Q'_S$  is a core query of  $Q'$ , and (2)  $Q_S(D)$  and  $Q'_S(D)$  are equal. IEQs that are not strong are classified as *weak IEQs*.

The strong IEQ definition essentially requires that both  $Q$  and  $Q'$  share a set of core relations such that  $Q(D)$  and  $Q'(D)$  are projected from the same set of selected tuples from these core relations. Thus, in Example 3,  $Q_1$  and  $Q_2$  are strong IEQs whereas  $Q_1$  and  $Q_3$  are weak IEQs.

Note that in our definition of strong IEQ, we only impose moderate restrictions on  $Q$  and  $Q'$  (relative to the weak IEQ definition) so that the space of strong IEQs is not overly constrained, and that the strong IEQs generated are hopefully both interesting as well as meaningful.

As in the case with weak IEQs, two strong IEQs can involve different sets of relations. As an example, suppose query  $Q$  selects pairs of records from two core relations, *Supplier* and *Part*, that are related via joining with a (non-core) *Supply* relation. Then it is possible for a strong IEQ  $Q'$  to relate the same pair of core relations via a different relationship (e.g., by joining with a different non-core *Manufacture* relation).

We believe that each of the notions of query equivalence has useful applications in different contexts depending on the available type of information about the input query and database. At one extreme, if both  $Q$  and the database integrity constraints are available, we can compute semantically strong equivalent queries. At the other extreme, if only  $Q(D)$  and the database  $D$  are available, we can only compute weak IEQs. Finally, if both  $Q$  and the database  $D$  are available, we can compute both weak and strong IEQs.

### 3.2 Precise vs Approximate IEQs

It is also useful to permit some perturbation so as to include IEQs that are “close enough” to the original query. Perturbations could be in the form of extra records or missing records or a combination thereof. Such generalizations are necessary in situations where there are no precise IEQs, and useful for cases where the computational cost for finding precise IEQs is considered unacceptably high. Moreover, a precise IEQ  $Q'$  might not always provide insightful characterizations of  $Q(D)$ , as  $Q'$  could be too “detailed” with many join relations and/or selection predicates.

The imprecision of a weak SPJ- or SPJU-IEQ  $Q'$  of  $Q$  (w.r.t.  $D$ ) can be quantified by

$|Q(D) - Q'(D)| + |Q'(D) - Q(D)|$ ; the imprecision of a strong IEQ can be quantified similarly. Thus,  $Q'$  is considered an approximate (strong/weak) IEQ of  $Q$  if its imprecision is positive; otherwise,  $Q'$  is a precise (strong/weak) IEQ. For SPJA-IEQs, the imprecision metric is more involved as it needs to take into account matching tuples in  $Q(D)$  and  $Q'(D)$  that differ only on their aggregated attribute value(s); the details are presented in Appendix B.

### 3.3 Complexity Results

In this section, we establish some complexity results for the QRE problem.

**Theorem 1** *Given an input query  $Q$ , we define  $QRE_S$  to be the problem to find the output query  $Q'$ , where  $Q'$  is a conjunctive query that involves only projection and selection (with predicates of the form “ $A_i \text{ op } c$ ”, where  $A_i$  is an attribute,  $c$  is constant, and  $\text{op} \in \{<, \leq, =, \neq, >, \geq\}$ ) such that: (1)  $Q' \equiv_D Q$ , and (2) the number of operators (AND, OR and NOT) used in the selection condition is minimized. Then  $QRE_S$  is unlikely to be in  $P$ .*

**Theorem 2** *Given an input query  $Q$ , we define  $QRE_U$  to be the problem to find an SPJU-IEQ  $Q'$  of  $Q$  w.r.t. a database  $D$  of the form  $Q' = Q_1 \text{ union } \dots \text{ union } Q_k$  where each  $Q_i$  is an SPJ query such that (1)  $Q' \equiv_D Q$ , and (2)  $k$  is minimized. Then  $QRE_U$  is NP-hard.*

**Theorem 3** *Given an input query  $Q$ , we define  $QRE_A$  to be the problem to find an SPJA-IEQ  $Q'$  of  $Q$  w.r.t. a database  $D$  where each aggregation operator (SUM or AVG) does not involve any arithmetic expression in the operator’s argument. Then  $QRE_A$  is NP-hard.*

Given the above results, we consider relational queries  $Q$  where the select-clause refers to either some relation’s attribute, or a value computed by an aggregation operator (COUNT, SUM, AVG, MIN, or MAX) that does not involve any arithmetic expression in the operator’s argument to ensure that  $Q'$  can be derived from  $Q(D)$  efficiently. We also require that  $Q(D) \neq \emptyset$  for the problem to be interesting. Other fragments of SQL queries (e.g., SPJA queries with HAVING clause, queries with arithmetic expressions) are beyond the scope of this paper.

## 4 Overview of Approach

In this section, we give an overview of our approach, named TALOS (Tree-based classifier with At Least One Semantics), for the QRE problem.

#### 4.1 Data Classification Formulation

Given an input result table  $Q(D)$ , to generate an SPJ  $Q'$  that is an IEQ of  $Q$ , we basically need to determine the three components of  $Q'$ :  $rel(Q')$ ,  $sel(Q')$ , and  $proj(Q')$ . Clearly, if  $rel(Q')$  contains a set of core relations of  $Q$ , then  $proj(Q')$  can be trivially derived from these core relations.<sup>3</sup> Thus, the possibilities for  $Q'$  depend mainly on the options for both  $rel(Q')$  and  $sel(Q')$ . Between these two components, enumerating different  $rel(Q')$  is the easier task, as  $rel(Q')$  can be obtained by choosing a subgraph  $G$  of the schema graph  $\mathcal{SG}$  such that  $G$  contains a set of core relations of  $Q$ :  $rel(Q')$  is then given by all the relations represented in  $G$ . Note that it is not necessary for  $rel(Q) \subseteq rel(Q')$ , as  $Q$  may contain some relations that are not core relations. The reason for exploring different possibilities for  $rel(Q')$  is to find interesting alternative characterizations of  $Q(D)$  that involve different join paths or selection conditions from those in  $Q$ . TALOS enumerates different schema subgraphs by starting out with minimal subgraphs that contain a set of core relations of  $Q$ , and then incrementally expanding the minimal subgraphs to generate larger, more complex subgraphs.

We now come to the most critical and challenging part of our solution, which is how to generate “good”  $sel(Q')$ ’s such that each  $sel(Q')$  is not only succinct (without too many conditions) and insightful, but also minimizes the imprecision between  $Q(D)$  and  $Q'(D)$  if  $Q'$  is an approximate IEQ. We propose to formulate this problem as a *data classification* task as follows.

Consider the relation  $J$  that is computed by joining all the relations in  $rel(Q')$  based on the foreign-key joins represented in  $G$ . Without loss of generality, let us suppose that we are looking for weak IEQs  $Q'$ . Let  $L$  denote the ordered listing of the attributes in  $proj(Q')$  such that the schema of  $\pi_L(J)$  and  $Q(D)$  are equivalent.<sup>4</sup>  $J$  can be partitioned into two disjoint subsets,  $J = J_{in} \cup J_{out}$  such that  $\pi_L(J_{in}) \subseteq Q(D)$  and  $\pi_L(J_{out}) \cap Q(D) = \emptyset$ . For the purpose of deriving  $sel(Q')$ , one simple approach to classify the tuples in  $J$  is to label the tuples in  $J_{out}$ , which do not contribute to the query’s result  $Q(D)$ , as *negative tuples*, and label the tuples in  $J_{in}$  as *positive tuples*.

*Example 4* To illustrate how we derive IEQs, consider the following query on the baseball database  $D$ :

$Q_4 = \pi_{name}(\sigma_{bats="R" \wedge throws="R"} Master)$ . Note that  $Q_4(D) = \{B, E\}$ . Suppose that the schema subgraph  $G$  considered contains both *Master* and *Batting*, i.e.,  $rel(Q'_4) =$

<sup>3</sup> Note that even though the definition of a weak IEQ  $Q'$  of  $Q$  does not require the queries to share a set of core relations, we find this restriction to be a reasonable and effective way to obtain “good” IEQs.

<sup>4</sup> If the search is for strong IEQs, then the discussion remains the same except that  $L$  is the ordered listing of the key attributes of a set of core relations  $S$  of  $Q$ , and we replace  $Q(D)$  by  $Q_S(D)$ .

$\{Master, Batting\}$ . The output of  $J = Master \bowtie_{PID} Batting$  is shown in Figure 2(a). Using  $t_i$  to denote the  $i^{th}$  tuple in  $J$ , we observe that  $J$  is partitioned into  $J_{out} = \{t_1, t_2, t_5, t_6\}$  and  $J_{in} = \{t_3, t_4, t_7\}$ , since  $\pi_{name}(J_{in}) = Q(D)$  and  $\pi_{name}(J_{out}) \cap Q(D) = \emptyset$ . We therefore label tuples  $t_3, t_4$  and  $t_7$  as positive tuples, and the other tuples  $(t_1, t_2, t_5, t_6)$  as negative tuples.  $\square$

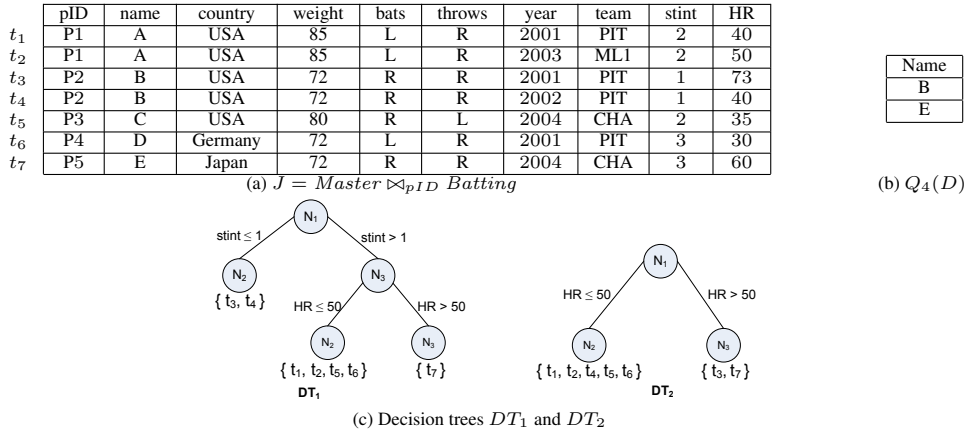
Given the labeled tuples in  $J$ , the problem of finding a  $sel(Q')$  can now be viewed as a data classification task to separate the positive and negative tuples in  $J$ :  $sel(Q')$  is given by the selection conditions that specify the positive tuples. A natural solution is to examine if an off-the-shelf data classifier can give us what we need. To determine what kind of classifier to use, we must consider what we need to generate our desired IEQ  $Q'$ . Clearly, the classifier should be efficient to construct and the output should be easy to interpret and express using SQL; i.e., the output should be expressible in axis parallel cuts of the data space. These criteria rule out a number of classifier systems such as neural networks,  $k$ -nearest neighbor classification, Bayesian classifiers, and support vector machines [21]. Rule based classifiers or decision trees (a form of rule-based classifier) are a natural solution in this context. TALOS uses a decision tree classifier for generating  $sel(Q')$ .

By computing IEQs in this way, our approach also generates a useful by-product: the set of labeled tuples in  $J$  could be used to derive the potential data provenance (i.e., the source tuples) of each output tuple in the query result.

##### 4.1.1 Decision Tree Construction

We now briefly describe how a simple binary decision tree is constructed to classify a set of data records  $D$ . For expository simplicity, assume that all the attributes in  $D$  have numerical domains. A decision tree  $DT$  is constructed in a top-down manner. Each leaf node  $N$  in the tree is associated with a subset of the data records, denoted by  $D_N$ , such that  $D$  is partitioned among all the leaf nodes. Initially,  $DT$  has only a single leaf node (i.e., its root node), which is associated with all the records in  $D$ . Leaf nodes are classified into pure and non-pure nodes depending on a given goodness criterion. Common goodness criteria include entropy, classification error and the Gini index [21]. At each iteration of the algorithm, the algorithm examines each non-pure leaf node  $N$  and computes the best split for  $N$  that creates two child nodes,  $N_1$  and  $N_2$ , for  $N$ . Each split is computed as a function of an attribute  $A$  and a split value  $v$  associated with the attribute. Whenever a node  $N$  is split (w.r.t. attribute  $A$  and split value  $v$ ), the records in  $D_N$  are partitioned between  $D_{N_1}$  and  $D_{N_2}$  such that a tuple  $t \in D_N$  is distributed into  $D_{N_1}$  if  $t.A \leq v$ ; and  $D_{N_2}$ , otherwise.

A popular goodness criterion for splitting is the Gini index, which is computed as follows. For a data set  $S$  with  $k$



**Fig. 2** Example of deriving IEQs for  $Q_4$  on  $D$

distinct classes, its Gini index is given by

$$Gini(S) = 1 - \sum_{j=1}^k (f_j^2) \quad (1)$$

where  $f_j$  denotes the fraction of records in  $S$  belonging to class  $j$ . Thus, if  $S$  is split into two subsets  $S_1$  and  $S_2$ , then the Gini index of the split is given by

$$Gini(S_1, S_2) = \frac{|S_1| Gini(S_1) + |S_2| Gini(S_2)}{|S_1| + |S_2|} \quad (2)$$

where  $|S_i|$  denotes the number of records in  $S_i$ . The general objective is to pick the splitting attribute whose best splitting value reduces the Gini index the most (i.e., the ideal goal is to reduce Gini to 0 resulting in all pure leaf nodes).

*Example 5* Continuing with Example 4 when we have labeled tuples  $t_3, t_4, t_7$  as positive and the other tuples as negative. Using the above algorithm, we build the decision tree  $DT_1$  that separates the positive and negative tuples in  $J$ , as shown in Figure 2(c). (Ignore for now the decision tree  $DT_2$  in this figure.) The IEQ derived from  $DT_1$  is given by  $Q'_4$ :  $\pi_{name} \sigma_{stint \leq 1 \vee (stint > 1 \wedge HR > 50)}(Master \bowtie Batting)$ .  $\square$

#### 4.1.2 An Overview of TALOS

The overall approach of TALOS is outlined in Algorithm 1. TALOS takes as inputs a query  $Q$ , a database instance  $D$ , and the query result  $Q(D)$  of  $Q$  on  $D$ . The output of TALOS is a set  $R$  of IEQs of  $Q$ . First, TALOS derives the set of core relations of  $Q$  (line 1). For each schema subgraph  $G$  that contains the set of core relations, TALOS computes the join  $J$  of relations in  $G$  (line 4). TALOS next enumerates a set of decision trees for  $J$  and derives an IEQ of  $Q$  for each of the decision trees (lines 5 - 8). Finally, TALOS returns the collection  $R$  of IEQs of  $Q$ .

As the search space for IEQs can be very large, particularly with large complex database schema where each relation has foreign-key joins with other relations, the search

---

#### ALGORITHM 1: TALOS( $Q, D, Q(D)$ )

---

- 1 let  $S$  be the set of core relations of  $Q$
- 2 let  $R$  be the set of IEQs of  $Q$  which is initialized to be empty
- 3 **foreach** schema subgraph  $G$  that contains  $S$  **do**
- 4     let  $J$  be the join of the relations in  $G$
- 5     Enumerate a set of decision trees for  $J$
- 6     **foreach** decision tree  $DT$  for  $J$  **do**
- 7         Derive the IEQ  $Q'$  corresponding to  $DT$
- 8         Add  $Q'$  into  $R$
- 9 **return**  $R$

---

space could be restricted by providing users with certain control knobs in the form of hint/preference specifications. The following five are some examples of control knobs to limit the search space. The first three knobs enable users to constrain the complexity of the derived IEQs in terms of the number of relations/conjuncts/disjuncts while the last two knobs enable users to specify more schema-specific information (relations/attributes) to control the search space.

**(K1)** Constraining the number of relations in the from-clause of each IEQ to be in the range  $[n_{min}, n_{max}]$ . This control knob constrains the number of vertices of each schema subgraph  $G$  to be in the range  $[n_{min}, n_{max}]$ .

**(K2)** Constraining the number of disjunctive clauses in the query's selection condition to be in the range  $[l_{min}, l_{max}]$ . Recall that the selection condition of an IEQ is expressed in disjunctive normal form where each disjunctive clause is a conjunction of selection predicates. This is implemented by limiting the number of leaf nodes that are selected in the derived decision trees to be in the range  $[l_{min}, l_{max}]$ .

**(K3)** Constraining the number of conjunctions in each disjunctive clause to be in the range  $[h_{min}, h_{max}]$ . This is implemented by limiting the height of the derived decision trees to be in the range  $[h_{min}, h_{max}]$ .

**(K4)** Specifying a specific set of relations to be included in (or excluded from)  $Q'$ . This knob controls the space of schema subgraphs considered.

**(K5)** Specifying a specific set of attributes to be included in (or excluded from) the selection predicates in  $Q'$ . This knob controls the space of attributes considered for decision tree construction.

We discuss metrics to rank IEQs for a query in Section 7.

## 4.2 Challenges

There are two key challenges in adapting decision tree classifier for the QRE problem.

### 4.2.1 Challenge 1: At Least One Semantics

The first challenge concerns the issue of how to assign class labels in a flexible manner without over constraining the classification problem and limiting its effectiveness. Contrary to the impression given by the above simple class labeling scheme, the task of assigning class labels to  $J$  is actually a rather intricate problem. The reason is that multiple tuples in  $J_{in}$  can be projected to the same tuple in  $\pi_L(J_{in})$ . Recall that in the simple class labeling scheme described, a tuple  $t$  is labeled positive if and only if  $t \in J_{in}$ . However, note that it is possible to label only a subset of tuples  $J_{in}^+ \subseteq J_{in}$  as positive (with tuples in  $J - J_{in}^+$  labeled as negative), and yet achieve  $\pi_L(J_{in}^+) = \pi_L(J_{in})$  (without affecting the imprecision of  $Q'$ ). In other words, the simple scheme of labeling all tuples in  $J_{in}$  as positive is just one (extreme) option out of many other possibilities. For instance, in Example 4, it is possible to label tuples in  $J_{in}^+ = \{t_3, t_7\} \subset J_{in}$  as positive and tuples in  $J - J_{in}^+$  as negative and yet achieve  $\pi_{name}(J_{in}^+) = \pi_{name}(J_{in})$ .

We now discuss more precisely the various possibilities of labeling positive tuples in  $J$  to derive different  $sel(Q')$ . Let  $\pi_L(J_{in}) = \{t_1, \dots, t_k\}$ . Then  $J_{in}$  can be partitioned into  $k$  subsets,  $J_{in} = P_1 \cup \dots \cup P_k$ , where each  $P_i = \{t \in J_{in} \mid \text{the projection of } t \text{ on } L \text{ is } t_i\}$ . Thus, each  $P_i$  represents the subset of tuples in  $J_{in}$  that project to the same tuple in  $\pi_L(J_{in})$ . Define  $J_{in}^+$  to be a subset of tuples of  $J_{in}$  such that it consists of at least one tuple from each subset  $P_i$ . Clearly,  $\pi_L(J_{in}^+) = \pi_L(J_{in})$ , and there is a total of  $\prod_{i=1}^k (2^{|P_i|} - 1)$  possibilities for  $J_{in}^+$ . For a given  $J_{in}^+$ , we can derive  $sel(Q')$  using a data classifier by labeling the tuples in  $J_{in}^+$  as positive and the remaining tuples in  $J - J_{in}^+$  as negative.

Based on the above discussion on labeling tuples, each tuple in  $J$  can be classified as either a *bound tuple* or *free tuple* depending on whether there is any freedom to label the tuple. A tuple  $t \in J$  is a *bound tuple* if either (1)  $t \in J_{out}$ , in which case  $t$  must be labeled negative, or (2)  $t$  is the only tuple in some subset  $P_i$ , in which case  $t$  must certainly be

included in  $J_{in}^+$  and be labeled positive. Otherwise,  $t$  is a *free tuple*; i.e.,  $t$  is in some subset  $P_i$  that contains more than one tuple. Thus, a free tuple could be labeled positive or negative. In Example 4,  $J_{in}$  is partitioned into two subsets:  $P_1 = \{t_3, t_4\}$  and  $P_2 = \{t_7\}$ , where  $P_1$  and  $P_2$  contribute to the outputs “B” and “E”, respectively. The tuples in  $J_{out}$  (i.e.,  $t_1, t_2, t_5, t_6$ ) and  $P_2$  (i.e.,  $t_7$ ) are bound tuples, while the tuples in  $P_1$  are free tuples.

In contrast to the conventional classification problem where each record in the input data comes with a well defined class label, the classification problem formulated for QRE has the unique characteristic where there is some flexibility in the class label assignment. We refer to this property as *at-least-one semantics* in the sense that at least one tuple from each subset  $P_i$  must be labeled positive. For instance, to derive an IEQ of  $Q_4$ , at least one of the free tuples in  $P_1$  must be labeled positive. To the best of our knowledge, we are not aware of any work that has addressed the data classification problem involving at-least-one semantics.

An obvious approach to solve the at-least-one semantics variant is to map the problem into the traditional variant by first applying some arbitrary class label assignment that is consistent with the at-least-one semantics. In our experimental study, we compare against two such static labeling schemes, namely, (1) NI, which labels all free tuples as positive, and (2) RD, which labels a random non-empty subset of free tuples in each  $P_i$  as positive.<sup>5</sup> However, such static labeling schemes do not exploit the flexible class labeling opportunities to compute optimal node splits for decision tree construction.

To avoid the limitations of static labeling schemes, TALOS employs a novel *dynamic class labeling scheme* to compute optimal node splits for decision tree construction without having to enumerate an exponential number of combinations of class labeling schemes for the free tuples. Continuing with Example 4, if  $t_3$  is labeled positive and  $t_4$  is labeled negative, then  $DT_2$  in Figure 2(c) is a simpler decision tree constructed by partitioning  $J$  based on a selection predicate on attribute  $HR$ . The IEQ derived from  $DT_2$  is  $Q_4'' : \pi_{name} \sigma_{HR > 50} (Master \bowtie Batting)$ .

**Exactly- $k$  semantics.** In a more general SQL setting where the query result is a bag instead of a set (i.e., the “distinct” keyword is not used in the select-clause), the at-least-one semantics becomes *exactly- $k$  semantics* in the following sense: if there are  $m$  instances of a tuple  $t_i$  in the query result  $Q(D)$ , then the *exactly- $k$  semantics* requires that  $J_{in}^+$  must contain exactly  $m$  tuples from the subset  $P_i \subseteq J$  corresponding to  $t_i$ . For example, consider the following query:

<sup>5</sup> We also experimented with a scheme that randomly labels only one free tuple for each subset as positive, but its performance is worse than NI and RD.



```

 $Q_{4,k}$ : SELECT name
        FROM Master, Batting
        WHERE Master.pID = Batting.pID
              AND bats = "R" AND throws = "R".

```

We have  $Q_{4,k}(D) = \{B, B, D\}$ . Suppose that the schema subgraph  $G$  considered to derive the IEQs for  $Q_{4,k}$  contains *Master* and *Batting*. The output of  $J = \text{Master} \bowtie_{pID} \text{Batting}$  is shown in Figure 2(a) (similar to the relation  $J$  for query  $Q_4$ ). The partition corresponding to tuple "B" is  $P_1 = \{t_3, t_4\}$ . The exactly- $k$  semantics requires that  $J_{in}^+$  contains exactly two tuples from  $P_1$ . In this case, both  $t_3$  and  $t_4$  must be labeled positive.

For simplicity and without loss of generality, we will mainly focus on the at-least-one semantics in the following discussion. We defer the discussion about the exactly- $k$  semantics to Section 5.2.

#### 4.2.2 Challenge 2: Performance Issues

The second challenge concerns the performance issue of how to efficiently generate candidates for  $rel(Q')$  and optimize the computation of the single input table  $J$  required for the classification task. To improve performance, TALOS exploits join indices (discussed in Section 6) to avoid a costly explicit computation of  $J$  and constructs mapping tables to optimize decision tree construction. TALOS also uses another type of indices, domain indices (discussed in Section 8), to efficiently identify covering attributes when the input query is unknown.

## 5 Data Classification with At-Least-One Semantics

In this section, we address the first challenge of TALOS and present a novel approach for classifying data with the at-least-one semantics and the exactly- $k$  semantics in deriving SPJ-IEQs (Sections 5.1 & 5.2). We then present techniques to derive more expressive IEQs beyond the basic SPJ queries (Section 5.3).

### 5.1 Computing Optimal Node Splits

The main challenge for classification with the at-least-one semantics is how to optimize the node splits given the presence of free tuples that offers flexibility in the class label assignment. We present a novel approach that computes the optimal node split *without* having to explicitly enumerate all possible class label assignments to the free tuples. The idea is based on exploiting the flexibility offered by the at-least-one semantics.

Let us consider an initial set of tuples  $S$  that has been split into two subsets,  $S_1$  and  $S_2$ , based on a value  $v$  of a

numeric attribute  $A$  (the same principle applies to categorical attributes as well); i.e., a tuple  $t \in S$  belongs to  $S_1$  iff  $t.A \leq v$ . The key question is how to compute the optimal Gini index of this split without having to enumerate all possible class label assignments for the free tuples in  $S$  such that the at-least-one semantics is satisfied. Without loss of generality, suppose that the set of tuples in  $S$  is partitioned into  $m + 2$  subsets,  $P_1, \dots, P_{m+2}$ , where  $P_1, \dots, P_m$  are the partitioned subsets of free tuples (as described in Section 4.2) such that each  $|P_i| > 1$  and at least one tuple in each of them must be labeled positive;  $P_{m+1}$  is the set of bound tuples that are labeled positive; and  $P_{m+2}$  is the set of bound tuples that are labeled negative.

Let  $n_{i,j}$  denote the number of tuples in  $P_i \cap S_j$ ,  $i \in [1, m + 2]$ ,  $j \in \{1, 2\}$ . Thus, the total number of free tuples in  $S_j$ ,  $j \in \{1, 2\}$ , is given by  $\sum_{i=1}^m n_{i,j}$ . Let  $f_i$  denote the number of free tuples in  $S_i$ ,  $i \in \{1, 2\}$ , that are labeled positive to satisfy the at-least-one semantics. After the free tuples have been labeled, the total number of positive and negative tuples in each  $S_j$ ,  $j \in \{1, 2\}$ , are,  $n_{m+1,j} + f_j$  and  $n_{m+2,j} + \sum_{i=1}^m n_{i,j} - f_j$ , respectively. By Equation 1, the Gini index of each  $S_j$ ,  $j \in \{1, 2\}$ , is given by:

$$Gini(S_j) = 1 - \left( \frac{n_{m+1,j} + f_j}{\sum_{i=1}^{m+2} n_{i,j}} \right)^2 - \left( \frac{n_{m+2,j} + \sum_{i=1}^m n_{i,j} - f_j}{\sum_{i=1}^{m+2} n_{i,j}} \right)^2$$

By Equation 2, the Gini index of the split  $S$  into  $S_1$  and  $S_2$  is given by:

$$Gini(S_1, S_2) = \alpha_1 \cdot Gini(S_1) + \alpha_2 \cdot Gini(S_2), \quad (3)$$

where  $\alpha_j = (\sum_{i=1}^{m+2} n_{i,j}) / (\sum_{k=1}^2 \sum_{i=1}^{m+2} n_{i,k})$ ,  $j \in \{1, 2\}$ . After simplification,  $Gini(S_1, S_2)$  is of the form

$$Gini(S_1, S_2) = H(f_1, f_2) = c - (a_1 \cdot f_1 + b_1)^2 - (a_2 \cdot f_2 + b_2)^2, \quad (4)$$

where  $f_1$  and  $f_2$  are two variables to be optimized to minimize  $Gini(S_1, S_2)$ , and  $c$ ,  $a_1$ ,  $a_2$ ,  $b_1$ , and  $b_2$  are constants defined as follows:

$$\begin{aligned}
 a_j &= \frac{\sqrt{2 \cdot \alpha_j}}{\sum_{i=2}^{m+2} n_{i,j}}, j \in \{1, 2\} \\
 b_j &= \sqrt{\frac{\alpha_j}{2}} \cdot \left( \frac{n_{m+1,j} - n_{m+2,j} - \sum_{i=1}^m n_{i,j}}{\sum_{i=1}^{m+2} n_{i,j}} \right), j \in \{1, 2\} \\
 c &= \sum_{j \in \{1,2\}} \alpha_j - \alpha_j \cdot \frac{n_{m+1,j}^2 + (n_{m+2,j} + \sum_{i=1}^m n_{i,j})^2 - d}{(\sum_{i=1}^{m+2} n_{i,j})^2} \\
 d &= \frac{(n_{m+1,j} - n_{m+2,j} - \sum_{i=1}^m n_{i,j})^2}{4}.
 \end{aligned}$$

For each partition of free tuples  $P_i$ ,  $i \in [1, m]$ , we classify  $P_i$  as a  $SP_1$ -set (resp.  $SP_2$ -set) if  $P_i$  is completely contained in  $S_1$  (resp.  $S_2$ ); otherwise,  $P_i$  is a  $SP_{12}$ -set (i.e.,

$n_{i,1} > 0$  and  $n_{i,2} > 0$ ). Let  $T_j$  denote the minimum number of free tuples in  $S_j$  to be labeled positive,  $j \in \{1, 2\}$ . Since for each  $SP_j$ -set, at least one of its tuples must be labeled positive to satisfy the at-least-one constraint,  $T_j$  is equal to the number of  $SP_j$  sets.

The following result establishes that the optimal value of  $Gini(S_1, S_2)$  is computed by considering only five combinations of  $f_1$  and  $f_2$  values.

**Theorem 4** *The optimal value of  $Gini(S_1, S_2)$  is given by  $\min\{H1, H2, H3, H4, H5\}$ , where*

$$\begin{aligned} H1 &= H(\sum_{i=1}^m n_{i,1}, \sum_{i=1}^m n_{i,2}), \\ H2 &= H(\sum_{i=1}^m n_{i,1}, T_2), \quad H3 = H(T_1, \sum_{i=1}^m n_{i,2}), \\ H4 &= H(T_1, m - T_1), \quad \text{and } H5 = H(m - T_2, T_2). \quad \square \end{aligned}$$

The five combinations of  $f_1$  and  $f_2$  values considered in Theorem 4 actually correspond to the different ways to maximize the number of positive or negative tuples in  $S_1$  and  $S_2$ :  $H1$  maximizes the number of positive tuples in both  $S_1$  and  $S_2$ ;  $H2$  maximizes the number of positive tuples in  $S_1$  and maximizes the number of negative tuples in  $S_2$ ;  $H3$  maximizes the number of negative tuples in  $S_1$  and maximizes the number of positive tuples in  $S_2$ ; and finally,  $H4$  and  $H5$  correspond to the two possibilities to maximize the number of negative tuples in both  $S_1$  and  $S_2$ .

## 5.2 Updating Labels & Propagating Constraints

Once the optimal  $Gini(S_1, S_2)$  is determined for a given node split, we need to update the split of  $S$  by converting the free tuples in  $S_1$  and  $S_2$  to bound tuples with either positive/negative class labels. The details of this updating depend on which of the five cases the optimal Gini value was derived from, and are summarized by the last four columns in Table 1.

For case H1, which is the simplest case, all the free tuples in  $S_1$  and  $S_2$  will be converted to positive tuples. However, for the remaining cases, which involve maximizing the number of negative tuples in  $S_1$  or  $S_2$ , some of the free tuples may not be converted to bound tuples. Instead, the maximization of negative tuples in  $S_1$  or  $S_2$  is achieved by propagating another type of constraints, referred to as ‘‘exactly-one’’ constraints, to some subsets of tuples in  $S_1$  or  $S_2$ . Similar to the principle of at-least-one constraints, the idea here is to make use of constraints to optimize the Gini index values for subsequent node splits without having to explicitly enumerate all possible class label assignments. Thus, in Table 1, the fourth and fifth columns specify which free tuples are to be converted to bound tuples with positive and negative labels, respectively; where a ‘-’ entry means that no free tuples are to be converted to bound tuples. The sixth and seventh columns specify what subsets of tuples in  $S_1$  and  $S_2$ , respectively, are required to satisfy the exactly-one con-

straint; where a ‘-’ entry column means that no constraints are propagated to  $S_1$  or  $S_2$ .

We now consider the exactly-one constraint, a special case of exactly- $k$  semantics, and explain why it is necessary. An *exactly-one constraint* on a set of free tuples  $S'$  requires that exactly one free tuple in  $S'$  must become labeled as positive with the remaining free tuples in  $S'$  labeled as negative. Consider case H2, which is to maximize the number of positive (resp. negative) tuples in  $S_1$  (resp.  $S_2$ ). The maximization of the number of positive tuples in  $S_1$  is easy to achieve by converting all the free tuples in  $S_1$  to positive. In this way, the at-least-one constraints on the  $SP_1$ -sets and  $SP_{12}$ -sets are also satisfied. Consequently, for each  $SP_{12}$ -set  $P_i$ , all the free tuples in  $P_i \cap S_2$  can be converted to negative tuples to maximize the number of negative tuples in  $S_2$ . This will not violate the at-least-one constraint on the  $SP_{12}$ -set  $P_i$ . However, for a  $SP_2$ -set  $P_i$ , the constraint to maximize the number of negative tuples in  $P_i$  while satisfying the at-least-one semantics translates to an exactly-one constraint on  $P_i$ . Thus, for case H2, an exactly-one constraint is propagated to each  $SP_2$ -set in  $S_2$ , and no constraint is propagated to  $S_1$ . A similar reasoning applies to the other cases.

Therefore, while the at-least-one constraint is applied to each subset of free tuples  $P_i$  in the initial node split, the exactly-one constraint is applied to each  $P_i$  for subsequent node splits. This second variant of the node split problem can be optimized by techniques similar to what we have explained so far for the first variant. In particular, the exactly-one semantics means that  $f_1 + f_2 = m$  if there are  $m$  partitions of free tuples. Consequently, the optimization of the Gini index value becomes simpler and only needs to consider cases H4 and H5.

*Example 6* To illustrate how class labels are updated and how constraints are propagated during a node split, consider the following query on the baseball database  $D$ :  $Q_5 = \pi_{stint}(\sigma_{country='USA'} Master \bowtie_{pID} Batting)$ . Suppose that the weak-IEQ  $Q'_5$  being considered has  $rel(Q'_5) = \{Master, Batting\}$ . Let  $J = Master \bowtie_{pID} Batting$  (shown in Figure 2(a)). Since  $Q_5(D) = \{1, 2\}$ , we have  $J_0 = \{t_6, t_7\}$ ,  $P_1 = \{t_1, t_2, t_5\}$  (corresponding to  $stint = 2$ ), and  $P_2 = \{t_3, t_4\}$  (corresponding to  $stint = 1$ ). The tuples in  $J_0$  are labeled negative, while the tuples in  $P_1$  and  $P_2$  are all free tuples.

Suppose that the splitting attribute considered is ‘‘weight’’, and the optimal splitting value for ‘‘weight’’ is 72. The  $Gini(S_1, S_2)$  values computed (w.r.t. ‘‘weight = 72’’) for the five cases, H1 to H5, are 0.29, 0.48, 0.21, 0.4 and 0.4, respectively. Thus, the optimal value of  $Gini(S_1, S_2)$  is 0.21 (due to case H3). We then split tuples with  $weight \leq 72$  (i.e.,  $\{t_3, t_4, t_6, t_7\}$ ) into  $S_1$ , and tuples with  $weight > 72$  (i.e.,  $\{t_1, t_2, t_5\}$ ) into  $S_2$ . Thus,  $P_1$  is a  $SP_2$ -set while  $P_2$  is a  $SP_1$ -set. Since the optimal Gini index computed is due to case H3 (i.e., maximizing negative tuples in  $S_1$  and maxi-

Case	Number of free tuples to be labeled positive		Labeling of free tuples		Exactly-One Constraint Propagation	
	$f_1$	$f_2$	positive	negative	$S_1$	$S_2$
H1	$\sum_{i=1}^m n_{i,1}$	$\sum_{i=1}^m n_{i,2}$	$S_1 \cup S_2$	-	-	-
H2	$\sum_{i=1}^m n_{i,1}$	$T_2$	$S_1$	$SP_{12}$ -sets in $S_2$	-	$SP_2$ -sets
H3	$T_1$	$\sum_{i=1}^m n_{i,2}$	$S_2$	$SP_{12}$ -sets in $S_1$	$SP_1$ -sets	-
H4	$T_1$	$m - T_1$	-	$SP_{12}$ -sets in $S_1$	$SP_1$ -sets	All subsets
H5	$m - T_2$	$T_2$	-	$SP_{12}$ -sets in $S_2$	All subsets	$SP_2$ -sets

Table 1 Optimizing Node Splits

mizing positive tuples in  $S_2$ ), all the free tuples in  $S_2$  (i.e.,  $t_1$ ,  $t_2$  and  $t_5$ ) are labeled positive, and an exactly-one constraint is propagated to the set of tuples  $P_2 \cap S_1$  (i.e.,  $\{t_3, t_4\}$ ). □

**Handling Exactly- $k$  Semantics.** In the following, we discuss how to extend the above technique to handle the *exactly- $k$*  semantics. The exactly- $k$  semantics is required when there is a constraint on the number of instances of a specific tuple in the query result. Recall that the *exactly- $k$*  semantics applied on a set of free tuples  $S'$  requires exactly  $k$  of its free tuples to be labeled positive and the remaining free tuples in  $S'$  to be labeled negative.

Suppose that there are  $m$  partitions of free tuples  $P_1, \dots, P_m$  such that exactly  $k_i$  tuples from each of  $P_i$  need to be labeled positive, where  $k_i \in [1, |P_i|]$ . Thus, the total number of free tuples to be labeled positive to satisfy the exactly- $k_i$  constraint is given by  $f_1 + f_2 = \sum_{i=1}^m k_i$ . Since each  $P_i$  needs exactly  $k_i$  free tuples to be labeled positive, minimizing the number of free tuples in  $P_i \cap S_1$  to be labeled positive maximizes the number of free tuples in  $P_i \cap S_2$  to be labeled positive, and vice versa. Therefore, the minimum number of free tuples to be labeled positive in each  $P_i \cap S_j$  is given by  $\max\{0, k_i - n_{i,3-j}\}$ , and thus we have  $T_j = \sum_{i=1}^m \max\{0, k_i - n_{i,3-j}\}$ ,  $j \in \{1, 2\}$ . Consequently, the optimization of the Gini index value becomes simpler and needs to consider only two cases H'4 and H'5 (analogous to H4 and H5): case H'4 corresponds to  $f_1 = T_1$  and  $f_2 = \sum_{i=1}^m k_i - T_1$ ; while case H'5 corresponds to  $f_1 = \sum_{i=1}^m k_i - T_2$  and  $f_2 = T_2$ .

In summary, TALOS is able to efficiently compute the optimal Gini index value for each attribute split value considered without enumerating an exponential number of class label assignments for the free tuples.

### 5.3 Supporting More Expressive IEQs

Our discussion so far has focused on IEQs that are simple SPJ queries. In this section, we present techniques to derive more expressive IEQs beyond the basic SPJ queries. The increased expressiveness is important to broaden the range of applications of QRE. Specifically, we discuss the derivation of IEQs that belong to the following two additional query

#### ALGORITHM 2: TALOS-SPJU ( $T, D, k$ )

```

1 foreach column  $C_i$  of  $T$  do
2    $CA_i \leftarrow \{(A, L) \mid A \text{ is an attribute of some relation } R,$ 
3      $L = \{t.rid \mid t \in T, t.C_i \in \pi_A(R)\}, L \neq \emptyset\};$ 
4    $SCA \leftarrow \{((A_1, L_1), \dots, (A_k, L_k)) \mid (A_i, L_i) \in CA_i\};$ 
5    $Q \leftarrow$  an empty query;
6   while ( $SCA \neq \emptyset$ )  $\wedge$  ( $T \neq \emptyset$ ) do
7     Pick the element  $((A_1, L_1), \dots, (A_k, L_k))$  from  $SCA$ 
8     such that  $|L_1 \cap L_2 \dots \cap L_k \cap \pi_{rid}(T)|$  is largest among
9     all possible elements from  $SCA$  and remove this element
10    from  $SCA$ ;
11     $T' \leftarrow \{t \in T \mid t.rid \in L_1 \cap L_2 \dots \cap L_k\};$ 
12    Derive an IEQ  $Q'$  using the set of covering attributes
13     $\{A_1, \dots, A_k\}$  w.r.t. the input table  $T'$ ;
14    if  $Q'$  exists then
15       $Q \leftarrow Q \cup Q'$ ;
16       $T \leftarrow T - Q'(D)$ ;
17  return  $Q$ ;
```

fragments: SPJ with union (SPJU) and SPJ with aggregation (SPJA).

In the default mode of operation, TALOS will attempt to derive IEQs that belong to a simpler fragment before proceeding to the more complex fragments in the following order: SPJ, SPJU, and SPJA. In this way, simpler IEQs (which are preferred) are generated before the more complex IEQs.

#### 5.3.1 Deriving SPJU-IEQs

Since an SPJU-IEQ  $Q'$  is a union of some  $n$  number of SPJ queries ( $n > 1$ ),  $Q'$  is conceptually derived by partitioning  $T = Q(D)$  into  $n$  non-empty subsets,  $T = T_1 \cup \dots \cup T_n$ , where each  $T_i$  is produced by some SPJ-IEQ  $Q'_i$ . It is desirable to generate a succinct SPJU-IEQ  $Q'$  where  $n$  is minimized. However, minimizing  $n$  is an NP-hard problem as shown by Theorem 2.

TALOS uses a simple greedy heuristic to derive the SPJU-IEQ by generating the  $Q'_i$ 's iteratively, where the largest candidate  $T_i$  is selected at each iteration to derive an IEQ  $Q'_i$ . Algorithm 2 shows the heuristic used by TALOS to derive a SPJU-IEQ for an output table  $T$  containing  $k$  columns  $\{C_1, \dots, C_k\}$  with respect to a database  $D$ .

The algorithm first identifies all partially covering attributes for each column  $C_i$  of  $T$  (line 2). Here, we use  $t.A$  to denote the value of attribute  $A$  of a tuple  $t$ , and use  $rid$  to

refer to an implicit attribute of each tuple that represents the tuple’s row identifier (RID).  $(A, L) \in CA_i$  if  $A$  is a partially covering attribute of  $C_i$ , and  $L$  represents the set of tuples in  $T$  whose  $C_i$  column values are “covered” by attribute  $A$ .

Next, the algorithm enumerates all combinations of partially covering attributes for all the  $k$  output columns and stores them in  $SCA$  (line 3). The  $k$  attributes in each combination  $((A_1, L_1), \dots, (A_k, L_k))$  can potentially form a join relation that “covers” a partition  $T_i$  of  $T$  to generate an IEQ  $Q_i$ . TALOS uses the metric  $|L_1 \cap L_2 \dots \cap L_k \cap \pi_{rid}(T)|$  as an estimate of the cardinality of the partition  $T_i$  corresponding to the combination.

The algorithm generates the SPJ-IEQs iteratively using a greedy heuristic. At each iteration, it picks the combination from  $SCA$  that has the largest estimated partition cardinality (line 6). The target partition of tuples in  $T$  is given by  $T'$  (line 7). The algorithm then tries to generate a SPJ-IEQ  $Q'$  (using the previously discussed technique) for the output table  $T'$  using  $\{A_1, \dots, A_k\}$  as the set of projected attributes for  $Q'$  (line 8). If the IEQ  $Q'$  exists,  $Q'$  is added as a component to the final SPJU-IEQ  $Q$  (line 10), and  $T$  is updated by removing the tuples in  $T$  that are covered by the output of  $Q'$  on  $D$  (line 11). Note that  $Q'(D)$  may not be equal to  $T'$  if  $Q'$  is an approximate IEQ for  $T'$ . The algorithm terminates when all the combinations in  $SCA$  have been considered or if all the tuples in  $T$  have been covered.

### 5.3.2 Deriving SPJA-IEQs

For each output column  $C_i$  in the output table  $T$ , let  $Cover(C_i)$  denote the set of attributes in the database that completely covers  $C_i$ . TALOS classifies each  $C_i$  as an *aggregated output column* if  $Cover(C_i)$  is empty; otherwise,  $C_i$  is classified as a *non-aggregated output column*. As the name implies, each aggregated output column in  $T$  will be generated by applying some aggregation function on some database attribute. Recall that we do not consider arithmetic expressions in an aggregation operator’s argument in this paper.

Specifically, if  $T$  consists of  $k+m$  output columns where  $C_1, \dots, C_k$  are non-aggregated output columns and  $C_{k+1}, \dots, C_{k+m}$  are aggregated output columns, the SPJA-IEQ  $Q'$  derived by TALOS for  $T$  is of the form

```
“SELECT     $B_1, \dots, B_k, \mathcal{F}_1(A_1), \dots, \mathcal{F}_m(A_m)$ 
FROM       $J(G)$ 
WHERE      $p$ 
GROUP BY   $B_1, \dots, B_k$ ”.
```

Here,  $J(G)$  refers to the join relation (corresponding to some schema subgraph  $G$ ) that is formed by joining all the relations contained in  $G$  based on their foreign-key joins, and  $p$  is some selection predicate on  $J(G)$ . The attributes  $B_1, \dots, B_k, A_1, \dots, A_m$  are from  $J(G)$  such that each  $B_i \in Cover(C_i)$ ,  $i \in [1, k]$ ; and each  $\mathcal{F}_i, i \in [1, m]$  is

---

#### ALGORITHM 3: TALOS-SPJA ( $T, C_g, C_a, D$ )

---

```
1 foreach schema subgraph  $G$  that contains at least some
   relation  $R_g, R_g.A_g \in Cover(C_g)$  do
2   Compute the join relation  $J(G)$ ;
3   foreach attribute  $A_x$  in  $J(G)$ ,  $A_x \neq A_g$  do
4     foreach aggregation function  $\mathcal{F} \in \{COUNT, SUM,$ 
        $AVG, MIN, MAX\}$  do
5       if  $\mathcal{F}$  is applicable with  $A_x$  as aggregation
         attribute then
6         Label the tuples in  $J(G)$  wrt  $A_x$  &  $\mathcal{F}$  to
           derive an IEQ  $Q'$ ;
7         if  $Q'$  exists then
8           return  $Q'$ ;
9 return null;
```

---

either a COUNT, SUM, AVG, MIN, or MAX aggregation function. We refer to each  $A_i$  as an *aggregation attribute*.

To simplify the following presentation, we shall assume that  $T$  consists of exactly two columns with one non-aggregated column and one aggregated column. Algorithm 3 shows an overview of the heuristic used by TALOS to derive SPJA-IEQs for an output table  $T$  (with respect to database  $D$ ) consisting of a non-aggregated output column  $C_g$  and an aggregated output column  $C_a$ . To find candidate aggregation attributes for  $C_a$ , TALOS enumerates different schema subgraphs  $G$  that contain at least a relation (denoted by  $R_g$ ) such that  $R_g$  has an attribute  $A_g$  that completely covers the non-aggregated column  $C_g$ . For each such schema subgraph  $G$ , TALOS computes the join relation  $J(G)$  by joining all the relations in  $G$  based on the foreign-key joins among the relations in  $G$ . Each of the attributes in  $J(G)$  (except for  $A_g$ ) will be considered as a candidate aggregation attribute. In the following, we elaborate on how an SPJA-IEQ is derived for a given candidate aggregation attribute  $A_x$  and a candidate aggregate function  $\mathcal{F}$  by an appropriate labeling of the tuples in  $J(G)$  (line 6).

Suppose that  $T$  contains  $n$  tuples of the form  $(g_i, a_i)$ ,  $i \in [1, n]$ , where  $g_i$  and  $a_i$  are values of columns  $C_g$  and  $C_a$ , respectively.  $J(G)$  can be partitioned into  $(n+1)$  disjoint partitions:  $J(G) = P_0 \cup P_1 \cup \dots \cup P_n$ , where each  $P_i$ ,  $i > 0$ , is the subset of tuples in  $J(G)$  that has the value  $g_i$  for attribute  $A_g$ ; i.e.,  $P_i = \{t \in J(G) \mid t.A_g = g_i\}$ . Thus,  $P_0$  refers to the subset of tuples that do not contribute to any of the output tuples in  $T$ ; i.e.,  $\pi_{C_g}(T) \cap \pi_{C_g}(P_0) = \emptyset$ .

The tuples in  $J(G)$  are labeled as follows. Since none of the tuples in  $P_0$  can contribute to any output tuple in  $T$ , the tuples in  $P_0$  are all labeled as negative tuples and the remaining tuples are initially labeled as free tuples. For a candidate aggregation function  $\mathcal{F}$  being considered for the IEQ  $Q'$ , TALOS needs to determine a subset of tuples  $P'_i \subseteq P_i$  for each partition  $P_i$ ,  $i > 0$ , such that when the tuples in  $P'_i$  are labeled as positive and the tuples in  $P_i - P'_i$  are labeled as negative, the result of  $\mathcal{F}$  on each  $P'_i$  is  $a_i$ . In the following, we elaborate on how the  $P'_i$ ’s are selected (i.e.,

how the selection predicate  $p$  on  $J(G)$  is determined) for each of the three aggregation functions.

For the COUNT aggregation function, it is applicable for an aggregation attribute  $A_x$  if the values in the output column  $C_a$  are all natural numbers such that  $a_i \in \{0, 1, \dots, |P_i|\}$  for each  $(g_i, a_i) \in T$ . The semantics of the COUNT aggregation function requires that  $|P'_i| = a_i$  for each partition  $P_i$ ,  $i > 0$ , which can be solved by imposing the *exactly-k* constraint for labeling tuples as described in Section 5.

For the SUM and AVG aggregation functions, the problem of selecting each  $P'_i \subseteq P_i$  such that  $\mathcal{F}(P'_i) = a_i$  can be mapped to the subset-sum/subset-average problem, which is a well-known NP-hard problem [8]. TALOS uses standard pseudo-polynomial algorithms to solve the subset-sum and subset-average problems with time complexities of  $O(Kn)$  and  $O(Kn^2)$ , respectively [8], where  $K = a_i$  and  $n = |P_i|$ ; or approximation algorithms [8] to derive approximate IEQs with lower time complexity.

For the MIN aggregation function, for each  $P_i$ , there must exist at least one tuple  $t_m \in P_i$  such that  $t_m.A_x = a_i$ . We further partition  $P_i$  as follows:  $P_i = P_i^- \cup P_i^{alos} \cup P_i^{free}$  where  $P_i^- = \{t \in P_i \mid t.A_x < a_i\}$ ,  $P_i^{alos} = \{t \in P_i \mid t.A_x = a_i\}$ , and  $P_i^{free} = \{t \in P_i \mid t.A_x > a_i\}$ . Clearly, at least one tuple in  $P_i^{alos}$  must be labeled positive in order for the MIN function applying on  $P_i$  to return the value  $a_i$ . Likewise, all tuples in  $P_i^-$  must be labeled negative. Otherwise, the result of the MIN function on  $P_i$  will be smaller than  $a_i$ . Finally, tuples in  $P_i^{free}$  have the flexibility to be assigned positive or negative. The reason is that if any tuples in  $P_i^{free}$  are assigned positive class labels, the MIN function applying on  $P_i$  still returns the value  $a_i$ . A similar scheme will be applied if the aggregated function is MAX.

## 6 Performance Issues

In this section, we address the second challenge of TALOS and discuss several optimization techniques to speed up the derivation of IEQs. We first explain how TALOS adapts a well-known decision tree classifier for performing data classification in the presence of free tuples, whose class labels are not fixed. We then explain the performance challenges of deriving  $Q'$  when  $rel(Q')$  involves multiple relations, and present optimization techniques to address these issues. For ease of presentation and without loss of generality, the discussion here assumes weak IEQs.

### 6.1 Classifying Data in TALOS

We first give an overview of SLIQ [16], a well-known decision tree classifier, that we have chosen to adapt for TALOS. We then describe the extensions required by TALOS to handle data classification in the presence of free tuples. Finally,

val	row
A	1
B	2
C	3
D	4
E	5

(a)  $AL_{name}$

nid	cid	sid
1	0	0
1	-1	1
1	-1	1
1	0	0
1	0	0
1	1	2

(b)  $CL$

$r_M$	$r_B$	$r_T$
1	1	1
2	3	1
2	4	2
3	5	3
4	6	1
5	7	3

(c)  $J_{hub}$

$r_M$	$S_{r_j}$
1	{1}
2	{2, 3}
3	{4}
4	{5}
5	{6}

(d)  $M_{Master}$

Fig. 3 Example data structures for  $Q_4(D)$

we present a non-optimized, naive variant of TALOS. It is important to emphasize that our approach is orthogonal to the choice of the decision tree technique.

**Overview of SLIQ.** SLIQ is an efficient decision tree classifier designed for handling large, disk-resident training data. To construct a binary decision tree on a set of data records  $D$ , SLIQ uses two key data structures.

First, a sorted attribute list, denoted by  $AL_i$ , is pre-computed for each attribute  $A_i$  in  $D$ . Each  $AL_i$  can be thought of as a two-column table  $(val, row)$ , of the same cardinality as  $D$ , that is sorted in non-descending order of  $val$ . Each record  $r = (v, i)$  in  $AL_i$  corresponds to the  $i^{th}$  tuple  $t$  in  $D$ , and  $v = t.A_i$ . The sorted attribute lists are used to speed up the computation of optimal node splits. To determine the optimal node split w.r.t.  $A_i$  requires a single sequential scan of  $AL_i$ .

Second, a main-memory array called *class list*, denoted by  $CL$ , is maintained for  $D$ . This is a two-column table  $(nid, cid)$  with one record per tuple in  $D$ . The  $i^{th}$  entry in  $CL$ , denoted by  $CL[i]$ , corresponds to the  $i^{th}$  tuple  $t$  in  $D$ , where  $CL[i].nid$  is the identifier of leaf node  $N$ ,  $t \in D_N$ , and  $CL[i].cid$  refers to the class label of  $t$ .  $CL$  is used to keep track of the tuples location (i.e., in which leaf nodes) as leaf nodes are split.

**Class List Extension.** In order to support data classification with free tuples where their class labels are assigned dynamically, we need to extend SLIQ with the following modifications. The class list table  $CL(nid, cid, sid)$  is extended with an additional column “sid”, which represents a subset identifier, to indicate which subset (i.e.,  $P_i$ ) a tuple belongs to. This additional information is needed to determine the optimal Gini index values as discussed in the previous section. Consider a tuple  $t$  that is the  $i^{th}$  tuple in  $D$ , the  $cid$  and  $sid$  values in  $CL$  are maintained as follows. If  $t$  belongs to  $J_{out}$ , then  $CL[i].cid = 0$  and  $CL[i].sid = 0$ . If  $t$  is a bound tuple in  $P_j$ , then  $CL[i].cid = 1$  and  $CL[i].sid = j$ . Otherwise, if  $t$  is a free tuple in  $P_j$ , then  $CL[i].cid = -1$  and  $CL[i].sid = j$ .

*Example 7* Figure 3 illustrates the two data structures created to compute IEQs for  $Q_4(D)$  in Example 4. Figure 3(a) shows the attribute list created for attribute *Master.name*, and Figure 3(b) shows the initial class list created for the relation  $J = \text{Master} \bowtie_{pID} \text{Batting}$ , where all the records are in a single leaf node (with *nid* value of 1).  $\square$

**Naive TALOS (TALOS<sup>-</sup>).** Before presenting the optimizations for TALOS in the next section, let us first describe a non-optimized, naive variant of TALOS (denoted by TALOS<sup>-</sup>). Suppose that we are considering an IEQ  $Q'$  where  $rel(Q') = \{R_1, \dots, R_n\}$ ,  $n > 1$ , that is derived from some schema subgraph  $G$ . First, TALOS<sup>-</sup> joins all the relations in  $rel(Q')$  (based on the foreign-key joins represented in  $G$ ) to obtain a single relation  $J$ . Next, TALOS<sup>-</sup> computes attribute lists for the attributes in  $J$  and a class list for  $J$ . TALOS<sup>-</sup> is now ready to construct a decision tree  $DT$  to derive the IEQ  $Q'$  with these structures. The decision tree  $DT$  is initialized with a single leaf node consisting of the records in  $J$ , which is then refined iteratively by splitting the leaf nodes in  $DT$ . TALOS<sup>-</sup> terminates the splitting of a leaf node when (1) its tuples are either all labeled positive or all labeled negative; or (2) its tuples have the same attribute values w.r.t. all the splitting attributes. Finally, TALOS<sup>-</sup> classifies each leaf node in  $DT$  as positive or negative as follows: a leaf node is classified as positive if and only if the ratio of the number of its negative tuples to the number of its positive tuples is no greater than a threshold value given by  $\tau$ .<sup>6</sup> The selection condition of the IEQ  $Q'$  is then derived from the collection of positive leaf nodes in  $DT$  as follows. Each internal node in  $DT$  corresponds to a selection predicate on some attribute of  $J$ , and each root-to-positive-leaf path  $P_j$  in  $DT$  corresponds to a conjunctive predicate  $C_j$  on  $J$ . Thus, each decision tree enumerated for  $G$  yields a selection predicate for  $Q'$  of the form  $C_1 \text{ or } C_2 \dots \text{ or } C_\ell$ . In the event that all the leaf nodes in  $DT$  are classified as negative, the computation of  $Q'$  is not successful (i.e., there is no IEQ for  $rel(Q')$ ), and we refer to  $Q'$  as a *pruned IEQ*.

## 6.2 Optimizations

The naive TALOS described in the previous section suffers from two drawbacks. First, the overhead of computing  $J$  can be high; especially if there are many large relations in  $rel(Q')$ . Second, since the cardinality of  $J$  can be much larger than the cardinality of each of the relations in  $rel(Q')$ , building decision trees directly using  $J$  entails the computation and scanning of correspondingly large attribute lists, which further increases the computation cost. In the rest of this section, we present the optimization techniques used by TALOS to address the above performance issues.

<sup>6</sup> To generate only precise IEQs,  $\tau = 0$ . In our experiments, we set  $\tau = 1$  to derive a reasonable number of approximate IEQs.

**Join Indices & Hub Table.** To avoid the overhead of computing  $J$  from  $rel(Q')$ , TALOS exploits pre-computed join indices [30], which is a well-known technique for optimizing joins. For each pair of relations,  $R$  and  $R'$ , in the database schema that are related by a foreign-key join, its join index, denoted by  $I_{R,R'}$ , is a set of pairs of row identifiers referring to a record in each of  $R$  and  $R'$  that are related by the foreign-key join.

Based on the foreign-key join relationships represented in the schema subgraph  $G$ , TALOS computes the join of all the appropriate join indices for  $rel(Q')$  to derive a relation, called the *hub table*, denoted by  $J_{hub}$ . Computing  $J_{hub}$  is much more efficient than computing  $J$ , since there are fewer number of join operations (i.e., number of relevant join indices) and each join attribute is a single integer-valued column.

*Example 8* Consider again query  $Q_4$  introduced in Example 4. Suppose that we are computing IEQ  $Q'_4$  with  $rel(Q'_4) = \{\text{Master}, \text{Batting}, \text{Team}\}$ . Figure 3(c) shows the hub table,  $J_{hub}$ , produced by joining two join indices: one for  $\text{Master} \bowtie_{pID} \text{Batting}$ , and the other for  $\text{Batting} \bowtie_{team,year} \text{Team}$ . Here,  $r_M, r_B$ , and  $r_T$  refer to the row identifiers for *Master*, *Batting*, and *Team* relations, respectively.  $\square$

**Mapping Tables.** Instead of computing and operating on large attribute lists (each with cardinality equal to  $|J|$ ) as in the naive approach, TALOS operates over the smaller pre-computed attribute lists  $AL_i$  for the base relations in  $rel(Q')$  together with small *mapping tables* to link the pre-computed attribute lists to the hub table. In this way, TALOS only needs to pre-compute once the attribute lists for all the base relations, thereby avoiding the overhead of computing many large attribute lists for different  $rel(Q')$  considered.

Each mapping table, denoted by  $M_i$ , is created for each  $R_i \in rel(Q')$  that links each record  $r$  in  $R_i$  to the set of records in  $J_{hub}$  that are related to  $r$ . Specifically, for each record  $r$  in  $R_i$ , there is one record in  $M_i$  of the form  $(j, S)$ , where  $j$  is the row identifier of  $r$ , and  $S$  is a set of row identifiers representing the set of records in  $J_{hub}$  that are created from  $r$ .

*Example 9* Figure 3(d) shows the mapping table  $M_{Master}$  that links the *Master* relation in Figure 1 and  $J_{hub}$  in Figure 3(b). The record  $(2, \{2, 3\})$  in  $M_{Master}$  indicates that the second tuple in *Master* relation (with *pID* of  $P2$ ), contributed two tuples, located in the second and third rows, in  $J_{hub}$ .  $\square$

**Computing Class List.** We now explain how TALOS can efficiently compute the class list  $CL$  for  $J$  (without having explicitly computed  $J$ ) by using the attribute lists, hub table, and mapping tables. The key task in computing  $CL$  is to partition the records in  $J$  into subsets ( $J_0, P_1, P_2$ , etc.), as described in the previous section.

For simplicity and without loss of generality, assume that the schema of  $Q(D)$  has  $n$  attributes  $A_1, \dots, A_n$ , where each  $A_i$  is an attribute of relation  $R_i$ . TALOS first initializes  $CL$  with one entry for each record in  $J_{hub}$  with the following default values:  $nid = 1$ ,  $cid = 0$ , and  $sid = 0$ . For each record  $r_k$  that is accessed by a sequential scan of  $Q(D)$ , TALOS examines the value  $v_i$  of each attribute  $A_i$  of  $r_k$ . For each  $v_i$ , TALOS first retrieves the set of row identifiers  $RI_{v_i}$  of records in  $R_i$  that have a value of  $v_i$  for attribute  $R_i.A_i$  by performing a binary search on the attribute list for  $R_i.A_i$ . With this set of row identifiers  $RI_{v_i}$ , TALOS probes the mapping table  $M_i$  to retrieve the set of row identifiers  $JI_{v_i}$  of the records in  $J_{hub}$  that are related to the records referenced by  $RI_{v_i}$ . The intersection of the  $JI_{v_i}$ 's for all the attribute values of  $r_k$ , denoted by  $P_k$ , represents the set of records in  $J$  that can generate  $r_k$ . TALOS updates the entries in  $CL$  corresponding to the row identifiers in  $P_k$  as follows: (1) the  $sid$  value of each entry is set to  $k$  (i.e., all the entries belong to the same subset corresponding to record  $r_k$ ), and (2) the  $cid$  value of each entry is set to 1 (i.e., tuple is labeled positive) if  $|P_k| = 1$ ; otherwise, it is set to  $-1$  (i.e., it is a free tuple).

*Example 10* We illustrate how TALOS creates  $CL$  for query  $Q_4$ , which is shown in Figure 3(b). Initially, each row in  $CL$  is initialized with  $sid = 0$  and  $cid = 0$ . TALOS then accesses each record of  $Q_4(D)$  sequentially. For the first record (with name = "B"), TALOS searches  $AL_{name}$  and obtains  $RI_B = \{2\}$ . It then probes  $M_{Master}$  with the row identifier in  $RI_B$ , and obtains  $JI_B = \{2, 3\}$ . Since  $Q_4(D)$  contains only one attribute, we have  $P_1 = \{2, 3\}$ . The second and the third rows in  $CL$  are then updated with  $sid = 1$  and  $cid = -1$ . Similarly, for the second record in  $Q_4(D)$  (with name = "E"), TALOS searches  $AL_{name}$  and obtains  $RI_E = \{5\}$ , and derives  $JI_E = \{6\}$  and  $P_2 = \{6\}$ . The sixth row in  $CL$  is then updated with  $sid = 2$  and  $cid = 1$ .  $\square$

## 7 Ranking IEQs

As a query generally has many possible IEQs, it is useful to rank and prioritize the presentation of the derived IEQs to the user. Since our preference for simpler and more precise IEQs involves conflicting objectives, we present three reasonable criteria for ranking IEQs: a metric based on the Minimum Description Length (MDL) principle [24], and two metrics based on the F-measure [23].

### 7.1 Minimum Description Length

The Minimum Description Length (MDL) principle argues that all else being equal, the best model is the one that minimizes the sum of the cost of describing the data given the

model and the cost of describing the model itself. If  $M$  is a model that encodes the data  $D$ , then the total cost of the encoding,  $cost(M, D)$ , is defined as:  $cost(M, D) = cost(D|M) + cost(M)$ . Here,  $cost(M)$  is the cost to encode the model (i.e., the decision tree in our case), and  $cost(D|M)$  is the cost to encode the data given the model. We can rely on succinct tree-based representations to compute  $cost(M)$ . The data encoding cost,  $cost(D|M)$ , is calculated as the sum of classification errors. Thus, an IEQ with a lower MDL encoding cost is considered to be better. The details of the encoding computations are given elsewhere [16].

### 7.2 F-measure

We now present two useful metrics based on the popular F-measure [23] that represents the precision of the IEQs. The first variant follows the standard definition of F-measure: the F-measure for two IEQs  $Q$  and  $Q'$  is defined as  $F_m = \frac{2 \times |p_a|}{2 \times |p_a| + |p_b| + |p_c|}$ , where  $p_a = Q(D) \cap Q'(D)$ ,  $p_b = Q'(D) - Q(D)$ , and  $p_c = Q(D) - Q'(D)$ . We denote this variant as *F-measure* in our experimental study. Here, an IEQ with a higher F-measure value is considered to be a more precise and better query.

Observe that the first variant of F-measure is useful only for approximate IEQs, and is not able to distinguish among precise IEQs, as this metric gives identical values for precise IEQs since  $p_b$  and  $p_c$  are empty. To rank precise IEQs, we introduce a second variant, denoted by  $F_m^{est}$ , which relies on estimating  $p_a$ ,  $p_b$ , and  $p_c$  using some data probabilistic models (as opposed to using the actual values from the data set).  $F_m^{est}$  captures how the equivalence of queries is affected by database updates, and the IEQ with high  $F_m^{est}$  is preferable to another IEQ with low  $F_m^{est}$ . For simplicity, we use a simple independent model to estimate  $F_m^{est}$ ; other techniques such as the Bayesian model by Getoor and others [10] can be applied too. The second variant has the benefit that estimates, which are computed from a global distribution model, may more accurately reflect the true relevance of the IEQs than one computed directly from the data. This of course pre-supposes that future updates follow the existing data distribution.

The details on computing the F-measure ( $F_m$  and  $F_m^{est}$ ) of a pair of IEQs  $Q$  and  $Q'$  are discussed in Appendix C.

## 8 Handling Unknown Input Query

This section considers the problem of how to derive IEQs  $Q'$  given only  $D$  and the query result  $T = Q(D)$  without knowledge of the input query  $Q$ . The key challenge is how to efficiently determine candidate core relations for  $T$ . We first present the overall framework in Section 8.1 followed

**ALGORITHM 4:** TALOS( $D, T$ )

---

```

1 let  $C_1, \dots, C_k$  be the columns in  $T$ 
2 foreach column  $C_i$  in  $T$  do
3    $S_i \leftarrow \{R.A \text{ is an attribute in } D \mid R.A \text{ covers } C_i\}$ 
4 if all  $S_i$ 's are non-empty then
5   foreach  $R_{i_1}.A_{j_1} \in S_1, \dots, R_{i_k}.A_{j_k} \in S_k$  do
6      $\mathcal{R} \leftarrow \{R_{i_1}, R_{i_2}, \dots, R_{i_k}\}$ 
7     Derive SPJ-IEQs using Algorithm 1 with  $\mathcal{R}$  as the set
      of core relations
8   Derive SPJU-IEQs using Algorithm 2
9   Derive SPJA-IEQs using Algorithm 3

```

---

by a discussion of an optimization technique using domain indices in Section 8.2.

### 8.1 An Overview of TALOS

The overall approach of TALOS is outlined in Algorithm 4 which takes as input a database  $D$  and an input table  $T$  to derive a set of IEQs  $Q'$  such that  $Q'(D) = T$ . Recall from Section 5.3 that in the default mode of TALOS, it will attempt to derive IEQs starting from the simplest SPJ IEQs to the more complex SPJU and SPJA IEQs.

To generate SPJ IEQs, TALOS first determines the set of schema attributes  $S_i$  that covers each column  $C_i$  in  $T$  (line 3). If each  $S_i$  is non-empty, TALOS then enumerates all possible sets of core relations from  $S_1, \dots, S_k$ , and invokes Algorithm 1 to derive SPJ IEQs for each set of core relations  $\mathcal{R}$  (lines 4 to 7). Note that  $\mathcal{R}$  is constructed such that each column in  $T$  is covered by at least some attribute of some relation in  $\mathcal{R}$ . SPJU- and SPJA-IEQs for  $T$  are generated using Algorithms 2 and 3, respectively (lines 8 & 9).

### 8.2 Optimization: Domain Indices

We now describe how TALOS uses a simple but effective indexing technique, called *domain indices*, to optimize the identification of covering attributes.

Recall that the definition of a covering attribute (in Section 2) is based on set semantics which ignores the presence of duplicate column values. That is, it is possible for an attribute  $A$  in relation  $R$  to cover an attribute  $C$  in relation  $T$  and yet there is some value  $v$  that occurs  $m$  times in  $T.C$  and  $n$  times in  $R.A$  with  $m > n$  and  $n \geq 1$ . This weaker definition is, however, sufficient for the purpose of identifying candidate core relations because after joining relation  $R$  with other relations in the schema subgraph to compute the join relation  $J$ , the frequency of a value  $v$  in  $J.A$  could be higher than that in  $R.A$ . In this section, for simplicity and without loss of generality, we assume that each column  $C$  of  $T$  does not contain any duplicate values; otherwise, we simply pre-process  $C$  to eliminate duplicates.

To determine whether a column  $A$  in relation  $R$  covers a column  $C$  in  $T$ , a straightforward approach is to compute the antijoin  $T \triangleright_{T.C=R.A} R$ . Then  $A$  covers  $C$  iff the antijoin result is empty. Thus, all the covering attributes for  $C$  could be identified by performing this procedure for every database attribute that has the same domain as  $C$ . We refer to this naive solution as TALOS<sup>-</sup>.

To avoid the cost of computing multiple antijoins, TALOS uses a simple yet effective indexing technique called *domain indices*. Unlike a conventional index which is defined on attribute(s) within a single relation, a domain index is defined on all the attributes in the database that have the same attribute domain.

For each attribute domain  $d$ , TALOS maintains a three-column mapping table  $M_d(v, attr, count_v)$ , where  $v$  is a value of type  $d$  in the database,  $attr$  is the schema attribute that contains  $v$  in its column, and  $count_v$  is the frequency of  $v$  in  $attr$ 's column. This table has one composite key consisting of  $v$  and  $attr$ , which is indexed by a B<sup>+</sup>-tree index, that we refer to as the *domain index* for  $d$ . The  $count_v$  field is used to facilitate index maintenance:  $count_v$  is updated accordingly when database records are modified, and when  $count_v$  drops to 0, the corresponding value  $v$  is removed from its mapping table  $M_d$  and domain index.

With the mapping tables and domain indices, TALOS identifies the covering attributes for a column  $C$  (with domain  $d$ ) in the output table  $T$  by computing the result  $\gamma_{attr, count(v)} \pi_{attr, v}(T \bowtie_{T.C=M_d.v} M_d)$ . Essentially, TALOS first joins  $T$  and  $M_d$  on  $T.C = M_d.v$  to find out the set of values in  $T.C$  that are covered by each database attribute  $attr$  that has the same domain as  $C$ , and then performs a group-by aggregation to count the number of distinct attribute values in  $T.C$  that are covered by each  $attr$ . Thus, an attribute  $A$  covers  $C$  iff the aggregated result contains the tuple  $(A, acount)$  and the number of distinct values of  $C$  is equal to  $acount$ . Here,  $acount$  represents the number of distinct attribute values in  $C$  that are covered by  $A$ . The implementation details of TALOS<sup>-</sup> and TALOS to identify covering attributes are discussed in Appendix D.

## 9 Handling Multiple Database Versions

In this section, we consider a more general setting of the QRE problem where the input consists of multiple versions of database instead of a single database. Specifically, we consider the following two practical scenarios for this generalization.

### 9.1 Multiple Database Versions & Single Unknown Result

In data exploratory/analysis scenarios, it is typically the case that the querying time when a user requests to reverse-engineer



a precise query  $Q'$  for a result table  $T$  occurs much later than the time when  $T$  was actually produced. Thus, it may not be meaningful or possible to derive  $Q'$  from the current version of the database as this could be very different from the version that  $T$  was generated from. Specifically, given a result table  $T$  and a sequence of database versions  $\langle D_1, D_2, \dots, D_\ell \rangle$ , where  $D_i$  is a more recent version than  $D_j$  if  $i > j$ , a specific goal may be to determine the most recent database version  $D_i$ ,  $i \in [1, \ell]$ , and query  $Q'$  such that  $Q'(D_i) = T$ . Depending on the applications, other variations of the problem (e.g., finding the earliest database version or all versions) are also possible. In this section, we focus on the problem of deriving IEQs with respect to the most recent database version. The performance challenge is how to efficiently determine both  $D_i$  as well as  $Q'$  for a given result  $T$ .

The most straightforward solution is to apply the previously described solution developed for a single database version to one database version at a time starting from the most recent version  $D_\ell$  and work progressively “backwards” to the next recent version and so on until an IEQ is derived (or none is found). However, if the multiple database versions are not stored independently but are instead organized using a *reference version* (either the earliest or latest version) together with a sequence of forward/backward deltas between successive versions (e.g., [27]), then it is possible to apply well-known join view maintenance techniques [5, 19] to optimize the IEQ derivations.

In this section, we assume, without loss of generality, that the database versions are stored using the backward delta storage organization. That is, given a sequence of database versions  $D_1, D_2, \dots, D_\ell$ , the database stores the most recent version  $D_\ell$  together with  $\delta_{\ell(\ell-1)}, \dots, \delta_{21}$ , where each  $D_j$  is derived from  $D_i$  and  $\delta_{ij}$ . For simplicity, we assume that each tuple update operation is modeled by a pair of tuple delete and insert operations. Thus, each delta  $\delta_{ij}$  consists of a set of tuple insert and delete operations. We use the notation “ $X \oplus \delta X$ ” to denote applying the insert/delete “delta” tuples in  $\delta X$  to update  $X$  where  $X$  is a database version or a relation. In the backward delta storage organization, we have  $D_i = D_{i+1} \oplus \delta_{(i+1)i}$ .

Assume that the current database version being considered is  $D_x$ ; i.e., the database versions  $D_\ell, D_{\ell-1}, \dots, D_{x+1}$  have already been considered but without any IEQs derived from them. The task at hand is to try to derive IEQs from  $D_x$ . An obvious approach is to first compute  $D_x$  from  $D_{x+1}$  and  $\delta_{(x+1)x}$ , and then try to derive IEQs from the materialized  $D_x$ . This simple approach, however, does not leverage any of the computations performed for previously considered database versions.

There are two computations for the current database version  $D_x$  that could be optimized by exploiting the computations performed for the previous database version  $D_{x+1}$ .

The first one is the computation of covering attributes and the second one is the computation of join relations. As both optimizations are based on the same ideas, we shall focus on the first optimization in the rest of this section and briefly discuss the second optimization at the end of this section.

Recall from Section 8.2 that TALOS maintains a mapping table  $M_d(v, attr, count_v)$  for each attribute domain  $d$ . The mapping table  $M_d$  is used to determine the covering attributes for a column  $C$  (with domain  $d$ ) in the output table  $T$  by computing  $\gamma_{attr, count(v)} TM_d$ , where  $TM_d = \pi_{attr, v}(T \bowtie_{T.C=M_d.v} M_d)$ . Here,  $TM_d$  determines the set of attribute values of  $T.C$  that are covered by each schema attribute  $attr$  with the same domain as  $T.C$ .

Let  $M_d^i$  and  $TM_d^i$  denote the relations  $M_d$  and  $TM_d$  with respect to database version  $D_i$ . The goal of our optimized approach is to compute  $TM_d^x$  from  $TM_d^{x+1}$  and  $\delta_{(x+1)x}$  without having to explicitly derive  $D_x$  and  $M_d^x$ . To facilitate this, we need to extend the schema of  $TM_d^i$  with an additional attribute  $count_v$  to  $TM_d^i(attr, v, count_v)$  where  $count_v$  indicates the number of times the value  $v$  occurs in the database attribute  $attr$ . Thus, when we start by considering database version  $D_\ell$ ,  $TM_d^\ell$  is computed as follows:

$$TM_d^\ell = \pi_{attr, v, count_v}(T \bowtie_{T.C=M_d^\ell.v} M_d^\ell).$$

Subsequently, when we consider database version  $D_x$ ,  $x < \ell$ ,  $TM_d^x$  is derived from  $TM_d^{x+1}$  and  $\delta_{(x+1)x}$ , which we explain in the following.

The derivation of  $TM_d^x$ ,  $x < \ell$ , consists of two steps. In the first step, we compute  $\delta M_d$ , which represents the delta tuples required to derive  $M_d^x$  from  $M_d^{x+1}$ . Specifically,  $\delta M_d$ , which is derived from  $\delta_{(x+1)x}$ , is a set of tuples of the form  $(attr, v, count_v)$  representing the net effect of the insert/delete delta tuples in  $\delta_{(x+1)x}$  on the frequency of occurrence (given by  $count_v$ ) of attribute values (indicated by value  $v$  of attribute  $attr$ ) that occur in the delta tuples. For example, if  $\delta_{(x+1)x}$  consists of  $m$  insert tuples that have a value of 10 for attribute  $A$  (with domain  $d$ ) and  $n$  delete tuples that have a value of 10 for attribute  $A$ , then  $(A, 10, m - n) \in \delta M_d$ . Tuples in  $\delta M_d$  with a value of zero for  $count_v$  are removed. Note that it is possible to derive  $M_d^x$  from  $M_d^{x+1}$  and  $\delta M_d$ .

Finally, in the second step, we compute  $TM_d^x$  from  $TM_d^{x+1}$  and  $\delta M_d$  as follows:

$$TM_d^x = \sigma_{count_v > 0} (\pi_{attr, v, count_v = TM_d^{x+1}.count_v + \delta TM_d.count_v} (TM_d^{x+1} \bowtie_{v, attr} \delta TM_d)), \text{ where}$$

$$\delta TM_d = \pi_{attr, v, count_v}(T \bowtie_{T.C=\delta M_d.v} \delta M_d).$$

$\delta TM_d$  represents the delta tuples required to derive  $TM_d^x$  from  $TM_d^{x+1}$ . The full outer join between  $TM_d^{x+1}$  and  $\delta TM_d$  (with equality join predicates on both  $attr$  and  $v$  attributes) is used to combine<sup>7</sup> the delta tuples in  $\delta TM_d$  with  $TM_d^{x+1}$ .

<sup>7</sup> Note that in the SQL implementation, null count values are first converted to zero values before being added.

Since  $\delta_{(x+1)x}$  is typically much smaller than  $D_x$ , the presented approach to compute  $\delta M_d$  and  $\delta TM_d$  to derive  $TM_d^x$  is less costly than the straightforward approach to compute  $D_x$  and  $M_d^x$  to derive  $TM_d^x$ .

Another area that could benefit from a similar optimization is the computation of the join relation  $J(G)$  corresponding to a selected schema subgraph  $G$ . Let  $J_i(G)$  denote the join relation for database version  $D_i$ . The direct approach to compute  $J_x(G)$  is to first derive  $D_x$  from  $\delta_{(x+1)x}$  and then compute  $J_x(G)$  from  $D_x$ . However, this can be optimized by instead deriving a set of delta tuples, denoted by  $\delta J(G)$ , from  $\delta_{(x+1)x}$  and computing  $J_x(G)$  from  $\delta J(G)$  and the previously computed  $J_{x+1}(G)$ . As the ideas are similar to the first optimization, we omit the elaboration of the second optimization.

## 9.2 Multiple Database and Result Versions

In the second scenario, the input to the QRE problem is a sequence of database versions and result pairs  $(D_1, T_1)$ ,  $(D_2, T_2)$ ,  $\dots$ ,  $(D_\ell, T_\ell)$ ; where each  $T_i$  is the result of the *same* unknown query  $Q$  on database version  $D_i$  (i.e.,  $T_i = Q(D_i)$ ). We assume that all the databases  $D_i$  have the same schema.

For example, the  $T_i$ 's could correspond to weekly reports generated by the same query on weekly versions of the database, or  $Q$  could be a continuous long standing query that is run periodically on different snapshots of the database. In this more general setting with multiple database and result versions, the challenge is how to efficiently reverse-engineer a query  $Q'$  such that  $Q'(D_i) = T_i$  for each  $i \in [1, \ell]$ .

The most straightforward solution to this problem is to independently derive a set of IEQs  $S_i$  for each database and result pair  $(D_i, T_i)$  and then intersect all the  $S_i$ 's to compute the set of IEQs for  $Q$ . We refer to this direct approach as  $TALOS^-$ .

In the following, we present a more efficient solution (denoted by  $TALOS$ ) that considers all database and result pairs collectively to derive the IEQs. The tradeoff of this more efficient approach is that it could miss even more precise IEQs compared to  $TALOS^-$ .

Similar to  $TALOS^-$ ,  $TALOS$  starts by identifying the set of covering attributes (and hence candidate core relations) for the columns in the output tables. Next, for each schema subgraph  $G$  that contains a set of candidate core relations,  $TALOS$  computes a join relation  $J_i(G)$  for each database version  $D_i$  by joining all the relations in  $D_i$  that are included in  $G$  based on their foreign-key joins. Note that if the multiple database versions are not stored independently but are instead organized using a reference version as discussed in the previous section, our previously discussed optimization to derive  $J_i(G)$  is applicable here as well. At this

Table	# Tuples
<i>adult</i>	45, 222
<i>Master</i>	16, 639
<i>Batting</i>	88, 686
<i>Pitching</i>	37, 598
<i>Fielding</i>	128, 426
<i>Salaries</i>	18, 115
<i>Team</i>	2, 535
<i>Manager</i>	3, 099

(a) Adult &amp; Baseball

Table	# Tuples
<i>lineitem</i>	6000000
<i>order</i>	1500000
<i>partsupp</i>	800000
<i>part</i>	200000
<i>customer</i>	150000
<i>supplier</i>	10000
<i>nation</i>	25

(b) TPC-H

**Table 2** Table sizes (number of tuples)

point,  $TALOS$  has computed a collection of join relations  $J_1(G), \dots, J_\ell(G)$  wrt  $G$ . Instead of deriving IEQs independently for each  $J_i(G)$  as in  $TALOS^-$ ,  $TALOS$  derives IEQs from a single unified join relation  $J(G)$  that is generated by combining all the join relations; i.e.,  $J(G) = \bigcup_{i=1}^{\ell} J_i(G)$ .

For convenience, let  $T$  denote the combined result table; i.e.,  $T = \bigcup_{i=1}^{\ell} T_i$ . For each tuple  $t_j \in T$ , let  $X(t_j) = \{i \in [1, \ell] | t_j \in T_i\}$ , and let  $P_{i,j}$  denote the maximal subset of tuples in  $J_i(G)$  that form  $t_j$  when they are projected on  $proj(Q)$ . Note that each  $P_{i,j} \subseteq J(G)$  since each  $J_i(G) \subseteq J(G)$ .

The tuples in  $J(G)$  are labeled as follows. The tuples that do not contribute to any result tuples in  $T$  (i.e.,  $J(G) - \bigcup_{t_j \in T, i \in [1, \ell]} P_{i,j}$ ) are labeled negative and the remaining tuples in  $J(G)$  are labeled free. To guarantee that  $t_j \in Q'(D_i)$  for each  $t_j \in T$  and  $i \in X(t_j)$ , we need to ensure that at least one free tuple in each  $P_{i,j}$  will be eventually labeled positive for the decision tree constructed with respect to  $J(G)$ . This can be achieved by imposing the at-least-one semantics on the set  $\bigcap_{i \in X(t_j)} P_{i,j} - \bigcup_{i \in [1, \ell], i \notin X(t_j)} P_{i,j}$  for the decision tree construction.

In case we need to derive an SPJA-IEQ, the same idea can be applied. For instance, to derive an SPJA-IEQ with SUM/AVG aggregation function, instead of applying the at-least-one semantics on the corresponding subset, i.e.,  $\bigcap_{i \in X(t_j)} P_{i,j} - \bigcup_{i \in [1, \ell], i \notin X(t_j)} P_{i,j}$ , we solve a subset-sum/subset-average problem defined on this set of tuples and label the tuples accordingly.

## 10 Experimental Study

This section presents our experimental study to evaluate the performance of our proposed approach,  $TALOS$ , for computing IEQs. We first examine its effectiveness in handling at-least-one semantics (Section 10.2) and generating IEQs (Section 10.3) and then evaluate the efficiency of its optimizations (Sections 10.4 & 10.5).

## 10.1 Methodology

**Algorithms.** To evaluate the effectiveness of TALOS in dynamically assigning the class labels for free tuples, we compare TALOS against two static class labeling techniques, denoted by NI and RD. Recall that NI labels all the free tuples as positive, while RD labels a random number of at least one tuple from each subset of free tuples as positive.

We also examine the effectiveness of our proposed optimizations by comparing against a non-optimized naive variant of TALOS (denoted by TALOS<sup>-</sup>).

**Data sets and Queries.** We used two real and one synthetic data sets of different sizes for our experiments: Adult, Baseball, and TPC-H with a scale factor of 1.0. The sizes of these data sets are shown in Table 2 and the test queries are given in Table 3 with the cardinality of their results shown in the last column. We also run experiments on three additional data sets: basket ball data set<sup>8</sup>, a single-relation data set containing information about laptops that are crawled from ebay<sup>9</sup>, and TPC-H with a scale factor of 5.0. The results for these additional data sets are given in Appendix F.

The Adult data set, from the UCI Machine Learning Repository, is a single-relation data set that has been used in many classification works. We use this data set to illustrate the utility of IEQs for the simple case where both the input query  $Q$  as well as the output IEQ  $Q'$  involve only one relation. There are five queries (A1 - A5) on this data set.<sup>10</sup>

The baseball data set is a more complex, multi-relation database that contains Batting, Pitching, and Fielding statistics for Major League Baseball from 1871 through 2006 created by Sean Lahman. The queries used for this data set ( $B_1 - B_6$ ) are common queries that mainly relate to baseball players' performance.

The TPC-H data set is a widely used benchmark. We use the TPC-H data set with a size of 1GB, and seven test queries  $T_1 - T_7$ .

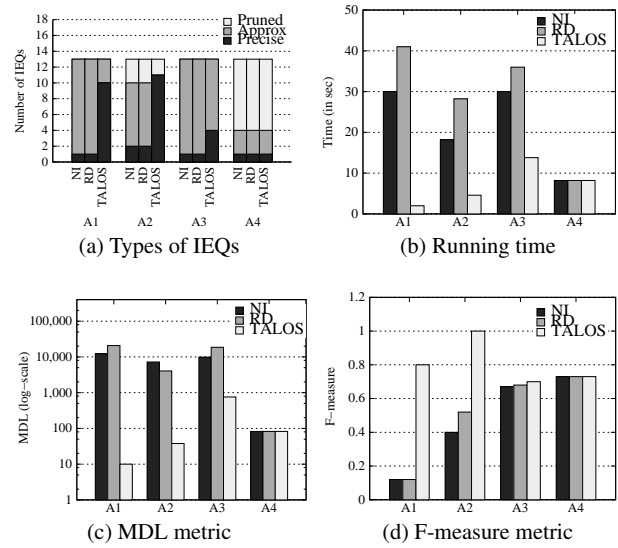
Queries  $A_5, B_5, B_6, T_5, T_6, T_7$  are examples of queries in more expressive fragments (SPJU, SPJA), while the remaining queries are examples of SPJ queries.

**Control Knobs.** In our experiments, we set the following two control knobs: (K1) the number of relations in the from-clause of the IEQs, and (K2) the number of selection predicates in each conjunction of  $sel(Q')$ . Table 4 shows the range of values set for these knobs for the queries in Adult, Baseball and TPC-H data sets; these values are chosen to ensure that the generated IEQs are not too complicated.

<sup>8</sup> <http://www.basketballreference.com/>.

<sup>9</sup> <http://www.ebay.com/>.

<sup>10</sup> We use *gain*, *ms*, *edu*, *loss*, *nc*, *hpw*, and *rs*, respectively, as abbreviations for capital-gain, marital-status, number of years of education, capital-loss, native-country, hours-per-week, and relationship.



**Fig. 4** Comparison of TALOS, NI and RD (Adult data set)

Control Knob	Adult	Baseball	TPC-H
(K1) # relations in the from-clause	[1, 1]	[2, 4]	[2, 3]
(K2) # selection predicates in each conjunction of the where-clause	[0, 5]	[0, 5]	[0, 5]

**Table 4** The control knob values

**Testing Platforms.** Our experiments were conducted on a machine running 64-bit Ubuntu OS with four-core Intel(R) Core(TM) i7 CPU Q820 @1.73GHz CPU and 4GB RAM. All the algorithms were coded in C++ and compiled and optimized with gcc; and data is managed using MySQL Server 5.0.51. Each experimental timing reported is an average timing over 5 different runs.

## 10.2 Effectiveness of At-Least-One Semantics

We compare TALOS against the two static class-labeling schemes, NI and RD, in terms of their efficiency as well as the quality of the generated IEQs. Recall that NI labels all the free tuples as positive, while RD labels a random number of at least one tuple from each subset of free tuples as positive. The comparison focuses on SPJ test queries for the Adult and Baseball datasets.<sup>11</sup>

For the Adult data set, only weak IEQs are considered as all the labeling schemes would have produced the same

<sup>11</sup> As all the class-labeling schemes are based on the same framework to derive SPJU-IEQs which differ only in how they label free tuples to derive SPJ-IEQs that form the subqueries for SPJU-IEQs, we do not consider SPJU queries here. For SPJA queries, since the free tuples are assigned fixed class labels to satisfy the aggregation conditions before being classified, all the three schemes would have returned the same results for SPJA-IEQs; therefore, SPJA queries are not considered. For the TPC-H queries, it turns out that all the free tuples are bound and therefore all the three schemes return the same set of IEQs; therefore, we do not report results for TPC-H datasets.

	Query	$ Q(D) $
$A_1$	$\pi_{nc} \sigma_{occ} = \text{"Armed-Force"} (adult)$	1
$A_2$	$\pi_{edu, occ} (\sigma_{ms} = \text{"Never-married"} \wedge 64 < age < 68 \wedge race = \text{"White"} \wedge gain > 500 \wedge sex = \text{"F"} (adult))$	4
$A_3$	$\pi_{nc, gain} (\sigma_{ms} = \text{"Never married"} (adult))$	137
$A_4$	$\pi_{id} (\sigma_{SKY-LINE}(gain \text{ MAX}, age \text{ MIN}) (adult))$	4
$A_5$	$\pi_{age, AVG}(hpw) \sigma_{nc} = \text{"US"} (adult)$	1
$B_1$	$\pi_{M.name} (\sigma_{team} = \text{"ARI"} \wedge year = 2006 \wedge HR > 10 (Master \bowtie Batting))$	7
$B_2$	$\pi_{M.name} (\sigma_{sum}(HR) > 600 (Master \bowtie Batting))$	4
$B_3$	$\pi_{M.name} (\sigma_{SKY-LINE}(HR \text{ MAX}, SO \text{ MIN}) (Master \bowtie Batting))$	35
$B_4$	$\pi_{M.name, T.year, T.rank} (\sigma_{team} = \text{"CIN"} \wedge 1982 < year < 1988 (Manager \bowtie Team))$	6
$B_5$	$\pi_{Master.name, Manager.ID} (Master \bowtie Manager)$ $\cup \pi_{Master.name, B.teamID} \sigma_{year=2006} (Master \bowtie Batting)$	583
$B_6$	$\pi_{name, SUM}(salary) (Master \bowtie Salaries)$	3540
$T_1$	$\pi_{S.name, N.name} \sigma_{S.acctbal} > 4000 \wedge N.regionkey < 4 (supplier \bowtie nation)$	4383
$T_2$	$\pi_{C.name, N.name} \sigma_{C.acctbal} > 3000 (customer \bowtie nation)$	95264
$T_3$	$\pi_{P.name, S.name} \sigma_{PS.avaiqty} > 3000 \wedge S.acctbal > 9500 (part \bowtie partsupp \bowtie supplier)$	24672
$T_4$	$\pi_{O.clerk, L.extendedprice} \sigma_{L.quantity} < 2 \wedge orderstatus = \text{"P"} (lineitem \bowtie order)$	3719
$T_5$	$\pi_{C.name, N.name} \sigma_{mktsegment} = \text{"BUILDING"} \wedge C.acctbal > 100 (customer \bowtie nation)$ $\cup \pi_{S.name, N.name} \sigma_{S.acctbal} > 4000 (supplier \bowtie nation)$	32554
$T_6$	$\pi_{P.name, SUM}(PS.supplycost) \sigma_{P.name} \sigma_{PS.retailprice} > 2000 (part \bowtie partsupp)$	4950
$T_7$	$\pi_{C.name, SUM}(O.totalprice) \sigma_{C.name} \sigma_{C.acctbal} > 3000 (order \bowtie customer)$	63533

**Table 3** Test queries for Adult ( $A_1 - A_5$ ), Baseball ( $B_1 - B_6$ ) & TPC-H ( $T_1 - T_7$ )

Query	Average height			Average size		
	NI	RD	TALOS	NI	RD	TALOS
$A_1$	14.9	19.8	2.1	5304	9360	4.7
$A_2$	16.1	21.8	6.5	3224	2970	19.2
$A_3$	16	20	12	4386	8103	334

**Table 5** Comparison of decision trees for NI, RD, and TALOS

strong IEQs due to the absence of free tuples in a single-relation database for computing strong IEQs.

Figure 4(a) compares the number of the three types of IEQs (precise, approximate, pruned) generated for Adult data set. As the number of decision trees considered is the same for all the schemes, the total number of IEQs generated by the schemes are the same. Observe that the number of precise IEQs from TALOS is consistently larger than that from NI and RD. The better quality IEQs from TALOS is due to its flexible dynamic class label assignment scheme for free tuples which provides more opportunities to derive precise IEQs. We note that it happens that all the tuples are bound for query  $A_4$ ; hence, the performance results are the same for all three labeling schemes.

Figure 4(b) compares the running time performance of the three schemes for Adult data set. The results show that TALOS is also more efficient than NI and RD. The reason is due to the flexibility of TALOS's dynamic labeling scheme for free tuples, which results in decision trees that are smaller than those constructed by NI and RD. Table 5 compares the decision trees constructed by TALOS, NI, and RD in terms of their average height and average size (i.e., number of nodes). Observe that the decision trees constructed by TALOS are significantly more compact than those built by NI and RD.

Figures 4(c) and (d) compare the quality of the IEQs generated for Adult data set using the MDL and F-measure metrics, respectively. The results show that TALOS produces much better quality IEQs than both NI and RD. For the

MDL metric (smaller value means better quality), the average value of the MDL metric for TALOS is low (under 700) while the values of both NI and RD are in the range of [4000, 22000]. For the F-measure metric (larger value means better quality), the average value for TALOS is no smaller than 0.7, whereas the values for NI and RD are only around 0.4.

Figure 5 shows the comparison results for the Baseball data set for strong IEQs. (The strong and weak IEQs for the queries  $B_1$  to  $B_4$  happen to be the same.) The results also demonstrate similar trends with TALOS outperforming NI and RD in both the running time as well as the number and the quality of IEQs generated for queries  $B_1 - B_3$ . It happens that all the tuples are bound for query  $B_4$ ; hence, the performance results are the same for all three algorithms.

We observe that the benefit of TALOS over NI and RD is higher for queries  $A_1 - A_3$  in Adult data set compared with that for queries  $B_1 - B_3$  in Baseball data set. For example, TALOS runs 3 - 10 times faster than NI and RD for queries  $A_1 - A_3$ , whereas TALOS runs 2 times faster than NI and RD for queries  $B_1 - B_3$ . As another example, the MDLs of the IEQs for  $A_1 - A_3$  returned by TALOS are 10 - 1000 times lower than those of NI (and RD); whereas the MDLs of the IEQs for  $B_1 - B_3$  returned by TALOS are 2 times lower than those of NI (and RD). The reason is that the number of free tuples per partition for queries on Baseball data set is smaller (in the order of 10) compared with that for queries on Adult data set (in the order of 100). The flexibility for TALOS, therefore, reduces in queries  $B_1 - B_3$ ; however, TALOS still produces higher quality IEQs than NI and RD.

### 10.3 Discussion on Generated IEQs

In this section, we discuss some of the interesting IEQs generated by TALOS on the running data sets, as shown in Ta-

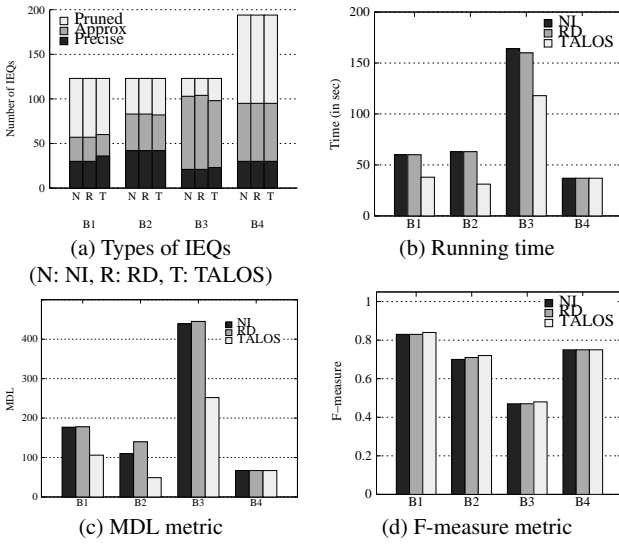


Fig. 5 Comparison of TALOS, NI and RD (Baseball data set)

ble 9. Discussions for other queries are given in Appendix E. We use  $X_{i,j}$  to denote an IEQ for the query  $X_i$  in Table 3, where  $X \in \{A, B, T\}$ . For each IEQ, we also show its value for the F-measure metric (i.e.,  $F_m^{est}$ ) given in terms of their  $|p_a|$ ,  $|p_b|$  and  $|p_c|$  values; recall that an IEQ is precise iff  $|p_b| = 0$  and  $|p_c| = 0$ .

**Adult.** In query  $A_2$ , we want to find the occupation and education of white females who are never married and with age in the range  $[64, 68]$  and with a capital gain greater than 500. The IEQ  $A_{2,1}$  provides additional insight about the query result: those who are in the age range  $[64, 66]$  are highly educated, whereas the rest (i.e., those in the age range  $[67, 68]$ ) have high capital gains.

Query  $A_4$  is a skyline query looking for people with maximal capital gain and minimal age. Interestingly, the precise IEQ  $A_{4,1}$  provides a simplification of  $A_4$ : the people selected by this skyline query are either (1) very young ( $age \leq 17$ ) and have capital gain in the range  $1055 - 27828$ , or (2) have very high capital gain ( $> 27828$ ), work in the protective service, and whose race is not classified as “others”.

Query  $A_5$  is an SPJA query looking for the average number of working hours per week of each age group of people who have native country as “US”. The precise IEQ  $A_{5,1}$  reports the number of working hours per week of a particular person in this group. Note that although  $A_5$  and  $A_{5,1}$  use different aggregated functions (SUM vs. AVG), these queries are still considered to be equivalent. The reason is that the difference between the aggregated values of the corresponding tuples in  $A_5(D)$  and  $A_{5,1}(D)$  is less than one, and is considered to be the same based on the similarity function that is used.

**Baseball.** In query  $B_2$ , we want to find the set of high performance players who have very high total home runs ( $>$

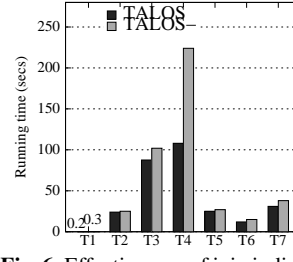


Fig. 6 Effectiveness of join indices

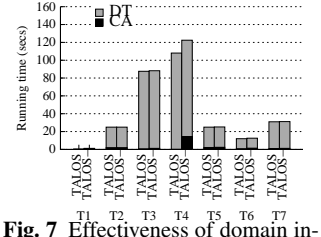


Fig. 7 Effectiveness of domain indices

600). The IEQ  $B_{2,1}$  indicates that one player in this group is highly paid with a left throwing hand.

Query  $B_4$  is an interesting query that involves multiple core relations. This query asks for the managers of team “CIN” from 1983 to 1988, the year they managed the team as well as the rank gained by the team. In this query, we note that TALOS found alternative join-paths to link the two core relations, Manager and Team. The first alternative join-path (shown by  $B_{4,1}$ ) involves Manager, Master, Batting, and Team. The second alternative join-path (not shown) involves Manager, Master, Fielding, and Team. The IEQ  $B_{4,1}$  reveals the interesting observation that there is one manager who was also a player in the same year that he managed the team with some additional information about this manager-player.

Queries  $B_5$  and  $B_6$  are examples of SPJU and SPJA queries. TALOS returns one IEQ for each of these two queries, and the IEQ is exactly the same as the original one. In other words, TALOS is able to reverse-engineer the original queries that are used to derive  $B_5(D)$  and  $B_6(D)$ .

**TPC-H.** Query  $T_1$  retrieves the names of suppliers whose *acctbal* is greater than 4000 and *regionkey* is smaller than 4.  $T_{1,1}$  is an approximate IEQ of  $T_1$  that does not include the selection condition on *acctbal* attribute.

Query  $T_3$  retrieves the name of suppliers and parts, where the *acctbal* of the supplier is greater than 9500 and the *availqty* of the part is greater than 3000.  $T_{3,1}$  is an approximate IEQ that provides details of one tuple in  $T_3(D)$ , i.e., the tuple has *availqty* = 3001 and *retailprice*  $\leq$  953.

Queries  $T_5 - T_7$  are examples of SPJU and SPJA queries. TALOS returns one IEQ for each of these queries, and the IEQ is exactly the same as the original one. In other words, TALOS is able to reverse-engineer the original queries that are used to derive  $T_5(D)$ ,  $T_6(D)$  and  $T_7(D)$ .

So far, we have studied the effectiveness of TALOS in generating IEQs. In the following, we study the efficiency of the proposed optimizations of TALOS. Our discussion focuses on TPC-H data set and the corresponding queries, as other data sets (Adult and Baseball) have small size, and the optimizations have little effect for those cases.

IEQ		$ p_a $	$ p_b $	$ p_c $
$A_{2,1}$ :	$\sigma_{p_1 \vee p_2}(\text{adult})$ $p_1 : 63 < \text{age} \leq 66 \wedge \text{edu} > 15 \wedge \text{ms} = \text{"NM"}; p_2 : 66 < \text{age} \leq 68 \wedge \text{ms} = \text{"NM"} \wedge \text{gain} > 2993$	5	0	0
$A_{4,1}$ :	$\sigma_{p_1 \vee p_2}(\text{adult})$ $p_1 : 1055 < \text{gain} \leq 27828 \wedge \text{age} \leq 17; p_2 : \text{gain} > 27828 \wedge \text{occ} = \text{P} \wedge \text{race} \neq \text{O}$	4	0	0
$A_{5,1}$ :	$\pi_{\text{age}, \text{SUM}(\text{hpw})} \sigma_{\text{age}=53 \wedge \text{edu}=\text{"Bachelor"} \wedge \text{occ}=\text{"Prof-specialty"}}(\text{adult})$	1	0	0
$B_{2,1}$ :	$\sigma_{\text{salary} > 21680700 \wedge \text{throws}=\text{"L"}}(\text{Master} \bowtie \text{Salaries})$	1	0	3
$B_{4,1}$ :	$\sigma_{21 < L < 22 \wedge SB < 0 \wedge 67 < W < 70}(\text{Manager} \bowtie \text{Master} \bowtie \text{Batting} \bowtie \text{Team})$	1	0	5
$T_{1,1}$ :	$\sigma_{\text{regionkey} \leq 3}(\text{supplier} \bowtie \text{nation})$	4383	3598	0
$T_{3,1}$ :	$\sigma_{3000 < \text{availqty} < 3001 \wedge \text{retailprice} < 953}(\text{part} \bowtie \text{partsupp} \bowtie \text{supplier})$	1	0	24671

**Table 6** Examples of IEQs generated by TALOS

#### 10.4 Effectiveness of Optimizations for One Database Version

We now consider the case of applying TALOS to generate IEQs given *one* database version.

**Effectiveness of Join Indices.** First, we evaluate the effectiveness of using join indices to compute the join relation for the decision tree classifier by comparing TALOS (which uses join indices) against TALOS<sup>-</sup> (without join indices) for deriving IEQs.

Figure 6 shows the running time comparison of TALOS and TALOS<sup>-</sup>. Note that the number and quality of IEQs produced by TALOS and TALOS<sup>-</sup> are the same, as these qualities are independent of the optimizations. The results show that TALOS is up to two times faster than TALOS<sup>-</sup>. The reason is that the computation of  $J_{hub}$  by TALOS using join indices is more efficient than the computation of join relation  $J$  by joining relations directly in TALOS<sup>-</sup>. In addition, the attribute lists accessed by TALOS, which correspond to the base relations, are smaller than the attribute lists accessed by TALOS<sup>-</sup>, which are based on  $J$ .

To illustrate these observations above, we show the running time comparison of TALOS and TALOS<sup>-</sup> to derive IEQs for queries  $T_3$  and  $T_4$  with respect to the three main steps of each algorithm: (1) deriving the join relation, (2) computing attribute lists, and (3) building decision trees. The results in Table 7 clearly demonstrate the effectiveness of TALOS over TALOS<sup>-</sup> in these steps. For instance, the step to derive the join relation for  $T_3$  by TALOS is 1.5 times faster than that of TALOS<sup>-</sup>, since TALOS only needs to derive the corresponding join indices of *part-partsupp* and *partsupp-supplier* consisting of integer-valued columns. In contrast, TALOS<sup>-</sup> needs to perform the join between *partsupp*, *part*, and *supplier* relations. In another example, the step to derive join relation for  $T_4$  by TALOS is 3.5 times faster than that of TALOS<sup>-</sup>, since TALOS only needs to read the corresponding join index (*lineitem-order*); whereas TALOS<sup>-</sup> needs to perform the join between *lineitem* and *order* relations.

The attribute lists used by TALOS are also more compact than those used by TALOS<sup>-</sup>. For instance, the attribute list constructed by TALOS<sup>-</sup> for attribute "*part.retailprice*" for query  $T_3$  is 4 times larger than that constructed by TALOS.

Thus the steps to derive attribute lists and building decision trees run more efficiently in TALOS than in TALOS<sup>-</sup>.

**Effectiveness of Domain Indices.** We evaluate the effectiveness of using domain indices to identify covering attributes by comparing TALOS (which uses domain indices) against TALOS<sup>-</sup> (without domain indices) for deriving IEQs where the input query is unknown.

For each test query  $T_i$ , we first compute its answer  $T_i(D)$  on the TPC-H data set  $D$  and then use  $T_i(D)$  and  $D$  as inputs to derive IEQs; thus, each input query for this experiment is actually  $T_i(D)$  and not  $T_i$ . For each query  $T_i(D)$ , we measured the time to derive the IEQs for  $Q_i(D)$ . We present each timing in terms of two components: the time taken to compute the covering attributes (denoted by  $CA$ ) and the remaining time taken to derive the IEQs after the covering attributes have been computed (denoted by  $DT$ ).

Figure 7 shows the comparison of TALOS and TALOS<sup>-</sup> for the TPC-H queries. The results show that TALOS is more efficient than TALOS<sup>-</sup> for computing covering attributes (i.e.,  $CA$  component) with TALOS taking a few seconds compared to 10 seconds incurred by TALOS<sup>-</sup>.

For TPC-H dataset with SF=5.0 (Appendix F), TALOS took 30 minutes to return three IEQs for query T4 which involved the two largest relations *lineitem* (30M tuples) and *orders* (7.5M tuples). Thus, further research is needed to scale the classification-based approach for very large data.

**Storage Overhead.** The storage overhead of TALOS consists of pre-computed join indices built for pairs of relations that have foreign-key relationships and the domain indices.

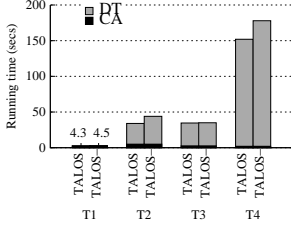
For join indices, basically, for every pair of relations  $R$  and  $S$  that have foreign-key relationship, TALOS builds a join index relation  $I_{R,S}(r_R, r_S)$  consisting of  $k$  pairs of integers, where  $k$  is the number of tuples in the join result of  $R$  and  $S$ . In addition, TALOS also builds two B<sup>+</sup>-indices on  $r_R$  and  $r_S$  column of  $I_{R,S}$  to speed up the joins of using join indices. In our experiments, the pre-computed join indices built for TPC-H data set consist of 450MB versus 1GB of the whole database.

For the domain indices, the size of domain indices was 272MB for the 1GB TPC-H database.<sup>12</sup>

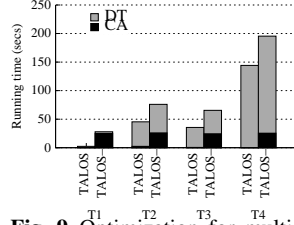
<sup>12</sup> We remove the *comment* attributes from each relation in TPC-H database.

Step	$T_3$		$T_4$	
	TALOS	TALOS <sup>-</sup>	TALOS	TALOS <sup>-</sup>
Join relation	25	36	20	70
Attribute list	1.6	3	36	63
Decision tree	61	63	52	91

**Table 7** Detailed running times of TALOS and TALOS<sup>-</sup>



**Fig. 8** Optimization for multiple database & result versions



**Fig. 9** Optimization for multiple database versions & single result

### 10.5 Effectiveness of Optimizations for Multiple Database Versions

In the last set of experiments, we evaluate the effectiveness of the optimizations proposed to derive IEQs for the two contexts with multiple database versions using the TPC-H data set and four queries ( $T_1$  to  $T_4$ ).

We created two database versions,  $D_1$  and  $D_2$ , where  $D_1$  was the original TPC-H database and  $D_2$  was created from  $D_1$  by combining with a set of delta tuples generated from four relations (*supplier*, *customer*, *partsupp*, *orders*)<sup>13</sup> as follows. For each of the four relations  $R$ , we randomly selected a subset of  $k$  tuples for deletion and also randomly generated a set of  $k$  new tuples for insertion to  $R$ , where  $k$  is equal to 5% of the cardinality of  $R$ . For each test query  $T_i$ , we run  $T_i$  on  $D_1$  and  $D_2$  to obtain two output tables  $T_i(D_1)$  and  $T_i(D_2)$ .

The performance timings presented in this section consist of two components: the time to compute the covering attributes (denoted by  $CA$ ), and the remaining time to derive the IEQs (denoted by  $DT$ ).

**Multiple Database & Result Versions.** Given the inputs  $(D_1, T_i(D_1))$  and  $(D_2, T_i(D_2))$ , we compare the performance of two algorithms, TALOS<sup>-</sup> and TALOS, to derive the IEQs for  $T_i$ .

The performance results in Figure 8 show that TALOS outperforms TALOS<sup>-</sup> by a factor of 1.4 times. While both TALOS<sup>-</sup> and TALOS incur the same time for computing covering attributes, TALOS<sup>-</sup> spends significantly more time for deriving the IEQs because it needs to derive one set of IEQs for each  $(D_j, T_i(D_j))$  and combine them to obtain the final result. In contrast, TALOS only needs to derive one set of IEQs using a more efficient approach. For all the test queries, both TALOS<sup>-</sup> and TALOS were able to successfully reverse-engineer the original test queries.

<sup>13</sup> These four relations are chosen because they appeared in the test queries  $T_1$  to  $T_4$ .

### Multiple Database Versions & Single Unknown Result.

Given the input  $T_i(D_1)$ , we compare the performance of two algorithms, TALOS<sup>-</sup> and TALOS, to derive the most recent database version  $D_j$  and IEQs  $Q'_i$  such that  $Q'_i(D_j) = T_i(D_1)$ .

The performance results in Figure 9 show that TALOS is on average 1.5 times faster than TALOS<sup>-</sup>. The reason is that TALOS<sup>-</sup> incurs a lot of time on computing covering attributes compared to TALOS.  $CA$  is in the order of 10 seconds in TALOS versus 60 seconds in TALOS<sup>-</sup>. To further study the usefulness of the optimization techniques used in TALOS, we present the details of the data structures to derive the covering attributes for  $T_2$  as an example. Recall from Section 9.1 that in order to compute the covering attributes which are categorical attributes in this case, TALOS first derives  $\delta TM_d$  as the join between  $T_2$  containing 95k tuples and  $\delta M_d$  containing 600k tuples. The resulting  $\delta TM_d$  contains 15k tuples. Lastly, TALOS performs an outer-join between  $\delta TM_d$  and  $TM_d^2$  containing 100k. This process is much cheaper than TALOS<sup>-</sup>, which first updates the table  $M_d^2$  containing 4000k by adding 600k tuples from  $\delta M_d$  and then joins the resulting  $M_d^1$  with  $T_2$ .

## 11 Related Work

The related work can be broadly classified into three areas: work on deriving instance equivalent queries, data mining techniques for related problems, and work on reverse query processing.

### 11.1 Instance Equivalent Queries

To the best of our knowledge, our work [29] was the first data-driven approach to generate instance equivalent queries from an input query and its result on a database. In this paper, we have extended the work to handle an unknown input query, support more expressive IEQs, and handle multiple database versions.

[25] examined a related problem called *View Definitions Problem (VDP)* which derives a view definition  $Q$  given an input database  $D$  and a materialized view  $V$ . As the focus there is on analyzing the complexity of variants of the VDP problem, it assumes a very basic setting where  $D$  consists of a single relation  $R$  and  $V$  has the same schema as  $R$ . Consequently, the derivation of  $Q$  consists of determining the selection predicates on  $R$  to generate  $V$ , and the paper examines the complexity of the problem for different query fragments in terms of the number of equality/range predicates and whether disjunctions are permitted. In contrast to the theoretical treatment in [25], the setting in our work is more general and focuses on a novel data classification-based approach and optimization techniques to derive IEQs.

An area that is related and complementary to our data-driven approach for QRE is intensional query answering (IQA) or cooperative answering. For a given query  $Q$ , the goal of IQA is to augment the query’s answer  $Q(D)$  with additional intensional information in the form of a semantically equivalent query<sup>14</sup> that is derived from integrity constraints on the data [9, 18]. While semantic equivalence is stronger than instance equivalence and can be computed in a data-independent manner using only integrity constraints (ICs), there are several advantages of adopting instance equivalence for QRE. First, in practice, many data semantics are not explicitly captured using ICs in the database for various reasons [11]. Hence, the effectiveness of IQA could be limited for QRE. Second, even when the ICs in the database are complete, it can be very difficult to derive semantically equivalent queries for complex queries (e.g., skyline queries that select dominant objects). Third, as IQA requires the input query  $Q$  to be known, IQA cannot be applied to QRE applications where only  $Q(D)$  (but not  $Q$ ) is available. Thus, we view IQA and our proposed data-driven approach to compute IEQs as complementary techniques for QRE.

## 11.2 Data Mining Techniques

A related technique to our approach of classifying tuples in a join relation is the CrossMine approach for multi-relational classification [33]. Given a target relation  $R$  with tuples that have fixed class labels (i.e., either positive or negative), CrossMine builds a decision tree classifier for tuples in  $R$  using the attributes in  $R$  as well as the attributes from other relations that have primary-foreign key relationships with  $R$ . Our data classification in TALOS differs from CrossMine in two key ways. First, TALOS has the notion of *free tuples* that are dynamically assigned positive or negative class labels with respect to constraints that are imposed by IEQ derivation problem (i.e., at-least-one semantics, exactly- $k$  semantics, and aggregation constraints). In contrast, CrossMine has the same setting as conventional data classification where all tuples have fixed class labels. Second, whereas CrossMine uses a greedy heuristic to compute node splits in the construction of decision trees, TALOS introduces a novel and efficient technique to compute optimal node splits by exploiting the flexibility enabled by the presence of free tuples.

A somewhat related problem to ours is the problem of *re-description mining* introduced in [22]. The goal in re-description mining is to find different subsets of data that afford multiple descriptions across multiple vocabularies covering the same data set. At an abstract level, our work is different from these methods in several ways. First, we are concerned

with a fixed subset of the data (the output of the query). Second, none of the approaches for re-description mining accounts for structural (relational) information in the data (something we explicitly address). Third, re-description mining requires multiple independent vocabulary descriptions to be identified. We do not have this requirement as we are concerned with deriving IEQs within an SQL context.

## 11.3 Reverse Query Processing

There are two recent directions that share the same broad principle of “reverse query processing” as QRE but differ completely in the problem objectives and techniques.

The first direction addresses the problem of generating databases to satisfy a set of constraints given an input set of queries and their results. [3] introduced the *Reverse Query Processing* problem, which given an input query  $Q$  and a desired result  $R$ , generates a database  $D$  such that  $Q(D) = R$ . [4] introduced the QAGen problem, which given an input query  $Q$  and a set of target cardinality constraints on the intermediate subexpressions in  $Q$ ’s evaluation plan  $P$ , generates a test database such that the execution of  $P$  on  $D$  satisfies the given cardinality constraints. The QAGen problem was generalized in [14] to consider a workload of queries with the objective of generating a minimal set of database instances such that the results of the workload queries on the database instances satisfy the given cardinality constraints. A more recent work [2] proposed probabilistically approximate algorithms for the QAGen problem. Unlike these works on generating databases, our work on QRE is to generate instance-equivalent queries.

The second direction addresses the problem of *Targeted Query Generation (TQGen)* which aims to generate test queries to meet a given set of cardinality constraints [7, 17]. Specifically, given a query  $Q$ , a database  $D$ , and a set of target cardinality constraints on the intermediate subexpressions in  $Q$ ’s evaluation plan, the objective of TQGen is to generate a modified query  $Q'$  (by modifying the constant values in  $Q$ ’s selection predicates) such that the evaluation plan of  $Q'$  on  $D$  satisfies the given cardinality constraints. Different from the TQGen problem that focuses on cardinality constraints, our work on QRE aims to generate instance-equivalent queries that satisfy the “content” constraints of the query results. In addition, unlike the TQGen problem, the input query may be unknown in our QRE problem and the IEQs generated could be very different from the input query (beyond modifications to selection predicate constants).

More broadly, there has also been work on reverse engineering various aspects of database systems: extracting data models from databases (e.g., [15, 20]), and mining database structural information that spans from data dependencies and constraints (e.g., [6, 26]) to higher-level semantic structures

<sup>14</sup> Two queries  $Q$  and  $Q'$  are *semantically equivalent* if for every valid database  $D$ ,  $Q(D) = Q'(D)$ .



(e.g., [1, 31]) to support browsing of databases with large, complex schemas and data integration applications.

## 12 Conclusion

In this paper, we have introduced the *Query Reverse Engineering* (QRE) problem to derive an instance-equivalent query that produces the same result as a given input result table. The QRE problem has useful applications in database usability, data analysis, and data security. We have proposed a data-driven approach, TALOS, that is based on a novel dynamic data classification formulation and extended the approach to efficiently support the three key dimensions of the QRE problem: whether the input query is known/unknown, supporting different IEQ fragments, and supporting multiple database versions.

There are several directions for further work on the QRE problem. With respect to our proposed method TALOS, one challenging issue that requires further study is the scalability of the approach for very large data sets; in particular, the computation and processing of the hub table could be a bottleneck when the relations involved are very large. It is also interesting to extend TALOS to derive more complex IEQs (e.g., queries with arithmetic expressions, top- $k$  queries). An interesting direction to explore beyond TALOS is a hybrid approach that includes an offline phase to mine for soft constraints in the database and an online phase that exploits both the database contents as well as mined constraints. Finally, a more general direction to explore is how to apply the idea of query reverse engineering for querying web data.

**Acknowledgements** We would like to thank the editors and reviewers for their constructive comments and suggestions to improve the paper's presentation. This research is supported in part by NUS Grant R-252-000-512-112.

## References

- Andritsos, P., Miller, R.J., Tsaparas, P.: Information-theoretic tools for mining database structure from large data sets. In: SIGMOD Conference, pp. 731–742 (2004)
- Arasu, A., Kaushik, R., Li, J.: Data generation using declarative constraints. In: SIGMOD Conference, pp. 685–696 (2011)
- Binnig, C., Kossmann, D., Lo, E.: Reverse query processing. In: ICDE, pp. 506–515 (2007)
- Binnig, C., Kossmann, D., Lo, E., Özsu, M.T.: QAGen: generating query-aware test databases. In: SIGMOD Conference, pp. 341–352 (2007)
- Blakeley, J.A., Larson, P.A., Tompa, F.W.: Efficiently updating materialized views. SIGMOD Rec. **15**(2), 61–71 (1986)
- Brown, P.G., Haas, P.J.: BHUNT: Automatic discovery of fuzzy algebraic constraints in relational data. In: Proceedings of the International Conference on Very Large (VLDB'03), pp. 668–679. ACM (2003)
- Bruno, N., Chaudhuri, S., Thomas, D.: Generating queries with cardinality constraints for dbms testing. IEEE TKDE **18**(12), 1721–1725 (2006)
- Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill Science (2001)
- Gaasterland, T., Godfrey, P., Minker, J.: An overview of cooperative answering. J. Intell. Inf. Syst. **1**(2), 123–157 (1992)
- Getoor, L., Taskar, B., Koller, D.: Selectivity estimation using probabilistic models. In: SIGMOD Conference, pp. 461–472 (2001)
- Godfrey, P., Gryz, J., Zuzarte, C.: Exploiting constraint-like data characterizations in query optimization. In: SIGMOD Conference, pp. 582–592 (2001)
- Johnson, T., Marathe, A., Dasu, T.: Database exploration and bellman. IEEE Data Eng. Bull. **26**(3), 34–39 (2003)
- Lenzerini, M.: Data integration: A theoretical perspective. In: PODS, pp. 233–246 (2002)
- Lo, E., Cheng, N., Hon, W.K.: Generating databases for query workloads. Proc. VLDB Endow. **3**(1), 848–859 (2010)
- Malpani, A., Bernstein, P., Melnik, S., Terwilliger, J.: Reverse engineering models from databases to bootstrap application development. In: ICDE, pp. 1177–1180 (2010)
- Mehta, M., Agrawal, R., Rissanen, J.: SLIQ: A fast scalable classifier for data mining. In: EDBT, pp. 18–32 (1996)
- Mishra, C., Koudas, N., Zuzarte, C.: Generating targeted queries for database testing. In: SIGMOD Conference, pp. 499–510 (2008)
- Motro, A.: Intensional answers to database queries. IEEE TKDE **6**(3), 444–454 (1994)
- Mumick, I.S., Quass, D., Mumick, B.S.: Maintenance of data cubes and summary tables in a warehouse. SIGMOD Rec. **26**(2) (1997)
- Petit, J.M., Toumani, F., Boulicaut, J.F., Kouloumdjian, J.: Towards the reverse engineering of denormalized relational databases. In: ICDE, pp. 218–227 (1996)
- P.N. Tan, M., V.Kumar: Introduction to Data Mining. Addison-Wesley (2006)
- Ramakrishnan, N., Kumar, D., Mishra, B., Potts, M., Helm, R.F.: Turning cartwheels: An alternating algorithm for mining re-descriptions. In: SIGKDD Conference, pp. 266–275 (2004)
- van Rijsbergen, C.J.: Information Retrieval. Butterworth (1979)
- Rissanen, J.: Modeling by shortest data description. Automatica **14**, 465–471 (1978)
- Sarma, A.D., Parameswaran, A., Garcia-Molina, H., Widom, J.: Synthesizing view definitions from data. In: ICDT, pp. 89–103 (2010)
- Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: GORDIAN: efficient and scalable discovery of composite keys. In: VLDB, pp. 691–702. ACM (2006)
- Stonebraker, M.: The design of the postgres storage system. In: VLDB, pp. 289–300 (1987)
- Tran, Q.T., Chan, C.Y.: How to ConQueR why-not questions. In: SIGMOD Conference, pp. 15–26 (2010)
- Tran, Q.T., Chan, C.Y., Parthasarathy, S.: Query by output. In: SIGMOD Conference, pp. 535–548 (2009)
- Valduriez, P.: Join indices. ACM Trans. Database Syst. **12**(2), 218–246 (1987)
- Wu, W., Reinwald, B., Sismanis, Y., Manjrekar, R.: Discovering topical structures of databases. In: SIGMOD Conference, pp. 1019–1030 (2008)
- Xiao, X., Tao, Y.: Output perturbation with query relaxation. Proc. VLDB Endow. **1**(1), 857–869 (2008)
- Yin, X., Han, J., Yang, J., Yu, P.S.: Efficient classification across multiple database relations: A crossmine approach. TKDE **18**, 770–783 (2006)