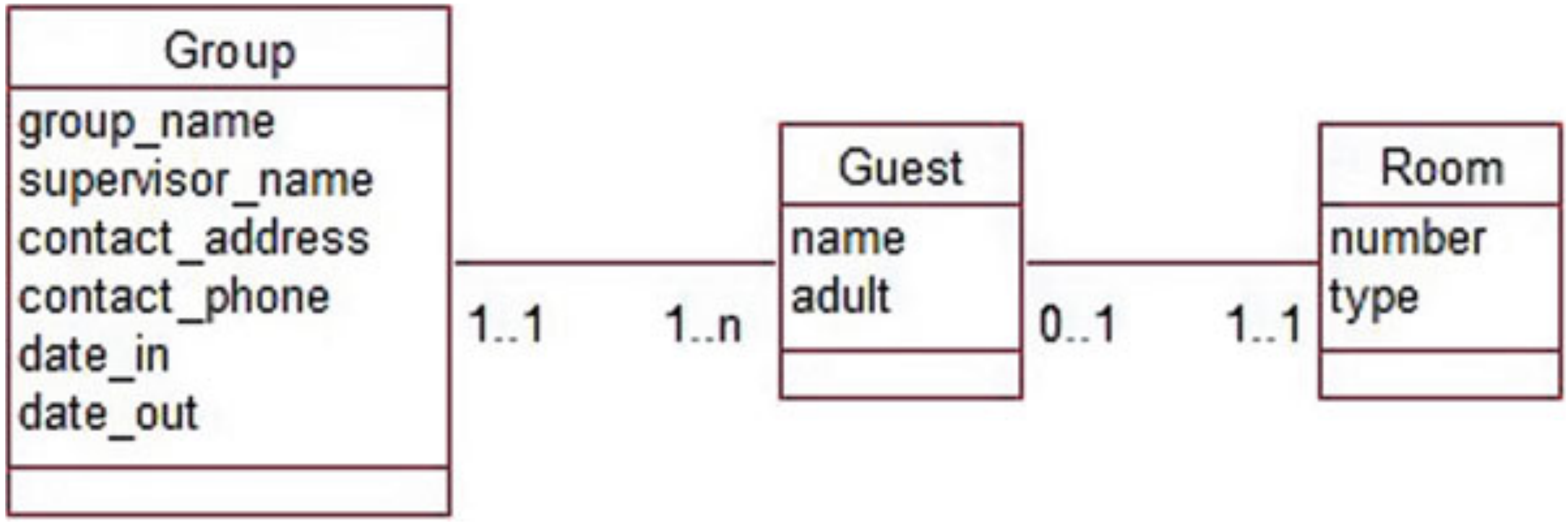
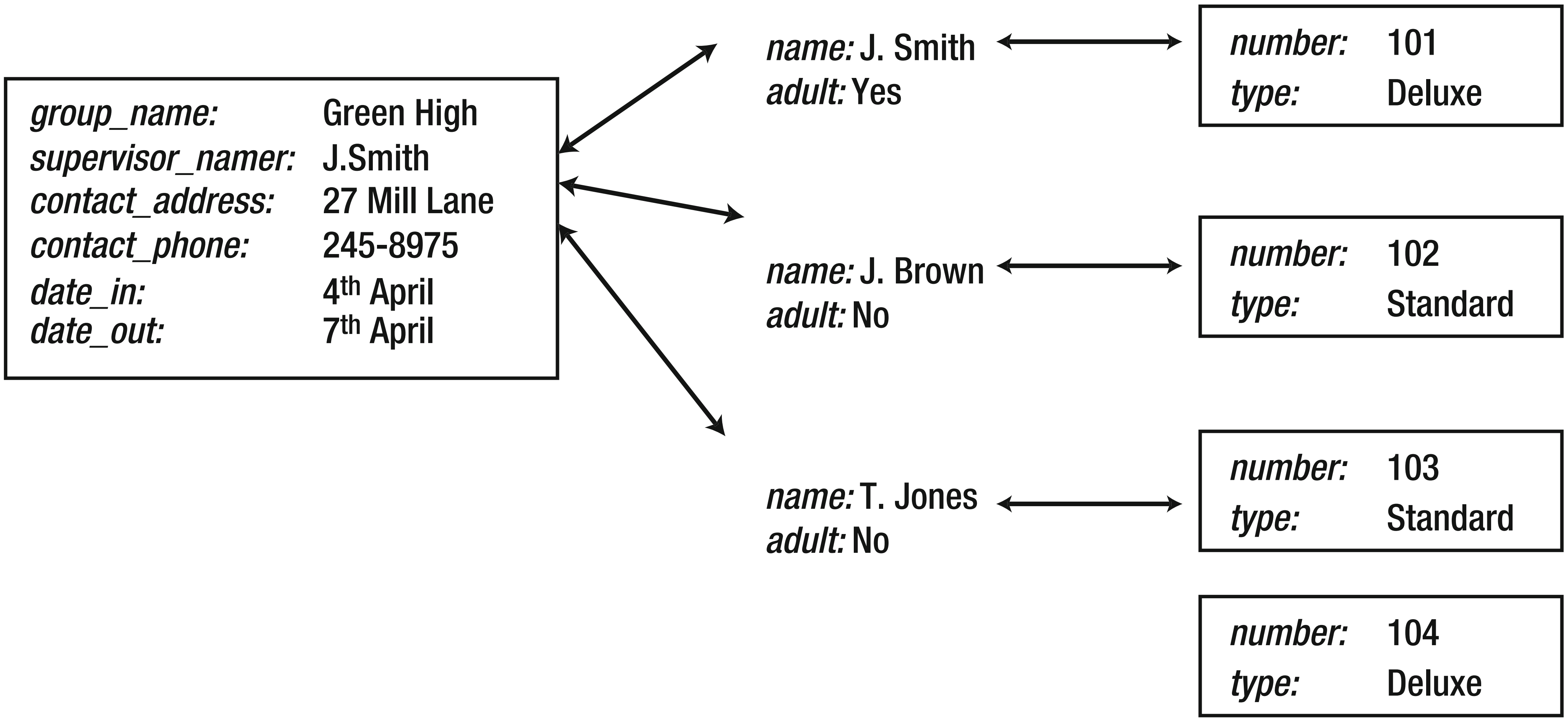
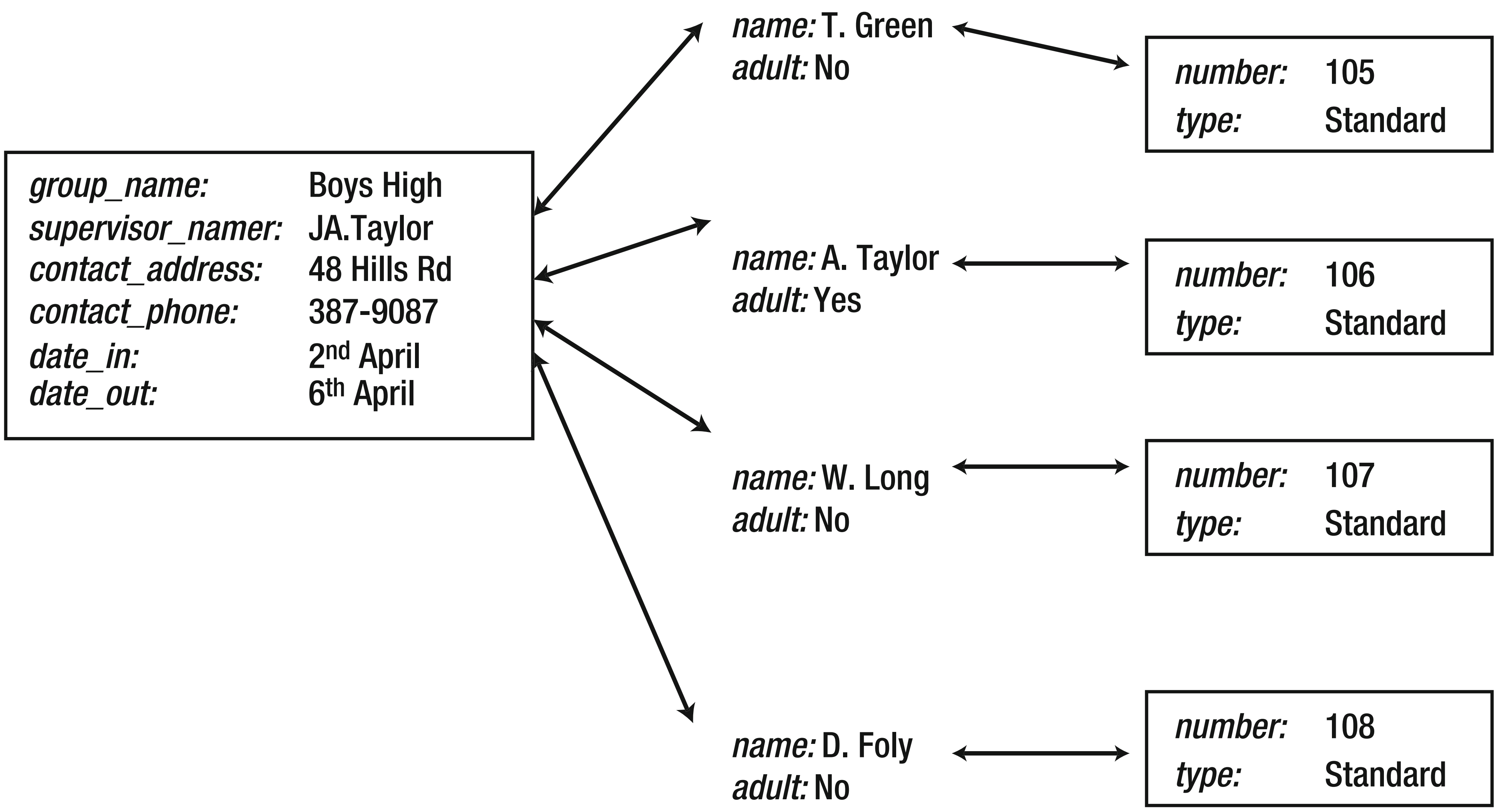


4-1. *Initial data model for the current occupancy of a small hostel*


group_name: Green High
supervisor_namer: J.Smith
contact_address: 27 Mill Lane
contact_phone: 245-8975
date_in: 4th April
date_out: 7th April
name: J. Smith
adult: Yes
name: J. Brown
adult: No
name: T. Jones
adult: No
number: 101
type: Deluxe
number: 102
type: Standard
number: 103
type: Standard
number: 104
type: Deluxe



4-2. *Objects and relationship instances consistent with Figure 4-1*

Student Course Example

Consider the data model in Figure 4-3, which shows a relationship between students and courses in which they enroll.

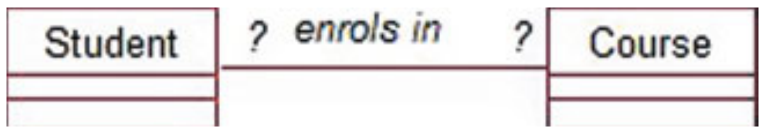


Figure 4-3. *Data model for students enrolling in courses*

Customer Order Example

Here is an easier example. (Or is it?) We keep information on customers and the orders they place. Our first instinct is to say that customers can place many orders and each order is placed by one customer. This can be represented as in Figure 4-4.

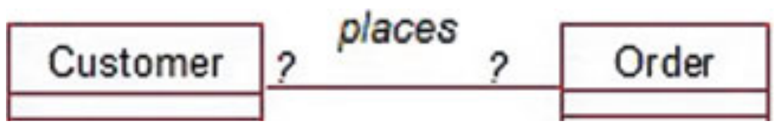


Figure 4-4. *Data model for customers placing orders*

Insect Example

Here is another example of how investigating the optionalities of a relationship can lead to questions about the scope of the problem. Figure 4-5 shows part of a possible data model from Example 1-3 in which farms were visited and several samples of insects were collected. A `Visit` object would contain information about the date and conditions of a particular visit and would be associated with several `Sample` objects. Each sample object would contain information about the number of insects collected.

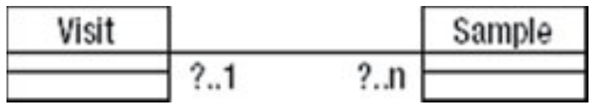


Figure 4-5. *Data model for collecting samples*

Insect Example

In the previous section we looked at the example of a scientist visiting a farm to collect insect samples. Some insects might behave differently if it is fine or raining so it may be important to record information about the weather when the sample was collected. To record the weather conditions consistently, the scientist may decide to choose from one of a number of categories. Introducing a `Weather` class with objects for the different conditions (e.g. fine, overcast, raining) can ensure that this information is recorded consistently. Part of a possible class diagram to represent the data is shown in Figure 4-6.

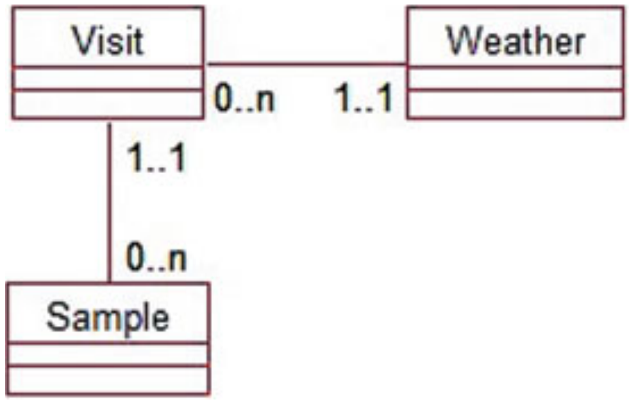


Figure 4-6. *Associating a weather category with a visit*

Reading the relationship between weather category and visit from left to right, it is reasonable that a visit will have one weather type that describes it, but there might also be occasions when a thunderstorm arrives while the last few samples are being collected. If so, do we care? The answer will, of course, depend on the client, but it is up to the analyst to ask the question and propose some possibilities.

At one extreme, the conditions under which each individual sample is collected may be vital. In this case, it might be more sensible to associate each sample with its own weather condition, as shown in Figure 4-7.

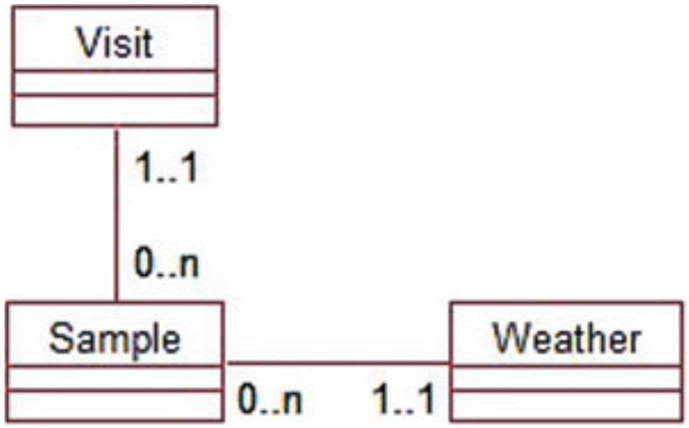


Figure 4-7. *Associating each sample with a weather category*

Sports Club Example

Here is another little snippet of a database problem. A local sports club may want to keep a list of its membership and the team for which each member currently plays (SeniorB, JuniorA, Veteran, etc.). One way to model this data is shown in Figure 4-8.

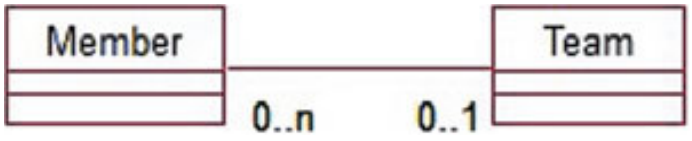
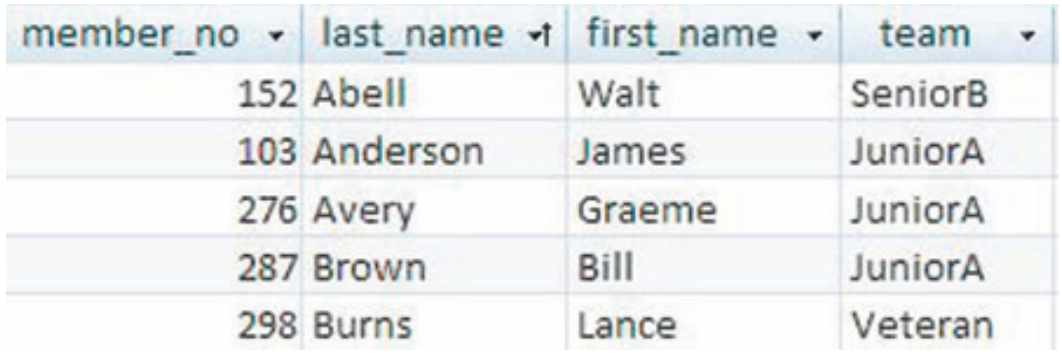


Figure 4-8. *Members and their current teams*

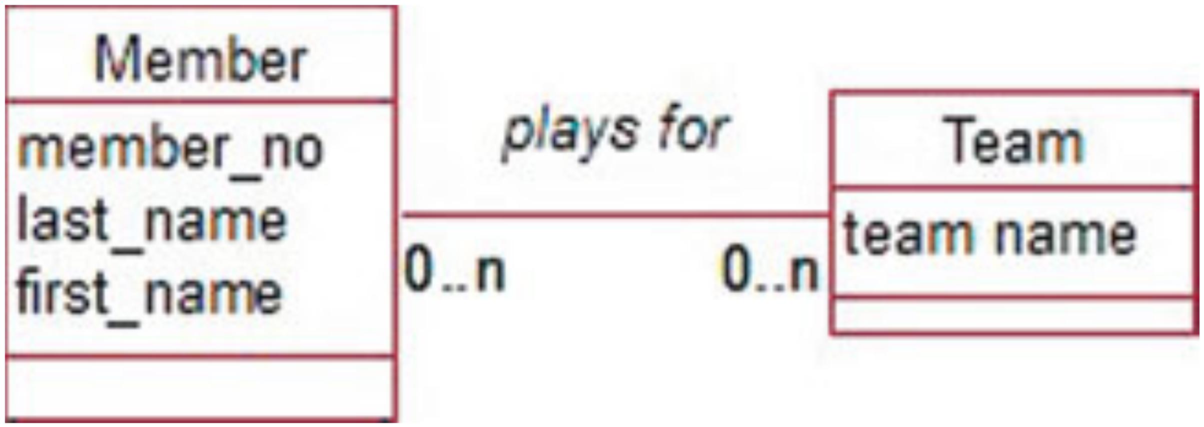
Sports Club Example

To illustrate how the sports club might lose its historical data, let's look at some simple data as it might be kept in a database table. If each member is associated with just one team, that team becomes a characteristic of the member, and the relationship can be represented as an attribute in the `Member` class as in Figure 4-9.



member_no	last_name	first_name	team
152	Abell	Walt	SeniorB
103	Anderson	James	JuniorA
276	Avery	Graeme	JuniorA
287	Brown	Bill	JuniorA
298	Burns	Lance	Veteran

Figure 4-9. *Members and their current teams*



4-10. *Members and the teams for which they play*

Departments Example

Figure 4-11 is an example that often appears in textbooks. Reading from left to right, we have that each department has one employee as its manager. But clearly this means one at a time. Over time, the department will have several different managers.

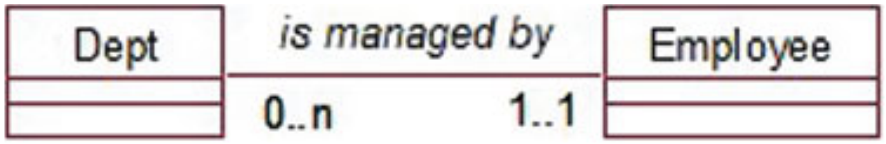


Figure 4-11. *Each department has a manager.*

The important question for this situation is, “Do we want to keep track of former managers?” Why are we keeping information about managers at all? If it is just to have someone to call when something goes wrong, probably the current manager is all that is required. However, if we want to know who was in charge when something went wrong last year, we will need to keep a history. The data model will need to change so that a department can be associated with several managers as in Figure 4-12.

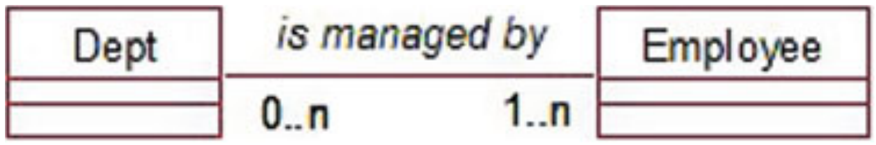


Figure 4-12. *A department has several managers over time.*

We will see in the next section how the introduction of an intermediate class will allow us to keep the dates for each manager.

Insect Example

Here is a real example of a problem arising in our scientific database of insect samples. To put the data in perspective, we need to know that the main objective of this long–term project was to see how the numbers of insects change as farming methods evolve over the years. The farms selected represented different farming types (organic, cropping, etc.). Throughout the duration of the project, each farm was visited several times to collect samples. Figure 4-13 shows part of an early attempt at a data model.

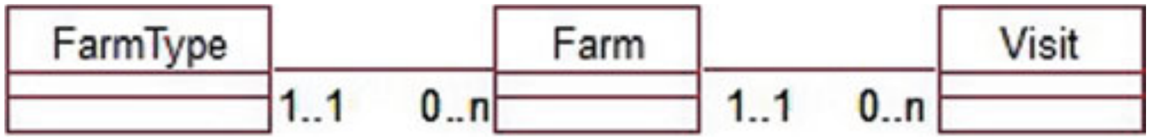


Figure 4-13. *Visits to farms of different types*

Is there any data that we need to record that depend on particular instances of each of the classes in our Many–Many relationship?

In this example, the question would be:

Is there any data that depend on a particular player and a particular team?

And the answer is:

Yes—the dates that player played for that team.

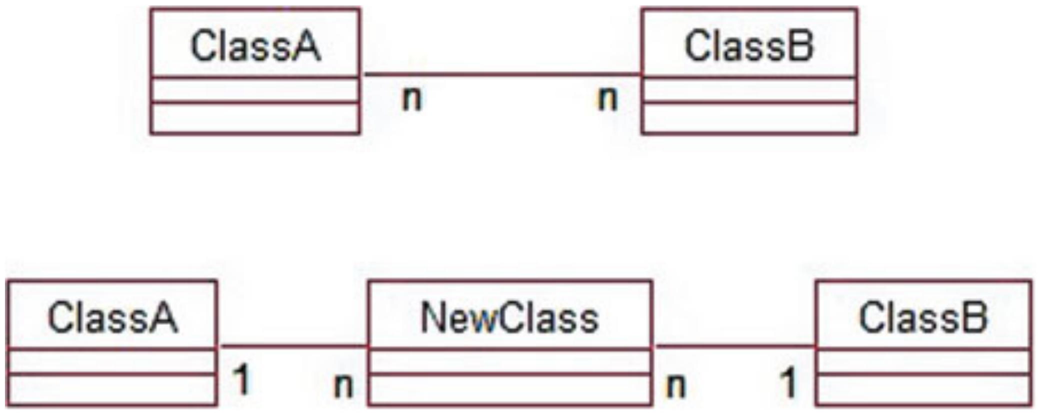


Figure 4-14. *Introducing a new class in a Many–Many relationship*

In situations where we have data that depend on instances of both classes in a Many–Many relationship, the Many–Many relationship is replaced by a new class and two 1–Many relationships. The many ends of the new relationships are always attached to the new intermediate class. We will see what this means for some of the examples we have already examined.

Sports Club Example

Let's reconsider the member and team problem. We'll put some attributes in the classes to make it clearer what information each is maintaining. The model is shown in Figure 4-15.

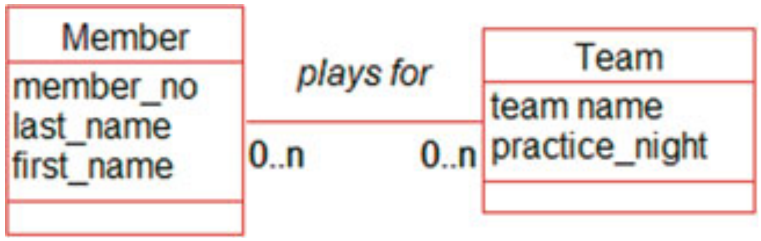


Figure 4-15. *Many–Many relationship between members and teams*

As we have already mentioned, the date that a particular member plays for a particular team cannot live in the `Member` class (because a member will play for many different teams over time) nor can it live in the `Team` class. Figure 4-16 introduces a new intermediate class, `Contract`, in the same way as was done in Figure 4-14.

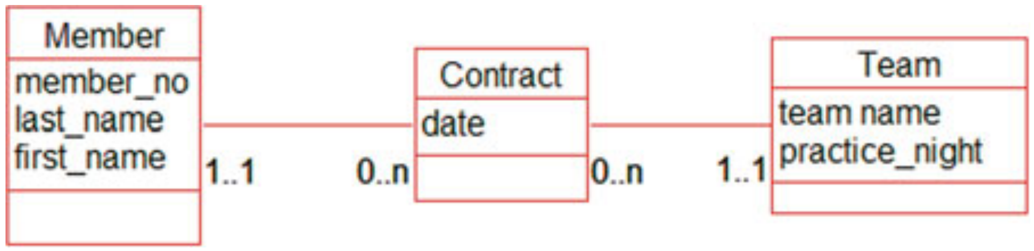


Figure 4-16. *Intermediate class, Contract, to accommodate the date a member played for a team*

Reading from the middle class outward, the model tells us that each contract is for exactly one team and exactly one member. Reading from the outside inward, we see that each member can have many contracts as can each team. Figure 4-17 shows some objects that might occur in such a data model.

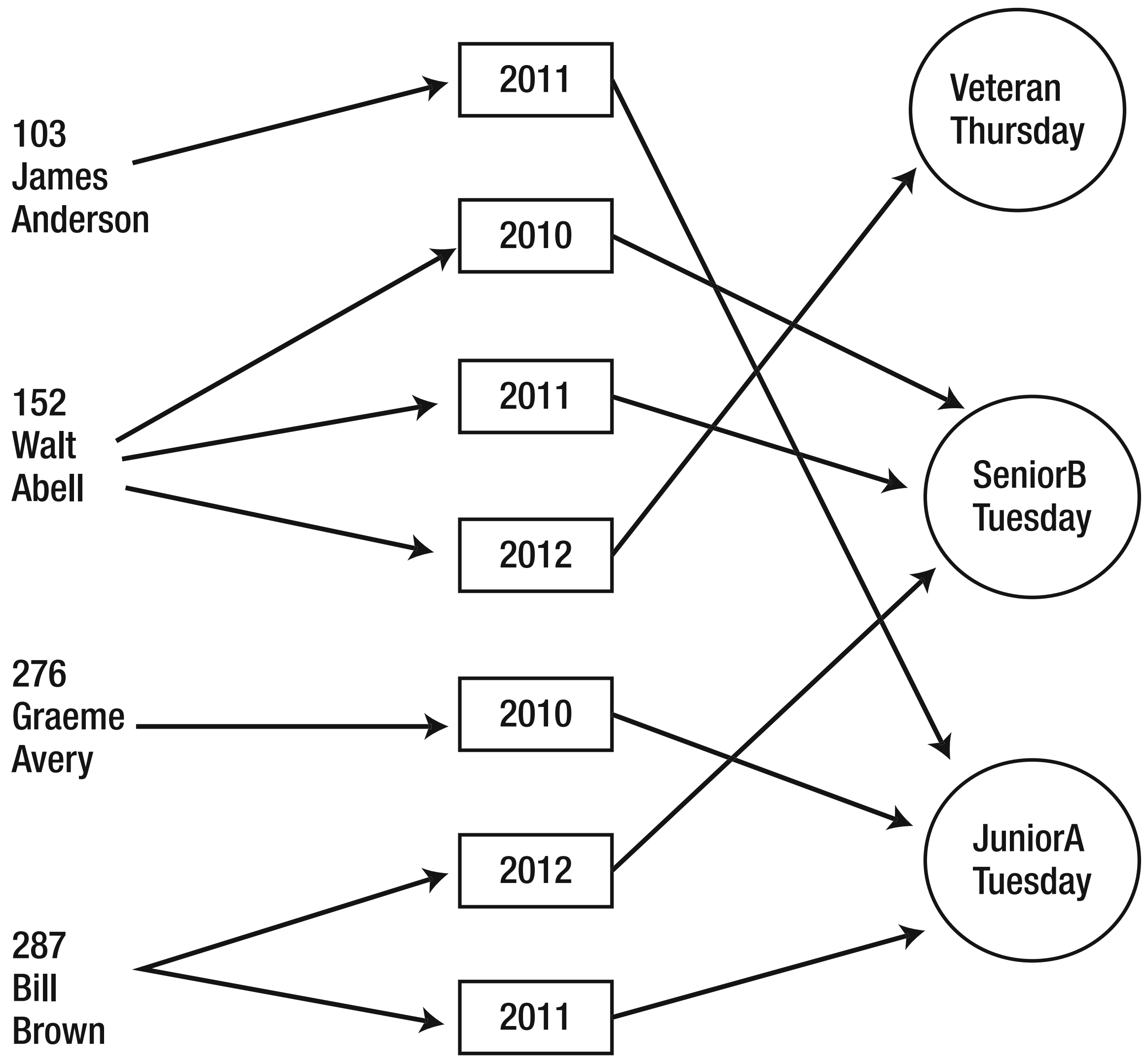
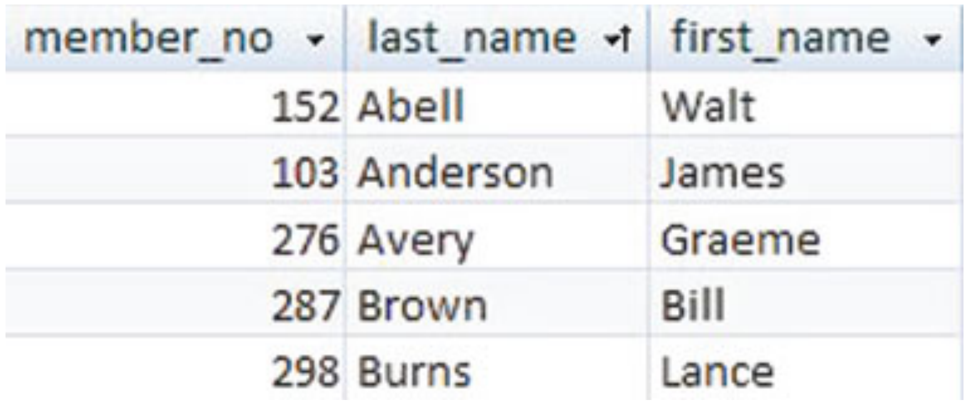


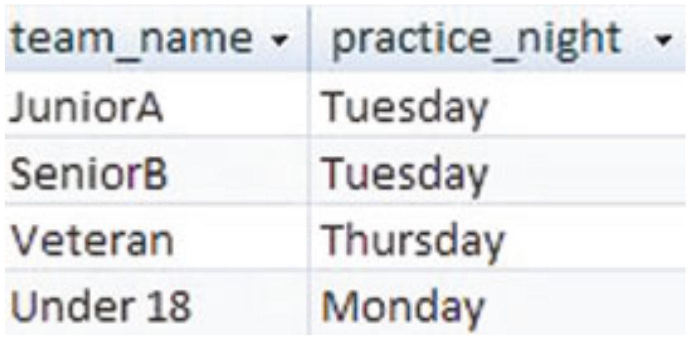
Figure 4-17. *Some possible objects of the Member, Contract, and Team classes*

We can now see what years members played for particular teams. We can see that Bill Brown (287) played for the JuniorA team in 2004 and for the SeniorB team in 2005. These data would be stored in database tables as shown in Figure 4-18.



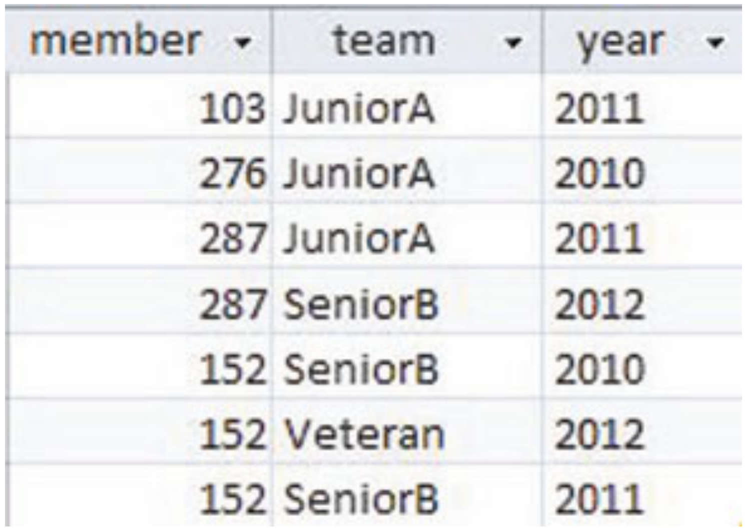
member_no	last_name	first_name
152	Abell	Walt
103	Anderson	James
276	Avery	Graeme
287	Brown	Bill
298	Burns	Lance

Member



team_name	practice_night
JuniorA	Tuesday
SeniorB	Tuesday
Veteran	Thursday
Under 18	Monday

Team



member	team	year
103	JuniorA	2011
276	JuniorA	2010
287	JuniorA	2011
287	SeniorB	2012
152	SeniorB	2010
152	Veteran	2012
152	SeniorB	2011

Contract

Figure 4-18. *Data for players, contracts, and teams*

Student Course Example

Let's now return to the Many–Many relationship of students enrolling in courses (Figure 4-3). This isn't just a historical problem, although we clearly will want to know when the student completed the course. But even if we were only keeping student enrollments for a single year or semester, we should still look to see whether there is missing information that might require an extra class. The question that needs to be asked is:

> *Are there any data that I want to keep that are specific to a particular student and his or her enrollment in a particular course?*

One obvious piece of data that fits the preceding criteria is the result or grade. Once again, we cannot keep the grade with the `Student` class (because it requires knowledge of which course) nor with `Course` class (because the grade depends on which student). In the same way as we dealt with this situation in Figure 4-14, we can introduce a new class, `Enrollment`, between the `Student` and `Course` classes as shown in Figure 4-19.

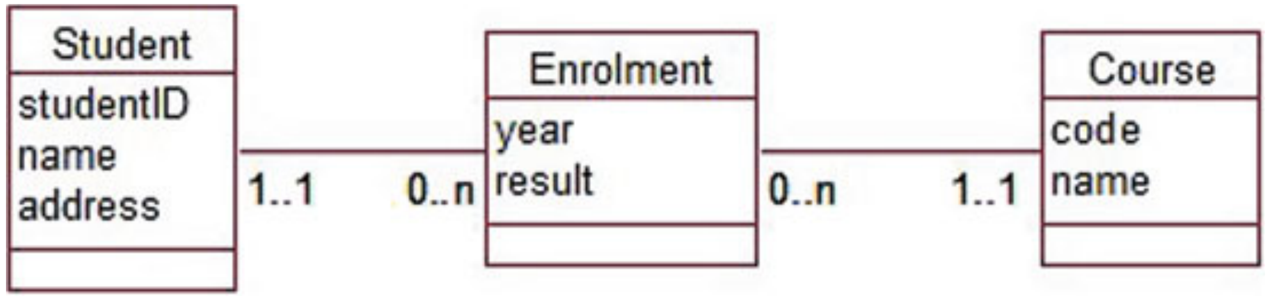


Figure 4-19. *Intermediate class to accommodate the result (and the year)*

Meal Delivery Example

As a final example of when we might need an additional class to keep information about a Many–Many relationship, let's look again at the meal delivery problem (Example 3-1) from the previous chapter. The initial data model had a Many–Many relationship between types of meal and orders. A particular type of meal (a chicken vindaloo, say) might appear on many orders, and a particular order may include many different meal types, as shown in Figure 4-20.

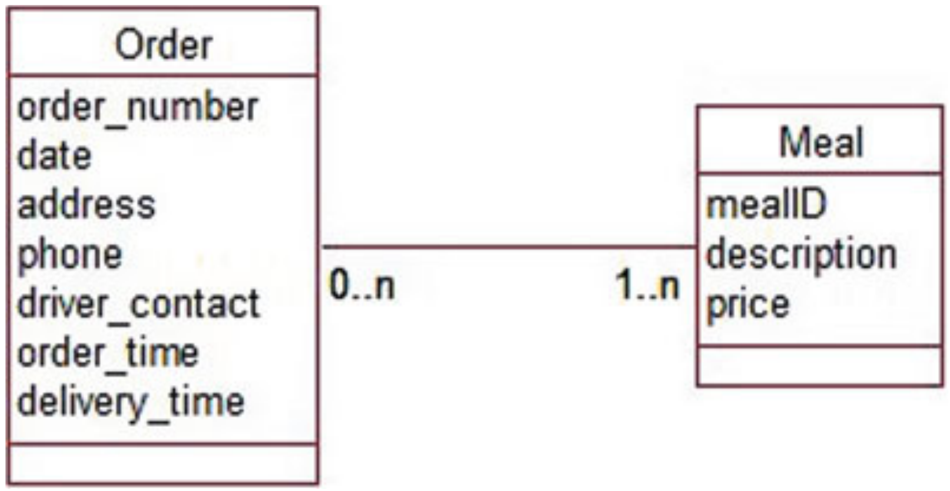


Figure 4-20. *Orders for different meal types*

Once again, our problem of where to put the additional data is solved by including a new class as shown in Figure 4-21.

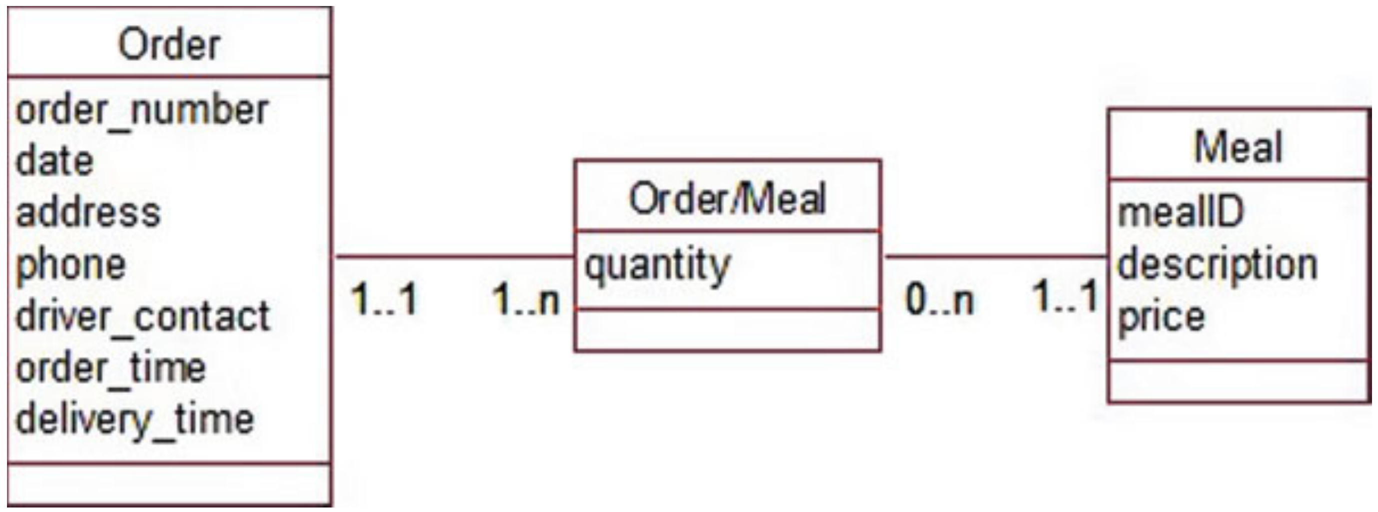


Figure 4-21. *Orders for different types of meal–with additional class to store quantities*

For some problems, it can be difficult to come up with a meaningful name for the intermediate class. In such a case, it is always possible to use a concatenation of the two original class names as we have done here with `Order/Meal`. We could maybe have called the class `Orderline`, in this example, as it represents each line in the order (i.e., a meal and the quantity). You might find it helpful to sketch some objects of the three classes in Figure 4-21 to clarify what is happening.

When a Many–Many Doesn’t Need an Intermediate Class

A few Many–Many relationships contain complete information for a problem without the need for an intermediate class in the data model. Problems that involve categories as part of the data often do not require an additional class. Example 1-1, “The Plant Database,” involved plants and uses to which they could be put. The original data model is repeated in Figure 4-22.

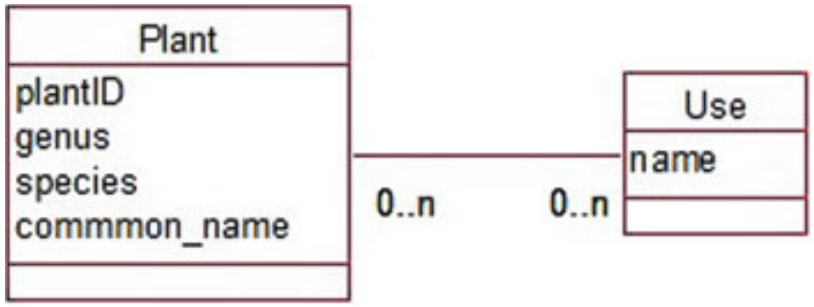
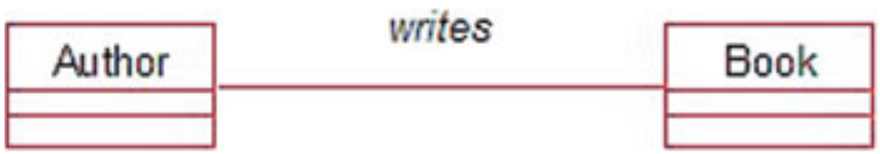


Figure 4-22. *Plants and their uses*

Exercise 4-1.

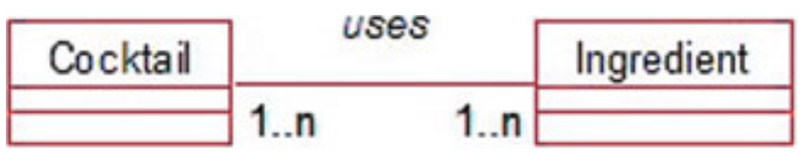
Figure 4-23 shows a first draft of modeling the situation where a publishing company wants to keep information about authors and books. Consider the possible optionalities at each end of the relationship `writes` and so determine some possible definitions for a book and an author.



ure 4-23. Consider possible optionalities for authors writing books.

Exercise 4-2.

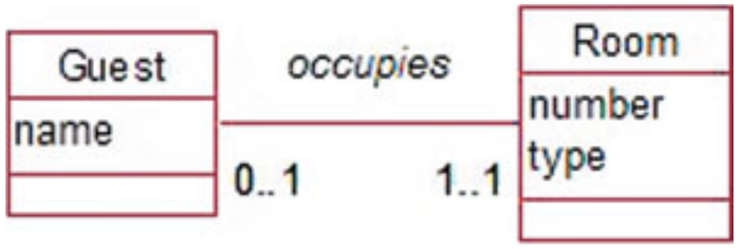
Figure 4-24 shows a possible data model for cocktail recipes. The Many–Many relationship `uses` can be navigated in either direction. To find out the ingredients in a Manhattan or to discover the possible uses for that bottle of Vermouth. What is missing?



ure 4-24. *Cocktails and their ingredients. What is missing?*

Exercise 4-3.

Part of the data model about guests at a hostel is shown in Figure 4-25. How could the model be amended to keep historical information about room occupancy?



ure 4-25. *How could this be amended to keep historical information about room occupancy?*