Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Editorial

Extending ER models to capture database transformations to build data sets for data mining

Carlos Ordonez [a,*], Sofian Maabout [b], David Sergio Matusevich [a], Wellington Cabrera [a]

[a] *University of Houston, Texas 77204, USA*
[b] *LaBRI, Bordeaux, France*

ARTICLE INFO

Article history:
Received 19 March 2012
Received in revised form 14 November 2013
Accepted 15 November 2013
Available online 19 December 2013

Keywords:
ER model
Data mining
Transformation
Denormalization
Aggregation

ABSTRACT

In a data mining project developed on a relational database, a significant effort is required to build a data set for analysis. The main reason is that, in general, the database has a collection of normalized tables that must be joined, aggregated and transformed in order to build the required data set. Such scenario results in many complex SQL queries that are written independently from each other, in a disorganized manner. Therefore, the database grows with many tables and views that are not present as entities in the ER model and similar SQL queries are written multiple times, creating problems in database evolution and software maintenance. In this paper, we classify potential database transformations, we extend an ER diagram with entities capturing database transformations and we introduce an algorithm which automates the creation of such extended ER model. We present a case study with a public database illustrating database transformations to build a data set to compute a typical data mining model.

1. Introduction

The entity-relationship (ER) model [1,2] provides diagram notation and methods to design a database, by defining its structure before storing information. On the other hand, the relational model provides a precise mathematical definition to store information in the form of tables (relations) interrelated by foreign keys, whose basic structure is determined by the ER model [1]. Relational algebra [2] provides a formal computation mechanism to query tables in a database combining select, project, join and aggregation (SPJA) operations. In this work, we are concerned with modeling all potential database transformations via relational queries to build a data set that is used as input by a typical data mining algorithm [3]. In general, such data set has a tabular form, where every row corresponds to an observation, instance or point (possibly varying over time) and every column is associated to an attribute or feature. The main issue is that relational queries in a data mining project create many temporary tables (static) or views (dynamic), which are not represented as entities in an existing ER model. Such collection of transformation tables and disconnected queries complicate database management, software development and software maintenance. The problem is even more acute when multiple data sets are derived. Our approach is related to an extract–load–transform (ELT) process, in which tables are cleaned and transformed *after* they are loaded into the database. On the other hand, traditional Extract–Transform–Load (ETL) tools can compute data transformations, but mainly to build data warehouses. In ETL tools most data transformation

* Corresponding author at: Department of Computer Science, University of Houston, Houston, TX 77204, USA.
URL: http://www2.cs.uh.edu/~ordonez/index.html (C. Ordonez).

0169-023X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.datak.2013.11.002

happens outside the DBMS, before loading data. In general, tools that perform data reverse engineering [4–6] do not give an abstract, well-defined representation of database transformations computed with relational queries, nor do they have a data set with variables (in a statistical sense) as the target of such transformations. We propose to promote derived data sets and associated temporary tables as an optional extended view of an ER model. By doing so, users can become aware of the content of the database and hence help them reuse what has already been computed. Indeed, understanding transformation tables in abstract form can be useful not only for optimizing the generation of new data sets, but also for tracing their provenance (lineage) [7]. Our approach goes further by considering several stages of data transformation, mathematical functions and the powerful SQL CASE statement, which does not have a 1–1 correspondence with relational algebra.

Based on the motivation introduced above, we extend an ER diagram with entities that represent database transformations used to build data sets. Our proposed extended ER model can help analysts understand previous database transformations and then motivate reuse of existing temporary tables, views and SQL scripts, thereby saving time and space. We introduce an algorithm to automate the process of extending an existing ER model based on a sequence of SQL transformation queries given in a script. The paper is organized as follows: Section 2 provides background information on the ER model and relational databases. Section 3 formalizes the problem with relational algebra, classifies database transformations, explains how to extend an existing ER model and introduces an algorithm to automate the ER model extension. In Section 4 we discuss a case study with a real database. We discuss closely related work in Section 5. Section 6 concludes the paper.

2. Preliminaries

We first present definitions and assumptions. Then we introduce a small database that will be used throughout the paper to illustrate our contributions.

2.1. ER model and its mapping to a relational model

A relational database is denoted by $D(T,I)$, where $T = \{S_1,\ldots,S_n\}$, is a set of n source tables and I is a set of integrity constraints. Columns in tables are denoted by $A_1,A_2,\ldots$, corresponding to attributes in an ER model. They are assumed to have the same name across all entities. The most important constraints are entity and referential integrity, which ensure validity of primary keys and foreign keys. Entity integrity requires that every relation should have a primary key. On the other hand, referential integrity is an assertion of the form $S_i(K) \rightarrow S_j(K)$, where K is the primary key of S_j and K is a foreign key in S_i. Referential integrity requires that the inclusion dependency $\pi_K(S_i) \subseteq \pi_K(S_j)$ holds.

As it has become standard in modern ER tools, each entity corresponds to a table and each relationship is represented by foreign keys. We assume that 1:1 relationships can be merged into one entity because they share the same primary key. Moreover, we assume that M:N (many to many) relationships eventually get mapped to a "linking" entity, which connects both entities taking their respective foreign keys as its primary key. Therefore, it is reasonable to assume that only 1:N (1 to many) and N:1 (many to 1) relationships exist between entities in a normalized database. Because relationships can be read in both directions it suffices to base our discussion on 1:N relationships. In short, we will work with the final ER model, ready to be deployed as physical relational tables, with a one to one correspondence between entities and tables. We use modern ER notation, which has been influenced by UML (Unified Modeling Language). Entities are represented by rectangles, attributes are grouped into key and non-key attributes and relationships are represented by lines. Entities are linked by relationships denoted with solid lines, where a crow feet indicates the "many" side of the relationship.

The ER to relational model mapping is the standard mechanism to convert an ER conceptual model into a physical model that can be deployed with data definition languages (DDL) in SQL. Database system textbooks (e.g. [2]) present a standard procedure for mapping (or converting) an ER model to a relational model. This mapping is not lossless, in the sense that not all semantics captured in ER are distinguishable in the relational model. In practice, an ER model is deployed in the form of a relational database schema, where entities and relationships are represented by tablesand referential integrity constraints.

2.2. Relational algebra and SQL

We assume that database transformations are computed with the SQL language. However, in order to have precise and short mathematical notation we study database transformation with relational queries, which have set semantics and obey first order logic. We use an extended version of relational algebra, where π, σ, $\bowtie$, $\cup$ and $\cap$ are standard relational operators and π is used as an extended operator to compute GROUP BY aggregation queries (e.g. grouping rows to get sum(), count()). Relational queries combine SQL aggregate functions, scalar functions, mathematical operators, string operators and the SQL CASE statement. Given its programming power and the fact that it is incompatible with relational algebra, we study the CASE statement as a special operator.

Outer joins are essential for the construction of data sets. However, full outer joins are not useful in data mining transformations because, generally speaking, the referencing table is the table whose rows must be preserved, whereas right outer joins can be rewritten as equivalent left outer joins. Consequently, an inner (and natural) join is a particular case of a left outer join returning less rows. In short, we consider left outer join a prominent operator needed to merge tables to build the desired data set.