Chapter V

Mapping Class Diagrams to Relational Schemas

This chapter first explains the need to map a class diagram to a relational schema. Then, most of the chapter is dedicated to presenting and demonstrating the mapping rules based on which a relational schema (made of normalized relations) is created. The mapping process is demonstrated with several comprehensive examples.

Why Map a Class Diagram to a Relational Schema?

The objects model can serve two purposes: (1) as a mean for planning, that is, to create a conceptual data model of the reality (this model, a class diagram, can be used to communicate between the users and the analysts/designers, similar to the role of an entity relationship diagram [ERD]), and (2) as a means for implementation, that is, the class diagram can be converted to an objects schema and then implemented in an object oriented database management system (OO-DBMS), as a substitute for a relational DBMS.

However, OO-DBMSs are not in widespread use. Although various OO-DBMSs have been developed and marketed (mainly during the 1990s, and mostly by small companies who tried to commercialize ideas created by researchers), they were not very successful. In fact, relational DBMSs continue to dominate the market. A reasonable explanation to this fact is that the big companies in the
DBMS market invested huge amounts on their relational systems and have many customers, and there seems to be little reason to abandon these investments and risk their customer base with the new technology. Another possible explanation is that for data processing systems (i.e., systems serving the business organizational world) there is no real need for a data model which is so different from the relational model. Despite claims regarding limitations of the relational model (see discussion in Chapter I) it seems that the model is adequate for most business-oriented data processing needs. Rather than seeing OO-DBMSs replacing the relational systems, we see that relational DBMSs are being enhanced with certain features of the objects model, thus enabling them to provide specific needs which they were unable to fulfill according to the “pure” relational model. The “new generation” relational systems are called object relational DBMS (OR-DBMS). Here are a few typical object oriented extensions to relational DBMSs:

- **Array (or multi-valued attribute):** A field (attribute) does not need to be “atomic” (meaning, contain only one value), but it can contain several values. For example, in a Students relation, it is possible to define an attribute array phone numbers (3), which allows us to save up to three phone numbers for each student (instead of having a separate relation for student phone numbers).

- **Structure (or nested relation):** It is possible to group several attributes in a relation and define them as a subrelation (subrecord, to be exact). This extension enables us to implement set attributes in a relation. For example, in the Students relation, it is possible to define an attribute nested relation registration to courses (course code, grade). This enables each student’s record to contain a subrecord of the student’s course registrations (instead of having a separate relation for course registrations).

- **Methods (or procedures):** In addition to attributes and “simple” constraints (e.g., key, unique, not-null) this extension enables to define methods which perform more complex constraint-checking operations on attributes in one or more relations. (This is actually an extension of the “check” procedure in relational DBMSs.)

- **Internal ID for a structure:** By analogy to object identification (OID) in the objects model, an OR-DBMS enables us to assign an internal ID to a structure (i.e., a subrecord). This enables us to save the subrecord apart from the main record (e.g., for the sake of efficiency), and use the internal ID to refer to it (like a reference attribute in the objects model). For example, instead of saving the nested relation registration to courses along with all other attributes of a student, it is possible to save it separately and refer to it (thus enable locating it) using its ID.
Related Content

Managing Complexity with MDSD
www.igi-global.com/chapter/managing-complexity-mdsd/25637?camid=4v1a

Improving OO Design Process Using Rules, Patterns and Refactoring
www.igi-global.com/chapter/improving-design-process-using-rules/25643?camid=4v1a

Planning and Managing the Human Factors for the Adoption and Diffusion of Object-Oriented Software Development Processes
www.igi-global.com/chapter/planning-managing-human-factors-adoption/25639?camid=4v1a

Design Patterns as Laws of Quality
www.igi-global.com/chapter/design-patterns-laws-quality/27360?camid=4v1a