J. Hlaváčová (Ed.): ITAT 2017 Proceedings, pp. 106–111
CEUR Workshop Proceedings Vol. 1885, ISSN 1613-0073, © 2017 J. Motl, P. Kordík

Foreign Key Constraint Identification in Relational Databases

Jan Motl, Pavel Kordík

Czech Technical University in Prague,
Thákurova 9, 160 00 Praha 6, Czech Republic,
jan.motl@fit.cvut.cz, pavel.kordik@fit.cvut.cz

Abstract: For relational learning, it is important to know the relationships between the tables. In relational databases, the relationships can be described with foreign key constraints. However, the foreign keys may not be explicitly specified. In this article, we present how to automatically and quickly identify primary & foreign key constraints from metadata about the data. Our method was evaluated on 72 databases and has F-measure of 0.87 for foreign key constraint identification. The proposed method significantly outperforms in runtime related methods reported in the literature and is database vendor agnostic.

1 Introduction

Whenever we want to build a predictive model on relational data, we have to be able to connect individual tables together [3]. In *Structured Query Language* (SQL) databases, the *relationships* (connections) between the tables can be defined with *foreign key* (FK) constraints. However, FK constraints are not always available. This can happen, for example, whenever we work with legacy databases or data sources, like *comma separated value* (CSV) files.

Identification of relationships from database belongs to *reverse engineering* from databases [14] and can be done manually or by means of handcrafted rules [2, 3, 7, 17]. Manual FK constraint discovery is very time-consuming for complex databases [11]. And handcrafted systems may overfit to small collections of databases, used for the training. Therefore we use machine learning techniques for this task and evaluate them on a collection of 72 databases.

Unfortunately, FK constraint identification is difficult. If we have n columns in a database, then there can be n^2 FK constraints, as each column can reference any column in the database, including itself[1]. Hence, there is n^2 candidate FK constraints.

Example 1. *If we have a medium-sized database with 100 tables, each with 100 columns, then we have to consider* 10^8 *candidate FK constraints.*

We can evaluate probability p that a single candidate FK constraint is a FK constraint with a classifier (e.g. logistic regression) in a constant time. Hence, if we assumed that the probability $p_{i,j}$, which denotes a probability that a column i references column j, is independent of all other candidate FK constraints in the database, the computational complexity of FK constraint identification would be $O(n^2)$. However, the probabilities do not appear to be independent.

Example 2. *If we had two columns A, B and we had known that $p_{A,B} = 0.9$ and $p_{B,A} = 0.8$ then under assumption of independence it would be reasonable to predict that column A references column B and also that column B references column A. However, directed cyclic references[2] do not generally appear in the databases as it would make updates inconveniently difficult [10]. Hence, our example database most likely contains only one FK constraint with A referencing B.*

If we accepted that the FK constraints are not independent of each other, we could generate each possible combination of FK constraints and calculate the probability that the candidate combination of FK constraints is the true combination of FK constraints. The computational complexity of such algorithm is $O(2^{n^2})$. Clearly, a practical algorithm must take simplifying assumptions to scale to complex databases.

The applications of the FK constraint discovery, besides relational learning, include *data quality* assessment [1] and *database refactoring* [11].

The paper is structured as follows: first, we describe related work, then we describe our method, then we describe our experiments and their results, discuss the results and provide a conclusion.

2 Related Work

Li et al. [8] formulated a related problem, attribute correspondence identification, as a classification problem.

Rostin et al. [16] formulate FK constraint identification as a classification problem.

Meurice et al. [11] compared different data sources for the FK constraint identification: database schema, Hibernate XML files, JPA Java code and SQL code. Based on their analysis, the database schema data source has four times higher recall than any other data source. In this article, we focus solely on the database schema data source. Furthermore, they introduce 4 rules for filtering the candidate FK constraints: the "likeliness" of the candidate FK constraint must be above a threshold, the FK constraints cannot be bidirectional, the column(s) of the selected FK

[1]We have not observed any instance of a column referencing itself. Nevertheless, SQL standard does not forbid it.

[2]However, undirected cyclic references are commonly used, for example, to model hierarchies.

constraints can be used only once and there can be only a single (undirected) path from FK constraints between any two tables.

Chen et al. [3] describe how to significantly accelerate FK constraint identification by pruning unpromising candidates at multiple levels. We inspire from them and use multi-level architecture as well. Furthermore, they introduce 4 rules for filtering the candidate FK constraints: explore FK constraints only between the tables selected by the user, only a single FK constraint can exist between two tables, directed cycles from FK constraints are forbidden and there can be only a single (undirected) path from FK constraints between any two tables. We inspire from Meurice's and Chen's articles and reformulate their rules as *integer linear programming* (ILP) problem.

3 Method

To make the relationship identification fast, a predictive model was trained only on the metadata about the data, which are accessible with *Java Database Connectivity* (JDBC) API. This approach has the following properties:

1. It is fast and scalable.
2. It is database vendor agnostic.
3. It is not affected by the data quality.

The problem of relationship identification was decomposed into two subproblems: identification of *primary keys* (PKs) and identification of FK constraints (Figure 1). The reasoning behind this decomposition is that identification of PKs is a relatively easy task. And knowledge of PKs simplifies identification of FK constraints because FK constraints frequently reference PKs[3].

The identification of the PKs is performed in two stages: scoring and optimization. During the scoring phase, a probability that an attribute is a part of a PK (a PK can be *compound* — composed of multiple attributes) is predicted with a classifier. The probability estimates are then passed into the optimization stage, which delivers a binary prediction.

The same approach is taken for FK constraint identification. During the scoring phase, a probability that a candidate FK constraint is a FK constraint is estimated with a classifier. The probabilities are then passed into an optimizer, which returns the most likely FK constraints.

3.1 Primary Key Scoring

All metadata that are exposed by JDBC[4] about attributes (as obtained with *getColumns* method) and tables (as obtained with *getTables*) were collected and considered as features for classification. For brevity, we describe and justify only features used by the final model.

[3]A FK may reference any attribute that is unique, not only PKs. Nevertheless, all FKs in the analyzed databases reference PKs.

[4]See docs.oracle.com for the documentation.

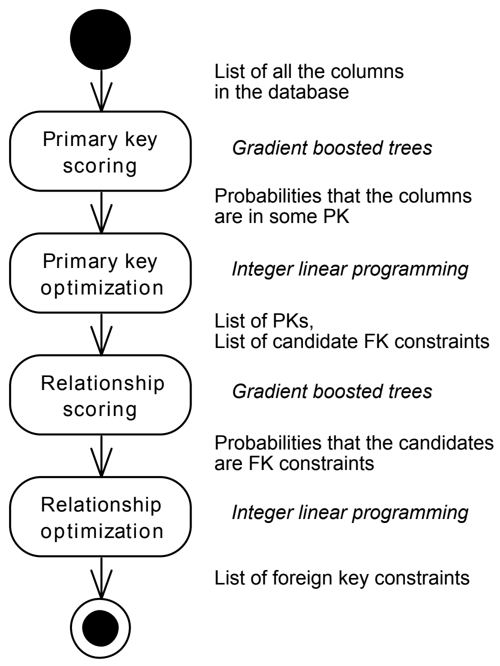
List of all the columns in the database
Primary key scoring
Gradient boosted trees
Probabilities that the columns are in some PK
Primary key optimization
Integer linear programming
List of PKs, List of candidate FK constraints
Relationship scoring
Gradient boosted trees
Probabilities that the candidates are FK constraints
Relationship optimization
Integer linear programming
List of foreign key constraints

Figure 1: The algorithm decomposition.

Data types like *integer* or *char* are generally more likely to be PKs than, for example, *double* or *text*. To promote portability of the trained model, we do not use database data types but JDBC data types, which have the advantage that they are the same regardless of the database vendor.

Since some databases offer only a single data type for numerical attributes, we also note whether numerical attributes can contain **decimals**, as PKs are unlikely to contain decimal numbers.

Doppelgänger is an attribute, which has a name similar to another attribute in the same table. For example, *atom_id1* is a doppelgänger to *atom_id2*. Doppelgängers frequently share properties, i.e. either both of them are in the PK or neither of them is in the PK.

Ordinal position defines the position of the attribute in the table. PKs are frequently at the beginning of the tables.

String distance between the column and table names are helpful for identification of PKs and FKs. Opinions on the best measure for PK and FK constraint identification vary. For example, [16] uses exact match while [3] uses Jaro-Winkler distance. After testing all string measures available in *stringdist* package [9], we found that Levenshtein distance delivers the best discriminative power on the tested databases.

Keywords like *id* or *pk* frequently mark PKs. The presence of the keywords is analyzed after the attribute/table name tokenization, which works with camel case and snake case notation.

JDBC also provides attributes that leak information about the presence of the PK, like *isNullable*, *isAutoIncrement* and *isGeneratedColumn*. Since it is unreasonable to assume that these metadata would be set correctly after importing data from CSV files, they were excluded from the model.

For comparison to features extracted from the data (and not metadata), two additional features were extracted: whether the attribute contains nulls (*containsNull*) and whether the attribute contains unique values (*isUnique*). These features are generally expensive to calculate [3]. Nevertheless, some databases, like PostgreSQL, automatically generate these statistics for each attribute in the database and provide a vendor-specific access to these statistics.

3.2 Primary Key Optimization

Since each table in a well-designed database should contain a PK, a single most likely PK is identified for each table in the database. If the single most likely PK attribute in a table is a doppelgänger, all its doppelgängers in the table are declared to be part of the PK as well, creating a compound key. The described optimization can be solved with an ILP solver, which we use mostly because foreign key optimization (section 3.4) is using ILP formulation as well.

3.3 Foreign Key Scoring

Features for FK constraints are a combination of features calculated for individual attributes from section 3.1 (prefixed with *fk* and *pk* respectively) with features unique for the FK constraints. The description of the unique features follows.

Data types between FK and PK attributes should match. Nevertheless, SQL permits FK constraints between *char* and *varchar* data types.

Data lengths between FK and PK attributes should match. Nevertheless, SQL explicitly permits FK constraints between attributes of different character lengths as defined in the SQL-92 specification, section 8.2.

String distance between FK column name and PK table name should be small because FK column names frequently embed a part of the PK table name. Similarly, FK column name should be similar to PK column name because FK column names frequently embed a part of the PK column name. On the other end, FK column name should generally differ from FK table name as they are not directly related.

Furthermore, to be able to compare metadata-based features to data-based features, we tested whether all non-null values in the FK are present in the PK (*satisfiesFKConstraint*). This is generally an expensive feature to calculate [3]. Nevertheless, some databases, like PostgreSQL, automatically calculate histograms for each attribute in the background and offer a vendor specific interface to access the histograms. And based on the range of the histograms many candidate FK constraints can be pruned. More advanced data-based features (e.g. similarity of the FK and PK distributions) were not explored as the focus of the article is on the metadata-based features.

3.4 Foreign Key Optimization

The optimization can be formulated as an integer linear optimization problem on a directed graph $G = (V, E)$, where V is the set of attributes in the database and E is the set of candidate FK constraints. The p_{ij} is the estimated probability that the candidate FK constraint referencing FK i to PK j is a FK constraint. The probabilities are estimated with a classification model trained on features described in section 3.3. Compound FKs are modeled as multiple FK constraints (one FK constraint for each attribute). We define variable x_{ij}:

$$x_{ij} = \begin{cases} 1 & \text{if the candidate FK constraint is a FK constraint} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The optimization problem is then:

$$\max_{x} \quad \sum_{[i,j]\in E} x_{ij} - 2 \sum_{[i,j]\in E} x_{ij}(1 - p_{ij}) \tag{2a}$$

s.t.

$$\sum_{j\in V} x_{ij} \leq 1, \qquad \forall i, \tag{2b}$$

$$\sum_{i\in S, j\in S, [i,j]\in E} x_{ij} \leq |S| - 1, \qquad \forall S \subseteq V, |S| \geq 1, \tag{2c}$$

$$x_{i_1 j_1} - x_{i_2 j_2} = 0, \qquad \forall P, F \subseteq V, i \in F, j \in P, |P| \geq 2, \tag{2d}$$

$$x_{i_1 j} - x_{i_2 j} = 0, \qquad \forall D \subseteq V, i \in D, j \in V, \tag{2e}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in V, j \in V. \tag{2f}$$

The objective function defines all FK constraint candidates x_{ij} with $p_{ij} > 0.5$ as FK constraints if it does not violate any of the following constraints.

Unity constraint 2b enforces that a FK can reference only a single PK. While a single FK can in theory reference multiple different PKs, no such occurrence appeared in the analyzed databases.

Acyclicity constraint 2c ensures that the graph is (directionally) acyclic. However, this formulation of acyclicity requires an exponential number of constraints. To deal with that, we generate acyclicity constraints lazily [15]. Acyclicity constraint is desirable because if p_{ij} is high, p_{ji} is generally high as well (particularly for $i = j$). But directed cycles (even over intermediate tables) do not appear in the analyzed databases.

Completeness constraint 2d says that if a PK is compound, then either all or neither attribute of the PK P is referenced from the FK table by attributes F. Completeness constraint ensures that compound FKs are syntactically correct.

Doppelgänger constraint 2e says that if attributes are doppelgängers to each other, then either all or neither attribute from the doppelgänger set D reference the (same) PK attribute.

Constraint 2f defines the problem as an integer programming problem.

It should be noted that if constraints 2d and 2e are removed, we get an optimization problem similar to the identification of minimum spanning tree in a graph [6]. Hence, the FKs can be efficiently optimized with Dijkstra algorithm with a modified termination condition (the algorithm terminates once the objective function starts to increase).

4 Results

Following paragraphs describe an empirical comparison of 5 classifies on 3 different sets of features from 72 databases.

4.1 Data

We used all 72 relational databases from relational repository [12]. The databases range from classical relational benchmarking databases (like *TPC-C* or *TPC-H*) to real-world databases used in challenges (e.g. from PKDD in 1999 or from Predictive Toxicology Challenge in 2000). The collection of the databases contains in total 1343 PKs, 1283 FK constraints, 6129 attributes and 788 tables. That means that on average approximately 1 of 5 attributes is part of a PK. The count of all *possible relationships* is 1,232,392 (in theory, a FK can reference any attribute in the database, including itself). That means that on average approximately 1 of 960 tested relationships are FK constraints.

4.2 Algorithm

Following classification algorithms were tested on the problem: decision tree, gradient boosted trees, naive Bayes, neural network and logistic regression as implemented in RapidMiner 7.5. Since the best results were obtained with gradient boosted trees, the reported results are for gradient boosted trees.

4.3 Measure

For evaluation of the classification models, AUC and F-measure [5] were used. Classification accuracy was omitted due to a significant class imbalance in FK identification task. AUC evaluates the ability of the model to rank. Hence, AUC is used to evaluate the quality of scoring. On the other end, F-measure is suitable for the evaluation of the quality of thresholding. Hence, F-measure is used to evaluate the quality of the optimization.

4.4 Validation

To measure the generalization of the models to new unobserved databases, *batch cross-validation* over databases [16, section 4.3] was performed. Since 72 databases were analyzed, it means that each model was trained and evaluated 72 times. The batch cross-validation has the advantage, in comparison to 10-fold cross-validation, that the samples from a single database are either all in the training set or all in the testing set. Hence, if the samples from a single database are more similar to each other than to samples from other databases, we may expect that batch cross-validation will deliver a less optimistically biased estimate of the model accuracy on new unobserved databases than 10-fold cross-validation.

4.5 Feature Importance

Generally, it is desirable to minimize the count of utilized features to make the model easier to understand and deploy. Table 1 depicts feature importance for PK identification as reported by gradient boosted trees for features that remained after backward selection.

Table 1: Feature importance for primary key identification for different feature sets. Higher weight means higher importance.

Feature	*All*	*Meta*	*Ordinal*	*Data*
ordinalPosition	3279	1970	2581	-
isDoppelgänger	142	99	-	-
isDecimal	80	81	-	-
containsNull	18	-	-	239
levenshteinToTable	15	124	-	-
dataType	14	71	-	-
isUnique	9	-	-	780
containsKeyword	7	21	-	-
AUC	**0.985**	0.970	0.934	0.784

The single most important feature for PK identification is the position of the attribute in the table. This is not so surprising because all non-compound PKs in the analyzed databases (with the exception of *Hepatitis* database) were the first attribute in the table. Indeed, if we always predicted that the first attribute in a table is a PK, we get F-measure equal to 0.934±0.007.

Table 2 lists feature importance for FK constraint identification. Interestingly, the knowledge whether the FK constraint is satisfiable is the least important feature from the selected features.

4.6 Optimization Contribution

The PK optimization improves F-measure of PK identification from 0.845±0.069 to 0.875±0.057. While FK optimization improves F-measure of FK constraint identification from 0.743±0.020 to 0.870±0.022.

Table 2: Feature importance for foreign key constraint identification. Higher weight means higher importance.

Feature	*All*	*Meta*	*Data*
levenshteinFkColToPkTab	298.1	301.2	-
levenshteinFkColToFkTab	245.8	246.0	-
fk_isDoppelgänger	210.6	210.4	-
levenshteinFkColToPkCol	182.2	182.2	-
fk_containsKeyword	160.2	160.3	-
dataLengthAgree	92.2	92.2	-
pk_isDoppelgänger	60.3	60.3	-
fk_isPrimaryKey	57.8	57.8	-
fk_ordinalPosition	48.4	48.2	-
dataTypeAgree	10.3	10.2	-
satisfiesFKConstraint	1.6	-	382
AUC	**0.990**	0.988	0.934

4.7 Runtime & Scalability

The time required to score all 72 databases is 55 seconds in total, where 95% of the runtime is due to the fact that JDBC collects metadata about the attributes for each table individually, causing many round trips between the algorithm and the database server. When we replaced JDBC calls with a single query to *information_schema*, which provides all the data at the database level, the total runtime decreased to 5 seconds.

Furthermore, the algorithm was tested on our university database with 909 tables. The runtime was 18 minutes, due to the quadratic growth of candidate FK constraints with the count of attributes in the database [3]. To keep the memory requirements manageable, FK candidates were scored on the fly and only the top *n* FK candidates with the highest probability were kept in a heap for FK optimization.

5 Discussion

Table 3 depicts a comparison of our approach to different approaches in the literature. Since the implementations of the referenced approaches are not available, we take and report the measurements for the biggest common denominator of the evaluated databases — the TPC-H database. The approaches differ in the utilized features (e.g. Kruse et al. utilize SQL scripts, while our approach does not) and objectives (e.g. Chen et al. aim to maximize precision at the expense of recall). The results of our method for all 72 databases are available for download at `https://github.com/janmotl/linkifier`.

Empirical comparison of our metadata-based approach to other metadata-based approaches is in Table 4. Oracle Data Modeler [13] estimates FK constraints based on the knowledge of PKs (it is assumed that a FK must reference a PK), equality of column names between the FK and the PK and equality of the data types between the FK and the PK. SchemaCrawler [4] is using an extended version of these three filters. SchemaCrawler assumes that a FK must reference either a PK or a column with a unique constraint. The column names must equal but differences in the presence/absence of *id* keyword and differences between singular and plural forms are ignored, improving the recall. And datatypes must equal including their length (except of *varchar* datatype), improving the precission.

5.1 Limitations

The metadata-based identification of PK and FK constraints is limited by the quality of the metadata. For example, if all the columns in the database had non-informative names and all the columns were typed as *text*, the accuracy of the predictions would suffer.

But even if the metadata are of hight quality, our metadata-base approach is not able to reliably reconstruct a hierarchy of subclasses. The problem is illustrated in Figure 2. Based on the table and PK names, we can correctly infer that *Person* and *Vendor* are subclasses of *BusinessEntity*. However, our metadata-based method has no means how to infer that *Customer* and *Employee* are subclasses of *Person* and not directly of *BusinessEntity*.

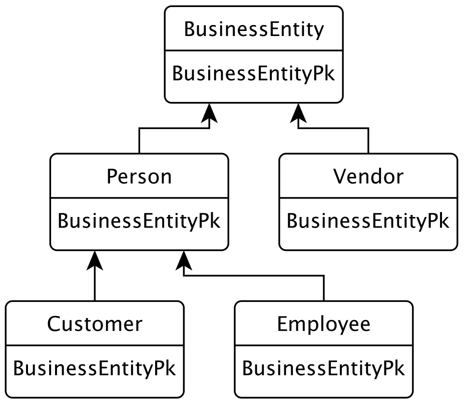

Figure 2: Example entity diagram with a hierarchy of subclasses.

Both these limitations can be addressed by extending the metadata-based approach by appropriate data-based features. For example, whenever a subclass can reference multiple superclasses, the superclass with the lowest row count, which still satisfies the FK constraint, should be selected.

6 Conclusions

We described a method for foreign key constraint identification, which does not put any assumptions on the schema normalization, data quality, availability of vendor specific metadata or human interaction. The code for primary & foreign key constraint identification was designed to be database vendor agnostic and was successfully tested against Microsoft SQL Server, MySQL, PostgreSQL and Oracle. The code with a graphical user interface is

Table 3: Literature review of different approaches to foreign key constraint identification on TPC-H 1GB database. Unknown values are represented with a question mark.

Reference	Features	Objective	Precision	Recall	F-measure	Runtime [s]
Zhang et al. [17]	Data	F-measure	**1.00**	**1.00**	**1.00**	501
Chen et al. [3]	Data, Metadata	Precision	**1.00**	**1.00**	**1.00**	14
Rostin et al. [16]	Data, Metadata	F-measure	?	?	0.95	450
Our	Metadata	F-measure	0.77	0.77	0.77	**1**

Table 4: Empirical evaluation of metadata-based approaches to foreign key constraint identification on 72 databases together.

Implementation	F-measure	Runtime [s]
Oracle Data Modeler	0.06	**2.07**
SchemaCrawler	0.17	4.65
Our	**0.87**	5.14

published at GitHub (`https://github.com/janmotl/linkifier`) under BSD license. The runtime is dominated by the connection lag to the server and if the requirement on the code portability is lifted, we are able to predict primary & foreign key constraints for all 72 tested databases in 5 seconds.

7 Acknowledgments

We would like to thank Aleš Fišer, Jan Kukačka, Jiří Kukačka, Manuel Muñoz and Batal Thibaut for their help. We furthermore thank the anonymous reviewers, their comments helped to improve this paper. This research was partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS17/210/OHK3/3T/18.

References

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015.

[2] John G. Bennett, Perry A. Gee, and Charles E. Gayraud. System and Methods Including Automatic Linking of Tables for Improved Relational Database Modeling with Interface, 1997.

[3] Zhimin Chen, Vivek Narasayya, and Surajit Chaudhuri. Fast Foreign-Key Detection in Microsoft SQL Server PowerPivot for Excel. *VLDB Endow.*, 7(13):1417–1428, 2014.

[4] Sualeh Fatehi. SchemaCrawler, 2017.

[5] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognit. Lett.*, 27(8):861–874, jun 2006.

[6] Dorit S. Hochbaum. Integer Programming and Combinatorial Optimization. *IEOR269 notes*, 2010.

[7] Sebastian Kruse, Thorsten Papenbrock, Hazar Harmouch, and Felix Naumann. Data Anamnesis: Admitting Raw Data into an Organization. *IEEE Data Eng. Bull.*, pages 8–20, 2016.

[8] Wen Syan Li and Chris Clifton. SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl. Eng.*, 33(1):49–84, 2000.

[9] Nick Logan. Package stringdist. Technical report, CRAN, 2016.

[10] Victor Markowitz. Safe referential integrity and null constraint structures in relational databases. *Inf. Syst.*, 19(4):359–378, jun 1994.

[11] Loup Meurice, Fco Javier Bermúdez Ruiz, Jens H. Weber, and Anthony Cleve. Establishing referential integrity in legacy information systems - Reality bites! *Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014*, pages 461–465, 2014.

[12] Jan Motl and Oliver Schulte. The CTU Prague Relational Learning Repository, 2015.

[13] Oracle. Oracle SQL Developer Data Modeler, 2017.

[14] Lurdes Pedro de Jesus and Pedro Sousa. Selection of Reverse Engineering Methods for Relational Databases. *Proc. Third Eur. Conf. Softw. Maint.*, pages 194–197, 1998.

[15] Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the TSP from pure integer solutions. *Cent. Eur. J. Oper. Res.*, pages 1–30, 2016.

[16] Alexandra Rostin, Oliver Albrecht, Jana Bauckmann, Felix Naumann, and Ulf Leser. A machine learning approach to foreign key discovery. *12th Int. Work. Web Databases (WebDB), Provid. Rhode Isl.*, (WebDB):1–6, 2009.

[17] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. On multi-column foreign key discovery. *Proc. VLDB Endow.*, 3(1-2):805–814, 2010.