Discovery and Application of Check Constraints in DB2

Jarek Gryz
Department of Computer Science
York University, Toronto
jarek@cs.yorku.ca

Berni Schiefer Jian Zheng Calisto Zuzarte
IBM Toronto
{schiefer, jerryz, calisto}@ca.ibm.com

Abstract

The traditional role of integrity constraints is to protect the integrity of data. But integrity constraints can and do play other roles in databases; for example, they can be used for query optimization. In this role, they do not need to model the domain; it is sufficient that they describe regularities that are true about the data currently stored in a database. In this paper we describe two algorithms for finding such regularities (in the syntactic form of check constraints) and discuss some of their applications in DB2.

1. Introduction

The traditional role of integrity constraints is to protect the integrity of data. To serve this purpose, integrity constraints are specified by a DBA and all updates are verified against them. But integrity constraints can and do play other roles in databases. In particular, they can be used for query optimization [15, 4, 6]. In this role, they do not need to model the domain; it is sufficient that they describe regularities that are true about the data currently stored in a database. Such discovered constraints may have the same syntactic form as traditional integrity constraints, and a similar meaning: they describe what is true in a database in its current state, as do integrity constraints. But unlike integrity constraints, they can be invalidated by updates. Since the discovered constraints can be used for many applications (besides query optimization) in the same way as the traditional integrity constraints, they can greatly enhance the use of such applications.

Although the discovered constraints describe regularities of the stored data, some of them may also characterize correctly the real world modeled by the database. Thus, the status of such constraints can be upgraded - most likely through human intervention - to the level of traditional (hard) integrity constraints. The remaining constraints can be divided into two groups, depending on their maintenance strategy. In some environments, such as data warehousing, data loading is strictly controlled and many discovered constraints can be safely assumed never to be inconsistent with the stored data. They need not be maintained. We call these constraints *informational*. The rest of the discovered constraints (which we call *soft constraints*) require some kind of maintenance strategy to be useful. Their maintenance can be immediate or deferred. With immediate maintenance, a constraint is modified/removed immediately after an update is found to be inconsistent with the constraint. With deferred maintenance, the correctness of a constraint is not verified immediately after each update. In this paper, we describe a method developed at IBM for the discovery of check constraints in DB2 and present examples of the novel applications of such constraints. The paper is organized as follows. Section 2 presents two algorithms for the discovery of check constraints. In Section 3, we describe briefly a few results of applying the algorithms on TPC-H benchmark. The applications of the constraints are presented in Section 4. Summary and related work and are in Section 5.

2 Discovery of Check Constraints

In this section we present two algorithms for the discovery of check constraints over pairs of attributes.

2.1 Correlations Between Attributes Over Ordered Domains

For the applications that we considered, we were interested in correlations between two attributes such that the value (or a range of values) of the first attribute determines a range of values of the second attribute. When the domains of both attributes are ordered, the correlation can be expressed as a linear relationship between the attributes in rules of the form:

$$Y = bX + a + [emin, emax]$$

This expression means:

$Y > bX + (a + emin)$ and $Y < bX + (a + emax)$

where X and Y are attribute values, a and b are constants, and $emin$ and $emax$ define the error, that is the expected range of Y for a given value of X. The algorithm for discovery of the constraints described above uses the simplest, but in many cases the most useful, linear regression statistic model, Ordinary Least Square (OLS) [8]. The algorithm takes a database and the desired error as input. It has the following structure:[1]

Algorithm 1

```
for all tables in the database
    for all comparable variable pairs
        (X and Y) in the table
        - apply OLS estimation to get
          function of the form:Y = bX+a
        - calculate max and min error:
          emax and emin
    endfor
endfor
```

2.2 Partitioning

The main drawback of the OLS algorithm is that it works only on numeric values from ordered domains, not on string or character variables (unless, of course, they encode numeric values). Thus, we designed another algorithm that partitions a range of a numeric attribute based on the values of a non-numeric attribute. The idea of the algorithm stems from data classification. The rules have the following form:

If $X = a$, then Y BETWEEN $emin$ AND $emax$

That is, for any pair of variables X and Y in the table, we classify (that is, group by) X into partitions, and calculate the new range of Y for each partition.

Algorithm 2

```
for all tables in the database
    for any qualifying variable pair
        (X and Y) in the table
        - calculate partitions using
          GROUP BY X statements
        - find the max and min value
          of Y for each partition
    endfor
endfor
```

A pair of attributes X and Y is considered by the algorithm (line 2 of Algorithm 2), when it satisfies some additional conditions (for example, we require that the number of distinct values in the domain of X is small).

[1]We only select comparable variable pairs from the database, that is, numeric vs numeric, numeric vs date, date vs date, where the numeric values come from an ordered domain

2.3 Measurement of the rule quality

One of the inputs for the first algorithm described above is the desired error (*random disturbance* in OLS terminology) in the discovered rules. By specifying the error, we are able to limit the number of rules generated by the algorithm; for example, setting the error to the size of the domain of Y would generate correlation rules for all arbitrary pairs of attributes (clearly, such rules would not be interesting). Indeed, the error must be a key component of any measure of the quality of the discovered rules. The most straightforward quality measure would be a comparison of an error (defined as emax minus emin) versus the size of the domain of Y. For example, if $Y = [0, 1000]$ in the database, and the discovered rule is: $Y > X + 100$, $Y < X + 600$, we can narrow the range of Y to 500 for any given value of X. If we define m = error/domain range, m is equal to 0.5 in this case. The quality measure for the second algorithm is similar to the one we used for the OLS algorithm; we require additionally that the number of distinct values in the domain of X is small. We found the quality measure described above particularly useful for mining rules used for semantic query optimization (described in Section 4.3). More complex measures are employed for rules used for cardinality estimates (described in Section 4.2).

3 Experiments in TPC-H

We ran a number of experiments using the algorithms in both real and synthetic databases. We describe several interesting rules discovered by the algorithms in the **Lineitem** table of the TPC-H benchmark [23]. For all the rules reported below, m refers to the quality measure described above. There is only one single table check constraint[2] defined in TPC-H over the **Lineitem** table: L_SHIPDATE < L_RECEIPTDATE. This constraint has been discovered by the OLS algorithm as the following rule:

- L_RECEIPTDATE = L_SHIPDATE + (1, 30), m = 0.0114.

The rule discovered by the algorithm is of better quality than the defined check constraint since it restricts the minimum and maximum values of the L_SHIPDATE given the value of L_RECEIPTDATE. Two other rules with very small value of m have been discovered by the algorithm. They are:

- L_COMMITDATE = L_RECEIPTDATE + (-119, 88), m = 0.084

[2]There are also two multitable check constraints involving the Lineitem table. However, we did not run the algorithm over joins of tables in TPC-H, hence the rules were not discovered by the algorithm.

and

- L_COMMITDATE = L_SHIPDATE + (-91, 89), m = 0.073

Both of these rules define a very tight correlation between L_COMMITDATE and L_RECEIPTDATE, and between and L_COMMITDATE and L_SHIPDATE which could not be accidental. Indeed, these relationships are enforced in the data-generating algorithm of TPC-H which is specified in Clause 4 [23]. The partitioning algorithm also generated a number of interesting rules. The following two rules describe a perfect correlation between the value of the LINESTATUS attribute and the SHIPDATE attribute.

- If L_LINESTATUS=F, then L_SHIPDATE = (01/04/1992, 06/17/1995), m = 0.50

- If L_LINESTATUS=O, then L_SHIPDATE = (06/19/1995, 12/25/1998), m = 0.50

These rules state that two types of lineitems, namely F and O, are divided by a specific date, that is, all F lineitems were shipped before that date and all O lineitems were shipped after that. This kind of observation might be very valuable from a data mining perspective.

Numerous rules describing weaker correlations were discovered. The following three rules describe a correlation between the RETURNFLAG attribute and the COMMITDATE attribute:

- If L_RETURNFLAG=A, then L_COMMITDATE = (02/01/1992, 08/15/1995), m = 0.52

- If L_RETURNFLAG=N, then L_COMMITDATE = (03/03/1995, 10/31/1998), m = 0.54

- If L_RETURNFLAG=R, then L_COMMITDATE = (01/31/1992, 08/22/1995), m = 0.53

[Comment: there will be a summary of the results from GWL added here]

4 Applications

4.1 DBA Wizard

Some of the discovered soft constraints correctly model the domain of a database and as such can be defined in a database as traditional integrity constraints. The algorithms described here will become part of a DBA wizard, a tool that will advise a DBA on a number on design issues.

4.2 Cardinality Estimation

Estimating cardinality of intermediate results in query evaluation is critical for designing a good access plan by query optimizer. However, such estimates are often unreliable because of existing correlations (unknown to the optimizer) between columns referenced in a query. We are able to extract such correlations from the check constraints discovered by the algorithms described above, thus providing dramatic improvements in cardinality estimates for results of the select operation. We developed two techniques for transforming a query into a form that allows the optimizer to estimate the cardinality of an intermediate table more accurately: Pseudo Predicate Transformation and Pseudo Predicate Elimination. We illustrate the first technique with an example. Consider a complex query issued against a hotel database, that among other things, requests the number of guests staying in the hotel on a given date. Typically, to obtain the actual number, the query would involve a predicate in one of these forms:

1. '1999-05-15' BETWEEN ARRIVAL_DATE AND DEPARTURE_DATE

or:

2. ARRIVAL_DATE $\leq$ '1999-06-15' AND DEPARTURE_DATE $\geq$ '1999-06-15'

Since this is part of a larger and more complex query, a good cardinality estimate is desirable, so that the database optimizer can choose an efficient access plan. More often than not, the ARRIVAL_DATE and the DEPARTURE_DATE columns are treated by the optimizer as "independent" columns for the purpose of cardinality estimation. This might involve multiplying probability or filter factor of the predicate (ARRIVAL_DATE $\leq$ '1999-06-15') and (DEPARTURE_DATE $\geq$ '1999-06-15). That is, without any correlation information about the values in the two columns, DB2 would compute ff, the filter factor of the conjunction of the two predicates as the product of their individual filter factors:

3. $ff = ff_1 * ff_2$

For example, if the date was approximately midway in the date ranges, we would estimate a quarter of all the guests that came in over the number of years stayed in the hotel at that date! One solution might be to keep histogram statistics on the combinations of values. This could be expensive and space consuming. A safe "hard" constraint (DEPARTURE_DATE $\geq$ ARRIVAL_DATE) cannot be used effectively to enhance the query to be able to improve on the cardinality estimates. Assume the mining of the soft check constraints yielded:

4. DEPARTURE_DATE $\geq$ ARRIVAL_DATE + (1 DAY, 5 DAYS)

In other words all guests stayed between 1 and 5 days. The database can use this information internally just for the cardinality estimation process by "twinning" or "replacing" the DEPARTURE_DATE predicate in (2) by an approximate predicate on ARRIVAL_DATE. A statistical probability of the average number of days stayed would yield a better estimate. However, considering the relationship DEPARTURE_DATE $\geq$ ARRIVAL_DATE + 5 DAYS, the "twin" predicate would be (ARRIVAL_DATE + 5 DAYS $\geq$ '1999-06- 10').

So for cardinality estimation purposes, we have:

5. ARRIVAL_DATE $\geq$ '1999-06-10' AND ARRIVAL_DATE $\leq$ '1999-06- 15'

In the BETWEEN form, this is:

6. ARRIVAL_DATE BETWEEN '1999-06-10' AND '1999-06-15'

BETWEEN specified in (6) is a common predicate on a column; we can give much better cardinality estimates in this situation. If ff_1 and ff_2 are the filter factors of the individual range predicates in (5), the overlap gives us a much better filter factor of the combination:

7. $ff = (ff_1 + ff_2 - 1)$

Using the "5-day" estimate would be significantly closer to the actual number than the estimate obtained using formula (3). We note that the constraints used for cardinality estimates need only be approximately true (that is, soft constraints with deferred maintenance strategy are acceptable). Hence, we do not have to pay a potentially high cost for maintaining such constraints.

4.3 Semantic Query Optimization

Semantic query optimization (SQO) uses integrity constraints associated with the database to improve the efficiency of query evaluation. One of the SQO techniques that relies primary on check constraints is Predicate Introduction (PI). The PI was discussed in literature as two different techniques: index introduction and scan reduction. The idea behind the index introduction is to add a new predicate to a query if there is an index on the attribute named in the predicate [15, 4, 6]. Consider the following query:

```
select sum(l_extendedprice * l_discount) as revenue
from   tpcd.lineitem
where  l_shipdate > '1994-01-01' AND
       l_shipdate < '1994-01-01' + 1 year AND
       l_discount BETWEEN .06 - 0.01 AND .06 + 0.01
       AND l_quantity < 24;
```

Since the check constraint, L_SHIPDATE $<$ L_RECEIPTDATE, has been defined for TPC-H, a new predicate, "L_RECEIPTDATE $>$ can be added to the where clause in the query without changing its answer set. Now, if the only index on **Lineitem** table is a clustered index with the search key L_RECEIPTDATE, a new, potentially more efficient evaluation plan is available for the query. Indeed, we showed in [6] that, with appropriate restrictions, PI leads to dramatic query performance improvements.

The addition of soft check constraints to the repertoire of semantic information available to the optimizer can enhance the power of PI even further. Consider a variant (described in Section 3) of the check constraint used in the query above: L_RECEIPTDATE = L_SHIPDATE + (1, 30). Now, yet another predicate: "L_RECEIPTDATE $<$ '1994-01-31' + 1 year" can be added to the query. This narrows the index range of index search for qulaifying tuples even further.

Besides PI, we have developed several other SQO techniques that perform query rewrites using check constraints. An SQO prototype has been implemented in DB2.

5 Summary and Related Work

Extracting semantic information from database schemas and contents, often called rule discovery, has been studied over the last several years. Rules can be inferred from integrity constraints [3, 2, 24] or can be discovered from database content using machine learning or data mining approaches [5, 7, 10, 21, 22, 24]. It has also been suggested that such rules be used for query optimization [11, 21, 22, 24] in a similar way that traditional integrity constraints are used in semantic query optimization [4, 15, 6]. Many algorithms for mining functional dependencies, which can be considered a special type of check constraints, have been developed over the last years [12, 1, 17, 20].

A lot of work has been devoted to the problem of estimating the size of the result of a query expression. Approaches based on sampling were explored in [9, 16] and histograms [13, 19]. [18] provides a survey of several techniques and [14] provides an analysis of error propagation in size estimation. Although the information about keys is often used in query result estimates, we are not aware of the use of check constraints for that purpose. In this paper we described two algorithms for finding regularities (in the form of check constraints) in data and the application of such regularities in DB2. We provided several examples of rules generated by the algorithms in TPC-H benchmark. We also described how these rules could be utilized for query optimization and for more accurate estimate of the query answer results.

Acknowledgements:

DB2 is a trademark of IBM Corporation.

TPC-H is a trademark of Transaction Processing Council.

References

[1] D. Bitton, J. Millman, and S. Torgersen. A feasibility and performance study of dependency inference. In *Proceedings of the Fifth ICDE, February 6-10, 1989, Los Angeles, California, USA*, pages 635–641. IEEE Computer Society, 1989.

[2] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Automatic generation of production rules for integrity maintenance. *TODS*, 19(3):367–422, 1994.

[3] S. Ceri and J. Widom. Deriving production rules for constraint maintanance. In *Proc.* 16^{th} *VLDB*, pages 577–589, 1990.

[4] U. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM TODS*, 15(2):162–207, June 1990.

[5] I.-M. A. Chen and R. C. Lee. An approach to deriving object hierarchies from database schema and contents. In *Proceedings of the 6th ISMIS*, pages 112–121, 1991.

[6] Q. Cheng, J. Gryz, F. Koo, C. Leung, L. Liu, X. Qian, and B. Schiefer. Implementation of two semantic query optimization techniques in DB2 UDB. In *Proc. of the 25th VLDB*, pages 687–698, Edinburgh, Scotland, 1999.

[7] W. Chu, R. C. Lee, and Q. Chen. Using type inference and induced rules to provide intensional answers. In *Proceedings of the 7th ICDE*, pages 396–403, 1991.

[8] W. H. Greene. *Econometric Analysis*. Prentice Hall, 1999.

[9] P. Haas, J. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of VLDB*, pages 311–322, 1995.

[10] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proceedings of the 18th VLDB*, pages 547–559, 1992.

[11] C. N. Hsu and C. A. Knoblock. Using inductive learning to generate rules for semantic query optimization. In *Advances in Knowledge Discovery and Data Mining*, pages 425–445. AAAI/MIT Press, 1996.

[12] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings of 14th ICDE*, pages 392–401, Orlando, FL, Feb. 1998.

[13] Y. E. Ioannidis. Universality of serial histograms. In *Proceedings of 19th VLDB*, pages 256–267, 1993.

[14] Y. E. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of the join results. *TODS*, 1993.

[15] J. King. Quist: A system for semantic query optimization in relational databases. *Proc. 7th VLDB*, pages 510–517, Sept. 1981.

[16] R. Lipton, J. Naughton, and D. Schneider. Practical selectivity estimation through adaptive sampling. In *Proc. of Sigmod*, pages 40–46, 1990.

[17] H. Mannila and K.-J. Raiha. Algorithms for inferring functional dependencies from relations. *Data and Knowledge Engineering*, 12(1):83–89, 1994.

[18] M. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database. *ACM Computing Surveys*, 20(3):191–221, 1988.

[19] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of SIGMOD*, pages 294–305, 1996.

[20] I. Savnik and P. Flach. Bottom-up induction of functional dependencies from relations. In G. Piatetsky-Shapiro, editor, *Knowledge Discovery in Databases*, pages 284–290. Morgan Kaufman Pub., 1993.

[21] S. Shekar, B. Hamidzadeh, A. Kohli, and M. Coyle. Learning transformation rules for semantic query optimization. *TKDE*, 5(6):950–964, Dec. 1993.

[22] M. Siegel. Automatic rule derivation for semantic query optimization. In *Proceedings of the 2nd International Conference on Expert Database Systems*, pages 371–386, 1988.

[23] Transaction Processing Performance Council, 777 No. First Street, Suite 600, San Jose, CA 95112-6311, www.tpc.org. *TPC Benchmark*^{TM} *D*, 1.3.1 edition, Feb. 1998.

[24] C. T. Yu and W. Sun. Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):362–375, Sept. 1989.