

Comp2400/Comp6240

Relational Databases

Lab 2

August 6, 2007

This is a slightly enhanced version of Exercise 3.32 from [EN07] (It's only in the 5th edition, not the 4th). The goal is to design and validate a database schema by modelling the data requirements then translating the model.

Note that the data requirements as presented almost directly specify a relational schema. As a result, the modelling process might not seem very helpful. When you start with much less well analysed and structured requirements, the value of modelling will be clearer. This exercise is to familiarise you with the process and language before tackling the similar but more difficult Assignment 1.

1 Data Requirements

Consider a `MAIL_ORDER` database in which employees take orders for parts from customers. The data requirements are summarized as follows:

- The mail order company has employees, each identified by a unique employee number, first and last name and ZIP [post] code.
- Each customer of the company is identified by a unique customer number, first and last name and ZIP code.
- Each part sold by the company is identified by a unique part number, a part name, price and quantity in stock.
- Each order placed by a customer is taken by an employee and is given a unique order number. Each order contains specified quantities of one or more parts. Each order has a date of receipt as well as an expected ship date. The actual ship date is also recorded.

2 Exercises

Work in pairs to complete the following tasks.

1. Complete the following scenario by adding (made up) details as specified in the general description above.

Employee Smith takes an order from customer Jones for 14 widgets and a whacking-mallet. There are no whacking-mallets left, so Smith phones the supplier who promises to deliver some the next day. When they arrive a week later, Smith ships the order.

2. Create a UML object diagram for this completed scenario.
3. Create a UML class diagram. The diagram should faithfully capture the data requirements above, and the object diagram should be an instance of it.
4. Ensure the class diagram is complete, following the procedure given in lecture 8 (see below).
5. Translate the class diagram into a database schema following the naive procedure given in lecture 8 (see below).
6. Create the database in PostgreSQL, and populate it with the scenario described by your object diagram.
7. Write SQL queries to determine
 - (a) Who works for the company?
 - (b) How many customers does the company have?
 - (c) Which orders were delivered when they were expected to be.
 - (d) What is the retail value of the parts in stock?
 - (e) Develop 2 more relevant business related questions, and code SQL queries to answer them.
8. (*optional*) Translate the class diagram again, using the better translation procedure (see below). Populate and modify or recode the queries. Note the differences, how has this changed them?

A Class Diagram Completion

1. For each class, is there a natural primary key? If so, underline the attributes, if not, add an artificial identifier.
2. For each attribute, define an appropriate domain or choose an appropriate datatype.
3. Ensure that each association end has an appropriate name (role name).

B Naive Class Diagram Translation

1. For each domain, create a database domain.
2. For each class, create a relation schema with
 - (a) a relation attribute for each class attribute
 - (b) specify the primary key, as underlined in the diagram
3. For each association, create a relation schema with
 - for each association end, create relation attributes for the primary key attributes of the class at that end.
 - If it is a single field key, use the association end name.
 - Otherwise, prefix each end class key field name with the association end name.Define these as foreign keys.

C A Better Class Diagram Translation

- Translate (UML) domains and classes as for the naive translation.
- For each association
 - If both ends have multiplicity greater than 1, translate as for the naive translation.
 - If both ends have multiplicity 1 (ie 1..1), merge the relation schemas of the classes at the associations ends. If the two relation schemas have attribute names in common, rename.
 - If exactly one end has multiplicity 1, add a non-null foreign key to the relation schema for the far end class. (The type of the foreign key is primary key of the relation schema of this end.)
 - For an association to which none of the previous three conditions apply, it has at least one end with multiplicity 0..1. Select one such end, and add a foreign key (nulls allowed) to the relation schema for the class at the far end.

References

- [EN07] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 5th edition, 2007.