Mining All Frequent Projection-Selection Queries from a Relational Table

Tao-Yuan Jen
ETIS - UMR CNRS
Université de Cergy-Pontoise
95302 Cergy-Pontoise
France
jen@u-cergy.fr

Dominique Laurent
ETIS - UMR CNRS
Université de Cergy-Pontoise
95302 Cergy-Pontoise
France
dlaurent@u-cergy.fr

Nicolas Spyratos
LRI - UMR CNRS
Université Paris-Sud
91405 Orsay
France
spyratos@lri.fr

ABSTRACT

In this paper we study the problem of mining *all* frequent queries in a given database table, a problem known to be intractable even for conjunctive queries. We restrict our attention to projection-selection queries, and we assume that the table to be mined satisfies a set of functional dependencies. Under these assumptions we define a pre-ordering $\preceq$ over queries and we show the following: (a) the support measure is anti-monotonic (with respect to $\preceq$), and (b) if we define $q \equiv q'$ iff $q \preceq q'$ and $q' \preceq q$ then *all* queries of an equivalence class have the *same* support.

With these results at hand, we further restrict our attention to star schemas of data warehouses. In those schemas, the set of functional dependencies satisfies an important property, namely, the union of keys of all dimension tables is a key for the fact table. The main contribution of this paper is the proposal of a level-wise algorithm for mining all frequent projection-selection queries in a data warehouse over a star schema. Moreover, we show that, in the case of a star schema, the complexity in the number of scans of our algorithm is similar to that of the well known Apriori algorithm, *i.e.,* linear with respect to the number of attributes.

1. INTRODUCTION

In this paper we study the problem of mining *all* frequent queries in a (relational) table Δ over an attribute set U, where a query is said to be *frequent* if the cardinality of its answer is above a given threshold.

This is an important but challenging problem of data mining, known to be intractable even for conjunctive queries [6]. Indeed, the size of the search space can be shown to be exponential not only in the number of attributes, but also in the number of tuples (since there are as many selection conditions as there are sub-tuples of tuples in Δ).

In this work we restrict our attention to projection-selection queries, and we assume that Δ satisfies a set of functional dependencies. A projection-selection query is of the form $\pi_X(\sigma_{Y=y}(\Delta))$ where X and Y are subsets of U and y is a tuple of $\pi_Y(\Delta)$. We define the *support* of such a query to be the number of tuples in the answer. Under these assumptions, we use the set of functional dependencies to define a pre-ordering $\preceq$ over queries and we show that (a) the support measure is anti-monotonic (with respect to $\preceq$), and (b) if we define $q \equiv q'$ if and only if $q \preceq q'$ and $q' \preceq q$ then *all* queries of an equivalence class have the *same* support.

With these results at hand, we restrict our attention to star schemas of data warehouses. In those schemas, the set of functional dependencies satisfies an additional important property, namely, the union of keys of all dimension tables is a key for the fact table. The main contribution of this paper is the proposal of a level-wise algorithm for mining all frequent projection-selection queries in a data warehouse over a star schema.

Our algorithm is inspired from the well known Apriori algorithm [1], and we show that its complexity is similar to that of Apriori. More precisely, whereas the complexity in the number of scans of the transaction table of Apriori is in $O(n)$ (where n is the number of items), we show that, in the case of a star schema, the complexity in the number of scans of the table of our algorithm is in $O(|U|)$, where $|U|$ is the number of attributes in U.

Let us illustrate the basic concepts of our approach through an example (that will serve as a running example throughout the paper).

Example 1. Consider the table Δ defined over the attribute set $U = \{Cid, Cname, Caddr, Pid, Ptype, Qty\}$, as shown in Figure 1, where:

- Cid, $Cname$ and $Caddr$ stand for Customer Identifier, Customer Name and Customer Address,
- Pid and $Ptype$ stand for Product Identifier and Product Type,
- Qty stands for Quantity (*i.e.,* number of products sold).

Moreover, assume that the table Δ satisfies the following set FD of functional dependencies:

$$FD = \{Cid \rightarrow Cname\,Caddr\,,\ Pid \rightarrow Ptype\,,\\ Cid\,Pid \rightarrow Qty\}.$$

(One can easily verify that the table Δ shown in Figure 1 does satisfy the above dependencies.)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EDBT'08, March 25–30, 2008, Nantes, France
Copyright 2008 ACM 978-1-59593-926-5/08/0003 ...$5.00.

Δ	Cid	Pid	$Cname$	$Caddr$	$Ptype$	Qty
	c_1	p_1	John	Paris	milk	10
	c_1	p_2	John	Paris	beer	10
	c_2	p_1	Mary	Paris	milk	1
	c_2	p_2	Mary	Paris	beer	5
	c_3	p_3	Paul	NY	beer	10
	c_4	p_4	Peter	Paris	milk	15

Figure 1: The table of the running example

We note that, in a star schema, the table Δ could be the result of joining the following three tables: $Customer(Cid, Cname, Caddr)$, $Product(Pid, Ptype)$, $Sales(Cid, Pid, Qty)$ where $Customer$ and $Product$ are dimension tables and $Sales$ is the fact table.

Assuming that *all* frequent projection-selection queries are computed using the table Δ, we argue that it is relevant to consider the confidence of rules of the form $q_1 \Rightarrow q_2$ (*i.e.,* the ratio of the support of q_2 over that of q_1), where q_1 and q_2 are frequent queries such that the support of q_2 is less than the support of q_1.

To illustrate this point, assume that q_1, q_2, q_3 and q_4 are frequent queries defined as follows:

– $q_1 = \pi_{Cid}(\sigma_{Caddr=\texttt{Paris}}(\Delta))$,
– $q_2 = \pi_{Cid}(\sigma_{Caddr\,Ptype=\texttt{Paris}\,\texttt{beer}}(\Delta))$,
– $q_3 = \pi_{Pid}(\sigma_{Ptype=\texttt{beer}}(\Delta))$,
– $q_4 = \pi_{Pid}(\sigma_{Caddr\,Ptype=\texttt{Paris}\,\texttt{beer}}(\Delta))$.

The fact that the confidence of $q_1 \Rightarrow q_2$ (respectively, $q_3 \Rightarrow q_4$) is 80% (respectively, 75%) means that 80% *of the customers from Paris buy beer* (respectively, 75% *of the products of type beer are bought by at least one customer from Paris*). Combining these two facts, we can state that 80% *of the customers from Paris buy* 75% *of the products of type beer.*

However, in this paper, we focuss on the computation of frequent projection-selection queries, and we leave the study of rules for future work.

Now, referring to Figure 1, it is easy to verify that we have $|ans(\pi_{Cid}(\Delta))| \geq |ans(\pi_{Caddr}(\Delta))|$, and that this inequality holds *because* the table Δ satisfies the dependency $Cid \rightarrow Caddr$ (which is a consequence of FD).

Similarly, it is easy to verify that we have $|ans(\pi_{Cid}(\Delta))| = |ans(\pi_{Cid\,Caddr}(\Delta))|$, and that this equality holds *because* the table Δ satisfies the dependencies $Cid \rightarrow Cid\,Caddr$ and $Cid\,Caddr \rightarrow Cid$ (which are consequences of FD).

Note that, although *not* equivalent according to standard query equivalence [14], the queries $\pi_{Cid}(\Delta)$ and $\pi_{Cid\,Caddr}(\Delta)$ have the same support. Thus, as long as we are interested only in computing supports, only *one* computation is necessary for these two queries.

On the other hand, it is easy to verify that we have $|ans(\sigma_{Pid=\texttt{p}_2}(\Delta))| \leq |ans(\sigma_{Ptype=\texttt{beer}}(\Delta))|$, and that this inequality holds *because* the table Δ satisfies the dependency $Pid \rightarrow Ptype$ and $\texttt{p}_2$ is associated with `beer`. Clearly, this inequality will hold in *any* table that satisfies the dependency $Pid \rightarrow Ptype$ and in which the Pid value $\texttt{p}_2$ is associated with the $Ptype$ value `beer`.

Now, consider the table Δ' obtained from Δ by replacing all occurrences of `beer` by `wine`. Then Δ' still satisfies FD but now $\texttt{p}_2$ is associated with `wine` instead of `beer`. As a result we have $|ans(\sigma_{Pid=\texttt{p}_2}(\Delta'))| > |ans(\sigma_{Ptype=\texttt{beer}}(\Delta'))|$, showing that these queries are not comparable in any table satisfying FD. Therefore, selection conditions have to be taken into account for a given table, when comparing the supports of selection-projection queries. □

In what follows, based on the observations of the previous example, we define a pre-ordering over the set of all projection-selection queries, and show that the support is anti-monotonic with respect to this pre-ordering.

Moreover, the pre-ordering induces an equivalence relation over projection-selection queries, and this equivalence relation plays a fundamental role in our approach, since two equivalent queries are shown to have the same support. The importance of this property lies in the fact that only *one* computation is necessary in order to obtain the support of *all* queries in the same equivalence class.

Finally, in the case where the set FD of functional dependencies is that of a star schema, we provide a level-wise algorithm (inspired from the Apriori algorithm [1]) showing that the problem of mining all frequent projection-selection queries becomes tractable in this particular case.

The paper is organized as follows: In Section 2, we briefly review previous work in the area, and in Section 3 we recall basic properties of projection-selection queries and introduce the associated pre-ordering. In particular, we show that the support measure is anti-monotonic with respect to this pre-ordering. In Section 4, we consider the equivalence relation induced by the pre-ordering, we characterize the content of equivalence classes, and we give a basic property for building the set of equivalence classes. In Section 5, we show how the general results of the earlier sections apply in the particular case of a star schema, and we give algorithms for mining all frequent projection-selection queries in this case. Section 6 concludes the paper and discusses future work.

2. RELATED WORK

The work in [6] considers *conjunctive* queries, as we do in this paper, and points out that without restrictions on the database schema, the problem is intractable. Although some hints on possible restrictions are mentioned in [6], no specific case is studied.

In [7], the authors consider restrictions on the frequent queries to be mined using the formalism of rule based languages. Although the considered class of queries is larger than in our approach, it should be pointed out that, in [7], (i) equivalent queries can be generated that can not be tested efficiently (a problem that does not exist in our approach), and (ii) functional dependencies are not taken into account, as we do in this paper.

The work of [8], although dealing with mining tree queries in a graph, is nevertheless closely related to ours. Indeed, in [8], a graph is represented by a binary relation, and frequent tree queries, expressed as SQL queries involving projections, selections and joins, are mined.

Therefore, the approach in [8] can be considered as being more general than ours because (i) particular joins are taken into account, which is not the case in our approach, and because (ii) for a given join, all frequent projection-selection queries are mined, as we do in our approach.

We also note that the consideration of joins, as done in [8], implies that selection conditions involve equalities between two attributes, which we do not consider in this paper.

However, in [8] frequent projection-selection queries are mined according to the standard notion of query equivalence

([14]), whereas in our approach, we introduce a new notion of query equivalence with the following characteristics:

1. it generalizes the standard one used in [8],
2. it is based on comparisons of cardinalities, instead of being based on set inclusion as in [8], and
3. it takes functional dependencies into account, which is not the case in [8].

In our previous work [11], we consider the particular case of a database in which relations are organized according to a star schema. In this setting, frequent selection-projection queries are mined from the associated weak instance ([14]). However, in [11], equivalence classes are defined based on projection only, and thus, only projection queries can be mined through one run of the proposed level-wise algorithm. Therefore, the present work can be seen as a generalization of [11], because in the present paper, we consider (i) equivalence classes for projection-selection queries, and (ii) any set of functional dependencies (not only those associated to star schemas).

In [3], a set of attributes, called the key, provides the set of values according to which the supports are to be counted. Then, using a bias language, the different tables involving the key attributes are mined, based on a level-wise algorithm. Our approach (as well as that of [7]) can be seen as a generalization of the work in [3], in the sense that we mine all frequent queries for *all* keys.

The work of [5] follows roughly the same strategy as in [3], except that joins are first performed in a level-wise manner; and for each join, frequent queries are mined, based also on a level-wise algorithm.

All other approaches dealing with mining frequent queries [4, 9, 12, 13] consider a fixed set of "objects" to be counted during the mining phase and only one table for a given mining task. For instance, in [13], objects are characterized by values over given attributes, whereas in [4], objects are characterized by a query, called the reference. On the other hand, except for [13], all these approaches are restricted to conjunctive queries, as is the case in the present paper.

To end this section, we emphasize that, to the best of our knowledge, this work along with that of [11], is the first attempt to consider explicitly constraints on the data set, such as functional dependencies, for optimizing the computation of frequent queries.

3. FREQUENT QUERIES

3.1 Preliminaries

We assume that the reader is familiar with the relational model for which we follow the notation of [14]. In particular, we consider a relational table over a fixed set of attributes U, each attribute A being associated with a domain of values, denoted by $dom(A)$. We extend the notion of domain to any subset X of U as follows: $dom(X) = \Pi_{A \in X}(dom(A))$.

For notational convenience, tuples will be denoted by lower case characters and their schema by the corresponding upper case characters. Given a tuple x over X, for every subset Y of X, $x.Y$ denotes the restriction of x over Y.

In this work, we consider a relational table Δ defined over an attribute set U and containing no null values. Moreover, we assume that Δ satisfies a set FD of functional dependencies.

As in [14], we denote by FD^+ the set of all functional dependencies that can be inferred from FD, based on Armstrong axioms ([2]). For every $X \rightarrow Y$ in FD^+ and for every tuple x over X, we denote by $\Delta_Y(x)$ the tuple over Y associated in Δ with x through $X \rightarrow Y$.

Referring back to Figure 1, $Cid \rightarrow Caddr$ is in FD^+, and we have, for instance, $\Delta_{Caddr}(\mathtt{c}_1) = \mathtt{Paris}$, as all tuples t such that $t.Cid = \mathtt{c}_1$ satisfy $t.Caddr = \mathtt{Paris}$. In this paper, given an attribute set X, we consider the notions of closure and keys of X as in [14], that is:

- X^+ denotes the *closure* of X (with respect to FD), namely, the set of all attributes A in U such that the dependency $X \rightarrow A$ is in FD^+,
- $keys(X)$ denotes the set of all *keys* of X^+, namely, the set of all minimal K (with respect to set inclusion) such that $K \subseteq X^+$ and $K \rightarrow X^+ \in FD^+$.

Moreover, given a query q, we denote by $ans(q)$ the answer to q in Δ and by $|ans(q)|$ the cardinality of $ans(q)$.

Functional dependencies allow for comparisons of the cardinalities of the answers to queries, as stated below.

LEMMA 1. *For all nonempty schemas X and Y such that $X \rightarrow Y$ is in FD^+:*

1. $|ans(\pi_X(\Delta))| \geq |ans(\pi_Y(\Delta))|$ and
2. $|ans(\sigma_{X=x}(\Delta))| \leq |ans(\sigma_{Y=\Delta_Y(x)}(\Delta))|$.

PROOF. 1. Since Δ satisfies $X \rightarrow Y$, there exists a total, onto function from $ans(\pi_X(\Delta))$ to $ans(\pi_Y(\Delta))$. Thus, $|ans(\pi_X(\Delta))| \geq |ans(\pi_Y(\Delta))|$.
2. For every t in $ans(\sigma_{X=x}(\Delta))$, $t.X = x$. Thus, $t.Y = \Delta_Y(x)$, showing that t is in $ans(\sigma_{Y=\Delta_Y(x)}(\Delta))$. Therefore, we have $|ans(\sigma_{X=x}(\Delta))| \leq |ans(\sigma_{Y=\Delta_Y(x)}(\Delta))|$. $\square$

In the standard relational model, schemas are assumed to be *nonempty* sets. However, for the purposes of this paper, we also consider the empty schema, denoted by $\emptyset$. The set $dom(\emptyset)$ is assumed to contain a single tuple, namely the *empty tuple*, denoted by $\top$. We consider functional dependencies involving $\emptyset$ along with the following rules, which can be shown to be consistent with Armstrong axioms.

- For every X (empty or not) X, every table Δ satisfies $X \rightarrow \emptyset$. Thus, for every x in $dom(X)$, $\top = \Delta_\emptyset(x)$.
- For every set of functional dependencies FD, $keys(\emptyset) = \{\emptyset\}$ and $\emptyset^+ = \emptyset$.

3.2 Queries

The projection-selection queries considered here are standard, conjunctive projection-selection relational queries, along with their extension to the empty schema.

DEFINITION 1. *A* conjunctive selection condition, *or a selection condition for short, is an equality of the form $Y = y$ where Y is a possibly empty relation schema and y a tuple in $dom(Y)$. Let $S = (Y = y)$ be a selection condition. A tuple t over U is said to* satisfy *S if: either $Y = \emptyset$ and $y = \top$, or $Y \neq \emptyset$ and $t.Y = y$.*

We denote by $\mathcal{Q}(\Delta)$ the set of all queries of the form $\pi_X(\sigma_{Y=y}(\Delta))$ such that $X \neq \emptyset$ or $Y \neq \emptyset$, and $y \in \pi_Y(\Delta)$.

In order to simplify notation, a projection-selection query of the form $\pi_X(\sigma_{Y=y}(\Delta))$ will be denoted by $\langle X, y\rangle$, and its answer in $\mathcal{Q}(\Delta)$ by $ans_\Delta(\langle X, y\rangle)$ (or simply $ans(\langle X, y\rangle)$ when Δ is understood). This answer is defined as usual if $X \neq \emptyset$, and if $X = \emptyset$ then $ans_\Delta(\langle \emptyset, y\rangle) = ans_\Delta(\langle Y, y\rangle)$.

We note that all queries in $\mathcal{Q}(\Delta)$ of the form $\langle X, \top\rangle$ where $X \neq \emptyset$ are simply the relational queries of the form $\pi_X(\Delta)$.

Therefore, in our approach, a query is defined by a relation schema X (which can be empty) together with a tuple y (which can also be empty). However, the case where X and Y are both empty is *not* considered here, simply because the resulting query would be meaningless.

It is also important to note that we restrict queries $\langle X, y\rangle$ to be such that y appears in Δ, simply because, otherwise, the answer is known to be empty, and thus not relevant regarding frequency computations.

Example 2. In the context of Example 1, the queries $q_1 = \langle Cid, \texttt{Paris}\rangle$ and $q_2 = \langle Cid, \texttt{Paris beer}\rangle$ are queries in $\mathcal{Q}(\Delta)$ and we have $ans(q_1) = \{\texttt{c}_1, \texttt{c}_2, \texttt{c}_4\}$ and $ans(q_2) = \{\texttt{c}_1, \texttt{c}_2\}$.

On the other hand, $\langle Cid, \top\rangle$, $\langle Caddr, \texttt{Paris beer}\rangle$ and $\langle \emptyset, \texttt{Paris beer}\rangle$ are also queries in $\mathcal{Q}(\Delta)$ for which we have $ans(\langle Cid, \top\rangle) = \{\texttt{c}_1, \texttt{c}_2, \texttt{c}_3, \texttt{c}_4\}$, $ans(\langle Caddr, \texttt{Paris beer}\rangle) = \{\texttt{Paris}\}$, and $ans(\langle \emptyset, \texttt{Paris beer}\rangle) = \{\texttt{Paris beer}\}$.

However, the query $\langle Cid\, Caddr, \texttt{NY 15}\rangle$ is *not* in $\mathcal{Q}(\Delta)$ because NY 15 is not a tuple in $\pi_{Caddr\, Qty}(\Delta)$. □

The notion of *frequent query* in our approach is defined in much the same way as in [6].

DEFINITION 2. *For every query q in $\mathcal{Q}(\Delta)$, the* support *of q in Δ, denoted by $sup_\Delta(q)$ (or simply $sup(q)$ when Δ is understood) is the cardinality of the answer to q,* i.e., *$sup_\Delta(q) = |ans_\Delta(q)|$.*

Given a support threshold min-sup, a query q is said to be frequent in Δ *(or simply* frequent *when Δ is understood) if $sup_\Delta(q) \geq$ min-sup.*

Referring back to queries q_1 and q_2 in Example 2, it is easy to see that $sup(q_1) = 3$, $sup(q_2) = 2$. Thus, for a support threshold equal to 3, q_1 is frequent whereas q_2 is not.

We notice that for every Δ and every $q = \langle X, y\rangle$ in $\mathcal{Q}(\Delta)$, we have $1 \leq sup(q) \leq |\Delta|$. Moreover, $sup(\langle U, \top\rangle) = |\Delta|$ and for every Y' such that $\emptyset \subseteq Y' \subseteq Y$, $sup(Y', y) = 1$.

3.3 Comparing Queries

In our approach, the queries of $\mathcal{Q}(\Delta)$ are compared as stated in the following definition.

DEFINITION 3. *Let $q = \langle X, y\rangle$ and $q_1 = \langle X_1, y_1\rangle$ be queries in $\mathcal{Q}(\Delta)$. Then q_1 is said to be* more specific than q in Δ, *denoted by $q \preceq_\Delta q_1$ (or $q \preceq q_1$ when Δ is understood) if one of the following cases holds:*

1. *$Y_1 \to X_1 \in FD^+$, or equivalently $X_1^+ \subseteq Y_1^+$,*
2. *q and q_1 are such that*
   - *$XY_1 \to X_1 \in FD^+$, or equivalently $X_1^+ \subseteq (XY_1)^+$,*
   - *$Y_1 \to Y \in FD^+$, or equivalently $Y^+ \subseteq Y_1^+$, and*
   - *$y = \Delta_Y(y_1)$.*

The assumptions in Definition 3 imply that all queries $\langle X, y\rangle$ such that $Y \to X \in FD^+$ are the most specific queries in $\mathcal{Q}(\Delta)$. Moreover, it can be easily seen that each most specific query has a support equal to 1.

Indeed, if $Y \to X \in FD^+$ and $sup(\langle X, y\rangle) > 1$, then Δ contains at least two distinct tuples t and t' such that $t.Y = t'.Y = y$ and $t.X \neq t'.X$, which is a violation of the dependency $Y \to X$ assumed to be satisfied by Δ.

We note that all queries of the form $\langle Y', y\rangle$ where $\emptyset \subseteq Y' \subseteq Y$ are such that $Y \to Y' \in FD^+$.

The following proposition shows that the relation $\preceq$ is a pre-ordering over $\mathcal{Q}(\Delta)$, *i.e.,* $\preceq$ is reflexive and transitive.

PROPOSITION 1. *The relation $\preceq$ is a pre-ordering over $\mathcal{Q}(\Delta)$.*

PROOF. It is easy to see that $\preceq$ is reflexive, thus we only show the transitivity of $\preceq$. Let $q = \langle X, y\rangle$, $q_1 = \langle X_1, y_1\rangle$ and $q_2 = \langle X_2, y_2\rangle$ be such that $q \preceq q_1 \preceq q_2$.

If $Y_2 \to X_2$ is in FD^+, then we have $q \preceq q_2$. If $Y_2 \to X_2 \notin FD^+$, then we show that $Y_1 \to X_1$ cannot be in FD^+ either. Indeed, if $Y_1 \to X_1 \in FD^+$ then, as we assume that $q_1 \preceq q_2$, $X_1Y_2 \to X_2$ and $Y_2 \to Y_1$ are in FD^+. So, $Y_2 \to X_1$ is in FD^+, and thus, $Y_2 \to X_2 \in FD^+$, which is a contradiction. Thus, in this case, we also have that $Y \to X$ is not in FD^+. Therefore, in this case, the following dependencies are in FD^+: $XY_1 \to X_1$, $Y_1 \to Y$, $X_1Y_2 \to X_2$ and $Y_2 \to Y_1$. As a consequence, $XY_2 \to XY_1 \in FD^+$, and so, using $XY_1 \to X_1$, $XY_2 \to X_1Y_2$ is in FD^+. As $X_1Y_2 \to X_2 \in FD^+$, we obtain that $XY_2 \to X_2 \in FD^+$. On the other hand, as $Y_2 \to Y_1$ and $Y_1 \to Y$ are in FD^+, so is $Y_2 \to Y$. Moreover, as we have $y = \Delta_Y(y_1)$ and $y_1 = \Delta_{Y_1}(y_2)$, we obtain $y = \Delta_Y(y_2)$, which shows that $q \preceq q_2$. □

Example 3. Consider the table Δ of Example 1 and the queries q_1 and q_2 of Example 2:

- $q_1 \preceq q_2$, since the dependencies $Cid\, Caddr\, Ptype \to Cid$ and $Caddr\, Ptype \to Caddr$ are in FD^+, and $\Delta_{Caddr}(\texttt{Paris beer}) = \texttt{Paris}$.
- $\langle Cid, \top\rangle \preceq q_1$, since the dependencies $Cid\, Caddr \to Cid$ and $Caddr \to \emptyset$ are in FD^+, and $\top = \Delta_\emptyset(\texttt{beer})$.
- $\langle Caddr, \texttt{Paris beer}\rangle \preceq \langle \emptyset, \texttt{Paris beer}\rangle$, since the dependencies $Caddr\, Ptype \to \emptyset$ and $Caddr\ Ptype \to Caddr\, Ptype$ are in FD^+, and $\Delta_{Caddr\, Ptype}(\texttt{Paris beer}) = \texttt{Paris beer}$.
- $\langle \emptyset, \texttt{Paris beer}\rangle \preceq \langle Caddr, \texttt{Paris beer}\rangle$ for the same reasons as above.

We also have $\langle Cid, \texttt{beer}\rangle \preceq \langle Qty, \texttt{p}_2\rangle$, because $Cid\, Pid \to Qty$ and $Pid \to Ptype$ are in FD^+, and $\Delta_{Ptype}(\texttt{p}_2) = \texttt{beer}$. □

Let $q = \langle X, y\rangle$ and $q_1 = \langle X_1, y_1\rangle$. Then, it is easy to see from Definition 3 that, if $X_1 \subseteq X$ and $Y \subseteq Y_1$ and $y = y_1.Y$, then $q \preceq q_1$ holds. In particular, for every nonempty X and every tuple y over Y, we have $\langle X, \top\rangle \preceq \langle X, y\rangle$. Moreover, it can also be seen that, if $X \subseteq Y$, then $\langle \emptyset, y\rangle \preceq \langle X, y\rangle$ and $\langle X, y\rangle \preceq \langle \emptyset, y\rangle$ hold.

We note that the pre-ordering $\preceq$ generalizes query containment $\sqsubseteq$ (see [14]), because, for all queries $q = \langle X, y\rangle$ and $q_1 = \langle X_1, y_1\rangle$ in $\mathcal{Q}(\Delta)$, $q_1 \sqsubseteq q$ implies $q \preceq q_1$.

Indeed, if $q_1 \sqsubseteq q$ then q and q_1 are defined over the same schema and thus $X = X_1$. Moreover, if $q_1 \sqsubseteq q$ then, for every table Δ over U (satisfying FD or not), $ans(q_1) \subseteq$

$ans(q)$. Thus, for every tuple t over U, $t.Y_1 = y_1$ implies that $t.Y = y$. Hence, we have $Y \subseteq Y_1$ and $y_1.Y = y$, which shows that $Y_1 \rightarrow Y$ is a trivial dependency in FD^+ and that, by Definition 3 above, $q \preceq q_1$.

As a consequence of Lemma 1, the following proposition states that the support is anti-monotonic with respect to $\preceq$.

Proposition 2. *For all q and q_1 in $\mathcal{Q}(\Delta)$, if $q \preceq q_1$ then $sup(q_1) \leq sup(q)$.*

Proof. The result holds if $Y_1 \rightarrow X_1$ is in FD^+, because $sup(q_1) = 1$ and $sup(q) \geq 1$.

In the case where $Y_1 \rightarrow X_1$ is not in FD^+, then:
– If $y_1 = \top$, as $q \preceq q_1$ and $Y_1 \rightarrow Y \in FD^+$, $Y = \emptyset$. Thus, $y = \top$ and the result follows from Lemma 1(1).
– If $y_1 \neq \top$, $ans(\langle U, y_1\rangle)$ satisfies $X \rightarrow Y_1$, and as $XY_1 \rightarrow X_1 \in FD^+$, $ans(\langle U, y_1\rangle)$ satisfies $X \rightarrow X_1$. As $ans(\langle X, y_1\rangle) = \pi_X(ans(\langle U, y_1\rangle))$ and $ans(\langle X_1, y_1\rangle) = \pi_{X_1}(ans(\langle U, y_1\rangle))$, by Lemma 1(1), we obtain $|ans(\langle X_1, y_1\rangle)| \leq |ans(\langle X, y_1\rangle)|$. As $Y_1 \rightarrow Y \in FD^+$ and $y = \Delta_Y(y_1)$, by Lemma 1(2), we have $|ans(\langle X, y_1\rangle)| \leq |ans(\langle X, y\rangle)|$. Therefore, we obtain $|ans(\langle X_1, y_1\rangle)| \leq |ans(\langle X, y\rangle)|$. $\square$

We recall from [1] that anti-monotonicity is an important property for computing frequent patterns using a level-wise algorithm. In our context we use Proposition 2 in a standard way as follows: Given a support threshold *min-sup* and two queries q and q_1 in $\mathcal{Q}(\Delta)$ such that $q \preceq q_1$, if q is not frequent, then q_1 is not frequent either. Thus, in this case, it is not necessary to compute $sup(q_1)$. Further optimizations are possible, based on *query equivalence,* as defined next.

4. QUERY EQUIVALENCE

4.1 Equivalence Relation

The pre-ordering $\preceq$ induces the following equivalence relation between queries:

> Let q and q_1 be two queries in $\mathcal{Q}(\Delta)$. Then q and q_1 are said to be *equivalent*, denoted by $q \equiv q_1$, if $q \preceq q_1$ and $q_1 \preceq q$ hold.

The equivalence class of q modulo $\equiv$ is denoted by $[q]$, and the set of all equivalence classes modulo $\equiv$, *i.e.,* the set $\mathcal{Q}(\Delta)/\equiv$, is denoted by $\mathcal{C}(\Delta)$.

The pre-ordering $\preceq$ over $\mathcal{Q}(\Delta)$ induces a partial ordering over $\mathcal{C}(\Delta)$, that we shall also denote by $\preceq$. This partial ordering is defined as follows:

> For all $[q]$ and $[q_1]$ in $\mathcal{C}(\Delta)$, $[q_1]$ is said to be *more specific than* $[q]$, denoted by $[q] \preceq [q_1]$, if $q \preceq q_1$ holds.

It is easy to see that $\preceq$ is indeed a partial ordering and that it is independent from the choice of representatives. This is why we use the same notation for the pre-ordering in $\mathcal{Q}(\Delta)$ and its associated partial ordering in $\mathcal{C}(\Delta)$.

Moreover, as the relation $\preceq$ has been shown to generalize query containment, it is easy to see that the equivalence relation $\equiv$ generalizes query equivalence, as defined in [14].

It can be seen from Definition 3 that all queries $\langle X, y\rangle$ such that $Y \rightarrow X \in FD^+$ form an equivalence class. We denote this class by $\perp$ and, for every $q \in \mathcal{Q}(\Delta)$, we have $[q] \preceq \perp$, meaning that $\perp$ is the most specific class in $\mathcal{C}(\Delta)$.

The following corollary is a direct consequence of Proposition 2:

Corollary 1. *For all q and q_1 in $\mathcal{Q}(\Delta)$, if $q \equiv q_1$ then $sup(q) = sup(q_1)$.*

For instance, in Example 3, we have $\langle Caddr, \texttt{Paris beer}\rangle \preceq \langle \emptyset, \texttt{Paris beer}\rangle$ and $\langle \emptyset, \texttt{Paris beer}\rangle \preceq \langle Caddr, \texttt{Paris beer}\rangle$. Therefore, these two queries are equivalent, and it follows from Corollary 1 above that their supports are equal (this can be also checked directly in Example 2).

Based on Corollary 1, given a class $[q]$ in $\mathcal{C}(\Delta)$, we denote by $sup([q])$ the support of $[q]$, *i.e.,* the support of any query in $[q]$. Thus, similarly to individual queries, given a support threshold *min-sup*, a class $[q]$ in $\mathcal{C}(\Delta)$ is said to be *frequent* if $sup([q]) \geq$ *min-sup.*

The following theorem follows easily from Proposition 2 and Corollary 1:

Theorem 1. *For all $[q]$ and $[q_1]$ in $\mathcal{C}(\Delta)$, if $[q] \preceq [q_1]$ then $sup([q_1]) \leq sup([q])$.*

The impact of Corollary 1 and Theorem 1 on the computation of frequent queries is as follows:

1. In each equivalence class only *one computation* of support is necessary. Thus, in the algorithms that we shall present shortly, we do not consider individual queries of $\mathcal{Q}(\Delta)$, but rather, equivalence classes of $\mathcal{C}(\Delta)$.
2. Frequent classes can be computed using a level-wise algorithm.

However, considering frequent classes is effective if equivalent queries can be characterized easily and if the set $\mathcal{C}(\Delta)$ can be constructed in an efficient manner. These two issues are addressed in the next two sections.

4.2 Equivalence Classes

In order to characterize the content of classes in $\mathcal{C}(\Delta)$, we first define the notion of *keys* of a query as follows.

Definition 4. *For every $q = \langle X, y\rangle$ in $\mathcal{Q}(\Delta)$, the set of* keys of q*, denoted by $Keys(q)$, is the set of all queries $q_0 = \langle X_0, y_0\rangle$ in $\mathcal{Q}(\Delta)$ such that*
- *$X_0 = K_0 \setminus Y^+$, where $K_0 \in keys(XY)$ and*
- *$y_0 = \Delta_{Y_0}(y)$, where $Y_0 \in keys(Y)$.*

It follows from Definition 4 that, for every $q = \langle X, y\rangle$ such that $Y \rightarrow X$ is in FD^+, we have:

$$Keys(q) = \{\langle \emptyset, y_0\rangle \mid Y_0 \in keys(Y) \wedge y_0 = \Delta_{Y_0}(y)\}.$$

Indeed, in this case, $X^+ \subseteq Y^+$ and thus, for every $X_0 \in keys(XY)$, $X_0 \subseteq (XY)^+ = Y^+$. Hence, $X_0 \setminus Y^+ = \emptyset$.

Example 4. In the context of Example 1, let q be the query $\langle Cid\,Cname, \top\rangle$. As $keys(Cid\,Cname) = \{Cid\}$, we have $Keys(q) = \{\langle Cid, \top\rangle\}$.

For $q' = \langle Cid\,Ptype\,Qty, \mathtt{p}_2\rangle$, $keys(Cid\,Ptype\,Qty\,Pid) = \{Cid\,Pid\}$ and $keys(Pid) = \{Pid\}$. As $Pid^+ = Pid\,Ptype$, we have $Keys(q') = \{\langle Cid, \mathtt{p}_2\rangle\}$. $\square$

The following proposition characterizes equivalent queries.

Proposition 3. *1. $\perp = \{\langle X, y\rangle \in \mathcal{Q}(\Delta) \mid X \subseteq Y^+\}$.*

2. For every $q = \langle X, y\rangle$ in $\mathcal{Q}(\Delta)$ but not in $\perp$ we have:

$$[q] = \{\langle X_1, y_1\rangle \mid (\exists \langle X_0, y_0\rangle \in Keys(q))((X_0 \subseteq X_1 \subseteq (X_0Y_0)^+) \wedge (Y_0 \subseteq Y_1 \subseteq Y_0^+) \wedge (y_1 = \Delta_{Y_1}(y)))\}.$$

Proof. 1. If $[q] = \bot$, the result follows from Definition 3.
2. If $[q] \neq \bot$, denoting by Q the set of queries as defined in the proposition, let $q_1 = \langle X_1, y_1\rangle$ be in $[q]$. It follows from Definition 3 that $(XY)^+ = (X_1Y_1)^+$, $Y^+ = Y_1^+$ and $y_1 = \Delta_{Y_1}(y)$. It can be seen that there exists $K \in keys(XY)$ such that $K \subseteq X_1Y_1 \subseteq (XY)^+$ and $Y_0 \in keys(Y)$ such that $Y_0 \subseteq Y_1 \subseteq Y^+$ and $y_0 = \Delta_{Y_0}(y)$. Moreover, for $X_0 = K \setminus Y_0^+$, as $X_0 \cap Y_0^+ = \emptyset$ and $Y_1 \subseteq Y_0^+$, $X_0 \cap Y_1 = \emptyset$. Since $X_0 \subseteq K \subseteq X_1Y_1$, we have $X_0 \subseteq X_1Y_1$, and thus, $X_0 \subseteq X_1$. Therefore $q_1 \in Q$.

Conversely, let q_1 be in Q. As $X_0 \subseteq X_1$, $X_1 \to X_0 \in FD^+$, and as $Y_0 \subseteq Y_1$, $Y_1 \to Y_0 \in FD^+$. Therefore, $X_1Y_1 \to X_0Y_0 \in FD^+$. As $X_1 \subseteq (X_0Y_0)^+$ and $Y_1 \subseteq Y_0^+$, $X_0Y_0 \to X_1Y_1$ and $Y_0 \to Y_1$ are in FD^+. Thus, $(X_0Y_0)^+ = (X_1Y_1)^+$ and $Y_0^+ = Y_1^+$. As we also have $y_1 = \Delta_{Y_1}(y_0)$, it can be seen that this entails that $q_1 \equiv \langle X_0, y_0\rangle$, and thus that $q_1 \in [q]$ (because it is easy to see that $q_0 \equiv q$). □

As a consequence of Proposition 3, for every $q = \langle X, y\rangle$ in $\mathcal{Q}(\Delta)$ such that $[q] \neq \bot$, $[q]$ contains exactly one representative $\langle X', y'\rangle$ such that $X' = X^+$, $Y' = Y^+$ and $Y' \subset X'$.

Thus, $\mathcal{C}(\Delta) \setminus \{\bot\}$ is isomorphic to the set of all queries $\langle X, y\rangle$ such that $X^+ = X$, $Y^+ = Y$ and $Y \subset X$. In the remainder of the paper, we identify every class of $\mathcal{C}(\Delta) \setminus \{\bot\}$ with this unique, particular representative.

For instance, consider again the queries q and q' of Example 4. We have:

– $[q] = \{\langle Cid, \top\rangle, \langle Cid\,Cname, \top\rangle, \langle Cid\,Caddr, \top\rangle, \langle Cid\,Cname\,Caddr, \top\rangle\}$
– $[q'] = \{\langle X, y\rangle \mid (Cid \subseteq X \subseteq U) \wedge (y = \mathtt{p}_2 \vee y = \mathtt{p}_2\,\mathtt{beer})\}$.

The classes $[q]$ and $[q']$ are respectively represented by:

– $\langle Cid\,Cname\,Caddr, \top\rangle$ and
– $\langle Cid\,Cname\,Caddr\,Pid\,Ptype\,Qty, \mathtt{p}_2\,\mathtt{beer}\rangle$.

4.3 Successors of an Equivalence Class

In this section, we show how to generate the set $\mathcal{C}(\Delta)$. First, given a class q in $\mathcal{C}(\Delta)$, call *successors* of q, denoted by $succ(q)$, the set of all classes $\langle X', y'\rangle$ such that:

1. $\langle X, y\rangle \preceq \langle X', y'\rangle$ and
2. if $\langle X'', y''\rangle$ is such that $\langle X, y\rangle \preceq \langle X'', y''\rangle \preceq \langle X', y'\rangle$, then $\langle X'', y''\rangle = \langle X, y\rangle$ or $\langle X'', y''\rangle = \langle X', y'\rangle$.

In order to characterize the set $succ(q)$, we introduce the following notation. We denote by $cl(FD)$ the set of all schemas X such that $X = X^+$ and, given a schema X in $cl(FD)$, $X^\downarrow$ (respectively $X^\uparrow$) denotes the set of all maximal (respectively minimal) schemas X' of $cl(FD)$ such that $X' \subset X$ (respectively $X \subset X'$). Notice however that $U^\uparrow$ and $\emptyset^\downarrow$ are not defined.

Proposition 4. *Given a class $q = \langle X, y\rangle$ in $\mathcal{C}(\Delta)$ different than $\bot$, $succ(q)$ is equal to the set of all classes computed according to the following:*

1. *$succ(q) = \{\bot\}$ if and only if, for every $Y' \in Y^\uparrow$, $X \subseteq Y'$.*
2. *For every nonempty schema Z, if*
   (a) $Y \subset (X \setminus Z)$, and
   (b) $(X \setminus Z) \in X^\downarrow$,
   then $succ(q)$ contains the class $\langle X', y'\rangle$ where $X' = (X \setminus Z)$, $Y' = Y$ and $y' = y$.
3. *For every tuple z over any nonempty schema Z such that $yz \in \pi_{YZ}(\Delta)$, if*
   (a) $Z \subseteq X$,
   (b) $YZ \in Y^\uparrow \setminus \{X\}$, and
   (c) there exists no X_0 in $cl(FD) \setminus \{\emptyset\}$ such that $Y \subseteq X_0 \subset X \subseteq (X_0Z)^+$,
   then $succ(q)$ contains the class $\langle X', y'\rangle$ where $X' = X$, $Y' = YZ$ and $y' = yz$.
4. *For every tuple z over any nonempty schema Z such that $yz \in \pi_{YZ}(\Delta)$, if*
   (a) $Z \not\subseteq X$,
   (b) $YZ \in Y^\uparrow$, and
   (c) there exists no X_0 in $cl(FD)$ such that $Y \subseteq X_0 \subset X$ and $(X_0Z)^+ = (XZ)^+$,
   then $succ(q)$ contains the class $\langle X', y'\rangle$ where $X' = (XZ)^+$, $Y' = YZ$ and $y' = yz$.

Proof. See Appendix A. □

We illustrate Proposition 4 through the following examples.

Example 5. Let us consider the case where $U = \{A, B\}$, $FD = \emptyset$ and $\Delta = \{ab\}$. We note that, in this case, every subset of U is in $cl(FD)$.

For $\langle AB, \top\rangle$, based on Proposition 4, the set $succ(\langle AB, \top\rangle)$ is computed as follows:

1. $succ(\langle AB, \top\rangle) \neq \{\bot\}$, because $A \in \emptyset^\uparrow$ and $AB \not\subseteq A$.

2. Let us consider $Z = B$, *i.e.,* the class $\langle A, \top\rangle$. Then we have (a) $\emptyset \subset (AB \setminus B) = A$ and (b) $A \in AB^\downarrow$. Notice that the same reasoning also holds if we consider $Z = A$, *i.e.,* the class $\langle B, \top\rangle$. Thus, $\langle A, \top\rangle$ and $\langle B, \top\rangle$ are in $succ(\langle AB, \top\rangle)$.

3. Proposition 4(3) does not apply. Indeed, for $Z = A$, although we have (a) $A \subseteq AB$ and (b) $A \neq AB$ and $A \in \emptyset^\uparrow$, it is the case that, (c) for $X_0 = B$, $B \in cl(FD)$ and $\emptyset \subseteq B \subset AB \subseteq (AB)^+ = AB$. Notice moreover that a similar argument holds for $Z = B$.

4. Proposition 4(4) does not apply either, because $X = AB = U$.

Thus, we have that $succ(\langle AB, \top\rangle) = \{\langle A, \top\rangle, \langle B, \top\rangle\}$. Now, considering the class $\langle A, \top\rangle$, based on Proposition 4, the set $succ(\langle A, \top\rangle)$ is computed as follows:

1. $succ(\langle A, \top\rangle) \neq \{\bot\}$, because $B \in \emptyset^\uparrow$ and $A \not\subseteq B$.

2. Proposition 4(2) does not apply, because removing A results in the empty set.

3. Proposition 4(3) does not apply, because the only Z to be considered is $Z = A$, and (b) $A \notin \emptyset^\uparrow \setminus \{A\} = \{B\}$.

4. Let us consider $Z = B$, *i.e.,* the class $\langle AB, b\rangle$. Then: (a) $B \not\subseteq A$, (b) $B \in \emptyset^\uparrow$, and (c) there exists no $X_0 \in cl(FD)$ such that $\emptyset \subseteq X_0 \subset B$ and $(X_0B)^+ = (AB)^+$ (because, in this case, $X_0 = \emptyset$ and $B^+ = B \neq (AB)^+ = AB$). Thus, $\langle AB, b\rangle$ is in $succ(\langle A, \top\rangle)$.

Therefore, $succ(\langle A, \top\rangle) = \{\langle AB, b\rangle\}$. Applying the same reasoning as above implies that $succ(\langle B, \top\rangle) = \{\langle AB, a\rangle\}$.

Going one step further, we have that $succ(\langle AB, a\rangle) = succ(\langle AB, b\rangle) = \{\bot\}$, because, as $A^\uparrow = B^\uparrow = \{AB\}$, Proposition 4(1) applies. □

Example 6. Let $U = \{A, B, C, D\}$, $FD = \{ABC \to D\}$ and $\Delta = \{abcd\}$ over U. The successors of $q_1 = \langle BC, \top\rangle$ are computed as follows:

1. $succ(q_1) \neq \{\bot\}$, because $B \in \emptyset^{\uparrow}$ and $BC \not\subseteq B$.

2. Let us consider $Z = B$, *i.e.,* the class $\langle C, \top\rangle$. Then: (a) $\emptyset \subset (BC \setminus B) = C$ and (b) $C \in BC^{\downarrow}$.

As the same reasoning holds for $Z = C$, *i.e.,* for the class $\langle B, \top\rangle$, the classes $\langle B, \top\rangle$ and $\langle C, \top\rangle$ are in $succ(q_1)$.

3. Let us consider $Z = B$, *i.e.,* the class $\langle BC, b\rangle$. Then: (a) $B \subseteq BC$, (b) $B \neq BC$ and $B \in \emptyset^{\uparrow}$, but (c) for $X_0 = C$, we have C in $cl(FD) \setminus \{\emptyset\}$, $\emptyset \subseteq C \subset BC \subseteq (BC)^+ = BC$.

Thus, Proposition 4(3) does not apply. Notice that the same reasoning holds for $Z = C$, *i.e.,* for the class $\langle BC, c\rangle$.

4. Let us consider $Z = A$, *i.e.,* the class $\langle (ABC)^+, a\rangle = \langle ABCD, a\rangle$. Then: (a) $A \not\subseteq BC$, (b) $A \in \emptyset^{\uparrow}$ and (c) there is no X_0 in $cl(FD)$ such that $\emptyset \subseteq X_0 \subset BC$ and $(X_0A)^+ = (ABC)^+$. Thus, $\langle ABCD, a\rangle$ is in $succ(q_1)$.

For $Z = D$, *i.e.,* for $\langle (BCD)^+, d\rangle = \langle BCD, d\rangle$, we have: (a) $D \not\subseteq BC$, (b) $D \in \emptyset^{\uparrow}$ and (c) there is no X_0 in $cl(FD)$ such that $\emptyset \subseteq X_0 \subset BC$ and $(X_0D)^+ = (BC)^+ = BC$. Thus, $\langle BCD, d\rangle$ is in $succ(q_1)$.

Therefore, $succ(\langle BC, \top\rangle) = \{\langle B, \top\rangle, \langle C, \top\rangle, \langle ABCD, a\rangle, \langle BCD, d\rangle\}$.

It can be seen, using similar arguments, that we have $succ(\langle BD, \top\rangle) = \{\langle B, \top\rangle, \langle D, \top\rangle, \langle ABD, a\rangle, \langle BCD, c\rangle\}$.

Moreover, using Proposition 4(2), we also have $\langle ABD, a\rangle \in succ(\langle ABCD, a\rangle)$. Thus, we obtain:

- $\langle BC, \top\rangle \preceq \langle B, \top\rangle$ and $\langle BC, \top\rangle \preceq \langle ABD, a\rangle$,
- $\langle BD, \top\rangle \preceq \langle B, \top\rangle$ and $\langle BD, \top\rangle \preceq \langle ABD, a\rangle$.

It follows that $\langle B, \top\rangle$ and $\langle ABD, a\rangle$ are two distinct least upper bounds of $\langle BC, \top\rangle$ and $\langle BD, \top\rangle$. Therefore, the set $\mathcal{C}(\Delta)$ is *not* a lattice.

Now, in order to illustrate a case where Proposition 4(3) applies, consider the class $\langle ABCD, \top\rangle$.

For $Z = D$, we have (a) $D \subseteq ABCD$, (b) $D \neq ABCD$ and $D \in \emptyset^{\uparrow}$, and (c) there is no $X_0 \in cl(FD) \setminus \{\emptyset\}$ such that $\emptyset \subseteq X_0 \subset ABCD \subseteq (X_0D)^+$.

Indeed, regarding point (c), the last inclusion above entails that $(X_0D)^+ = ABCD$, meaning that the only possibility for X_0 is $X_0 = ABC$. However, this is not possible because $ABC \notin cl(FD)$, and thus, $\langle ABCD, d\rangle$ is in $succ(\langle ABCD, \top\rangle)$. □

It is important to note that, as shown in Example 6, the set $\mathcal{C}(\Delta)$ is not a lattice in general. However, despite this negative result, it is still possible to design a level-wise algorithm for computing all frequent classes, in some important cases. In the following section we present such a case study for star schemas in data warehouses.

5. A CASE STUDY: STAR SCHEMAS

5.1 Basics

An *N-dimensional star schema* consists of a distinguished table φ with schema F, called the *fact table*, and N other tables $\delta_1, \ldots, \delta_N$ with schemas $D_1, \ldots, D_N$, called the *dimension tables*, such that:

1. If $K_1, \ldots, K_N$ are the (primary) keys of $\delta_1, \ldots, \delta_N$, respectively, then $K = K_1 \cup \ldots \cup K_N$ is the key of φ;

2. For every $i = 1, \ldots, N$, $\pi_{K_i}(\varphi) \subseteq \pi_{K_i}(\delta_i)$ (thus each K_i is a foreign key in the fact table φ).

The attribute set $M = F \setminus K$ is called the *measure* of the star schema.

Clearly, the table Δ of Figure 1 can be seen as the result of joining the following three tables: $Customer(Cid, Cname, Caddr)$, $Product(Pid, Ptype)$, $Sales(Cid, Pid, Qty)$ where $Customer$ and $Product$ are the dimension tables and $Sales$ is the fact table. Moreover, in this example, $K = \{Cid, Pid\}$.

From now on, we consider an N-dimensional star schema with the corresponding set of functional dependencies FD. In this setting, we show how to mine all frequent projection-selection classes from the table Δ obtained by joining all tables of the star schema.

We note that Example 6 above shows that, in the case of a star schema, the set $\mathcal{C}(\Delta)$ is not a lattice. This is so because the functional dependency $ABC \to D$ in Example 6 can be considered as specifying the fact that the key of the fact table of a 3-dimensional star schema is $K = ABC$.

However, we show in this section that, nevertheless, frequent queries can be mined based on standard level-wise algorithms in this case.

First, the following lemma shows that the schemas in $cl(FD)$ can easily be characterized.

LEMMA 2. *In the case of a star schema, X is in $cl(FD)$ if and only if: either $X = U$, or $K \not\subseteq X$ and for every $K_i \in X$, $D_i \subseteq X$.*

PROOF. – If $K \subseteq X$, then $X^+ = U$ as K is a key of U. As K is the only key of U, U is the only schema in $cl(FD)$ containing K.
– If $K \not\subseteq X$, then the result holds because in this case, the only dependencies of FD^+ that can be applied are the form $K_i \to Z$ such that $K_i \in X$. □

5.2 Successors in Star Schemas

Based on Proposition 4, for every $q \in \mathcal{C}(\Delta)$, we characterize the set $succ(q)$ in the case of a star schema. Given an attribute set X in $cl(FD)$ different than U and $\emptyset$, we have:

- $X^{\downarrow}$ is the set of all schemas $X \setminus A$ where A is an attribute in X such that
  – $A \in K$, or
  – $(\exists i \in \{1, \ldots, N\})(K_i \notin X$ and $A \in D_iM)$.
- $X^{\uparrow}$ is the set of all schemas XA where A is an attribute not in X such that
  – $A \notin K$, or
  – $(\exists i \in \{1, \ldots, N\})(A = K_i$ and $(D_i \setminus K_i) \subseteq X)$.

Moreover, $U^{\downarrow} = \{U \setminus A \mid A \in K\}$ and $\emptyset^{\uparrow} = \{A \mid A \notin K\}$.

Now, given a class $\langle X, y\rangle$ different than $\bot$, we use Proposition 4 in order to explicitly characterize the set $succ(\langle X, y\rangle)$ in the case of a star schema.

PROPOSITION 5. *Let $q = \langle X, y\rangle$ be a class in $\mathcal{C}(\Delta)$ different than $\bot$. Then, $succ(q)$ is the set of all classes $\langle X', y'\rangle$ defined as follows:*

1. *$succ(q) = \{\bot\}$ if and only if $Y^{\uparrow} = \{X\}$.*

2. *If $A \in (X \setminus Y)$ is such that $Y \subset (X \setminus A)$ and $(X \setminus A) \in X^{\downarrow}$, then $X' = (X \setminus A)$ and $y = y'$.*

3. *If A is such that $A \in (X \setminus Y)$, $YA \in Y^{\uparrow} \setminus \{X\}$ and $(X \setminus A) \notin cl(FD) \setminus \{\emptyset\}$, then $X' = X$, $Y' = YA$ and $y' = ya$ where $ya \in \pi_{YA}(\Delta)$.*

4. If A is such that $A \notin X$, $YA \in Y^\uparrow$ and y' is any tuple over Y' such that $y' = ya$ and $ya \in \pi_{YA}(\Delta)$, then
(a) if $A \in K$, $(K \setminus A) \subseteq X$ and $(M \cap (X \setminus Y)) = \emptyset$, then $X' = U$ and $Y' = YA$,
(b) if $A \notin K$ then $X' = XA$ and $Y' = YA$.

PROOF. See Appendix B. □

Example 7. In the context of Example 1, we illustrate Proposition 5 through the computation of $succ(\langle X, \texttt{beer}\rangle)$ where $X = Cid\,Cname\,Caddr\,Ptype$. In this case, we have:

1. $succ(\langle X, \texttt{beer}\rangle) \neq \{\bot\}$ because $Ptype^\uparrow = \{Pid\,Ptype, Ptype\,Qty\} \neq \{X\}$.

2. For $A = Cid$, we have that $Cid \in (X \setminus Ptype)$ and $Cname\,Caddr\,Ptype \in X^\downarrow$. Thus, $\langle Cname\,Caddr\ Ptype, \texttt{beer}\rangle \in succ(\langle X, \texttt{beer}\rangle)$.

This item does not apply to any other A in $(X \setminus Ptype)$, because, as $X^\downarrow = \{Cid\,Cname\,Caddr, Cname\,Caddr\,Ptype\}$, we would not have $(X \setminus A) \in X^\downarrow$.

3. For $A = Cname$, we have $Cname \in (X \setminus Ptype)$, $(X \setminus Cname) \notin cl(FD) \setminus \{\emptyset\}$ and $Ptype\,Cname \in Ptype^\uparrow$. Thus the classes $\langle X, \texttt{John beer}\rangle$, $\langle X, \texttt{Mary beer}\rangle$ and $\langle X, \texttt{Paul beer}\rangle$ are in $succ(\langle X, \texttt{beer}\rangle)$.

Moreover, a similar reasoning with $A = Caddr$ shows that $\langle X, \texttt{Paris beer}\rangle$ and $\langle X, \texttt{NY beer}\rangle$ belong to $succ(\langle X, \texttt{beer}\rangle)$.

4. The only attributes not in X are Pid and Qty.
– For $A = Pid$, we have $Pid \in Cid\,Pid$, $(Qty \cap (X \setminus Ptype)) = \emptyset$, $((Cid\,Pid) \setminus Pid) \subseteq X$ and $Pid\,Ptype \in Ptype^\uparrow$. Therefore, $\langle U, \texttt{p}_2\,\texttt{beer}\rangle$ and $\langle U, \texttt{p}_3\,\texttt{beer}\rangle$ belong to $succ(\langle X, \texttt{beer}\rangle)$.
– For $A = Qty$, we have $Qty \notin Cid\,Pid$, and $Ptype\,Qty \in Ptype^\uparrow$. Thus, $\langle U \setminus Pid, \texttt{beer 10}\rangle$ and $\langle U \setminus Pid, \texttt{beer 5}\rangle$ belong to $succ(\langle X, \texttt{beer}\rangle)$.

Consequently, $succ(\langle X, \texttt{beer}\rangle)$ is the following set:

$\{\langle Cname\,Caddr\,Ptype, \texttt{beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype, \texttt{John beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype, \texttt{Mary beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype, \texttt{Paul beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype, \texttt{Paris beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype, \texttt{NY beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Pid\,Ptype\,Qty, \texttt{p}_2\,\texttt{beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Pid\,Ptype\,Qty, \texttt{p}_3\,\texttt{beer}\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype\,Qty, \texttt{beer 10}\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype\,Qty, \texttt{beer 5}\rangle\}$. □

5.3 Algorithms

In order to avoid generating all candidate classes, in our algorithms we consider *generic classes* defined as follows:

> Given two schemas X and Y in $cl(FD)$, the generic class $\langle X, Y\rangle$ is the set of all classes $\langle X, y\rangle$ such that y is in $\pi_Y(\Delta)$, *i.e.*, $\langle X, Y\rangle = \{\langle X, y\rangle \in \mathcal{C}(\Delta) \mid y \in \pi_Y(\Delta)\}$.

A generic class $\langle X, Y\rangle$ is said to be frequent if it contains a frequent class $\langle X, y\rangle$. Moreover, the set of successors of a generic class $\langle X, Y\rangle$, denoted by $succ(\langle X, Y\rangle)$, is computed based on Proposition 5, except that, for generic classes, the different tuples defining the selection conditions have not to be considered.

Referring back to Example 7, the generic class to be considered is $\langle X, Ptype\rangle$ where $X = Cid\,Cname\,Caddr\,Ptype$. The set $succ(\langle X, Ptype\rangle)$ is then the following:

$\{\langle Cname\,Caddr\,Ptype, Ptype\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype, Cname\,Ptype\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype, Caddr\,Ptype\rangle,$
$\langle Cid\,Cname\,Caddr\,Pid\,Ptype\,Qty, Pid\,Ptype\rangle,$
$\langle Cid\,Cname\,Caddr\,Ptype\,Qty, Ptype\,Qty\rangle\}$.

Generic classes are built up and the supports of the corresponding classes are computed during the *same* scan of Δ.

In other words, given a set of candidate generic classes at a given level of the computation, a scan of Δ performs the following two tasks: (i) build up all generic classes and (ii) compute the supports of the corresponding classes.

It is important to note from a computational point of view, that the size of the set of all generic classes is exponential in the size of the attribute set U, but *independent from the size of* Δ, contrary to the set $\mathcal{C}(\Delta)$, whose size is exponential in the size of U and also in the size of Δ.

Based on these considerations, the main algorithm we propose, shown in Figure 2, follows the same strategy as the Apriori algorithm ([1]). Namely at each level:

1. The set of all candidate generic classes, denoted by C, is generated from the set of all frequent generic classes computed at the previous level, denoted by L.

2. The set C is pruned using the so-called Apriori trick, that is: if in C, $\langle X, Y\rangle$ is such that there exists a generic class $\langle X', Y'\rangle$ for which $\langle X, Y\rangle \in succ(\langle X', Y'\rangle)$ and $\langle X', Y'\rangle \notin L$, then $\langle X, Y\rangle$ is not frequent.

3. During the scan of Δ, every remaining generic class $\langle X, Y\rangle$ in C is built up and the supports of the resulting classes are computed. The resulting set of frequent classes, denoted by L_{Freq}, is added into the current set of frequent classes, denoted by $Freq$.

We note that the second item above is shown to be correct as follows. If $\langle X', Y'\rangle$ is not in L then, for every y' in $\pi_{Y'}(\Delta)$, the class $\langle X', y'\rangle$ is not frequent. Thus, for every y in $\pi_Y(\Delta)$, there exists a non frequent class $\langle X', y'\rangle$ in $\langle X', Y'\rangle$ such that $\langle X, y\rangle \in succ(\langle X', y'\rangle)$.

Algorithm 1
Input: The table Δ associated to an N-dimensional star schema and a support threshold *min-sup*.
Output: The set $Freq$ of all frequent classes.
Method:

```
if $|\Delta| < min\text{-}sup$ then
    //no computation as for every $q \in \mathcal{C}(\Delta)$, $|\Delta| \geq sup(q)$
    $Freq = \emptyset$
else //the computation starts with $\langle U, \top\rangle$
    $L = \{\langle U, \emptyset\rangle\}$ ; $Freq = \{\langle U, \top\rangle\}$
    while $L \neq \emptyset$ do
        $C$ = generate($L$)
        $C$ = prune($C, L$)
        scan($C, L, L_{Freq}$)
        //all generic classes of $L$ are instanciated and the
        //supports of the corresponding classes are computed
        $Freq = Freq \cup L_{Freq}$
    end while
end if
return $Freq$
```

Figure 2: The main algorithm

```
Algorithm generate
Input: A set $L$ of frequent generic classes at level $l$.
Output: The set $C$ of all candidate generic classes obtained from $L$ at level $l+1$.
Method:
$C = \emptyset$
for each $\langle X, Y\rangle \in L$ do
    if $Y^{\uparrow} = \{X\}$ then
        //Proposition 5(1)
        $C = \{\perp\}$
    else
        for each $A \in (X \setminus Y)$ do
            if $(X \setminus A) \in X^{\downarrow}$ then
                //Proposition 5(2)
                $C = C \cup \{\langle (X \setminus A), Y\rangle\}$
            end if
            if $(X \setminus A) \notin cl(FD) \setminus \{\emptyset\}$ and $YA \in Y^{\uparrow}$ then
                //Proposition 5(3)
                $C = C \cup \{\langle X, YA\rangle\}$
            end if
        end for each
        for each $A \in (U \setminus X)$ do
            if $A \in K$ and $(K \setminus A) \subseteq X$ and $(M \cap (X \setminus Y)) = \emptyset$
            and $YA \in Y^{\uparrow}$ then
                //Proposition 5(4.a)
                $C = C \cup \{\langle U, YA\rangle\}$
            end if
            if $A \notin K$ and $YA \in Y^{\uparrow}$ then
            //Proposition 5(4.b)
                $C = C \cup \{\langle XA, YA\rangle\}$
            end if
        end for each
    end if
end for each
return $C$
```

Figure 3: The algorithm for generating candidates

As a consequence, the anti-monotonicity of the support stated in Theorem 1 implies that $\langle X, y\rangle$ cannot be frequent. Hence, the generic class $\langle X, Y\rangle$ is not frequent.

In the algorithm of Figure 2, each of the three steps calls one algorithm, called `generate`, `prune` and `scan` are given in figures 3, 4 and 5, respectively. The following comments are in order regarding these algorithms and their complexity.

Algorithm `generate`. It is easy to see that this algorithm follows Proposition 5. It should be noticed that the different tests on schemas can be efficiently implemented by representing every schema X as an array of booleans the length of which being the cardinality of U.

To this end, we order the attributes as follows: Based on a fixed ordering of the dimensions, we first consider all key attributes according to the dimension ordering, then all non-key dimensional attributes appear at consecutive positions according to the dimension ordering, and then, all measure attribues are considered.

With this ordering at hand, it is possible to implement Algorithm `generate` in such a way that candidate generation is processed in a non redundant manner, *i.e.,* a given candidate is generated in C only once, even if it is a successor of more than one generic class in L.

Algorithm `prune`. It can be seen that this algorithm follows Proposition 5, but in a different manner, since in this algorithm, we are interested in computing *predecessors* of a generic class $\langle X, Y\rangle$, *i.e.,* the set of those classes $\langle X', Y'\rangle$ such that $\langle X, Y\rangle$ belongs to $succ(\langle X', Y'\rangle)$.

```
Algorithm prune
Input: A set $L$ of frequent generic classes at level $l$, the set $C$ of candidate generic classes generated from $L$.
Output: The pruned set $C$.
Method:
//$C$ is assumed to be different than $\{\perp\}$
for each $\langle X, Y\rangle \in C$ do
    for each $A \notin X$ such that $XA \in X^{\uparrow}$ do
        if $\langle XA, Y\rangle \notin L$ then                //Proposition 5(2)
            $C = C \setminus \{\langle X, Y\rangle\}$
        end if
    end for each
    for each $A \in Y$ do
        if $\big((X \setminus A) \notin cl(FD) \setminus \{\emptyset\}$ and $(Y \setminus A) \in Y^{\downarrow}$ and
            $\langle X, (Y \setminus A)\rangle \notin L\big)$          //Proposition 5(3)
        or $\big(A \in K$ and $X = U$ and $(Y \setminus A) \in Y^{\downarrow}$ and
            $\langle (U \setminus ((M \setminus Y)A)), (Y \setminus A)\rangle \notin L\big)$
                                                  //Proposition 5(4.a)
        or $\big(A \notin K$ and $(Y \setminus A) \in Y^{\downarrow}$ and
            $\langle (X \setminus A), (Y \setminus A)\rangle \notin L\big)$   //Proposition 5(4.b)
        then
            $C = C \setminus \{\langle X, Y\rangle\}$
        end if
    end for each
end for each
return $C$
```

Figure 4: The algorithm for pruning candidates

Therefore, for each item in Proposition 5, if one of the predecessors of the generic class $\langle X, Y\rangle$ under consideration is not in L, then $\langle X, Y\rangle$ is pruned.

The only case to be clarified is that of Propostion 5(4.a). In this case, under the hypotheses stated in this part of the algorithm, it can be shown that $X' = (U \setminus ((M \setminus Y)A))$ is the only schema such that $A \notin X'$, $(K \setminus A) \subseteq X'$ and $(M \cap (X' \setminus (Y \setminus A))) = \emptyset$. Thus, X' is the only schema for which Propostion 5(4.a) shows that $\langle X, Y\rangle \in succ(\langle X', (Y \setminus A)\rangle)$.

Algorithm `scan`. We recall that, while scanning Δ, Algorithm `scan` performs the following two tasks: (*i*) build up the generic classes in C and (*ii*) compute the supports of all associated classes.

Task (*i*) is achieved as follows: Every $\langle X, Y\rangle$ in C is associated to a set of classes, denoted by $L(\langle X, Y\rangle)$ and set to $\emptyset$ in the first loop of Algorithm `scan`. Then, for every tuple t in Δ, when considering $\langle X, Y\rangle$, if $\langle X, t.Y\rangle$ is not in $L(\langle X, Y\rangle)$, then it is added and its support is intialized to 1; otherwise, the support of $\langle X, t.Y\rangle$ is updated.

We note that, at this step no pruning is performed, as done when computing the set C. The reason why we skip this step is that, when the number of candidates is huge (which is the case in our approach when dealing with classes in $\mathcal{C}(\Delta)$), computing the supports of all candidates is faster than pruning and computing the supports of the remaining candidates. However, we plan to investigate the two options when implementing and testing the algorithms.

Regarding task (*ii*) of Algorithm `scan`, it is important to note that, contrary to the standard Apriori algorithm ([1]), the main difficulty in counting the supports is that a tuple over a given schema may appear in several rows of Δ, but has to be counted only once. Since we have to consider several classes at the same time (*i.e.,* the set of all generic classes in C), it is not possible to sort Δ accordingly, as done in [3, 4]. Note also that this remark shows that an FP-growth technique ([10]) can not be efficiently used in our appoach, since it would also require that the database be sorted for each candidate.

This explains why, in Algorithm `scan`, using the notation $pred(t)$ for the set of all tuples appearing in Δ *before* the tuple t under consideration, the test "`if` $t.X \notin pred(t)$" has to be performed. In this way, given the class $\langle X, t.Y\rangle$, it is checked whether the value $t.X$ has been previously counted. If so (*i.e.,* if $t.X$ appeared in Δ associated with $t.Y$ before the current occurrence in t), then the test fails and the support count is not changed. Otherwise, the support count is incremented by 1.

We point out in this respect that, contrary to standard level-wise algorithms, counting the supports of all candidates at a given level cannot be processed through one simple scan of Δ. However, we emphasize that it is possible to use indexing techniques so as to avoid scanning Δ for the test on $t.XY$. In this case, counting the supports requires a number of tests in $O(|\Delta|)$, as in standard level-wise algorithms.

Once the supports of all classes in all generic classes of C are computed (*i.e.,* once Δ has been scanned), all classes $\langle X, y\rangle$ whose support is less than the support threshold *min-sup* are removed from $L(\langle X, Y\rangle)$ and the remaining classes are added to the set L_{Freq}. Moreover, all generic classes $\langle X, Y\rangle$ such that $L(\langle X, Y\rangle) \neq \emptyset$ are inserted into L.

Complexity issues. As usual for level-wise algorithms, the complexity of our algorithms is expressed in terms of the number of scans of Δ, because, as previously argued, the supports of all classes at a given level can be computed through one scan of Δ.

Therefore, the complexity of our algorithms is given by the maximum number of levels that have to be processed, and this number is equal to the length S of one of the longuest sequences of classes $(q_1, \ldots, q_S)$ such that q_1 are q_S are respectively the less and the most specific classes in $\mathcal{C}(\Delta)$, and $q_{i+1} \in succ(q_i)$ for $i = 1, \ldots, S-1$.

We argue that S is in $O(|U|)$. Indeed, Proposition 5 shows that for all classes $q = \langle X, y\rangle$ and $q' = \langle X', y'\rangle$ such that $q' \in succ(q)$ and $q' \neq \perp$, we have either $|X' \setminus Y'| = |X \setminus Y|$ (due to Proposition 5(4) where $X' = XA$ and $Y' = YA$), or $|X' \setminus Y'| = |X \setminus Y| - 1$ (due to Proposition 5(2) where $X' = X \setminus A$ and $Y' = Y$, and to Proposition 5(3) where $X' = X$ and $Y' = YA$).

Moreover, for $i = 1, \ldots, S$, we denote by d_i the cardinality of $(X_i \setminus Y_i)$, where X_i and Y_i are such that $q_i = \langle X_i, y_i\rangle$. Since $\langle U, \top\rangle$ and $\perp$ are respectively the less specific and the most specific classes in $\mathcal{C}(\Delta)$, we have $d_1 = |U|$ and $d_S = 0$ (as all queries $\langle X, x\rangle$ where $X \in cl(FD)$ are in $\perp$).

Thus, $(d_1, \ldots, d_S)$ is a decreasing sequence of integers bounded by $|U|$ and 0, and in which at most $|U|$ terms are equal (because Proposition 5(4) cannot be applied twice with the same attribute A in the sequence). Therefore, we have $S \leq 2.|U| + 1$. On the other hand, let $(q_1, \ldots, q_s)$ be the sequence built up as follows, starting with $q_1 = \langle U, \top\rangle$:

1. Reach a class of the form $\langle A, \top\rangle$ where $A \in (D_{i_0} \setminus K_{i_0})$ $(i_0 \in \{1, \ldots, N\})$ by removing from X_i one by one all key attributes in K and then, all remaining attributes but A. This step produces $|U| - 1$ classes such that $q_{i+1} \in succ(q_i)$ (by Proposition 5(2)), thus giving a sequence of length $|U|$.
2. Then, expand the sequence by adding one by one to X_i and Y_i all attributes different than A and K_{i_0} (considering first non key attributes, and then key attributes). This step produces $|U| - 2$ classes such that $q_{i+1} \in succ(q_i)$ (by Proposition 5(4)) and $q_{2.(|U|-1)} = \langle U \setminus K_{i_0}, y\rangle$ where y is over $U \setminus (K_{i_0} A)$. As $succ(q_{2.(|U|-1)}) = \{\perp\}$ (by Proposition 5(1)), $s = 2.|U| - 1$.

Therefore, $2.|U| - 1 \leq S \leq 2.|U| + 1$, and thus, *the complexity in the number of scans of our algorithms is in $O(|U|)$.*

```
Algorithm scan
Input: The set $C$ of candidate generic classes.
Output: The set $L$ of frequent generic classes in $C$ and the set $L_{Freq}$ of all associated frequent classes.
Method:
$L = \emptyset$ ; $L_{Freq} = \emptyset$
for each $\langle X, Y\rangle \in C$ do
    $L(\langle X, Y\rangle) = \emptyset$
end for each
for each tuple $t \in \Delta$ do
    for each $\langle X, Y\rangle \in C$ do
        if $\langle X, t.Y\rangle \notin L(\langle X, Y\rangle)$ then
            //Task $(i)$
            $L(\langle X, Y\rangle) = L(\langle X, Y\rangle) \cup \{\langle X, t.Y\rangle\}$
            $sup(\langle X, t.Y\rangle) = 1$
        else
            //Task $(ii)$
            if $t.X \notin pred(t)$ then
                $sup(\langle X, t.Y\rangle) = sup(\langle X, t.Y\rangle) + 1$
            end if
        end if
    end for each
end for each
for each $\langle X, Y\rangle \in C$ do
    $L(\langle X, Y\rangle) = L(\langle X, Y\rangle) \setminus$
                    $\{\langle X, y\rangle \mid sup(\langle X, y\rangle) < min\text{-}sup\}$
    if $L(\langle X, Y\rangle) \neq \emptyset$ then
        $L_{Freq} = L_{Freq} \cup L(\langle X, Y\rangle)$
        $L = L \cup \{\langle X, Y\rangle\}$
    end if
end for each
return $L$ and $L_{Freq}$
```

Figure 5: The algorithm for scanning Δ

6. CONCLUSION AND FURTHER WORK

In this paper, we have considered the problem of mining all projection-selection queries from a given relational table satisfying a given set of functional dependencies. In this setting, we defined a pre-ordering with respect to which the support measure is anti-monotonic, and showed that this pre-ordering allows to consider equivalence classes of queries instead of individual queries.

In the case where the functional dependencies are those considered in a star schema, we have provided algorithms for computing all frequent queries, and we have shown that the complexity in the number of scans of these algorithms is linear with respect to the size of the attribute set.

We are currently implementing these algorithms. It is important to note that, regarding efficiency issues, the following points are to be carefully considered: First, as mentioned in the previous section, the option of pruning or not the set of candidate equivalence classes in $\mathcal{C}(\Delta)$ has to be studied. Second, efficient data structures are necessary for storing the sets C and L of candidate and frequent generic classes as well as the sets $L(\langle X, Y\rangle)$. Hashing techniques are expected to work efficiently in this case.

Regarding possible extensions of the present approach, we plan to investigate the following issues: First, the case of a database consisting of several tables, instead of a single table, will be considered as done in [11], *i.e.,* under the weak instance semantics that allows to represent a database as a single table ([14]). Second, extending the selection conditions to equalities of the form $Y = Y'$ where Y and Y' are two relation schemas, as done in [8], is another point that we plan to investigate in the context of the present work.

7. REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 309–328. AAAI-MIT Press, 1996.

[2] W. Armstrong. Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583. North Holland, 1974.

[3] L. Dehaspe and L. D. Raedt. Mining association rules in multiple relations. In *7th International Workshop on Inductive Logic Programming*, *LNCS* 1297, pages 125–132. Springer Verlag, 1997.

[4] C. T. Diop, A. Giacometti, D. Laurent, and N. Spyratos. Composition of mining contexts for efficient extraction of association rules. In *EDBT'02*, *LNCS* 2287, pages 106–123. Springer Verlag, 2002.

[5] A. Faye, A. Giacometti, D. Laurent, and N. Spyratos. Mining rules in databases with multiple tables: Problems and perspectives. In *3rd International Conference on Computing Anticipatory Systems (CASYS)*, 1999.

[6] B. Goethals. Mining queries. In *Workshop on inductive databases and constraint based mining. http://www.informatik.uni-freiburg.de/~ml/IDB/talks/Goethals_slides.pdf*, 2004.

[7] B. Goethals and J. V. den Bussche. Relational association rules: getting warmer. In *ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining*, *LNCS* 2447, pages 125–139. Springer-Verlag, 2002.

[8] B. Goethals, E. Hoekx, and J. V. den Bussche. Mining tree queries in a graph. In *11th ACM SIGKDD-KDD*, pages 61–69, 2005.

[9] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. Dmql : A data mining query language for relational databases. In *SIGMOD-DMKD'96*, pages 27–34, 1996.

[10] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8:53–87, 2004.

[11] T. Y. Jen, D. Laurent, N. Spyratos, and O. Sy. Towards mining frequent queries in star schemes. In *International Workshop on Knowledge Discovery in Databases (KDID)*, *LNCS* 3933, pages 104–123. Springer Verlag, 2005.

[12] R. Meo, G. Psaila, and S. Ceri. An extension to sql for mining association rules. *Data Mining and Knowledge Discovery*, 9:275–300, 1997.

[13] T. Turmeaux, A. Salleb, C. Vrain, and D. Cassard. Learning caracteristic rules relying on quantified paths. In *PKDD*, *LNCS* 2838, pages 471–482. Springer Verlag, 2003.

[14] J. Ullman. *Principles of Databases and Knowledge-Base Systems.* Comp. Science Press, 1988.

APPENDIX

A. PROOF OF PROPOSITION 4

Regarding the proof of item 1 in the proposition, we first note that, as $\bot$ is the most specific class, if $\bot \in succ(\langle X, y\rangle)$, then $succ(\langle X, y\rangle) = \{\bot\}$.

Assume first that, for every $Y' \in Y^{\uparrow}$, $X \subseteq Y'$ and that there exists $\langle X'', y''\rangle$ such that $\langle X, y\rangle \prec \langle X'', y''\rangle \prec \bot$. As comparisons are strict, $Y \rightarrow X$ and $Y'' \rightarrow X''$ are not in FD^{+}. Thus, $Y \subset X$, $Y'' \subset X''$, $X'' \subseteq (XY'')^{+}$, $Y \subseteq Y''$, $y''.Y = y$, and one of the last two inclusions is strict.
– If $Y = Y''$ then $X'' \subset (XY'')^{+} = X$. As in this case, $Y \subset X''$, there exists $Y' \in Y^{\uparrow}$ such that $Y \subset Y' \subseteq X''$. Thus, $X \subseteq Y' \subseteq X''$, a contradiction with $X'' \subset X$.
– If $Y \neq Y''$, then there exists $Y' \in Y^{\uparrow}$ such that $Y \subset Y' \subseteq Y''$. Thus, $X \subseteq Y' \subset Y''$, and so, $X'' \subseteq (XY'')^{+} = Y''$, which is not possible. Therefore, we have shown that, if for every $Y' \in Y^{\uparrow}$, $X \subseteq Y'$, then $succ(\langle X, y\rangle) = \{\bot\}$.

Conversely, let $\langle X, y\rangle$ be such that $succ(\langle X, y\rangle) = \{\bot\}$ and assume that there exists Y' in $Y^{\uparrow}$ such that $X \not\subseteq Y'$. Thus, $X \neq Y'$ and:
– If $Y' \subset X$ then, for any y' in $\pi_{Y'}(\Delta)$ such that $y'.Y = y$, we have $\langle X, y\rangle \prec \langle X, y'\rangle \prec \bot$, and so, $\bot \notin succ(\langle X, y\rangle)$.
– If $Y' \not\subset X$ then $Y' \subset XY' \subseteq (XY')^{+}$. In this case, with a tuple y' defined as above, we have $\langle X, y\rangle \prec \langle (XY')^{+}, y'\rangle \prec \bot$, and so, $\bot \notin succ(\langle X, y\rangle)$.
Consequently, item 1 in the proposition holds.

In order to prove that the other items hold, assuming that $succ(\langle X, y\rangle) \neq \{\bot\}$, we denote by $\Sigma(\langle X, y\rangle)$ the set of all classes defined by the last three items in the proposition, and we show that $\Sigma(\langle X, y\rangle) = succ(\langle X, y\rangle)$.

Assuming that there exists a class $\langle X'', y''\rangle$ such that $\langle X, y\rangle \prec \langle X'', y''\rangle \prec \langle X', y'\rangle$, we successively consider the last three items in the proposition. In this case, we have $X'' \subseteq (XY'')^{+}$, $X' \subseteq (X''Y')^{+}$, $Y \subseteq Y''$ and $Y'' \subseteq Y'$.

2. If $Y = Y'$, then $Y = Y' = Y''$. Thus, $X'' \subset X$ and that $X \setminus Z \subset X''$, a contradiction with the definition of Z.

3. If $X = X'$ and $Y' = YZ$, then $X'' \subseteq (XY'')^{+}$, $X \subseteq (X''YZ)^{+}$ and $Y \subseteq Y'' \subseteq YZ$. Thus, $X \subseteq (XY''YZ)^{+} = (XY'')^{+}$. Moreover, as $Y'' \in cl(FD)$, by item 3(b), we have either $Y'' = Y$ or $Y'' = Y' = YZ$.

If $Y'' = Y$ then $X'' \subset X \subseteq (X''YZ)^{+} = (X''Z)^{+}$. Thus, $Y'' \subseteq X'' \subset X \subseteq (X''Z)^{+}$, a contradiction with item 3(c).

If $Y'' = Y' = YZ$ then $X'' \subseteq (XZ)^{+} = X$ and $X \subset (X''YZ)^{+} = (X''Y'')^{+} = X''$. Thus, $X \subset X'' \subseteq X$, which is impossible.

4. If $X' = (XZ)^+$ and $Y' = YZ$, then $Y \subseteq Y'' \subseteq YZ$ and thus, by item 4(b), either $Y'' = Y$ or $Y'' = Y' = YZ$.

If $Y'' = Y$ then $X'' \subset (XY)^+ = X$ and $(XZ)^+ \subseteq (X''YZ)^+ = (X''Z)^+$. Thus, $Y \subseteq X'' \subset X$ and $(XZ)^+ \subseteq (XZ)^+$. As $X'' \subset X$, we have $(X''Z)^+ \subseteq (XZ)^+$. Hence, $(X''Z)^+ = (XZ)^+$, which is a contradiction with item 4(c).

If $Y'' = Y' = YZ$ then $X'' \subseteq (XZ)^+$, $YZ \subseteq X''$ and $(XZ)^+ \subset (X''YZ)^+ = X''$. Hence, $X'' \subseteq (XZ)^+ \subset X''$, which is impossible.

Therefore, we have shown that $\Sigma(\langle X, y\rangle) \subseteq succ(\langle X, y\rangle)$. We now proceed to the proof of the converse inclusion, *i.e.,* $succ(\langle X, y\rangle) \subseteq \Sigma(\langle X, y\rangle)$.

Let $\langle X', y'\rangle \in succ(\langle X, y\rangle)$, then we have $\langle X, y\rangle \prec \langle X', y'\rangle$ and there does not exist $\langle X'', y''\rangle$ in $\mathcal{C}(\Delta)$ such that $\langle X, y\rangle \prec \langle X'', y''\rangle \prec \langle X', y'\rangle$. Thus, $X' \subseteq (XY')^+$, $Y \subseteq Y'$, $y = \Delta_Y(y')$ and at least one of the two inclusions is strict.

If $y = y'$, and so, if $Y = Y'$, then $X' \subset (XY')^+ = X$. Therefore X' can be written as $X \setminus Z$, where $Y \subseteq X \setminus Z$ and $(X \setminus Z) \in cl(FD)$. Thus, 2(a) in the definition of $\Sigma(\langle X, y\rangle)$ is statisfied. Let us assume that 2(b) is not satisfied, *i.e.,* that there exists X_0 in $cl(FD)$ such that $(X \setminus Z) \subset X_0 \subset X$. Then, we have $\langle X, y\rangle \prec \langle X_0, y\rangle$ and $\langle X_0, y\rangle \prec \langle X', y\rangle$, which is a contradiction with the fact that $\langle X, y\rangle$ is in $succ(\langle X, y\rangle)$. Thus, in this case $\langle X', y'\rangle$ satisfies item 2 of the definition of $\Sigma(\langle X, y\rangle)$, which shows that $\langle X', y'\rangle \in \Sigma(\langle X, y\rangle)$.

If $y \neq y'$, then $Y \subset Y'$, and thus, $y' = yz$ where Z is such that $Y' = YZ$ and $YZ \in cl(FD)$. We consider the two cases according to which YZ is or not a subset of X.

− Let us assume that $YZ \subseteq X$. Notice that if $X = YZ$ then, as $\langle X, y\rangle \prec \langle X', y'\rangle$, $X' \subseteq (XY')^+$ can be written as $X' \subseteq YZ$, meaning that $X' \subseteq Y'$. This is not possible since we assume that $\langle X', y'\rangle \neq \perp$. Thus, $X \neq YZ$.

If there exists Z' such that $YZ' \in cl(FD)$ and $Y \subset YZ' \subset YZ$, then we have $\langle X, y\rangle \prec \langle X, yz'\rangle \prec \langle X', yz\rangle$, where $yz' = yz.(YZ')$. Indeed, on the one hand, $X \subseteq (XYZ')^+$, $Y \subset YZ'$ and $y = \Delta_Y(yz')$, and on the other hand, $X' \subseteq (XYZ)^+$, $YZ' \subset YZ$ and $yz' = \Delta_{YZ'}(yz)$. This case is impossible since $\langle X', y'\rangle \in succ(\langle X, y\rangle)$, and therefore item 3(b) in the definition of $\Sigma(\langle X, y\rangle)$ is satisfied.

If there exists X_0 in $cl(FD)\setminus\{\emptyset\}$ such that $Y \subseteq X_0 \subset X \subseteq (X_0Z)^+$, then we have $\langle X, y\rangle \prec \langle X_0, y\rangle \prec \langle X', yz\rangle$. Indeed, $X_0 \subset (XY)^+ = X$, $X' \subseteq (X_0YZ)^+$ (because $X' \subseteq (XYZ)^+$ and $YZ \subseteq X$ imply that $X' \subseteq X$, and $X \subseteq (X_0Z)^+ \subseteq (X_0YZ)^+$), $YZ \subset Y$ and $y = \Delta_Y(yz)$. Again, this case is impossible since $\langle X', y'\rangle$ is assumed to be in $succ(\langle X, y\rangle)$, and therefore item 3(c) in the definition of $\Sigma(\langle X, y\rangle)$ is satisfied. Thus, item 3 in the definition of $\Sigma(\langle X, y\rangle)$ is satisfied, which shows that $\langle X', y'\rangle \in \Sigma(\langle X, y\rangle)$.

− Let us assume that $YZ \not\subseteq X$. If there exists Z' such that $YZ' \in cl(FD)$ and $Y \subset YZ' \subset YZ$, then we have $\langle X, y\rangle \prec \langle (XZ')^+, yz'\rangle \prec \langle X', yz\rangle$, where $yz' = yz.(YZ')$. Indeed, we have on the one hand $(XZ')^+ \subseteq (XYZ')^+$ (because, as $Y \subseteq X$, $XZ' = XYZ'$), $Y \subset YZ'$ and $y = \Delta_Y(yz')$, and on the other hand, $X' \subseteq (XYZZ')^+$ (because, as $YZ' \subseteq YZ$ and $Y \subseteq X$, $XYZZ' = XZ$ and $X' \subseteq (XY')^+ = (XYZ)^+ = (XZ)^+$), $YZ' \subset YZ$ and $yz' = \Delta_{YZ'}(yz)$. This case is impossible since $\langle X', y'\rangle$ is assumed to be in $succ(\langle X, y\rangle)$, and therefore item 4(b) in the definition of $\Sigma(\langle X, y\rangle)$ is satisfied.

If there exists X_0 in $cl(FD)$ such that $Y \subseteq X_0 \subset X$ and $(XZ)^+ = (X_0Z)^+$, then $\langle X, y\rangle \prec \langle X_0, y\rangle \prec \langle X', yz\rangle$. Indeed, we have on the one hand $X_0 \subset (XY)^+$, and on the other hand $X' \subseteq (X_0YZ)^+$ (because $X' \subseteq (XY')^+$, $Y' = YZ$, $Y \subseteq X$, $(X_0Z)^+ = (XZ)^+$ and $Y \subseteq X_0$ imply that $X' \subseteq (X_0Z)^+ = (X_0YZ)^+$), $Y \subset YZ$ and $y = \Delta_Y(yz)$. This case is impossible since $\langle X', y'\rangle$ is assumed to be in $succ(\langle X, y\rangle)$, and therefore item 4(c) in the definition of $\Sigma(\langle X, y\rangle)$ is satisfied. Thus, item 4 in the definition of $\Sigma(\langle X, y\rangle)$ is satisfied, which shows that $\langle X', y'\rangle \in \Sigma(\langle X, y\rangle)$. Hence, $succ(\langle X, y\rangle) \subseteq \Sigma(\langle X, y\rangle)$, and the proof is complete.

B. PROOF OF PROPOSITION 5

The items in the proposition respectively correspond to those of Proposition 4. We consider each of them below.

1. As every $Y' \in Y^\uparrow$ is of the form YA where $A \notin Y$, and as $Y \subset X$, $X \subseteq YA$ is equivalent to $X = YA$. The result follows from item 1 in Proposition 4.

2. It is easy to see that item 2 in Proposition 4 corresponds to that in Proposition 5.

3. It is easy to see that items 3(a) and 3(b) in Proposition 4 correspond to $A \in (X\setminus Y)$ and $YA \in Y^\uparrow \setminus \{X\}$, respectively. Thus, we have to show that item 3(c) in Proposition 4 is equivalent to $(X \setminus A) \notin cl(FD) \setminus \{\emptyset\}$.
− Let $A \in (X\setminus Y)$ be such that $(X\setminus A) \in cl(FD)\setminus\{\emptyset\}$. Then $Y \subseteq (X \setminus A) \subset X \subseteq ((X \setminus A)A)^+$, which shows that, with $X_0 = (X \setminus A)$, item 3(c) in Proposition 4 is not satisfied.
− Conversely, if item 3(c) in Proposition 4 is not satisfied, then let X_0 be in $cl(FD) \setminus \{\emptyset\}$ such that $Y \subseteq X_0 \subset X \subseteq (X_0A)^+$. In this case, $A \notin X_0$ because, otherwise, we would have $X_0 \subset X \subseteq X_0$.

If $(X_0A)^+ = X_0A$, then we have $X_0 = (X \setminus A)$, showing that $(X \setminus A) \in cl(FD) \setminus \{\emptyset\}$.

If $(X_0A)^+ \neq X_0A$, then as $X_0 \in cl(FD) \setminus \{\emptyset\}$, we have that A is in K. Let $A = K_i$ ($i \in \{1, \ldots, N\}$), since $K_i \in X$ and $X \in cl(FD)$, $D_i \subseteq X$. Thus, $(X \setminus K_i) \in cl(FD) \setminus \{\emptyset\}$, which shows that $(X \setminus A) \in cl(FD) \setminus \{\emptyset\}$.

4. It is easy to see that items 4(a) and 4(b) in Proposition 4 correspond to $A \notin X$ and $YA \in Y^\uparrow$, respectively. Thus, we have to show that item 4(c) in Proposition 4, (a) either is equivalent to the fact that $(M\cap(X\setminus Y)) = \emptyset$ and $(K\setminus A) \subseteq X$ if $A \in K$, or (b) holds if $A \notin K$.

(a) If $A \in K$, let $i \in \{1, \ldots, N\}$ such that $A = K_i$. Moreover
− Let us assume that $(M \cap (X \setminus Y)) \neq \emptyset$ and $(K \setminus K_i) \subseteq X$. Then, for $X_0 = (X \setminus (M \cap (X \setminus Y)))$, we have $Y \subseteq X_0 \subset X$ and $(K \setminus K_i) \subseteq X_0$. Therefore, $(XK_i)^+ = (X_0K_i)^+ = U$, which shows that item 4(c) in Proposition 4 is not satisfied.
− Conversely, if item 4(c) in Proposition 4 is not satisfied, then let X_0 be in $cl(FD)$ such that $Y \subseteq X_0 \subset X$ and $(XK_i)^+ = (X_0K_i)^+$. Since $YK_i \in Y^\uparrow$, we have $(D_i \setminus K_i) \subseteq Y$ and thus, $(D_i \setminus K_i) \subseteq X$ and $(D_i \setminus K_i) \subseteq X_0$.

If $(K \setminus K_i) \not\subseteq X$, then we also have $(K \setminus K_i) \not\subseteq X_0$. Thus, $(XK_i)^+ = XK_i$ and $(X_0K_i)^+ = X_0K_i$, which is a contradiction with $(XK_i)^+ = (X_0K_i)^+$.

Hence, $(K \setminus K_i) \subseteq X$, which entails that $(XK_i)^+ = U$, and thus that $(XK_i)^+ = (X_0K_i)^+ = XK_iM$. Moreover, as X and X_0 are in $cl(FD)$, for every $j \neq i$, $D_j \subseteq X$ and $D_j \subseteq X_0$. Thus, $(X \setminus X_0) \subseteq M$, and as $Y \subseteq X_0 \subset X$, we have $(X \setminus X_0) = ((X \setminus Y) \setminus X_0)$. Thus, $M \cap (X \setminus Y) \neq \emptyset$.

(b) If $A \notin K$ and $A \notin X$, then $(XA)^+ = XA$. Thus, if $Y \subseteq X_0 \subset X$, then $A \notin X_0$ and so, $(X_0A)^+ = X_0A$. Therefore, $(XA)^+ \subset (X_0A)^+$ showing that there exists no X_0 such that $X_0 \subset X$ and $(XA)^+ = (X_0A)^+$. Thus, item 4(c) in Proposition 4 holds, and the proof is complete.