MANY-TO-MANY

CECS 323

Many-to-Many – Design Pattern

Design pattern: many-to-many (order entry)

There are some modeling situations that you will find over and over again as you design real databases. We refer to these as **design patterns.** If you understand the concepts behind each one, you will in effect be adding new "tools" to your design toolbox that you can use in building the model of an enterprise.

Many-to-Many Relationships

Introduction

As we learned in the article on associations, objects interact with each other and we use relationships in our model of these objects to capture these interactions. In that section we learned about 1-to-many relationships. There are also many-to-many and one-to-one relationships; as you can see these three types of relationships are named after the cardinality constraints. There is a fourth type, the many-to-one relationship, however, that is just the reverse direction of the one-to-many relationship. This article introduces a design pattern for the many-to-many relationship, and shows the UML class diagram which is then mapped to the relational model.

Sales DB Example

The sales database

So far our sales database has customers and orders. To finish the design pattern, we need products to sell. We'll first describe what the Product class means, and how it is associated with the Order class. Below is how Product might be described

> "A product is a specific type of item that we have for sale. Each product has a descriptive name; we distinguish similar products by the manufacturer's name and model number. For each product, we need to know its unit list price and how many units of this product we have in stock. "

It is important to understand exactly what is meant by a product so that it can be modeled correctly. An example product (an instance of the Product class, a Product object) might be named "Blender, Commercial, 1.25 Qt.", manufactured by Hamilton Beach, model number 908. This is a type of product, not an individual boxed blender that is sitting on our shelves. The same manufacturer probably has different blender models (909, 918, 919), and there are probably blenders that we stock that are made by other companies. Each such blender would be a different instance of this class.

The following would be a typical description of how products relate with orders.

> "An order can include many products, though a product will only be in an order once. Each time an order is placed for a product, we need to know how many units of that product are being ordered and the price we are actually selling the product for. The latter is needed as the sale price might vary from the list price by customer discount, special sale, price changes, etc. "

More about Order

The following would be a typical description of how products relate with orders.

> "An order can include many products, though a product will only be in an order once. Each time an order is placed for a product, we need to know how many units of that product are being ordered and the price we are actually selling the product for. The latter is needed as the sale price might vary from the list price by customer discount, special sale, price changes, etc. "

From the above description, we can extract the following sentences which focus on the constraints that will need to be modeled.

- "Each Order must contain one and at most *many* Products." We use the verb "must" here because it doesn't make sense to have an order with no products; that is, for the order to exist it is mandatory for it to contain at least one product. The rest tells us that an order can contain many products.
- "Each Product may be contained in *many* Orders". Here, we use the verb "may" because a product can exist in the database and not have been ordered by anyone, so it's optional for it to be in an order. Of course, we need to allow a product to be in many orders.

Result of The Association

There are significant differences in this modeling problem compared to what we have seen before; these are listed below.

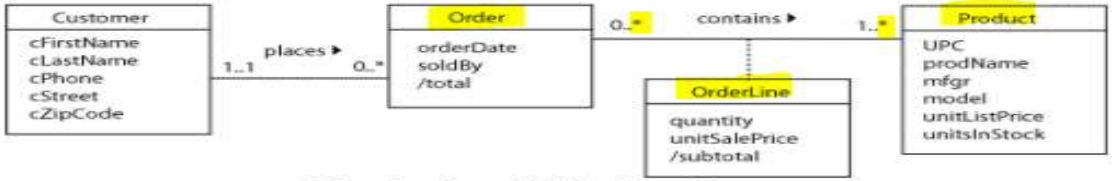
- The maximum multiplicity (i.e. the cardinality) in each direction is "many", this is called a **many-to-many** relationship between Orders and Products. We will see how to model this in UML and then how to map this to the relational model.
- There are two attributes described, the sale price and the quantity of the product being ordered, that cannot possibly be attributes of either the Products or Orders classes. These attributes are a result of the association between the Order and the Product, they describe each individual instance of the association. Below, we learn how to model such attributes in UML and then how to implement these in the relational model.

Class Diagram for Many-to-Many

Class diagram

The many-to-many relationship will be modeled with an association that specifies a many cardinality in the multiplicity constraints of both ends of the association.

UML gives us a way to model relationship attributes by showing them in an **association class** that is connected to the association by a dotted line. This is shown in the figure below with the OrderLine association class and the two attributes quantity and unitSalePrice as described above. If there are no attributes that result from an association, there is no association class. Association classes can be attached to associations of different cardinalities, not just many-to-many associations.



Sales database UML class diagram.
Other views of this diagram:
Large image - Data dictionary (text)

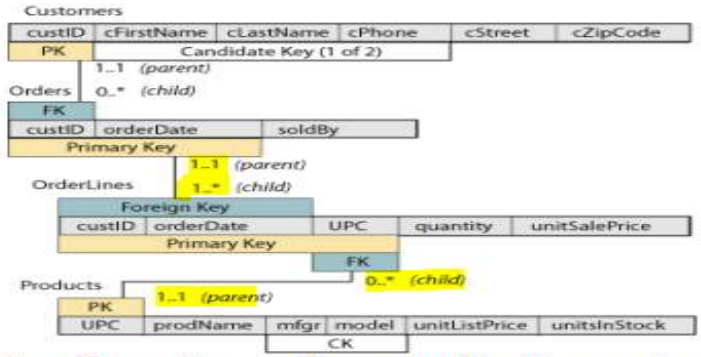
There are two additional attributes shown in the class diagram that we haven't talked about yet. We need to know the subtotal for each order line (that is, the quantity times the unit sale price) and the total dollar value of each order (the sum of the subtotals for each line in that order). As just described, the values of these two attributes can be computed by using values in other attributes; in the class diagram we precede their name by a "/" to denote that they are **derived attributes**.

Since these values can be computed, they don't need to be stored in the database, and they are not included in the relation scheme.

Relation Scheme Diagram

Relation scheme diagram

We can't represent a many-to-many association directly in a relation scheme, because two tables can't be children of each other—there are no parent/child roles that can be assigned as both cardinalities are many. So for *every* many-to-many association, we will need an additional table, a **junction table**, in the database that will store all the relationship instances. The relation scheme of such junction tables are also part of the database scheme diagram. The scheme of a junction table (sometimes also called a join table or linking table) will contain two sets of FK attributes, one for each side of the association. If there is no association class, then the scheme only contains these two FKs. If there is an association class (as there is here with OrderLines), its attributes will go into the junction table scheme.



Sales database relation scheme diagram demonstrating the mapping of a many-to-many association and association class.

Other views of this diagram:

Large image - Data dictionary (text)

English Description

In the relational model, the many-to-many association between Orders and Products has turned into a one-to-many relationship between Orders and Order Lines, plus a many-to-one relationship between Order Lines and Products. However, this does not mean the UML class diagram can be updated to show a one-to-many association and a many-to-one association with the respective classes. The UML class diagram given above is the correct model for the problem. The list below provides the English description matching these two relational model relationships along with what the constraints say about the data in those tables

- **Each Order must be related to *one and at most many* OrderLines**. This means every PK value in the Orders table must appear at least once as a FK value in the OrderLines table. If such a FK value appears more than once in the OrderLines table, it is because the order corresponding to it includes several products so there need to be several rows in OrderLines.
- **Each OrderLine must be related to *one and only one* Order**. These constraints mean that each of value of the FK {custID, orderDate} must refer to at most one row in the Orders table and that row must exist. This is the referential integrity constraint which databases enforce.
- **Each OrderLine must be related to *one and only one* Product**. This is another case of the referential integrity constraint, this time it has to do with the FK {UPC} in OrderLines referring to the PK {UPC} in Products.
- **Each Product may be related to *zero or many* OrderLines**. These constraints state that a product's PK value does not need to appear in the OrderLines table FK and that it could appear there many times. This is logical as a product might not have been purchased by anyone and it could be ordered by many.

Keys

Unique identifiers

The UPC is an external key. UPCs are defined for virtually all grocery and manufactured products by a commercial organization called the Uniform Code Council, Inc.® We will use it as the primary key of our Products table, which also has a candidate key here: {mfgr, model}.

With Orders now a parent of OrderLines, we might have decided that it needs a surrogate key (order number) to be copied into the OrderLines. In fact, most sales systems do this, as you know if you've ever tried to check on the status of something you've ordered from a company. For this example, it seems to be just as easy to stick with the existing PK of Orders, since it already has a surrogate key from Customers, and the order date doesn't add much size.

To uniquely identify each order line, we need to know both which order this line is contained in, and which product is being ordered on this line. The two FKs, from Orders and Products, together form the only candidate key of this relation and therefore the primary key. There is no need to look for a smaller PK, since OrderLines has no children.

Data Example

Data representation

The key to understanding how a many-to-many association is represented in the database is to realize that each row of the junction table (in this case, OrderLines) connects *exactly one* row from the left table (Orders) with *exactly one* row from the right table (Products). Each PK of Orders can be copied many times to OrderLines; each PK of Products can also be copied many times to OrderLines. But the *same* pair of FKs in OrderLines can only occur once. In the figure below we show only the PK and FK columns for sake of space.

Orders, OrderLines, and Products Tables highlighting the data linking them.

Other views of this diagram: