

# Logical database design: from conceptual to logical schema

Alexander Borgida

Rutgers University, <http://www.cs.rutgers.edu/> borgida

Marco A. Casanova

PUC - Rio, <http://www.inf.puc-rio.br/> casanova

Alberto H. F. Laender

Universidade Federal de Minas Gerais, <http://www.dcc.ufmg.br/> laender

**SYNONYMS** “*Logical Schema Design*” “*Data Model Mapping*”

## **DEFINITION**

Logical database design is the process of transforming (or mapping) a conceptual schema of the application domain into a schema for the data model underlying a particular DBMS, such as the relational or object-oriented data model. This mapping can be understood as the result of trying to achieve two distinct sets of goals: (i) representation goal: preserving the ability to capture and distinguish all valid states of the conceptual schema; (ii) data management goals: addressing issues related to the ease and cost of querying the logical schema, as well as costs of storage and constraint maintenance. This entry focuses mostly on the mapping of (Extended) Entity-Relationship (EER) diagrams to relational databases.

## **HISTORICAL BACKGROUND**

In the beginning, database schema design was driven by analysis of the prior paper or file systems in place in the enterprise. The use of a conceptual schema, in particular Entity Relationship diagrams, as a preliminary step to logical database design was proposed by P. P. Chen in 1975 [2, 3], and the seminal paper on the mapping of EER diagrams to relational databases was presented by Teorey et al. in 1986 [7]. Major milestones in the deeper understanding of this mapping include [6] and [1], which separate the steps of the process and pay particular attention to issues such as naming and proofs of correctness. Among others, the correct mapping of subclass hierarchies requires a careful definition of table keys [5], and the *maintenance* of optimized relational representations of ER schemas is discussed in [4].

## ENGINEERING FUNDAMENTALS

The mapping from an Extended Entity Relationship schema to a relational (logical) schema handles the issue of representing the different states of the conceptual schema through a function that provides for every “object set”  $O$  (entity set or relationship set) in the EER schema a relational table  $T_O$ . The data management-related issues are handled by merging, partitioning or otherwise reorganizing the tables obtained in the previous step, while making sure that there is no loss of information. (See “information capacity” entry).

To begin with, some notation and assumptions concerning the conceptual schema expressed in EER notation are required. A unique set of identifying attributes  $id(E)$  is assumed to be available for every strong entity set  $E$  (i.e., one that is not a weak entity or a subclass). Each object set  $O$  participating in a relationship set  $R$  can be marked as being: **total**, indicating that each instance of  $O$  must participate in at least one relationship instance of  $R$ ; **functional**, indicating that an instance of  $O$  can participate in at most one relationship instance of  $R$ ; and as playing a particular **role** in the relationship, if  $O$  (or its sub/super-class) can participate in other relationships or as different arguments of  $R$ . (See EER entry.) For every relationship set  $R$ ,  $id(R)$  is assumed to return a subset of participants that uniquely determine each relationship instance. This is needed in examples such as “Exactly one faculty advises each student for each major area”, where  $id(advises)=\{Student, Major\}$ , because students can major in several areas. Of course, if the relationship set  $R$  has one functional participant  $O$ , then  $id(R)$  returns  $O$ ; when there are multiple functional participants,  $id(R)$  is assumed to return one that is total, if available. Entities can be organized in a ISA/subclass hierarchy, with the ability to declare subclasses as **disjoint**, and/or **covering** the superclass.

In a relational schema, for each table  $T$ , one must specify its attributes/columns, its primary key, any foreign key referential constraints, and non-null constraints on columns. The notation  $T[\underline{KY}]$  is used to refer to a table named  $T$ , with columns  $KY$  and primary key  $K$ ;  $key(T)$  returns  $K$ .

### The Basic E2R Mapping

The initial mapping from entity and relationship sets in the conceptual schema to relational tables (referred to as the E2R mapping) is defined recursively as follows: For a strong entity set  $E$ , the attributes of  $E$  form the columns of table  $T_E$ , and its key is set to  $id(E)$ . For example, for entity set **Student**, with attributes **sid**, **name** and **age**, and  $id(Student)=sid$ , the E2R mapping will generate a table  $T\_Student[sid, name, age]$ .

To preserve first-normal form,  $T_E$  does not include multi-valued attributes of  $E$ . For any such attribute  $M$  of  $E$ , one adds a separate table  $T_{EM}$ , whose attributes are those in  $id(E)$  plus a new attribute  $\hat{M}$  that holds the individual values that occur in  $M$ ; the full set of columns forms the key of this table, and a foreign key constraint is added from column  $id(E)$  of  $T_{EM}$  to  $T_E$ . Thus, if the entity set **Student** has a multi-valued attribute **phone**, then the E2R mapping will generate a table  $T\_StudentPhone[sid, phone]$ , with **sid** being a foreign key with respect to  $T\_Student$ .

If  $E$  is a subclass of some class  $F$ , then the columns of  $T_E$  consist of the attributes of  $E$  together with those in  $key(T_F)$  — the key “inherited” from  $T_F$ . Therefore  $key(T_F)$  is the key of  $T_E$ , and a foreign key reference to  $T_F$  is needed to ensure that every subclass instance is in the super-class. For example, supposing that **GradStudent** is given as a subclass of **Student**, then  $T\_GradStudent$  will have, among others, a column **sid**, which will also be its key and a foreign key

referencing `T_Student`. In case subclasses are disjoint or cover the super-class, appropriate SQL assertions need to be added to check these constraints.

For a relationship set  $R$ , `T_R` has as attributes all the attributes of  $R$ , as well as the union of the sets of attributes  $X_O = key(T_O)$  of all object sets  $O$  participating in  $R$ ; for each such set of attributes  $X_O$ , a foreign key constraint from `T_R` to `T_O` is added, in order to avoid dangling references. Moreover, a `NOT NULL` constraint is added to the corresponding columns of  $X_O$ . The keys of the tables generated from the object sets in  $id(R)$  jointly become the key of `T_R`. For example, suppose that `admits` is defined as a relationship set with participating entity sets `Professor` and `Student`, plus attribute `Year`. Assuming that  $id(Professor) = pid$ , then the relational representation of `admits` would be `T_admits[pid,sid,year]`, with `pid` and `sid` being foreign keys that reference `T_Professor` and `T_Student`, respectively. Additional constraints found in some EER schemas, such as numeric lower and upper bounds on relationship participation (e.g., a child has between 2 and 2 parents), can only be enforced using general SQL assertions.

For a weak entity set  $W$ , `T_W` includes the attributes of  $W$ , any attributes of the identifying relationship set of  $W$ , and the identifying attributes  $key(T_E)$  of the entity set  $E$  that “owns”  $W$ . The key of `T_W` is the union of the local identifier  $id(W)$  of  $W$  with the key attributes  $key(T_E)$  of  $E$ . A foreign key constraint from `T_W` to `T_E` must also be added. For example, suppose that `Section` is specified as a weak entity set with respect to `Course`, with local identifier `sectionNr` and identifying relationship set `sectionOf`; furthermore, assume that  $key(T\_Course) = courseNr$ . Then, the relational representation of `Section` would be `T_Section[courseNr,sectionNr]`, with `courseNr` as a foreign key referencing `T_Course`.

## Refinements

In the creation of tables for relationship sets and for weak entity sets, special attention needs to be paid when duplicate column names arise (because the same object can be involved in a relation in multiple ways). In this case, role names need to be used to disambiguate the columns. For example, if  $id(Professor) = id(Student) = ssn$  (because both are subclasses of `Person` say), then `T_admits` should have columns named `prof_ssn` and `student_ssn`, or `admitter_ssn` and `admitted_ssn`.

The treatment of subclass hierarchies which are not trees and where subclasses may inherit different identifiers from different parents also requires special care, and is discussed in [5].

The previous constructions provides a relational schema that allows every instance of the conceptual schema to be captured precisely. Relational schema restructuring by relation merging and sometimes partitioning is then undertaken for a number of reasons. The cardinal rule of all techniques is the need to be able to recover precisely the original relation instances from the merged instance (viz. lossless join). An additional rule observed in the techniques here is to avoid duplicating information in ways that lead to “update anomalies”. (See FD entry).

## Table merging

The most familiar technique replaces tables by their outer join, with the goal of making it easier to express and evaluate queries by avoiding the join. For example, one can merge table `student[sid,name,age]` with `minorsIn[sid,minor]`, to obtain `student2[sid,name,age,minor]`. Conceptually, such a change usually merges a functional relationship with the entity it is about, or a sub-class into its super-class. The main disadvantage of this change is the need to store null values as part of the outer join (in the above example, for students who do not have a minor).

The rule being applied in this case can be stated as:

*Rule 1:* Tables  $T[\underline{KX}]$  and  $R[\underline{\hat{K}Y}]$ , where  $\hat{K}$  is a foreign key referencing  $T$ , can be replaced by table  $T_R[\underline{KXY}]$ , whose intended use is as the left outer join of  $T$  and  $R$ .

Applying the above transformation one must be mindful of a number of potential problems. First, identical column names occurring in  $X$  and  $Y$  cause conflicts, which must be avoided by renaming. Second, this merge may prevent making previously possible distinctions when there is no guarantee that for every tuple in table  $T$  there is a corresponding tuple in  $R$  referencing it. For example, in the original schema one can distinguish the case of a student who has a minor that is not known (represented by a tuple with NULL in column `minor` of table `minorsIn`) from the case of a student without a minor (represented by the absence of a the student's `sid` in `minorsIn`); but in `student2` both cases have NULL in the `minor` column. To capture such distinctions one can add a Boolean attribute `isInMinorsIn?` to table `student2`. This technique becomes essential when merging the table of a subclass (e.g., `T_GradStudent`) into that of the superclass (e.g., `T_Student`) in case all the additional attributes of the subclass (e.g., `advisor`) may have null values, and therefore cannot be used to detect membership in the subclass.

Third, previously easy-to-state constraints may now become more convoluted. For example, if graduate students must have both an `advisor` and a `department` attribute (hence these columns have NOT NULL constraints in `T_GradStudent`), then since nulls must be allowed into these columns of `T_Student` after the merge, one must ensure (using an SQL check constraint) that NULLs in the two columns *correlate*. Moreover, foreign key references to table `T_GradStudent` now become references to `T_Student`, and must be supplemented by SQL assertions verifying that some attribute associated with `GradStudents` has a non-null value.

### Table partitioning

Tables can also be reorganized by so called “horizontal splitting” as stated by the following rule:

*Rule 2:* Table  $T[\underline{KX}]$  can be replaced by tables  $T_1[\underline{KX}], T_2[\underline{KX}], \dots$ , with the intended use of  $T_1, T_2, \dots$  being as a partition of the tuples in  $T$ .

In logical (as opposed to physical) schema design, the partition tables usually have semantic interpretation as subclasses of relationship sets (e.g., `T_admitted` replaced by `T_currentlyAdmitted` and `T_previouslyAdmitted`) or of entitie sets (e.g., `T_Student[sid,name,age]` replaced by `gradStudent0[sid,name,age]`, `undergradStudent0[sid,name,age]`, and `student0[sid,name,age]`, — the latter with students that are neither undergraduate nor graduate). Note that after such a split, one can merge `gradStudent0[sid,name,age]` with `gradStudent[sid,office]` to get another variant of mapping subclass hierarchies to tables; this one is particularly good for cases when the subclasses cover the superclass.

Table partitioning can be seen as encoding into each table selection criteria, which therefore once again facilitates query statement and evaluation. The down side is that once again built-in constraints, such as foreign keys, now need to be stated as more complex SQL inter-table assertions.

### Mapping from non-ER Conceptual Schemas

UML class diagrams are increasingly popular for the specification of conceptual schemas. The correspondences “class”  $\Leftrightarrow$  “entity set”, “association”  $\Leftrightarrow$  “relationship set” make it easy to reformulate the E2R mapping as a UML-to-Relational (U2R) mapping. Higher arity relationships (such as `assignedTo(Professor, Course, Semester)`) need to be reified in UML (i.e., represented as classes of objects `Assignment` related by functional associations  $f_1, f_2$  and  $f_3$  to `Professor, Course`

and *Semester* respectively), but these end up producing a similar relational schema as E2R because tables  $T_{f_i}$  are merged into  $T_{\text{Assignment}}$ ; the artificial key  $id(\text{Assignment})$  should however be removed.

As in the above example, the main difficulty in U2R is dealing with identifying (“key”) attributes for entities, which are not mandated by the UML data model, since it assumes objects have intrinsically unique identity.

## KEY APPLICATIONS

### *Database design*

The mapping of the conceptual schema into a logical schema is the central step of the database design process. A carefully crafted mapping will guarantee that the logical schema correctly represents the application domain that the conceptual schema models.

## FUTURE DIRECTIONS

The publication of the SQL:2003 language standard provides additional mapping opportunities, exploiting some of the the object-oriented features of the language.

As a simple example, SQL:2003 introduces the **MULTISET** data type, which can be used to avoid defining a separate table to accommodate multi-valued attributes of entity or relationship sets.

More importantly, SQL:2003 supports the declaration of an “*identity attribute*” for a table (designated with the special keyword **IDENTITY**). The value of an identity attribute is unique and automatically generated whenever a new row is inserted into the table. This construct is useful to the E2R, and especially U2R mapping process since it avoids the selection of an *artificial* key for entity  $E$  by adding an identity attribute for table  $T_E$ .

## URLs

DBDesigner (<http://fabforce.net/dbdesigner4/>) is an open-source database design system, available from fabFORCE.net, which integrates EER modeling with the derivation and maintenance of a relational schema for **mySQL**.

## CROSS REFERENCES

“Information Capacity”, “Extended Entity Relationship model”, “Functional dependency”.

## References

- [1] Marco A. Casanova, Luiz Tucherman, Alberto H. F. Laender: “On the Design and Maintenance of Optimized Relational Representations of Entity-Relationship Schemas”. *Data and Knowledge Engineering* 11(1): 1-20 (1993)
- [2] Peter P. Chen: “The Entity-Relationship Model: Toward a Unified View of Data”. *Proceedings of the First Int. Conf. on Very Large Data Bases*: 173 (1975)
- [3] Peter P. Chen: “The Entity-Relationship Model - Toward a Unified View of Data”. *ACM Trans. on Database Systems* 1(1): 9-36 (1976)
- [4] Altigran Soares da Silva, Alberto H. F. Laender, Marco A. Casanova: “An Approach to Maintaining Optimized Relational Representations of Entity-Relationship Schemas.” *Proc. 15th Int. Conf. on Conceptual Modeling (ER'96), LNCS 1157*: 292-308 (1996)
- [5] Altigran Soares da Silva, Alberto H. F. Laender, Marco A. Casanova: “On the relational representation of complex specialization structures.” *Information Systems* 25(6-7): 399-415 (2000)
- [6] Victor M. Markowitz, Arie Shoshani: “Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach”. *ACM Trans. on Database Systems* 17(3): 423-464 (1992)
- [7] Toby J. Teorey, Dongqing Yang, James P. Fry: “A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model”, *ACM Computing Surveys* 18(2): 197-222 (1986)