Representing and Querying Conceptual Graphs with Relational Database Management Systems is Possible

Gunther Schadow MD PhD, Michael R. Barnes MD, Clement J. McDonald MD

Regenstrief Institute and Indiana University School of Medicine, Indianapolis, IN

This is an experimental study on the feasibility of maintaining medical concept dictionaries in production grade relational database management systems (RDBMS.) In the past, RDBMS did not support transitive relational structures and had therefore been unsuitable for managing knowledge bases. The revised SQL-99 standard, however, may change this. In this paper we show that modern RDBMS that support recursive queries are capable of querying transitive relationships in a generic data model. We show a simple but efficient indexed representation of transitive closure. We could confirm that even challenging combined transitive relationships can be queried in SQL.

Introduction

Conceptual graphs are an approach to organize categorical knowledge about concepts. Terminologies, ontologies, and conceptual graphs are quite active areas of research in medical informatics. Many research systems to manage these conceptual graphs (e.g., GRAIL[1], PROTÉGÉ-II[2], LUME[3]) are stand-alone systems (often implemented in LISP) that support expressive description logics and inference, but that have limited data management features. These systems mostly depend on the entire conceptual graph to be loaded into memory. Their innovativeness and power notwithstanding, these specialized research systems do not blend very well into a production data management system.

Relational Database Management Systems (RDBMS) on the other hand are a theoretically sound and industrially proven technology for managing large amounts of shared data,[4,5] but RDBMS have not been used for managing knowledge bases. This is because most RDBMS do not support transitive closure queries, which are an indispensable tool for managing concept networks.[6] However, RDBMS are catching up and it is time to revisit the practicality of maintaining extensive medical knowledge in RDBMS.

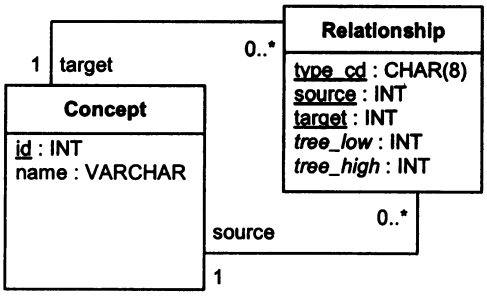

Figure 1: Example generic data model.

In this paper, we outline an approach for managing conceptual graphs in an industry supported RDBMS. As we are currently migrating the Regenstrief Medical Record System[7] (RMRS) and its medical concept dictionary to industry standard relational data architecture, this work is a study in the possibilities and limitations of RDBMS meeting the needs of generic data modeling and conceptual graphs. The RMRS dictionary contains approximately 25000 concepts, with approximately 1000 concepts added every year since 1978. In addition we maintain the rapidly growing Logical Observations Identifiers Names and Codes[8] (LOINC) and we are referring to external terminologies, such as ICD-9, SNOMED, and Medispan GPI for use in our medical record system. In addition, having been influential in transforming the HL7 Reference Information Model (RIM) to an abstract generic data model,[9,10] we are investigating the feasibility and costs of straightforwardly implementing this model in an RDBMS.

Example Problems

Throughout this paper we will use an example database defined by the following SQL data definition (see also Figure 1):

```
CREATE TABLE Concept (
  id   INTEGER NOT NULL PRIMARY KEY,
  name VARCHAR(64),
  text LONG VARCHAR);

CREATE TABLE Relationship (
  type_cd   CHAR(8) NOT NULL,
  source    INT NOT NULL
              REFERENCES concept(id),
  target    INT NOT NULL
              REFERENCES concept(id),
  CONSTRAINT pk PRIMARY KEY (
              type_cd, source, target));
```

This data model pattern underlies many generic databases found in the literature, including the HL7 RIM. Databases according to this model are directed graphs with labeled nodes and arcs. In our example we call the node "Concept" and the arc "Relationship".

Attributes as Relationships

The generic relationship-table handles all relationships between concepts, including those simple n-to-1 rela-

tionships that could be represented as direct attributes (self-referential foreign keys) in the concept table without using the relationship table.

The advantage of always using the link table is that we can query our concept database for any kind of relationships between concepts. Thus, we can isolate concepts in the database that are never used (and eliminate those concepts), or we can focus the maintenance effort on those concepts that are most frequently referenced.

Transitive Closure Queries

The generalization relationship ('ISA') is probably the most commonly used relationship in concept databases. One common use case for querying ISA links is to test whether a given concept matches an expected concept. This requires querying the ISA relationship between the two concepts.

The ISA relationship however is transitive. For example, if gastric mucosa ISA columnar epithelia, and columnar epithelia ISA epithelia, the gastric mucosa ISA$^+$ epithelia (we note transitive relationships using a superscript plus ($^+$) after the relationship type symbol.)

With the SQL-92 standard,[11] implemented by most commercial RDBMS today, one cannot query the complete transitive closure. Instead one would either write complex queries that allow a transitive chain up to a certain length, or one would need to write a program that would walk the links to form the transitive closure. Programmatically following links is not only cumbersome (requiring programming instead of just ad-hoc queries,) it is also slow since many queries and their responses need to be communicated between the database client and server.

In the new standard SQL-99[12] (also known as "SQL 3") transitive closure can be queried using named temporary tables. For example, to find out if one concept (*spec*) is the transitive specialization of another concept (*gen*) we construct the successor set of the transitive ISA relationship and test whether *gen* is in that successor set:

```
WITH tmp(id) AS (
    VALUES spec
  UNION ALL
    SELECT next.target
      FROM tmp, Relationship AS next
     WHERE next.type_cd = 'ISA'
       AND next.source  = tmp.id
) SELECT * FROM tmp WHERE id = gen;
```

Transitive closure queries in SQL-99 use the WITH-clause that defines a temporary table (here named "tmp") as the union of two queries. The first part constructs a starter set, while the second part joins the current temporary table with other tables to add more items into the temporary table. This process is repeated until no more items to are found to join. The final query takes the completed temporary table and answers the original question – in this case whether *gen* is among the generalizations of *spec*.

This example shows that when querying the relationship table, a join with the concept table itself is normally not needed. Thus, the slight disadvantage that we found when modeling attribute links with the relationship table is of no concern if we perform extensive graph traversal.

Inheritance

The ISA link logically implies that the specialization inherits all relationships from its generalizations. To find all the transitively inherited relationships of a concept *spec* we construct the successor set of the ISA$^+$ relationship and then join with the other relationships:

```
WITH tmp(id) AS (
    VALUES spec
  UNION ALL
    SELECT next.target
      FROM tmp, Relationship AS next
     WHERE next.type_cd = 'ISA'
       AND next.source  = tmp.id
) SELECT link.*
    FROM tmp, Relationship AS link
   WHERE link.source = tmp.id;
```

Besides ISA relationships, other important transitive relationships exist, such as PART-OF (e.g., the aortic valve is PART-OF the heart and the heart is PART-OF the cardiovascular system, hence is the aortic valve PART-OF$^+$ the cardiovascular system.) The causal link (CAUSES) is also transitive ("chain of causality".)

Transitive Closure Optimization

Transitive closure queries are expensive, even with support from an SQL-99 compliant RDBMS and for online applications one must consider techniques for optimization. Many transitive closure algorithms had been described in the recent years,[13] yet most of these algorithms are most sensibly implemented in the core of the RDBMS. As a client-side program, performance of these algorithms is severely impaired by the fact that all of those algorithms have to read the entire relationship table at least once, and this is slowed down by both disk access and network traffic.

Materialized Transitive Closure

One way of optimizing transitive closure is to materialize all transitive relationships in an "ancestor-descendant-table." To do this, we would add a column to our relationship table distinguishing a direct relationship from a cached transitive relationship:

```
ALTER TABLE Relationship
  ADD COLUMN trans_ind CHAR;
```

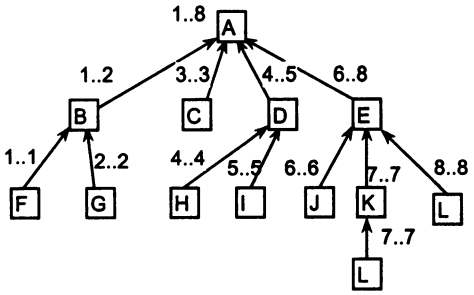

Figure 2: The interval method of transitive closure optimization allows testing for the transitive relationship between L and E by testing whether the interval 7..7 is in the interval 6..8.

If the trans_ind value is '+' we know that the link is materialized transitive link. One could update the transitive links either on a regular basis or using a database trigger that computes the transitive closure whenever a new relationship is inserted or an existing relationship deleted. One can build the initial transitive closure table using a variant of the query above:

```
INSERT INTO Relationships
  WITH tmp(type_cd, source, target,
           trans_ind) AS (
      SELECT Relationship.*, ''
        FROM Relationship
       WHERE type_cd IN
     (VALUES 'ISA', 'PART-OF', 'CAUSES')
    UNION ALL
      SELECT next.type_cd, next.source,
             next.target,'+'
        FROM tmp, Relationship AS next
       WHERE next.type_cd = tmp.type_cd
         AND next.source  = tmp.target
  ) SELECT * FROM tmp;
```

The materialized representation is simple, and queries written against the simple relationship table will continue to work against the transitive relationship table. The disadvantage, however, is that the materialized transitive closure can grow very large and may be impractical with large concept databases.

Tree Addresses

Many hierarchical coding systems use tree addresses to encode transitive relationships (e.g., HELP, MeSH, or ICD-9.) A tree-address is an encoding of the path leading from the root to the so addressed node. For example, node "H" in **Figure 2** would have a tree address like "A.D.H", which immediately reveals that "H" is a descendent of "A".

ISO Object Identifiers (OID) or IP addresses are the two prototypical examples of tree-address encodings: OIDs are simply dotted lists of numbers with no limitation to the size of each number or the length of the list. IP addresses are fields of 32 bit, where the bits are grouped to address each step in the path. Such bit-fields are efficient to compare and operate on but allow only a limited depth and limited breadth in each level of the graph. On the other hand, OID type addresses are less efficient to operate on and can easily grow quite long.

The Interval Method

The interval method assigns an integer-interval to each link in the graph. For simplicity we require the graph to be a proper tree, i.e., there must be a single root and besides the root every node in the tree must have exactly one outgoing link. A more general interval method exists that works for all graphs,[13,14] but with the tree assumption the representation and algorithms become very simple and practically useful.[15]

We can then assign an integer interval to all the links in the tree as shown in Figure 2. Each link's interval is the union of the intervals of that link's successor set.* We cannot store the intervals in the concept node itself because different intervals are assigned to each node, one interval for each transitive link type. To support the interval method, we add two columns to the link table

```
ALTER TABLE Relationship
  ADD COLUMN tree_low  INTEGER
  ADD COLUMN tree_high INTEGER;

CREATE INDEX ON
  Relationship(tree_low, tree_high);
```

The transitive closure query to find all the generalizations of a concept *x* is then as simple as:

```
SELECT gen.*
 FROM Relationship spec,
      Relationship gen
 WHERE spec.type_cd   = 'ISA'
   AND gen.type_cd    = spec.type_cd
   AND spec.source    = x
   AND gen.tree_low  <= spec.tree_low
   AND gen.tree_high >= spec.tree_high;
```

This query can be answered rapidly using only two index scans: one quick lookup to find the first ISA link for the concept *X* followed by an index scan to find only the concepts which include the given concept's interval.

Discussion

The interval method is a very efficient transitive closure optimization, because unlike tree addresses the intervals' storage requirements depend only on the number of nodes in the graph, independent of the depth or balance issues. Both tree addresses and the interval method have some limitations. First, the graph must be a proper tree with a single root. Completing a poly-tree ("forrest") structure to a singly rooted tree is easy, we just add a root concept (with id = –1). For each transi-

* An algorithm for assigning the intervals is described by Kamfonas[15], we found a way to assign the intervals using only SQL statements but we still require an explicit iteration.

tive relationship type we link all concepts that are not the source of such relationship directly to this root node. We require these root links to store the interval numbers for the concepts at the source of each link.

In the simple form shown here, the interval method does not work for cyclic graphs and it does not support, multiple outgoing links per concept (no multiple inheritance.) For concept relationships, we believe multiple inheritance to be much less valuable than it might seem from the many uses of the ISA relationship reported in the literature. For example, it is tempting to define *heart* ISA *cardiovascular organ* when in fact what's being said is that the *heart* is PART-OF the *cardiovascular system*.

We think the ISA link should be restricted to expressing only substantial relationships defined by the very nature of primary concepts. To that end, concepts that essentially are groupings of other concepts should not be defined as abstract concepts at all. For example, one should define the *cardiovascular system* (concrete concept) but not *cardiovascular organ* (abstract concept), whose only definition is: an *organ* that is PART-OF the *cardiovascular system*. The same argument can be made for diseases; instead of defining *myocardial infarction (M.I.)* ISA *cardiovascular disease*, one should say that *M.I.* is CAUSED-BY *ischemia* of the *myocardium*, which is PART-OF the *heart*, etc. Interestingly, while the anatomy section of SNOMED RT is quite well designed regarding our criterion, the disease section unnecessarily uses the ISA relationship; e.g., *M.I.* ISA *ischemic heart disease*.

While one can create arbitrary classifications of concepts based on any attribute or relationship, the ISA link should be reserved for substantial relationship only. Hence, we do not believe that abandoning the option for multiple inheritance is a great loss. When we encounter a concept with two ancestors, we ask whether this is due to arbitrary classification. If both relationships appear substantial, we ask whether the two ancestor concepts might themselves be related (thus the multiple ISA links might be a materialized transitive relationship.)

Interactions between Relationships

Different kinds of relationship types may interact with each other. As noted earlier all relationships are inherited through ISA links. Sometimes, however, relationships are conducted through other links. Horrocks et al.[16] give the example of the HAS-LOC[ation] link that transfers through PART-OF links (e.g., if fracture of femur-shaft HAS-LOC femur-shaft and femur-shaft is PART-OF femur, then a femur-shaft-fracture HAS-LOC femur.) The authors go on to propose that relationships can be chained. For example (HAS-LOC ° PART-OF$^+$) is the composed relationship that expresses this rule of the conducted relationship.

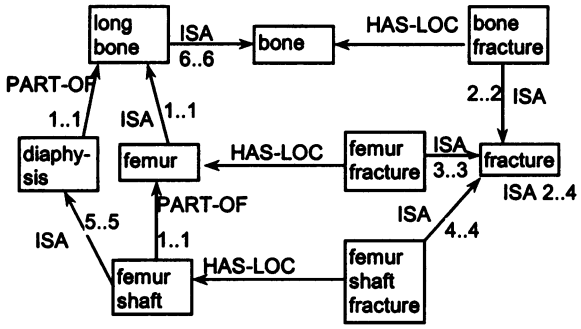

Figure 3:.Example conceptual graph with multiple interacting relationship types.

To test for such composite relationships in our representation we combine the intransitive HAS-LOC relationship with the transitive closure query for the transitive PART-OF relationship and the inheritance through ISA links. Given the conceptual graph in Figure 3 to test whether a femur-shaft-fracture (*fsf*) is a fracture (*frac*) of the femur (*f*) we have to find whether *fsf* ISA$^+$ *frac* and *fsf* (HAS-LOC ° PART-OF$^+$) *femur*, i.e., that there exist two concepts *x* and *y* where *fsf* ISA$^+$ *x* which HAS-LOC *y* which is PART-OF$^+$ *femur*.

```
SELECT Px.*, Py.*
  FROM Relationship Ifsf,
       Relationship Ifrac,
       Relationship Ix,
       Relationship Lx,
       Relationship Py,
       Relationship Pf,
 WHERE Ifsf.type_cd  = 'ISA'
   AND Ifsf.source   = fsf
   AND Ifrac.type_cd = 'ISA'
   AND Ifrac.source  = frac
   AND Ifsf.tree_low >= Ifrac.tree_low
   AND Ifsf.tree_high<= Ifrac.tree_high
   AND Ix.type_cd    = 'ISA'
   AND Ifsf.tree_low >= Ix.tree_low
   AND Ifsf.tree_high<= Ix.tree_high
   AND Lx.type_cd    = 'HAS-LOC'
   AND Lx.source     = Ix.source
   AND Py.type_cd    = 'PART-OF'
   AND Py.source     = Lx.target
   AND Pf.type_cd    = 'PART-OF'
   AND Pf.source     = f
   AND Py.tree_low   >= Pf.tree_low
   AND Py.tree_high  <= Pf.tree_high);
```

This query is a join of the same relationship table repeated six times. The complexity however is not as high as it seems, since all but *x* and *y* are bound to specific concepts, so the search space is not large and all searches are efficient index lookups.

Consider the question "is femur-shaft-fracture a fracture of a bone?" The first part of the question is answered as shown above, i.e. *fsf* ISA$^+$ *frac* and *fsf* ISA$^+$ *x* where *x* HAS-LOC *y*. However, answering whether *y*

is part of a bone requires the transitive closure over the combination of at least one PART-OF link and optional ISA links in any order. For example, possible paths between *y* and *bone* could be "y ISA$^+$ diaphysis PART-OF$^+$ long-bone ISA$^+$ bone"; and "y PART-OF$^+$ long-bone ISA$^+$ bone." This query cannot be answered in a simple join but requires recursion again:

```
WITH tmp AS (
  SELECT * FROM Relationship
   WHERE type_cd IN ('ISA', 'PART-OF')
     AND source = y
 UNION ALL
  SELECT link.*
    FROM tmp, Relationship this,
              Relationship next
   WHERE this.type_cd IN
                      ('ISA', 'PART-OF')
     AND this.source    = tmp.target
     AND next.type_cd   = this.type_cd
     AND next.tree_low <= this.tree_low
     AND next.tree_high>= this.tree_high
) SELECT * FROM tmp WHERE target = bone;
```

The above query is only a sketch of the working query; in reality we need a longer query statement preventing infinite recursion and testing whether the path really contains at least one PART-OF link.[17]

Conclusion

We have shown that modern RDBMS that support recursive queries are capable of querying certain conceptual graphs structures with only reasonable restrictions. We have further shown a simple but efficient indexed representation of transitive closure. The restrictions on the graph structures (tree-requirement) are no severe shortcoming, in part because the total graph is partitioned per each relationship type. Many more transitive closure optimizations are available, yet none of them is as simple as the one shown. Although there is still much potential left for improving the storage and query optimizers for transitive relational structures in RDBMS, we could confirm that even challenging combined relationships can be managed and queried in a commercial grade RDBMS. We do not pretend to have implemented a sophisticated subsumption and classification algorithm; however our current EMR system does not depend on such algorithms and classifiers. For us and at this time it is more important to free the data from specialized applications and into an open industry supported data architecture.

Acknowledgements

This work has been performed at the Regenstrief Institute for Health Care with support in part by the National Library of Medicine (Contract #N01-LM-6-3546).

[1] Rector AL, Bechhofer S, Goble CA, Horrocks I, Nowlan WA, Solomon WD. The GRAIL concept modelling language for medical terminology. *Artif Intell Med.* 1997 Feb;9(2):139-71.

[2] Musen MA, Gennari JH, Eriksson H, Tu SW, Puerta AR. PROTEGE-II: computer support for development of intelligent systems from libraries of components. Medinfo. 1995;8 Pt 1:766-70.

[3] Schulz S, Romacker M, Hahn U. Knowledge engineering the UMLS. *Stud Health Technol Inform.* 2000;77:701-5.

[4] Codd EF. A relational model of data for large shared data banks. *Communications of the ACM.* 1970 Jun;13(6):377-387.

[5] Date CJ, Darwen H. *Foundations for object/relational databases: The third manifesto.* Reading, MA; Addison-Wesley, 1998.

[6] Zhe L, Ross KA. On the cost of transitive closures in relational databases [technical report CUCS-004-93]. 1993 Feb. New York; Computer Science Department, Columbia University.

[7] McDonald CJ, Overhage JM, Tierney WM, Dexter PR, Martin DK, Suico JG, Zafar A, Schadow G, et al. The Regenstrief Medical Record System: a quarter century experience. *Int J Med Inf.* 1999 Jun;54(3):225-53.

[8] Huff SM, Rocha RA, McDonald CJ, De Moor GJ, Fiers T, Bidgood WD, Forrey AW, Francis WG, et al. Development of the Logical Observation Identifier Names and Codes (LOINC) vocabulary. *J Am Med Inform Assoc.* 1998 May-Jun;5(3):276-92.

[9] Schadow G, Russler DC, McDonald CJ. Conceptual integration of guidelines and workflow into the electronic health record. Stud Health Technol Inform. 2000;77:807-11.

[10] Russler DC, Schadow G, Mead C, Snyder T, Quade LM, McDonald CJ. Influences of the Unified Service Action Model on the HL7 Reference Information Model. Proc AMIA Symp. 1999;:930-4.

[11] International Organization for Standardization: Information technology – Database languages – SQL [ISO/IEC 9075:1992]. 1992, Geneva, Switzerland; The organization.

[12] International Organization for Standardization: Information technology – Database languages – SQL – Part2: Foundation [ISO/IEC 9075-2:1999]. 1992, Geneva, Switzerland; The organization.

[13] Nuutila E. Efficient transitive closure computation in large digraphs [dissertation]. 1995. Helsinki, Finnland; Helsinki University of Technology.

[14] Agrawal R, Borgida A, Jagadish HV: Efficient management of transitive relationships in large data and knowledge bases. ACM-SIGMOD 1989 Int'l Conf. on Management of Data, Portland, Oregon, 1989.

[15] Kamfonas MJ. Breaking the relational taboo [online]. Intel Enterp Database Progr Design. URL: http://www.dbpd.com/vault/9811/kamfn.shtml.

[16] Horrocks I, Rector AL, Goble C. A description logic based schema for the classification of medical data. 1996. *Proceedings of the 3rd Workshop KRDB'96*:24-28.

[17] All queries discussed here are available from: http://aurora.regenstrief.org/recursive-SQL.html