CoDS: A Representative Sampling Method for Relational Databases

Teodora Sandra Buda[1], Thomas Cerqueus[1], John Murphy[1], and Morten Kristiansen[2]

[1] Lero, Performance Engineering Lab
School of Computer Science and Informatics, University College Dublin.
teodora.buda@ucdconnect.ie, thomas.cerqueus@ucd.ie, j.murphy@ucd.ie
[2] IBM Collaboration Solutions, IBM Software Group, Dublin, Ireland.
morten_kristiansen@ie.ibm.com

Abstract. Database sampling has become a popular approach to handle large amounts of data in a wide range of application areas such as data mining or approximate query evaluation. Using database samples is a potential solution when using the entire database is not cost-effective, and a balance between the accuracy of the results and the computational cost of the process applied on the large data set is preferred. Existing sampling approaches are either limited to specific application areas, to single table databases, or to random sampling. In this paper, we propose CoDS: a novel sampling approach targeting relational databases that ensures that the sample database follows the same distribution for specific fields as the original database. In particular it aims to maintain the distribution between tables. We evaluate the performance of our algorithm by measuring the representativeness of the sample with respect to the original database. We compare our approach with two existing solutions, and we show that our method performs faster and produces better results in terms of representativeness.

Keywords: Relational database, Representative database sampling.

1 Introduction

Nowadays applications are generally faced with the challenge of handling large number of users that produce very large amounts of data up to terabytes in size. The storage space, administration overhead of managing large datasets and the analysis of this data is a real challenge in different fields. For instance, in data mining, a balance between the accuracy of the results and the computational cost of the analysis is generally preferred to overcome this challenge. Moreover, in software validation, the operational data available for a system under development could serve as a realistic testing environment. However these databases consist of large amount of data, which is computationally costly to analyze.

Database sampling is a potential solution to this problem: a smaller database can be used instead of the original one. Olken's major contribution to random

sampling from large databases proves sampling to be a powerful technique [14]. Database sampling methods aim to provide databases that (i) are smaller in size, (ii) are consistent with the original database (e.g. conformance of the schema), (iii) contain data from the original database, (iv) are representative of the original database. The last criteria is crucial because the accuracy of the results of the following analysis to be performed on the sample is expected to be significantly higher if the sample is representative of the original database. For instance, a representative sample of the production environment would determine the sample contain realistic test data, encompassing a variety of scenarios the user created. In particular, in functional testing, a small realistic sample of the production environment would suffice to test the core functionality of the system under development, while maintaining the accuracy of the results. The problem raised in this work is to define a method that produces a representative database sample targeting relational databases.

Existing databases sampling methods involve random sampling [6], target single-table databases [11], or they are specific to an application area [10, 4]. For instance, in [13], the reader is presented a representative sampling approach that aims to handle scalability issues of processing large graphs. However, most of today's structured data is stored in relational databases, consisting of multiple tables linked through various constraints. Single table sampling methods applied on relational databases produce an inconsistent sample database with regards to the referential integrity. Moreover, we expect that random samples provide poor accuracy in the results of the analysis to be performed on the large data set (e.g. testing purposes, data mining methods, approximate query evaluation). For instance a random sample of the production environment could sample only one test case and not detect high priority errors of the system.

In this paper, we propose the CoDS system: a novel approach for database sampling, targeting relational databases, with the purpose of creating a smaller representative sample, that respects the referential integrity constraints. We consider that a sample is representative if it follows the same distribution for specific fields. The fields considered by CoDS are the foreign key constraints. A foreign key constraint in a database is used to create and enforce a link between the data in two tables. Thus, these constraints represent invaluable inputs for our system to depict the relationships between data and produce a representative sample. If the sample database follows the same distribution as the original database for these fields, it is feasible to expect that the results of the following analysis to be performed on the sample will produce the same results as the ones performed on the original database. The sampling mechanism proposed is independent of the application area and will result in producing a consistent representative sample.

The remainder of this paper is organized as follows: Section 2 discusses related work and describes various application areas in which representative sample database may be of interest. Section 3 describes the main contribution of this work: the representative database sampling system. Sections 4 describe the experimental evaluation of CoDS, and its comparison with previous approaches. Section 5 concludes the paper.

2 Related Work

Several database sampling methods have been proposed in specific application areas proving sampling to be a useful and powerful technique. However, most of them are designed for specific application areas: software testing, data mining, query approximation. Before presenting methods built for these different areas, we present general approaches.

General approaches The database sampling approach presented in [3] is oriented towards relational databases focusing on the advantage of using prototype databases populated with operational data. Data items that follow a set of integrity constraints (e.g. foreign-key constraints, functional dependencies, domain constraints) are randomly selected from the original database, so that the resulting sample database is consistent with the original database. Furthermore, there are a few commercial applications that support sampling from databases. For instance, IBM *Optim*[1] is used for managing data within many database instances. Its component, *Move*, can be used for sampling by using the option to select every n^{th} row of each table from the original database. *Optim* ensures that the referential integrity is respected by the sample database. As a recognized value of database sampling, Oracle DBMS supports the possibility to query a sample of a given table instead of the whole table by using the *Sample* statement[2].

Software testing Analyzing the production environment, its constraints, and generating relevant testing data are just a few of the challenges encountered during the testing process. Existing methods for populating the testing environment commonly generate synthetic data values or use some type of random distribution to select the data that must be included in the resulting database [18, 15]. In [20] the reader is presented with a privacy-preserving approach that uses the operational data available for testing purposes focusing on the importance of the representativeness of the data as it can increase the probability of detecting crucial faults of the system. Moreover, the testing environment would encompass scenarios created by the user, useful for testing the core functionality of the system. However, the production environment generally consists of large amounts of data. Database sampling is a potential solution to overcome this challenge.

Data mining Various sampling approaches have been proposed in the data mining community, proving that sampling is a powerful technique for achieving a balance between the computational cost of performing data mining on a very large population and the accuracy of the results [17, 12]. However, the approaches devised in this community are generally oriented towards the data mining algorithm used on the sample [4, 19, 16] and most of the standard methods for data mining are built on the assumption that data is stored in single-table databases. In [11], the authors propose a static sampling approach which uses the distribution of the sample data as an evaluation criterion to decide whether the

[1] http://www-01.ibm.com/software/data/data-management/optim-solutions/
[2] http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_10002.htm

sample reflects the large dataset. However, it is limited to single-table datasets and to univariate analysis. Some recent work in the data mining community [8, 21] avoided this shortcoming and target relational databases. In [21], authors present a sampling algorithm for relational databases that focuses on improving the scalability and accuracy of multi-relational classification methods.

Approximate Query Evaluation Numerous papers proposed random sampling for approximate query answering [2, 9], and statistics estimation for query size result [5], allowing approximate but faster answers to queries. A more recent approach that extended the table-level sampling to relational database sampling is presented in [7]. The authors propose a sampling mechanism called *Linked Bernoulli Synopses* based on *Join Synopses* [1] aiming to provide fast approximate query answers for join queries over multiple tables. Their solution imply maintaining the foreign key integrity of the synopses. Both approaches are probabilistic and require the processing of each tuple in a database. In the case of JS, each tuple from the set of tables is sampled with a probability equal to the sampling rate. After this insertion of tuples in the sample database, JS ensures the referential integrity of the sample database remains intact by visiting all the tables, starting with the root, and adding the missing referenced tuples in the sample database. LBS is run only one time over the entire database. The decision of whether or not to include the row in the sample is different in LBS. LBS requires the retrieval of every tuple from each table and calculates the probability of a tuple t, being inserted in the sample database based on the probabilities of the tuples referencing tuple t to be inserted in the sample. The computation of this probability is described in detail in [7]. In the case that one of referencing tuples has already been included in the sample, the tuple under analysis is also included in the sample, thus avoiding the referential integrity to be broken.

3 CoDS: a Representative Sampling System

The CoDS[3] system proposes a method to produce a representative α-sample of an original database, where α represents the sampling rate for the original database and is given as an input by the user of the system. The objective is to maintain the distribution between the tables of a database to ensure the representativeness property, while maintaining the referential integrity of the data. The system targets relational databases, in third normal form. We assume that the schema of the original database forms a connected graph. CoDS aims to analyze and preserve the distributions between a starting table and the rest of the tables of the database, through various joins when needed. CoDs computes a set of identifiers that need to be sampled from the starting table to preserve these distributions, along with a representative error measure when a perfect representative sample could not be created. CoDS is composed of four phases:

– The system identifies the starting table (section 3.2).

[3] Chains of Dependencies-based Sampling

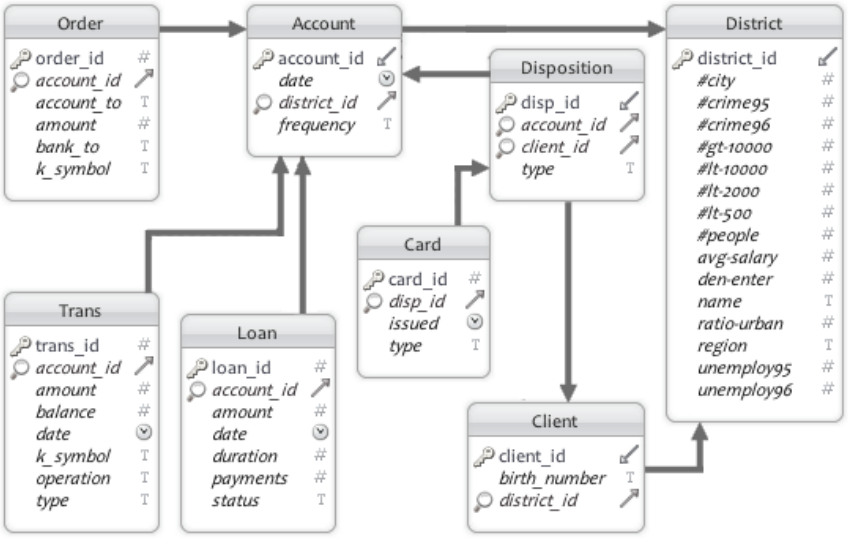

Fig. 1. The *Financial* database schema.

- The system detects the relationships of the starting table with the rest of the tables of the database by following the foreign key constraints from the metadata. Then it generates the scatter plots associated to these relationships (section 3.3).
- The system analyzes the generated scatter plots between the starting table and the rest of the tables in order to compute a set of identifiers of the starting table that need to be sampled to preserve the distribution of these relations (section 3.4).
- Finally, the system proceeds in sampling the tuples associated with the set of identifiers of the starting table computed in the previous step, and to sampling all the related tuples from the rest of the tables (section 3.5).

Before presenting each phase in detail, we introduce the formal model and definitions used in the remaining of the document.

3.1 Model and definitions

Relational database A relational database is a set of n tables $T = \{t_1, ..., t_n\}$. Each table t_i of the database is composed of a set of attributes $A_i = \{a_1, ..., a_m\}$. The set of attributes that allow to uniquely identify a tuple in table t_i is called the primary key noted PK_i. A foreign key is a set of attributes that refers to another table's attributes. For instance, in Fig. 1, an example of such foreign key is *client_id*, declared in table *Disposition*. A table may contains several foreign keys. We denote by FK_i^j the set of attributes of table t_i that reference table t_j. When a table t_i defines a foreign key constraint to another table t_j, we say that t_i directly

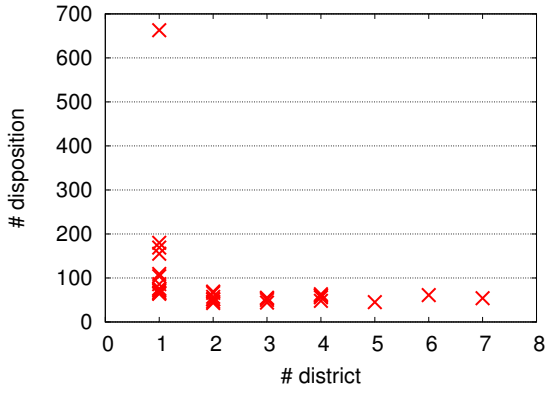

Fig. 2. Scatter plot associated to chain $\langle District, Client, Disposition\rangle$

references t_j and we denote it by: $t_i \rightarrow t_j$. Symmetrically, t_j is directly referenced by t_i, and denote it by $t_j \leftarrow t_i$.We refer by children of table t by $children(t)$ to the set of tables that t references: $children(t) = \{t_i \in T : t \rightarrow t_i\}$. We refer by $desc(t)$ to the set of all the descendants of t:

$$desc(t) = children(t) \cup \bigcup_{t_i \in children(t)} desc(t_i)$$

Similarly, we refer to parents of t by: $parents(t) = \{t_i \in T : t \leftarrow t_i\}$. We define the set of related tables of t_i as follows: $RT(t) = parents(t) \cup children(t)$. We denote by $O(t)$ and $S(t)$ the tuples of t in the original and the sample database.

Chain of dependencies A chain of dependencies is a sequence of tables $\langle t_1, \ldots, t_k\rangle$ such that $\forall i \in [1, k-1]$, $t_i \in RT(t_{i+1}) \ \wedge \ \forall i, j \in [1, k], i \neq j \Rightarrow t_i \neq t_j$. An example of a chain of dependencies in Fig. 1 is $\langle District, Client, Disposition\rangle$. The set of all chains of dependencies of t (i.e. t_1=t) is denoted by $\mathcal{C}h(t)$.

Scatter plot and data point Given a chain $\langle t_1, \ldots, t_k\rangle$ we consider the scatter plot associated to this chain between table t_1 and table t_k. We denote by $\mathcal{S}p(t)$ the set of all scatter plots associated to $\mathcal{C}h(t)$. A scatter plot is composed of a set of points corresponding to that plot. Each point of a scatter plot is called a *data point*. A data point situated at the coordinate (x, y) means that: (i) if $t_1 \leftarrow t_2$: x tuples of table t_1 are indirectly referenced by y tuples of table t_k, (ii) if $t_1 \rightarrow t_2$: x tuples of table t_k are indirectly referenced by y tuples of table t_1. For instance, for the scatter plot of chain $\langle District, Client, Disposition\rangle$ presented in figure 2, we can see that only one district is indirectly referenced by 663 dispositions, or that 7 districts are indirectly referenced by 54 dispositions. Each data point is uniquely identified by its y value, and contains identifiers of table t_1 (i.e. contains a set of values of PK_1) from the original database. For instance, in Fig. 2, the data point with $y = 663$ contains the identifier of the single district that is indirectly referenced by 663 dispositions.

3.2 Starting table selection

The objective of this phase is to select a starting table for the sampling, which we denote by $t_\star$. In CoDS, a leaf table (i.e. a table that has no children) is chosen as starting table. If the database has more than one leaf tables the system chooses the one with the maximum number of tuples. The reason for this is to avoid choosing a leaf with few tuples, as this would critically impact the sampling method by having very little influence on the tuples selected from the related tables, and thus on the representativeness of the sample database. CoDS selects a leaf table as a starting table in order to reduce the computational cost of analyzing the chains generated by using a bottom-up approach. We show in section 3.3 that the computational cost of analyzing a chain using a bottom-up approach is lower in contrast with a top-down approach. Moreover, we expect that using a leaf table produces less errors related to the sample size and representativeness.

3.3 Generation of chains

In the second phase, we aim to discover the relationships between the starting table and the rest of the tables of the database, generate the set of chains of dependencies of the starting table and construct their associated scatter plots. These scatter plots will be used for the selection of identifiers of the starting table. The system generates $\mathcal{C}h(t_\star)$ by following all the possible paths through the arrows between the tables, starting with $t_\star$. Note that each table is visited only once in this representation and the shortest path is preferred. If two chains with equal t_k have the same length but are composed of different tables, both chains are considered. Let us consider the relational database presented in Fig. 1. CoDS generates the following chains of dependencies for $t_\star = District$: $\mathcal{C}h(District) = \{\langle District, Client\rangle, \langle District, Client, Disposition\rangle, \langle District, Client, Disposition, Card\rangle, \langle District, Account\rangle, \langle District, Account, Disposition\rangle, \langle District, Account, Order\rangle, \langle District, Account, Trans\rangle, \langle District, Account, Loan\rangle, \langle District, Account, Disposition, Card\rangle\}$.
For each chain of dependency discovered, $ch = \langle t_1, t_2, \ldots, t_k\rangle$, a scatter plot is generated with the following properties:
If $t_1 \rightarrow t_2$: The scatter plot is interpreted as x tuples of table t_k are indirectly referenced by y tuples of table t_1 (i.e. the x-axis corresponds to the t_k, while the y-axis corresponds to t_1). The reason for this is that t_1 is directly or indirectly referencing each table in the chain. Thus, table t_1 is in relation 1:1 or 1:N with table t_2 and indirectly with all tables from ch. In this case, if the x-axis corresponds to t_1, the scatter plot would be formed of a single point, corresponding to all of the identifiers of t_1. Each scatter plot is composed of a set of data points, which are composed of the set of identifiers of table t_1. In order to compute these identifiers in this case, the following query is run by the system: `SELECT` $t_k.PK_k$`,` $t_1.PK_1$ `FROM` $t_1 \bowtie \ldots \bowtie t_k$. For each value of $t_k.PK_k$, CoDS will count the number of distinct values for $t_1.PK_1$ from the previous query and this will determine the values for y. For each value of y, CoDS will count how many identifiers of $t_k.PK_k$ (i.e. x value) are associated with y distinct tuples of

t_1. A nested SQL query in this case would result in losing information about the identifiers of each data point, or would require an extra query for each value on the y-axis. In order to avoid multiple queries, CoDS constructs the set of data points from the above query. The method constructs the data point with the values of $t_1.PK_1$. The data point will appear at coordinates (x, y).
If $t_1 \leftarrow t_2$: The scatter plot is interpreted as x tuples of table t_1 are indirectly referenced by y tuples of table t_k (i.e. the x-axis will correspond to the t_1, while the y-axis will correspond to t_k). The reason for this is that each table in the chain is directly or indirectly referencing t_1. Symmetrically with the previous case, if we considered the axes inverted, the scatter plot would consist of a single point. The data points associated to this scatter plot are computed using the following query:

`SELECT` $t_1.PK_1$`, COUNT(DISTINCT` $t_k.PK_k$`) AS y FROM` $t_1 \bowtie \ldots \bowtie t_k$
`GROUP BY` $t_1.PK_1$

The query is distinct in this case as the grouping of values of the y-axis is performed by the identifiers of t_1. Thus, after this query is performed, CoDS constructs for each value of y the set of identifiers of t_1 associated with y number of tuples of table t_k and a data point dp composed of the identifiers discovered with coordinates $(\|dp\|, y)$. In this case, we also do not use a nested query in order to be able to instantiate each data point with the associated values for $t_1.PK_1$ without using additional queries. For instance for $ch = \langle District, Client, Disposition \rangle$ the following query is constructed:

`SELECT` $District.district\_id$`,COUNT(DISTINCT` $Disposition.disp\_id$`) AS y`
`FROM` $District \bowtie Client \bowtie Disposition$ `GROUP BY` $District.district\_id$

The system proceeds in counting how many districts (i.e. x value) have the same y number of dispositions associated. For each value of y it then constructs the data point with the associated values of $district\_id$. Each unique data point will appear at the computed coordinates (x, y) on the scatter plot associated. The scatter plot is presented in Fig. 2. Finally, we observe that a bottom-up approach will determine the processing of smaller results for the queries used and will require less internal processing of data by CoDS, delegating this task to the database management system.

3.4 Identification of tuples to sample

The third phase of the system consists of the selection of identifiers from the starting table to sample so that the size of the starting table will be $\alpha \cdot \|O(t_\star)\|$. The output of this phase is a set of identifiers from the starting table that are required to be included in the sample for preserving the distribution along the discovered chains. We refer to this set of identifiers to sample with Id_S. The input of this phase is a set of chains of dependencies generated previously by the system, with their associated scatter plots and data points. A key point is identifying data which has the same characteristics across all the scatter plots,

as they represent the same scenario. As data points consist of a set of identifiers with the same characteristic on the y and x axis, CoDS considers each data point as a group of identifiers from the starting table with the same characteristics. However, as data points are distributed across multiple scatter plots, a set of identifiers grouped in one scatter plot might be distributed in another. The objective is to produce an α-sample of each of these data points. The *current* number of identifiers of a data point dp represents the number of identifiers of $t_\star$ that have been included in Id_S. It is calculated using the following formula:

$$CurrentNo(dp) = \|Id_S \cap dp\|$$

The *expected* number of tuples of data point dp represents the number of identifiers dp should contain in the sample database. It is defined as:

$$ExpectedNo(dp) = \alpha \cdot \|dp\|$$

The objective of CoDS is to meet the following condition:

$$\forall dp \in \cup_{sp \in \mathcal{S}p(t_\star)} sp : CurrentNo(dp) = ExpectedNo(dp)$$

It is not always feasible that all data points in all scatter plots verify this condition. The system proceeds in checking for each data point dp of all scatter plots whether this condition is met or not. While the latter is true the system calls `balance(`dp`)`. In order to avoid an infinite loop, the maximum number of iterations for calling `balance(`dp`)` is: $\lceil ExpectedNo(dp) - CurrentNo(dp) \rceil$. The function `balance` represents the core functionality of the sampling algorithm. The function is presented in detail in algorithm 1, where $\|dp\|$ represents the number of identifiers that the data point dp contains. In order to decide which identifier should be added to a data point, the system computes for each data point dp the set of related data points, RDP(dp) by intersecting dp with all the data points from all the rest of the generated scatter plots:

$$RDP(dp) = \{dp' \in \cup_{sp' \in \mathcal{S}p(t_\star) \setminus sp} sp' : dp' \cap dp \neq \emptyset\} \quad (1)$$

This information is used to calculate the impact factor of an identifier $id \in dp$:

$$IF(id) = \sum_{dp' \in RDP(dp)} \frac{CurrentNo(dp')}{ExpectedNo(dp')}$$

The impact factor suggests how much impact adding an identifier will have. Adding an identifier with low impact factor will not trigger major differences between the current number and the expected number of any of the related data points, facilitating a balanced insertion. Situations when no identifier is found to insert in Id_S (i.e. as this would disrupt the distribution with the current number of a related data point higher than the expected number) are best avoided using this strategy.

After all data points are balanced by CoDS, if $\|S(t_\star)\| \neq \alpha \cdot \|O(t_\star)\|$, the system finally checks for each value of $PK_\star$ in $O(t_\star)$ whether it can be added to Id_S. The reason for this is to try to fill data points that have the current number 1, and expected number 0 or 1 as these are hardly influenced by `balance(`dp`)`.

Algorithm 1: `balance(`dp`)`

```
1  if CurrentNo(dp) <ExpectedNo(dp) then
2      c ← 0;
3      RDP(dp) ← computeRDP(dp) ;                    // see equation (1).
4      while c < ‖dp‖ do
5          // Retrieve identifier with the c-th smallest impact factor
6          id ← dp.getIdNthSmallestIF(c);
7          Id_S ← Id_S ∪ {id};
8          // Checking whether adding id disrupts any scatter plot
9          if ∃dp' ∈ RDP(dp): CurrentNo(dp') > ⌈ExpectedNo(dp')⌉ then
10             Id_S ← Id_S \ {id};
11         else break;
12         c++ ;
```

3.5 Creation of the database sample

The final phase consists in creating and populating the tables in the sample database. For each table of the original database, we create a new table in the sample database following the same specifications (attributes, types, primary key, foreign keys, etc.). After the insertion of the tuples corresponding to the Id_S of the $t_\star$, `fillRT(`$t_\star$`)` is called. This method ensures that the related tables of $t_\star$ will be filled with referencing or referenced tuples of $\mathrm{S}(t_\star)$. The algorithm is presented in detail in algorithm 2 and it represents a bottom-up breadth-first recursive approach. In this algorithm, $isFilled(t)$ determines whether a table t has already been filled, and $filled(T)$ defines the set of tables of T that have been filled: $filled(T) = \{t_i \in T : isFilled(t_i)\}$. Note that a table t with multiple children is not filled until either all its children or all its children reachable by the already filled tables have been filled. The reason for this is to avoid the space overhead that might be triggered between the children of t. For instance, in Fig. 1, $children(Disposition) = \{Account, Client\}$. Considering $t_\star = District$, filling table *Account* will trigger inserting tuples in table *Disposition*. Filling table *Disposition* with tuples referencing existing tuples in *Account* might trigger inserting tuples in *Client* to avoid missing references. This would trigger inserting tuples in *District* to avoid missing references, and results in a cyclic insertion flow that should be avoided. The function `buildAndExecuteQuery(`t_1, T'`)` (algorithm 2, lines 3 and 8) is used to insert tuples in table t_1 based on already the inserted tuples in tables $T' = \{t_2, t_3, \ldots, t_j\}$. The function executes one of the following queries:

If $t_1 \leftarrow t_2$**:** `INSERT INTO` $S.t_1$ `(SELECT * FROM` $O.t_1$ `WHERE` PK_1 `IN (SELECT` FK_2^1 `FROM` $S.t_2$`) AND ... AND` PK_1 `IN (SELECT` FK_j^1 `FROM` $S.t_j$`))`

If $t_1 \rightarrow t_2$**:** `INSERT INTO` $S.t_1$ `(SELECT * FROM` $O.t_1$ `WHERE` FK_1^2 `IN (SELECT` PK_2 `FROM` $S.t_2$`) AND ... AND` FK_1^j `IN (SELECT` PK_j `FROM` $S.t_j$`))`

Algorithm 2: `fillRT(`t`)`

```
1 Crt ← ∅;
2 for t_i ∈ children(t): NOT isFilled(t_i) do
3 |   buildAndExecuteQuery(t_i, filled(parents(t_i)));
4 |   Crt ← Crt ∪ {t_i};
5 for t_i ∈ Crt do fillRT(t_i);
6 for t_i ∈ parents(t) AND NOT isFilled(t_i) do
7 |   if (∀t_j ∈ children(t_i) : isFilled(t_j) OR ∄ d ∈ desc(t_j): isFilled(d)) then
8 |   |   buildAndExecuteQuery(t_i, filled(children(t_i)));
9 for t_i ∈ parents(t): isFilled(t_i) do fillRT(t_i);
```

4 Evaluation

In this section we evaluate our method and compare it to the Join Synopses approach (JS) [1], and Linked Bernoulli Synopsis approach (LBS) [7]. Both methods aim to construct a consistent database sample of a relational database and are described in detail in section 2.

4.1 Environment and dataset

JS, LBS, and CoDS were implemented against MySQL databases, using Java 1.6. CoDS was deployed on a machine with quad-core 2.5GHz processor, 16GB RAM, and 750GB Serial ATA Drive with 7200 rpm. Each experiment was run with 12GB maximum size of the memory allocation pool. We consider the *Financial* database[4] from PKDD'99 Challenge Discovery (see Fig. 1). It contains typical bank data, such as its clients information, their accounts, transactions, loans, and credit cards. The database contains 8 tables, and a total of $1,079,680$ tuples. The sizes of the tables range from 77 (table *District*) to $1,056,320$ tuples (table *Trans*). The average number of tuples per table is $134,960$.

4.2 Measures

Representativeness In this work, we aim to produce a representative sample of a relational database. In order to measure the accuracy of our approach, we propose to measure the representativeness of a sample as follows. We evaluate the sample database by comparing the distributions between consecutive linked tables in the graph representation of the database (e.g. in Fig. 1: *District* and *Account*, *Account* and *Order*, *Disposition* and *Card*, ...) with their associated distributions in the original database. The representativeness error of the relationship between two tables:

$$\delta(t,t') = \frac{1}{\|sp_t^{t'}\|} \sum_{dp_t^{t'} \in sp_t^{t'}} \min\left(\frac{|S - \lfloor E \rfloor|}{\lfloor E \rfloor}, \frac{|S - \lceil E \rceil|}{\lceil E \rceil}\right)$$

[4] http://lisp.vse.cz/pkdd99/Challenge/berka.htm

where $\|sp_t^{t'}\|$ represents the number of data points in the scatter plot between table t and table t', $S = CurrentNo(dp_t^{t'})$, and $E = ExpectedNo(dp_t^{t'})$. The average representativeness error for a sample database:

$$\delta(T) = \frac{1}{\|T\|} \sum_{t \in T} \left(\frac{1}{\|RT(t)\|} \sum_{t' \in RT(t)} \delta(t, t') \right)$$

Sample size error When sampling a database $O(T)$ with a sampling rate α, we expect that each table will be reduced in size by α. As a consequence, we expect the database size to be reduced by a factor α. An accurate sampling method should produce a sample database $S(T)$ with size $\alpha \cdot \|O(T)\|$. We measure the *global sample size error* of a sample with respect to a database as:

$$global\_sample\_size\_error(T) = \frac{S_T - \alpha \cdot O_T}{\alpha \cdot O_T}$$

where $O_T = \sum_{t \in T} \|O(t)\|$ and $S_T = \sum_{t \in T} \|S(t)\|$.

Time We measure the time needed to sample a database in seconds.

4.3 Results and observations

In this section, we present the results of running CoDS, JS, and LBS with regards to the metrics described in the previous section. The starting table identified by CoDS is the leaf table *District*. A diamond pattern described in detail in [7] is contained in the *Financial* database between the following tables: *District, Client, Account, Disposition* (see Fig. 1). The proposed solutions for applying LBS in this situation is to store the *District* table completely, or switch to JS method. For comparison purposes and due to the small number of tuples of the *District* table, we have chosen to store it completely when applying LBS.

Representativeness error Figure 3 shows the results for the average representativeness error for the sample database. We observe that CoDS method performs best for $\alpha \in [0.1, 0.8]$, while JS performs best for $\alpha = 0.9$. The error varies between 31.7% and 2.3% for the LBS method, resulting in LBS being the less accurate method for this measure. JS method is more accurate than LBS, with the representative error varying between 26.1% and 2.3%. CoDS method is less sensible to the variation of α, with the error varying between 6.5% and 4.9%. We observe that the CoDS method generally produces the most representative sample.

Sample size error Figure 4 shows the results for the global sample size error. The global sample size error can be negative in the case that not enough tuples have been inserted in the sample database. We observe that the LBS technique produces the best sample database with a global sample size error close to 0 for all values of α. The error varies between 199% and 11% for the JS, resulting in

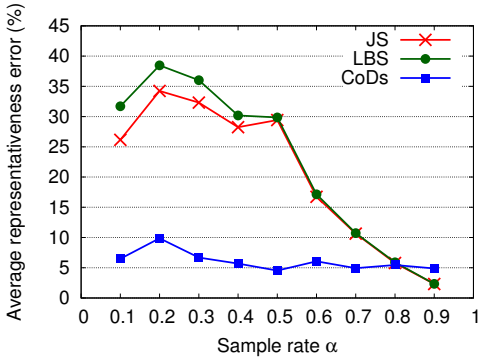

Fig. 3. Representativeness error for JS, LBS, and CoDs.

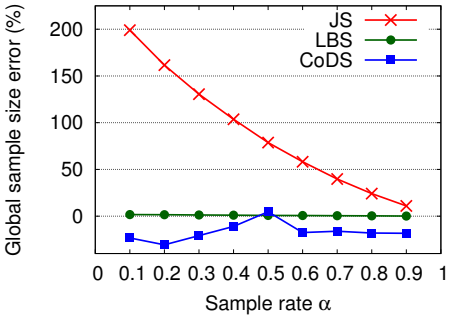

Fig. 4. Global sample size error.

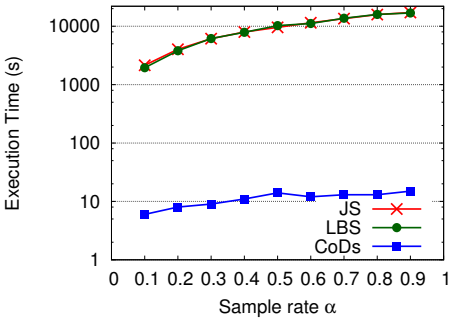

Fig. 5. Execution time.

JS being the less accurate method for this measure. CoDS method is less sensible to the variation of α, with the global sample size error varying between -23.3% and -18.3%. The worst case for all methods occurs when $\alpha = 0.1$. This is unfortunate as generally the desired sample database is less than 50% of the original database to reduce the computational cost of analyzing the data at least by half. The reason why CoDS generally produces a sample database with less tuples than desired is because the method is cautious, and does not insert tuples that might disturb the representativeness of the sample.

Execution time Figure 5 shows the execution time for CoDS, JS, and LBS methods. We observe that CoDS outperforms JS, and LBS producing a sample database 300-1000 times faster, for α ranging from 0.1 to 0.9. The execution time in the case of JS and LBS is dependent on the processing of each tuple of each table in the original database.

In conclusion, we observe that CoDS produces the best results in terms of representativeness except for $\alpha = 0.9$. We observe that CoDS is very close to the best solution in terms of global sample size error and outperforms JS and LBS

method with regards to the execution time for all values of α by producing a sample database between 300 and 1000 times faster.

5 Conclusion

In this paper, we proposed CoDS, a novel approach for relational database sampling. CoDS aims to produce a representative consistent sample by taking into consideration the dependencies between the data in a relational database. To do so, CoDS analyses the distribution between a certain table (called the starting table) and all the other tables. We conducted experiments on the *Financial* database. Results show that CoDS outperforms the previous existing consistent sampling approaches in terms of representativeness and also in terms of execution time. The sampling algorithm aims to significantly decrease the storage space needed for the original database, while achieving a balance between the computational cost of running the analysis on the original database and the accuracy of the results by preserving the properties of the original database.

As future work, we plan to extend our method to take into account other characteristics of the database. In particular we aim to consider the distribution of attributes values in order to produce a sample that is realistic not only at the table-level, but also at the attribute-level. We plan to study how to improve our method's accuracy in terms of sample size error, while maintaining the representativeness of the sample. Last but not least, we plan to apply our approach to populate testing environments. This work will be done in collaboration with IBM. The objective is to significantly decrease the time it takes to populate the testing environment, and demonstrate in a real situation that the representativeness of a sample allows to find more anomalies in the code in comparison with random-based samples.

6 Acknowledgments

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

1. S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *International Conference on Management of Data (SIGMOD)*, pages 275–286, 1999.
2. S. Agarwal, A. P. Iyer, A. Panda, S. Madden, B. Mozafari, and I. Stoica. Blink and it's done: interactive queries on very large data. *VLDB Endowment*, 5(12):1902–1905, 2012.
3. J. Bisbal, J. Grimson, and D. Bell. A formal framework for database sampling. *Information and Software Technology*, 47(12):819–828, 2005.
4. V. T. Chakaravarthy, V. Pandit, and Y. Sabharwal. Analysis of sampling techniques for association rule mining. In *12th ACM International Conference on Database Theory (ICST)*, pages 276–283, 2009.

5. S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *ACM international conference on Management of Data (SIGMOD)*, pages 287–298, 2004.
6. E. Ferragut and J. Laska. Randomized sampling for large data applications of SVM. In *11th IEEE International Conference on Machine Learning and Applications (ICMLA)*, volume 1, pages 350–355, 2012.
7. R. Gemulla, P. Rösch, and W. Lehner. Linked bernoulli synopses: Sampling along foreign keys. In *20th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 6–23, 2008.
8. B. Goethals, W. Le Page, and M. Mampaey. Mining interesting sets and rules in relational databases. In *25th ACM Symposium on Applied Computing (SAC)*, pages 997–1001, 2010.
9. P. J. Haas and C. König. A bi-level bernoulli scheme for database sampling. In *ACM International Conference on Management of Data (SIGMOD)*, pages 275–286, 2004.
10. Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *25th International Conference on Very Large Data Bases (VLDB)*, pages 174–185, 1999.
11. G. John and P. Langley. Static versus dynamic sampling for data mining. In *2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 367–370, 1996.
12. H. Köhler, X. Zhou, S. Sadiq, Y. Shu, and K. Taylor. Sampling dirty data for matching attributes. In *ACM International Conference on Management of Data (SIGMOD)*, pages 63–74, 2010.
13. X. Lu and S. Bressan. Sampling connected induced subgraphs uniformly at random. In *24th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 195–212, 2012.
14. F. Olken. *Random Sampling from Databases*. PhD thesis, University of California at Berkeley, 1993.
15. C. Olston, S. Chopra, and U. Srivastava. Generating example data for dataflow programs. In *International Conference on Management of Data*, pages 245–256, 2009.
16. C. R. Palmer and C. Faloutsos. Density biased sampling: an improved method for data mining and clustering. In *ACM International Conference on Management of Data (SIGMOD)*, pages 82–92, 2000.
17. F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *5th ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 23–32, 1999.
18. K. Taneja, Y. Zhang, and T. Xie. MODA: Automated test generation for database applications via mock objects. In *IEEE/ACM International Conference on Automated Software Engineering*, 2010.
19. H. Toivonen. Sampling large databases for association rules. In *22th International Conference on Very Large Data Bases (VLDB)*, 1996.
20. X. Wu, Y. Wang, S. Guo, and Y. Zheng. Privacy preserving database generation for database application testing. *Fundamenta Informaticae*, 78(4):595–612, 2007.
21. X. Yin, J. Han, J. Yang, and P. Yu. Efficient classification across multiple database relations: a crossmine approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(6):770 –783, 2006.