Efficiently Detecting Inclusion Dependencies

Jana Bauckmann[†] Ulf Leser[†] Felix Naumann[‡] Véronique Tietz[†]

[†]Department for Computer Science
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
{bauckmann,leser,vtietz}
@informatik.hu-berlin.de

[‡]Hasso-Plattner-Institut
University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
naumann@hpi.uni-potsdam.de

Abstract

Data sources for data integration often come with spurious schema definitions such as undefined foreign key constraints. Such metadata are important for querying the database and for database integration.

We present our algorithm SPIDER *(Single Pass Inclusion DEpendency Recognition) for detecting inclusion dependencies, as these are the automatically testable part of a foreign key constraint. For IND detection all pairs of attributes must be tested.* SPIDER *solves this task very efficiently by testing all attribute pairs in parallel. It analyzes a* 2 *GB database in* ~ 20 *min and a* 21 *GB database in* ~ 4 *h.*

1. Schema Discovery for Data Integration

In large integration projects one is often confronted with undocumented data sources. One important schema information are foreign key constraints, which are necessary for meaningful querying and for database integration.

One example is the popular life science database Protein Data Bank (PDB) that can be imported into a relational schema using the OpenMMS[1] schema and parser. The OpenMMS schema defines 175 tables with 2,705 attributes but not a single foreign key constraint.

If foreign key constraints are not defined explicitly they are given implicitly by the data. Thus, we want to identify inclusion dependencies (INDs) in a schema as they are a precondition for foreign keys. An inclusion dependency $A \subseteq B$ means that all values of the *dependent* attribute A are contained in the value set of the *referenced* attribute B. We call a pair of attributes A and B an IND candidate. An IND is *satisfied* if the IND requirements are met and *unsatisfied* otherwise. Obviously, a satisfied IND is only the automatically testable part of a foreign key. Whether a special IND corresponds to a semantically correct foreign key must be decided in a second step.

The challenge in detecting INDs is the potentially large number of IND candidates as each pair of attributes must be tested. With n attributes this results in $(n-1)^2$ IND candidates. To the best of our knowledge all previous approaches for unary IND detection restrict IND candidates by the attribute's data type as proposed in [5], i. e., only IND candidates are created with both attributes sharing the same data type. This restriction reduces the problem complexity heavily, but in life science databases we cannot use this restriction, because one cannot rely on datatypes.

In this paper we present our algorithm SPIDER to detect unary INDs. SPIDER works in two phases: (i) all attribute value sets are sorted inside a RDBMS, and (ii) all IND candidates are tested in parallel while reading each attribute's value at most once. Its strength is the data structure that synchronizes all IND candidate tests efficiently without running into deadlocks or missing an IND.

There are two approaches in related work on detecting unary INDs exactly: Bell and Brockhausen use SQL join statements to test single IND candidates [3]. They leverage filters on data type, maxima and minima to prune IND candidates and utilize already finished IND tests for further pruning exploiting transitivity of IND. Marchi et al. propose to preprocess all data by assigning to each value in the database all attribute's names that include this value [6]. The results are stored in tables – one table per data type. Afterwards all IND candidates are tested in parallel exploiting the sets of attribute names. We shall show in Sec. 3 that SPIDER outperforms both approaches up to orders of magnitude.

In [2] we tested the performance of several possible SQL statements for IND tests, i. e., join, minus, and not in. The fastest approach was to join the IND candidate's attributes

[1] openmms.sdsc.edu

and to compare the join cardinality with the dependent attribute's cardinality. However, we showed that all SQL approaches are much slower for large databases than algorithms based on sorted attribute value lists.

2. Detecting Unary INDs

To compute all unary INDs exactly we must test all pairs of attributes. The test can be performed using the following procedure: (i) Sort and "distinct" the value sets of all attributes using an arbitrary but fixed sort order. (ii) Iterate over the sorted value sets of each pair starting from the smallest item using cursors. Let dep be the current dependent value and ref be the current referenced value of an IND candidate. If dep $=$ ref or dep $>$ ref move both cursors or move the referenced cursor, respectively. Otherwise, if dep $<$ ref stop the execution immediately, because we found a dependent value not contained in the referenced attribute. An IND candidate is shown to be a satisfied IND if all dependent values are found in the referenced value set.

The two following algorithms apply this approach. Both use the database to sort and "distinct" the values of all attributes, and then write the sorted lists to disk. However, the order in which those sets are read is entirely different.

Brute Force This algorithm creates and tests all IND candidates sequentially, i. e., one by one. Compared to a SQL join, the main advantages are (i) the single sort of each attribute and (ii) the implemented early stop for unsatisfied INDs as, of course, most IND candidates are unsatisfied and the test can often stop after comparing only a few or even only a single value pair. The brute force algorithm has the disadvantage that each attribute's values are read as often as the attribute is part of an IND candidate.

Single Pass Inclusion DEpendency Recognition (SPIDER) This algorithm eliminates the need to read data multiple times. All IND candidates are created and tested in parallel. SPIDER first opens all attribute files on disk and starts reading values using one cursor per file. The challenge is to decide when the cursor for each file can be moved. All dependent attributes affect the point in time at which the cursor of a referenced attribute can be moved. But also all referenced attributes influence when the cursor of a dependent attribute can be moved. Despite this mutual dependency, it is possible to synchronize the cursor movements without running into deadlocks or missing some IND candidate tests, because we use *sorted* data sets.

Each attribute is represented in SPIDER as an *attribute object* providing the attribute's sorted values and a cursor to the current value. An attribute can be covered in multiple IND candidates as referenced attribute and / or as dependent attribute. Thus, each attribute object potentially has two roles. For short, we call attribute objects with a dependent role DEP and with a referenced role REF.

IND candidates are represented (and tested) in the dependent attribute role. Each DEP maintains two lists of REFs: (i) satisfiedRefs contains REFs that are known to contain this DEP's current value, (ii) unsatisfiedRefs contains all REFs that are not (yet) known to contain this value but that contain all previous values of this DEP.

All attribute objects are stored in a min-heap sorted by their current values. We remove sets of attribute objects with minimal but equal values (Min) and process them as follows until the heap is empty: (i) Inform all DEP in Min of all REFs in Min. This way, the DEPs can track and test the IND candidates by moving REFs from unsatisfiedRefs to satisfiedRefs. (ii) Test for each DEP in Min if it has a next value. If so, update the the lists satisfiedRefs and unsatisfiedRefs in the following way: All IND candidates given by this DEP and all REFs in unsatisfiedRefs can be excluded (i. e. empty unsatisfiedRefs), because they do not contain the DEP's current value (we cannot have missed it). All REFs in satisfiedRefs contain (till now) all values of the dependent attribute object. To prepare the next iteration, move all REFs from satisfiedRefs to unsatisfiedRefs. For all DEP in Min with no next value output the INDs given by their REFs as satisfied IND. (iii) For all attribute objects in Min (DEPs and REFs) having a next value: Move the cursor to this next value and insert the attribute object into the heap.

Using the min-heap, we synchronize the processing of all values of all attributes. Therefore, we cannot miss a IND or run into a deadlock.

3. Experimental Results

We tested the algorithms on real life databases from the life sciences and on a synthetic database: *UniProt*[2] is a database of annotated protein sequences [1] available in several formats. We used the BioSQL[3] schema and parser, creating a database of 16 tables with 85 attributes and a total size of 667 MB. *PDB* is a large database of protein structures [4]. We used the OpenMMS software for parsing PDB files into a relational database. The PDB populates 115 tables over 1,711 attributes in the OpenMMS schema. There are no specified foreign keys. The total database size is 21 GB. To achieve a better idea of the scalability of our approach, we also used a 2.7 GB fraction of the PDB obtained by removing some extremely large tables. Finally, we also verify our results by using a generated *TPC-H* database (scale factor 1.0).

We run all experiments on a Linux system with 2 Intel Xeon processors (2.60 GHz) and 12 GB RAM. We use a

[2] www.pir.uniprot.org
[3] obda.open-bio.org

commercial object-relational database management system.

Experiments on restricted IND candidates In our first class of experiments we only create IND candidates with an unique referenced attribute, such that we can infer foreign keys. Second, we exclude intra-table references. Third, we prefiltered the IND candidates by the number of their distinct values: If the number of distinct values in the dependent attribute is larger than the number of distinct values in the referenced attribute, then there is at least one dependent value that is not contained in the referenced attribute. Thus, the IND is unsatisfied.

Results can be seen in Table 1. Brute Force and SPIDER outperform the fastest SQL approach (join). For low numbers of IND candidates and small data sets, i. e., UniProt and TPC-H, there is just a small difference between Brute Force and SPIDER. For large schemas with high numbers of IND candidates the improvement of SPIDER over Brute Force is considerable.

	UniProt	TPC-H	PDB	
DB size	667 MB	1.3 GB	2.7 GB	21 GB
# IND cand.	910	477	139,356	158,432
# INDs	36	33	30,753	34,988
join	15 m 03 s	29 m 22 s	> 7 days	–
Brute Force	2 m 01 s	6 m 56 s	3 h 13 m	20 h 21 m
SPIDER	1 m 38 s	6 m 40 s	19 m 06 s	3 h 52 m

Table 1. Run-time performance on testing restricted IND candidates.

We compared SPIDER to the approaches of Bell and Brockhausen [3] and Marchi et al. [6] (described in Sec. 1) using our own implementation. Those approaches only test candidates of same data types. In our life sciences setting, we often find schemas with only `string` attributes and thus must test all pairs of attributes. For a fair comparison we assigned the same data type to all attributes. The algorithm provided by Bell and Brockhausen [3] runs 4 m 39 s on UniProt data, which is three times slower than SPIDER. On the 2.7 GB fraction of PDB it did not finish within 24 hours. Marchi et al. proposed to preprocess all data values and test all IND candidates afterwards in parallel. However, the preprocessing on UniProt already takes 9 h 45 m (the actual IND test only takes 2 m 10 s).

Experiments on unrestricted IND candidates In the second class of experiments we test all IND candidates, i. e., without restricting IND candidates to have unique referenced attributes. The results are given in Table 2. In comparison to Table 1 the runtime increases only slightly, although many more INDs are tested and satisfied. This implies that more attribute values have to be handled, because attributes cannot be excluded from the SPIDER heap as early. For PDB there is the additional effect that the much larger number of INDs has to be saved. Thus, SPIDER is applicable for very large databases with huge numbers of IND candidates.

	UniProt	TPC-H	PDB	
DB size	667 MB	1.3 GB	2.7 GB	21 GB
# IND cand.	2,235	1,616	729,674	831,984
# INDs	116	86	84,232	99,669
SPIDER	1 m 45 s	6 m 55 s	23 m 27 s	4 h 05 m

Table 2. Run-time performance on testing unrestricted IND candidates.

4. Conclusion

We described the SPIDER algorithm for detecting inclusion dependencies in an RDBMS with no previously known schema information. The algorithm is divided into two phases. The first phase leverages optimized sort operations of the DBMS but avoids the constraints of SQL. The second phase tests all IND candidates in parallel such that all data values are read only once and tests are stopped early. We showed its superiority to other approaches by experiments on different data sets. SPIDER is the only method that makes feasible the detection of INDs in databases with large numbers of attributes and data values.

Acknowledgments. This research was supported by the German Ministry of Research (BMBF grant no. 0312705B) and by the German Research Society (DFG grant no. NA 432).

References

[1] A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. Martin, D. Natale, C. O'Donovan, N. Redaschi, and L. Yeh. The universal protein resource (UniProt). *Nucleic Acids Res*, 33(Database issue):D154–9, 2005.

[2] J. Bauckmann, U. Leser, and F. Naumann. Efficiently computing inclusion dependencies for schema discovery. In *Second International Workshop on Database Interoperability. In Workshop-Proceedings of the ICDE 06*, 2006.

[3] S. Bell and P. Brockhausen. Discovery of data dependencies in relational databases. In Y. Kodratoff, G. Nakhaeizadeh, and C. Taylor, editors, *Statistics, Machine Learning and Knowledge Discovery in Databases, ML–Net Familiarization Workshop*, pages 53–58, 1995.

[4] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242, 2000.

[5] M. Kantola, H. Mannila, K.-J. Rih, and H. Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7:591–607, 1992.

[6] F. D. Marchi, S. Lopes, and J.-M. Petit. Efficient algorithms for mining inclusion dependencies. In *8th International Conference on Extending Database Technology (EDBT '02)*, pages 464–476. Springer-Verlag, 2002.