Enhancing ER Diagrams to View Data Transformations Computed with Queries

Carlos Ordonez
University of Houston, USA

Ladjel Bellatreche
LIAS/ISAE-ENSMA, France

ABSTRACT

Transforming relational tables to build a data set takes most of the time in a machine learning (ML) project centered around a relational database. The explanation is simple: a relational database has a collection of tables that are joined and aggregated with complex relational queries, and whose columns are transformed with complex SQL expressions, in order to build the required data set. In general, such data is wide, gathering many ML variables together. Such complicated data pre-processing results in a large set of SQL queries that are independently developed from each other for different ML models. The database grows with important tables and views that are absent in the original ER diagram. More importantly, similar SQL queries tend to be written multiple times, creating problems in database evolution, disk space utilization and software maintenance. In this paper, we go in opposite direction from a physical level (tables) to a logical level (entities) representation, providing a unifying diagram of both levels. Specifically, we propose minimal, but powerful, extensions to an ER diagram in UML notation to represent most common database transformations. Our "transformation" ER diagram helps analytic users understanding complex transformations, consolidating columns representing analytic variables into fewer tables (i.e. eliminating redundant tables), reusing existing SQL queries (i.e. avoid forking new queries) and explaining data provenance (where data originated from).

1 INTRODUCTION

The entity-relationship (ER) model provides methods (step by step) and diagram notation to design a database, by defining its structure before storing data values. On the other hand, the relational database model provides a precise mathematical definition to store and query data in the form of tables (relations) linked by foreign keys (FKs), whose basic structure is sketched in an ER model. We are concerned with exploiting a DBMS to analyze data sets with exploratory analysis, machine learning and statistics [2]. We are interested in modeling all potential database transformations via relational queries to build a specific data set that can be analyzed by machine learning (ML) algorithms. In general, such data set has a tabular form, where every row corresponds to an observation, instance or point and every column is associated to a variable or feature. The main side effect is that relational queries produce many new tables, which do not appear as entities in the existing ER diagram. Such collection of transformed tables and disconnected queries complicate database management, software development, and computing further ML models.

Based on the motivation discussed above, we introduce minor, but powerful, changes to ER diagrams to represent data transformations computed by queries, used to build data sets for ML models. Our main contributions are the following: (1) labeling ER entities to distinguish source and transformation entities; (2) numbering entities to understand a sequence of transformations; (3) showing SPJA queries as an optional zoom-in exploratory feature; (4) an algorithm to automate extending and maintaining the transformation ER diagram. We believe our ER diagram has potential usefulness. Existing tuned queries can be reused and extended. Redundant copies of data can be eliminated by reusing existing queries. Provenance of attributes can be tracked. Nevertheless, we do not intend to represent every query in a single ER diagram since that would create a huge diagram. Instead, we propose to represent each set of tuned queries to build each data set separately. Each set of queries is similar to a data mart with the main difference being that table structure is gradually refined by queries, not by database design.

2 DEFINITIONS

2.1 ER Diagram for a Relational Database

In modern database design tools an ER diagram is refined and consolidated into a collection of "structured" entities represented in a simplified UML diagram, closer to a physical model. In the resulting relational database schema entities and relationships are mapped to tables linked by referential integrity constraints. As it is standard in most modern ER tools, each entity corresponds to a table and each relationship is represented by foreign keys (FKs).

We assume 1:1 relationships are consolidated into one entity because they share the same primary key (PK). Moreover, we assume M:N (many to many) relationships get mapped to a "link" entity, which connects 2+ entities, merging their respective FKs into a new PK. Therefore, a refined database ER diagram has only 1:N (1 to many) and N:1 (many to 1) relationships. In short, we assume there exists an ER diagram ready to be converted and deployed as physical relational tables, with a 1-1 correspondence between entities and tables. Such restricted ER diagram allows representing tables returned by queries as entities.

A database is defined as $D(\mathcal{T}, I)$, where $\mathcal{T} = \{T_1, \ldots, T_n\}$, is a set of n tables and I is a set of integrity constraints (entity for primary key (PK), referential for foreign key (FK)). We use the term "table" to make an explicit connection to the physical level and SQL. Each table has a set of columns, coming from different domains. The entity constraint asserts that every table should have a primary key. Referential integrity is denoted by $T_i(K) \rightarrow T_j(K)$ where K is the primary key (PK) of T_j and K is a foreign key (FK) in T_i. The common key attribute K is assumed to have the same name on both tables.

2.2 Relational Queries

Database transformations are assumed to be computed only with SQL queries, mixing joins ($\bowtie$), aggregations (extended π operator) and selection (σ). Relational queries combine aggregate functions (built-in or UDFs), scalar functions (built-in or UDFs), mathematical operators, string/date operators and the powerful SQL CASE statement. However, in order to have a precise mathematical foundation we study data transformation with relational queries, which combine π, σ and $\bowtie$ operators, where π is used

as a GROUP BY aggregation and derivation operator. Notice SQL PIVOT and UNPIVOT operators available in some DBMSs are great to manipulate matrices, but they are not relational. Fortunately, these operators can be represented with tables with extra columns storing subscripts generated by the DBMS. We assume *well formed* queries are used to create data sets, which include a primary key to identify rows and at least one non-key attribute (variable, feature).

We consider left outer join as a prominent operator needed to merge partial tables sharing the same PK to build the data set for analysis. In query terms, referential integrity between two tables T_i, T_j means $\pi_K(T_i) \subseteq \pi_K(T_j)$.

2.3 Running Example

Our example is based on the benchmark TPC-H database, shown on Figure 1. Consider the following representative ML problem: predicting whether a product will be returned or not, based on historic sales information. A long database transformation process is needed to build the data set to be used as input in a regression or classification model. Our objective is to extend the existing ER diagram to capture this data transformation process.

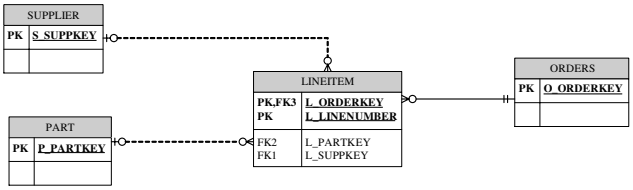

Figure 1: Input database ER diagram for example.

3 DATA TRANSFORMATIONS

3.1 Data Set

The main objective is to create a single table, that will be the input for an ML algorithm. This table will be the output of a sequence of SPJA queries filtering, merging, transforming and aggregating data.

The data set is represented by an entity with two sets of attributes $X(K, A)$, where K is the primary key (generally simple, but it could be composite) and A is a set of p non-key attributes (numerical & categorical variables). From a database ER modeling standpoint the data set corresponds to a weak entity that depends on some strong entity [1], defining an "is-a" relationship (i.e. the data set is a sub-type of some strong entity). In a machine learning context, X will be the input for a mathematical model like linear regression, PCA dimensionality reduction, Bayesian classification, decision trees or clustering.

The first consideration is K, which can be simple or composite. The most common case is that K is simple with a 1-1 correspondence to some existing source entity (e.g. customer id). In such case, we will use i instead of K to make reference to the ith point or the ith instance. On the other hand, if K is composite then it takes primary keys from several entities (e.g. customer id, product id), similar to a data cube with a composite key or it is the result of adding a key attribute that was not originally part of any entity (e.g. month id). In the latter case, there is likely already a "dimension" table where the new key attribute is already available (e.g. a cube dimension table). Evidently, given k source tables (entities) there should be $k-1$ foreign keys linking them in order to enable $k-1$ joins. In other words, the primary key of the data set comes from putting together the primary keys of a subset of the k source tables.

We now discuss A, the set of non-key attributes. Assume $T = T_1 \bowtie \ldots T_2 \cdots \bowtie T_k$ gathers all raw attributes. The goal is to further transform T into X. We emphasize T is not in 3NF. On the other hand, if T has a composite key it will not be in 2NF either. Each attribute in A can come from: a table, an aggregation or a scalar expression (arithmetic, logical, string). Therefore, a transformed attribute must either be the result of an aggregation summarizing several rows into one or computing some derived expression (denormalization). Below we explore data transformations in more depth.

3.2 Representing Data Transformations

We explain database transformations at two levels: entity (table) and attribute (column), in order to define an ER diagram at two levels of abstraction.

Entity Level. We extend an existing ER diagram with new entities providing a diagram representation of data transformations. We call existing entities *source* entities and added entities are called *transformation* entities (computed with SPJA queries). We emphasize that a single SQL query can create several temporary tables during its evaluation with one table per relational operator. In order to define a faithful representation each nested SQL query will correspond to one temporary table, which in turn will be represented by one relational algebra query and one transformation entity. By a generalization, the data transformation process will create a sequence of transformation tables that will be successively joined and aggregated. The data set X will be the final output. Such database transformation process with nested queries (based on views or temporary tables) will result in a tree (similar to a parse tree from a query), where internal nodes are joins or projections (aggregations), leaves are tables and the root is one final query returning the data set. In the extended ER diagram, the new entities are linked with existing entities or previous transformation entities.

The π operator eliminates duplicates, if any. Even though SQL has bag semantics, allowing duplicate rows, such semantics do not make sense from a machine learning perspective. The reason is simple: each represented object must be identifiable. Assume a well formed query $T = T_1 \bowtie T_2 \cdots \bowtie T_k$ on appropriate foreign keys. We propose the following rules to enable query composition. If we project columns from T they must either include the primary key of T (being the primary key of some table T_i) or they include a GROUP BY key to compute aggregations. When computing data transformations with SQL queries the output table cannot be used in future queries if it does not include the primary key of T and it does not have aggregations to derive non-key attributes. That is, π must include both GROUP BY attributes and a list of aggregation functions.

Attribute Level. There exist two mutually exclusive data transformation classes: (1) Denormalization (via join), which gathers attributes from several entities into a single transformation entity or simply derives new attributes from existing attributes. (2) Aggregation (via projection with aggregation), which creates a new aggregation value attribute grouping rows by a primary key and computing some summarization. An aggregation cannot be considered denormalization because, in general, we cannot reason about functional dependencies beyond 1NF between the grouping column(s) and the aggregated column.

Denormalization: When denormalization brings attributes from other entities the attribute values themselves are not transformed. That is, the table structure changes, but the column values remain the same. When attributes come directly, without transforming values, from other entities they can have three roles: being part of the primary key, being part of a foreign key or being non-key whatsoever. Attributes being part of the PK or a FK can later be used to group rows to compute aggregations, which are our second class of data transformations. In general, non-key attributes interact together using all SQL operators, programming constructs and functions (scalar, aggregate, UDFs). Arithmetic expressions and string expressions fall into this category. The only non-key denormalization data transformation that deserves special attention is the SQL CASE statement. There are two important issues introduced by the CASE statement: (1) it may create new values, not present on any previous column. (2) it may introduce nulls which were not present before. Therefore, an SQL query with CASE statements cannot be evaluated with an equivalent SPJA query.

Aggregation: Aggregations are expressed with the π operator having a grouping attribute(s) and a list of functions (e.g. sum(),count()). In SQL, the GROUP BY caluse partitions output attributes into key attributes and aggregated attributes. In general, aggregations return numbers, but notice "max()" in SQL aggregations can have strings or dates as argument. Global aggregations without a GROUP BY clause (e.g. a count() or rows, a total sum()) can be represented with a π operator with an artificial column with a constant value. Representing such aggregation in the ER diagram is problematic because there is no given primary key; this is a research issues in database modeling. In meantime, we propose to "pin" such aggregation to the entity name as a small bag of aggregations (e.g. counts, sums, statistics).

3.3 Extending ER Diagram

We propose extensions to ER diagram notation to represent database transformations to build one data set, which can be exploited by multiple ML models. We clarify there will be multiple ER diagrams, corresponding to different data sets (i.e. having different PK) and that we aim to represent only polished tuned queries. That is, we do not propose to build a single ER diagram representing all temporary tables since that would create a huge ER diagram, impossible to interpret. We emphasize an ER diagram works at a conceptual/logical level and SQL works at a physical level. The ER diagram helps designing a coherent database structure, whereas SQL queries help processing the database. Therefore, our proposed notation brings both paradigms closer, enriching the ER diagram with the ability to represent transformations on database attributes, but which mixes database modeling with database processing.

The first consideration is notation for queries. SQL has bag semantics and SQL queries tend to be verbose. On the other hand, given its precise mathematical definition and conciseness relational algebra is the best choice. The second aspect is defining our diagram notation, a thorny and subjective aspect. After exploring several alternatives keeping in mind the large number of temporary tables and columns created by SQL queries, we propose two ER diagrams: (1) a high level "abstract" ER diagram displaying only entities and (2) a low level (fine granularity) "query" ER diagram displaying all attributes, the data transformations and the corrsponding query. The high level diagram can be used to explore the connection among tables and understand the overall flow of transformations. On the other hand, the low level diagram provides a specific idea on how transformations make up a table. The low level view can be selectively displayed in a "zoom in" view on each entity.

ER diagram notation in the literature has many variants: evidently the classical, but highly intuitive, notation with small boxes for entities and external ellipses for attributes [1] is inadequate and cumbersome as it does not scale to a large number of entities and much less to a large number of attributes. That is why we defend the idea of using UML notation as most modern ER tools do. A transformation entity represents a weak entity in a strict sense since each object existence depends on the source entity. The classical ER notation uses dotted lines. Since we intend to use UML notation and we intend to extend it further we prefer to show transformation entities with solid lines. As introduced above, we classify entities as source or transformation, where transformation entities are either GROUP BY aggregations or denormalization. Therefore, each entity will be labeled as "source:<name>", or "denormalization:<name>", or "aggregation:<name>". A complicated aspect is where to place attribute transformations. This is not an easy choice since a typical modern ER diagram may have hundreds of entities and thousands of attributes, exploding further with the analytic database transformations. Given the column-oriented programming approach in SQL and the fact that modern ER diagrams display one attribute per line we decided to show denormalization expressions and aggregations to the right of the attribute name. Putting all these elements together, our proposal is to show database transformations as an additional column inside an ER entity box. Such column can substitute the data type definitions in a typical physical model (convertible to SQL Data Definition Language). A major requirement is understanding where columns come from in a given transformation entity. This information must necessarily come from the $\bowtie$ operator in a query. We believe the best place to display a query is next to the entity name, simply showing which tables participate in $\bowtie$. Attributes provenance will be shown with single dot or double dot notation (e.g. T.A or T..B).

Finally, we must consider attributes (table columns). A major requirement is to track provenance. We propose the following notation: a single dot to indicate the source table if such table participates in a join in the corresponding query (e.g. $T.A$). Otherwise, we propose to use a double dot to indicate the column originates from a previous temporary table (e.g. $T..A$). Based on the fact that SPJA queries cannot express CASE computations it is necessary to define a special notation to represent CASE statements. Basically, there are two elements: a predicate combining comparisons with and/or/not and the value(s) to be assigned. Therefore, we can think of the CASE statement as a functional form which returns a value based on the predicate being true or false. Given its similarity to the C++ "?:" operators, it is the notation we use. For the example mentioned above it will be shown as "($A >= 0$) ? 'positive':'negative'", which is shorter than the CASE syntax and which is intuitive to C++ or Java programmers.

The actual SQL queries, including each temporary table are selectively displayed on each trasformation entity with a "zoom in" low level view. That is, we avoid showing all SQL queries in the ER diagram.

Extended ER Diagram Properties. From a theory perspective, since we use relational algebra to represent database transformations our extended ER diagram is guaranteed to be: (1) complete and (2) consistent. From a database design perspective, our model

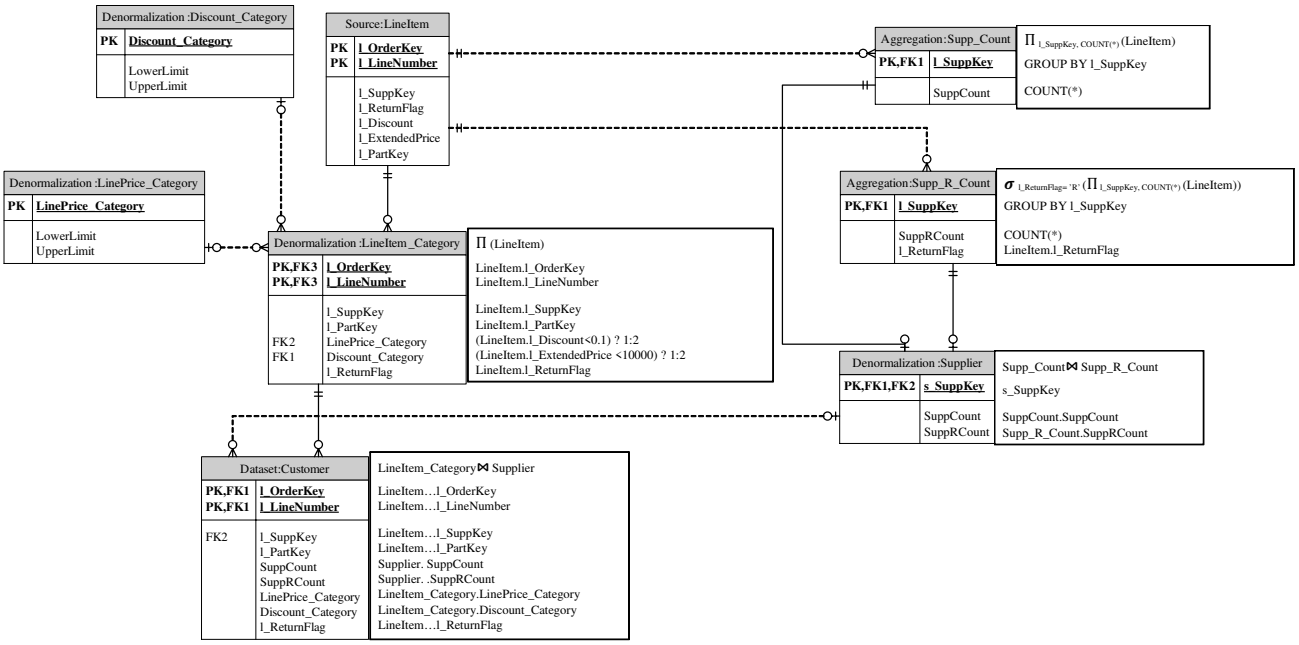

Figure 2: Transformation ER diagram for TPC-H database (low level).

can improve database evolution it can track attribute provenance, it can help reusing existing tables and it motivates reusing transformation queries.

3.4 Extended ER Diagram Example

We now illustrate our extended ER diagram notation and the algorithm with the sample database introduced in Section 2. Figure 2 presents the low level view with entities, data transformations at the attribute level and the query.

3.5 Algorithm to Extend an ER Diagram

Given an ER diagram (generally corresponding to a tuned OLTP or OLAP database) and an existing set of polished SQL queries that create data set X, the following steps help building an extended ER diagram.

(1) Initialize extended ER diagram with the original ER diagram; labeling each entity as "source" entity (S).
(2) Create a transformation (T) entity for every intermediate table; consider nested queries and views as additional temporary tables. Label each intermediate table as T< 99 >, where 99 is an increasing integer, resulting from an incremental computation.
(3) Label each attribute as key or non-key.
(4) For each non-key attribute associate to either: a derived expression or an aggregation. Indicate provenance (lineage) of attributes coming from the denormalization process. For aggregations use the same function name provided by SQL in a relational algebra expression.
(5) Add a final main data set entity joining all intermediate tables; this data set entity will be highlighted in the ER diagram and labeled "data set".

4 CASE STUDY

Due to lack of space we do not present experiments. Intead we summarize a case study with a real database with bike sales information, that comes as a test database for the Microsoft SQL Server DBMS (AdventureWorks). We manually wrote SQL queries to build a data set to predict if a customer will buy or not any product in the next 6 months, given past sales history. This database consists of approximately 70 tables with bike stores sales over 4 years. Our program, written in C#, using 12 SQL queries as input, produced a sequence of relational schemas with appropriate primary and foreign keys, which were labeled as transformation entities (with our algorithm) in a couple of seconds (since this involves only schema data). The data set was used as input in several ML predictive models, including Naive Bayes and logistic regression. Using a standard CSV format, these entities can be visualized in automated ER diagram drawing tools (e.g. smartdraw, Lucidchart).

Measuring disk space savings, reduction in development time, time to automatically produce an ER diagram and easiness to maintain the extended ER diagram requires a long-term case study, which we will conduct in the future.

5 RELATED WORK

Research on ER modeling can be classified as models for transactional databases and models for analytic databases (data warehouses). Models for transactional databases ensure the database can be maintained in a valid state (complete and consistent), whereas models for database analysis enable multidimensional analysis on cubes and basic machine learning. Since we are concerned with analyzing the database rather than updating it our extended ER model is more closely related to models for data warehouses. However, there are fundamental differences between both kinds of models. Cube dimensions and measures are identified early on the database design to denormalize tables, whereas attributes in the data set are built later, during the iterative data mining process. Generally speaking, data transformations for ML analysis are more complex since they also involve mathematical functions and complex logic in the CASE statement. On the other hand, provenance identifies the sources of information in a data warehouse. We have adapted provenance to trace the

source tables an attribute in the data set comes from. A closely related work that studied the problem of transforming tables from a modeling perspective is [5], where the authors study the transformation of a database schema into another equivalent one in order to facilitate information exchange with another database; [5] compares alternatives schemas, but does not consider queries combining aggregations, denormalization and math functions on attributes to transform tables like we do. The closest research we found that studies how to represent a data set to compute a machine learning model is [7], where the authors exploit UML notation to extend an existing data warehouse model to represent attributes used in classification, a prominent statistical learning problem; we emphasize [7] focuses more on the final data set, rather than studying the data transformations to build it. The idea to represent data transformations in the ER model was proposed in [3], where simple data transformations are classified and SQL queries are shown in the ER diagram. Our paper takes a step further by focusing on the ER diagram (coining the term "transformation" ER diagram), formally studying the problem with relational notation instead of SQL, considering a broader class of queries (e.g. representing σ and views), and introducing an algorithm to automate ER diagram construction.

The closest approach in the literature is reverse data engineering, where the main purpose is to produce an ER diagram using a database schema as input. In contrast, we assume queries use tables already stored on the database. Moreover, we consider attribute-level data transformations beyond denormalization with joins. ML models generally require many tables to be joined, aggregated and transformed to derive variables (features). In [4] there is a proposal to analyze inclusion and functional dependencies in denormalized tables built with relational queries. However, the goal is to transform tables into a 3NF database model representable in ER form, rather than providing an abstract representation of data transformation queries.

Our work shares similarities with research on ETL (Extract-Transorm-Load) [6] where records are transformed and cleaned with traditional programming languages (including UDFs) before being loaded; this includes conceptual modeling for ETL [**? ?**] and ELT, where data records are transformed with queries after being loaded. Important similarities include representing data processing with a diagram, capturing a comprehensive set of data transformations and meeting users requirements. On the other hand, important differences include the following. We assume referential integrity problems have been solved. The input to our diagram is a set of queries, which work strictly on tables, not on files or documents. We draw a clear boundary on data transformations based on joins (only derivation) or on aggregation (some aggregation function). Instead of using a flow diagram to represent data processing we propose to represent such processing with "processing" entity boxes. We believe our "zoom-in" view of transformation entities with relational queries is novel and meaningful.

6 CONCLUSIONS

We introduced minimal ER diagram extensions to represent data transformations in abstract form, bridging the gap between a logical database model and a physical database. We focused on studying database transformations to build a data set for machine learning analysis. We assume such data set is wide, gathering many ML variables (features) in one place. Our extended ER model has two kinds of entities: source entities and transformation entities, which correspond to standard tables coming from an ER diagram and temporary "analytic" tables created with SQL queries, respectively. Data transformations are further classified into two main categories: denormalization and aggregations. Due to the large number of entities and attributes involved, we use modern UML diagram notation, where all entities are boxes and entities are linked by 1:1 and 1:N relationships. We extended ER diagram entity boxes with a new box where transformations are expressed with extended relational algebra notation (SPJA queries). We introduced an algorithm to extend an existing ER model, using data transformation queries as input where the data set is the final database goal. We emphasize that the extended ER diagram is automatically created for each ML project, to be relevant and useful. That is, different data sets, having different primary keys, have separate extended ER diagrams.

Our proposed ER diagram extended with data transformation offers many opportunities for future research. It is necessary to study data transformations deeper, considering coding, pivoting and cubes. We want to further explore UML diagram constructs to enrich ER to represent database transformations. We want to determine when it is feasible to design transformation entities before writing SQL queries to close the loop. From a normalization perspective, it is necessary to study aggregations choosing a best table schema to minimize storage of redundant information. The SQL CASE statement is powerful, but it introduces many issues from a theoretical and database modeling perspective. We plan to explore provenance aspects in depth. We intend to develop metrics to quantify savings in software development effort (e.g. lines of source code, number of functions or classes) and database maintenance (number of tables that can be reused, number of attributes that can be used by multiple ML models, or number of tables that can serve multiple analytic goals).

REFERENCES

[1] Z. Akkaoui, J.N. Mazón, A.A. Vaisman, and E. Zimányi. Bpmn-based conceptual modeling of ETL processes. In *Proc. DaWaK Conference*, pages 1–14, 2012.

[2] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Book.* Prentice Hall, 2nd edition, 2008.

[3] B. Oliveira and O. Belo. Using REO on ETL conceptual modelling: a first approach. In *Proc. ACM DOLAP*, pages 55–60, 2013.

[4] C. Ordonez. Can we analyze big data inside a DBMS? In *Proc. ACM DOLAP Workshop*, 2013.

[5] Carlos Ordonez, Sofian Maabout, David Sergio Matusevich, and Wellington Cabrera. Extending ER models to capture database transformations to build data sets for data mining. *Data & Knowledge Engineering*, 2013.

[6] J.M. Petit, F. Toumani, J.F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *Proc. ICDE Conference*, pages 218–227, 1996.

[7] Alexandra Poulovassilis and Peter McBrien. A general formal framework for schema transformation. *Data & Knowledge Engineering (DKE)*, 28(1):47–71, 1998.

[8] P. Vassiliadis, A. Simitsis, and E. Baikousi. A taxonomy of ETL activities. In *DOLAP*, pages 25–32, 2009.

[9] J. Zubcoff and J. Trujillo. Conceptual modeling for classification mining in data warehouses. In *Proc. DaWaK Conference*, LNCS, pages 566–575. Springer, 2006.