10. From Data Model to Relational Database Design

Fall 2018

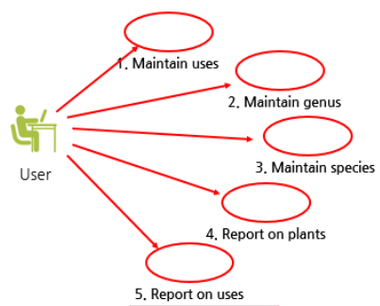
- We started with use cases to describe the basic requirements of a problem and developed an initial data model.
- By looking carefully at the details of the model, we were able to develop questions to help understand further subtleties and complexities of the real-world problem.
- We then looked at a number of situations that occur in many models in the hope that these would be useful when difficult situations arose in other contexts.
- The outcome we are seeking is agreement on a model that accurately reflects the essential requirements of the real-world problem.
- This will involve numerous iterations as the use cases adjust to reflect the improved understanding and the changing scope.
- Having arrived at a set of use cases and a data model with which everyone is comfortable,

analysis →

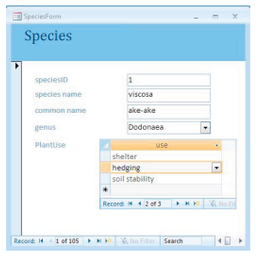
Problem

Real world

Abstract world

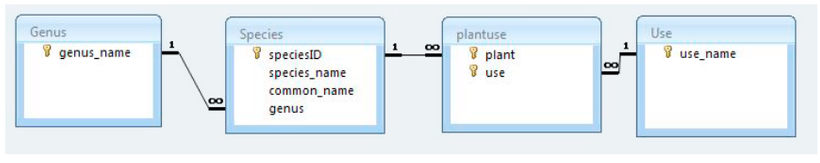


Solution



PlantUse

Use	ID	Genus	Species Name	Common Name
bird food				
	4	Acacia	melanoxylon	Tasmanian blackwood
	7	Alnus	incana	grey alder
	28	Eucalyptus	nicholii	Black peppermint gum
coppicing				
	30	Eucalyptus	gunnii	cider gum
	4	Acacia	melanoxylon	Tasmanian blackwood
	28	Eucalyptus	nicholii	Black peppermint gum
firewood				
	6	Alnus	glutinosa	Black alder
	3	Jugians	nigra	black walnut
hedging				
	1	Dodonaea	viscosa	ake-ake



Today's Lecture

- Representing the Model
- Representing Classes and Attributes
- Primary Key
- Representing Relationships
- Exercises
- Summary

Representing the Model

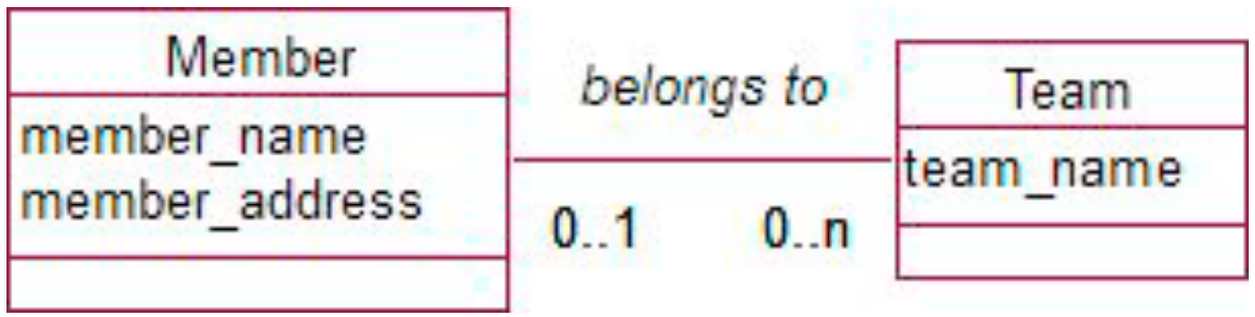
- We have gone to a great deal of trouble to capture as much detail as possible in the data model. Much of this detail can be represented and enforced by standard techniques built into relational database management software.
- A good model, implemented using the standard techniques, allows us to capture many of the constraints implied by the relationships between classes without recourse to programming or complex interface design.
- How many of the aspects of the data model can be captured by standard database functionality.

Feature in Model	Technique Used in Relational Database
Class	Add a table with a primary key.
Attribute	Add a field with an appropriate data type to the table.
Object	Add a row of data to the table.
1–Many relationship	Use a foreign key, i.e., a reference to a particular row (or object) in the table at the 1 end of the relationship.
Many–Many relationship	Add a new table with two foreign keys.
Optionality of 1 at the 1 end of a relationship	Make the value of the foreign key required.
Parent and child classes (inheritance)	Add a table for the parent class. Add tables for each child class with a primary key that is also a foreign key referencing the parent table (not an exact representation but OK).

- More complex constraints may require some additional procedures or checking at data input time,
- but with a good model this can be minimized.
- By using the built-in facilities of the database product, the time required for implementation, maintenance, and expansion of the application is greatly reduced.

Representing Classes and Attributes

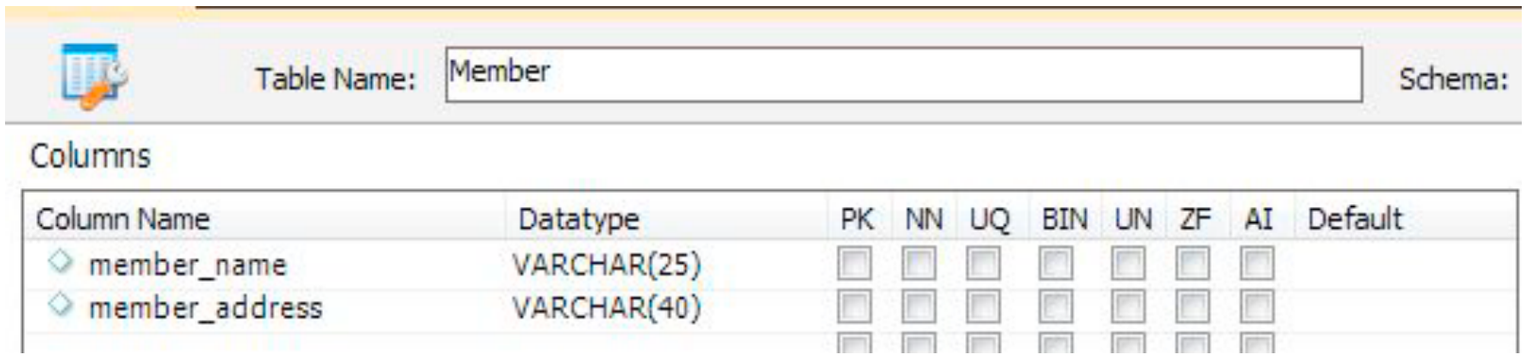
- Consider the figure, which represents a small part of a data model for members and teams.



- The first step is to design a database table for each class. The attributes of the class will become the field or column names of the table, and when the data is added, each row, or record, in the table will represent an object.
- We would create a table called **Member** for the **Member** class.
- The table would have two fields or columns, one for a member's name and one for the address.
- We would then add a row to the table for each member object (e.g., John Smith, 83 SomePlace, Christchurch).

- All relational databases allow you to create a table using SQL, which is a language for creating, updating, and querying databases.
- First we need a database that stores all our tables.
- To create a table, we need to provide a name for the table and a name and domain for each of the columns.
- The domain specifies the set of values that is allowed for that particular column.

```
CREATE TABLE Member (
member_name VARCHAR(25),
member_address VARCHAR(40)
)
```



Member

Field Name	Data Type
member_name	Text
member_address	Text

Text
Memo
Number
Date/Time
Currency
AutoNumber
Yes/No
OLE Object
Hyperlink
Lookup Wizard...

General | Lookup

Field Size	40
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

- Once the table has been created, we can enter data. In the case of the Member table, we would enter a row for each member object.

```
INSERT INTO Member (name, address)
VALUES ('Green, Ruth' '36 Some Street, Christchurch' )
```

member_name	member_address
Green, Ruth	36 Some Street Christchurch

Choosing Data Types

- Each attribute in a class becomes a field or a column in a table.
- When we create the table, we need to provide a name for the field (e.g., name, address) and specify the type of data that will be stored in that field.

Character types: These allow you to enter any combination of characters—numbers, letters, and punctuation. They are used for names, addresses, descriptions, and so on.

Integer types: These types are for entering numbers with no fractional part. They are great for ID numbers such as customer numbers and for anything that you can count.

Numbers with a fractional part: These are used for things that you measure (heights, weights, etc.) and also for numbers that result from calculations such as averages.

Dates: No prizes for guessing the type of data you can put in fields with these types.

- There are three main reasons why it is important to choose an appropriate data type for each of your fields:

  Constraints on the values: A character field type has no constraints on what you can enter; however, most other fields do.

  Ordering: Different types of fields have different ways for comparing or ordering values.

  Calculations: Your database product can do arithmetic and perform other functions on your data, but only if it is the correct type.

Domains and Constraints

- A domain is a set of values allowed for an attribute or field.
- For something like a product description it might be sufficient for the domain to be any set of characters up to some specified length. Other attributes might have more specific domains. For example, a gender attribute might only allow the values “M” or “F”; a day of the week might be constrained to the characters “Mon,” “Tue,” “Wed,” “Thu,” “Fri,” “Sat,” and “Sun”; possible marks for an exam might be whole numbers between 0 and 100.
- The user defined domain or type can then be used in all the tables in the database.
- The difference between a constraint and a domain is that the former has to be specified in every table whereas the latter only needs to be declared once in the database.

```
CREATE TABLE Member (
member_name VARCHAR(25),
member_address VARCHAR(40),
gender VARCHAR(1) CHECK gender IN ('M', 'F')
)
```

- One very important constraint is specifying whether a value is required or can be left empty.

```
CREATE TABLE Member (
member_name VARCHAR(20) NOT NULL,
member_address VARCHAR(45),
gender VARCHAR(1) CHECK gender IN ('M', 'F')
)
```

- Even if you think a value is going to be essential for the accuracy of your data, do not underestimate the likelihood that disallowing nulls might cause an incorrect value to be entered.

Checking Character Fields

- Character fields are a bit different from other field types. With a character field, we can enter anything we like (if there are no other constraints), and so it is possible to enter several values into a field. Other fields such as numbers and dates only ever allow one value to be entered.
- A good rule of thumb is that any data that you are likely to want to search for, sort by, or extract in some way should be in a field all by itself.

member_name	member_address
Green, Ruth	36 Some Street Christchurch

last_name	first_name	title	street_address	city	post_code
Turner	John	Mr	Flat 1, 6 Moa Street	Christchurch	8033
Green	Ruth	Mrs	36 Some Street	Christchurch	8041

- If the accuracy of values in a field is really crucial to the project, maybe that particular piece of information should actually be in a class of its own. You might recall that we separated genus out of our Plant class because its accuracy was important and we didn't want any misspellings.

Primary Key

- We have one constraint that is so important. This involves choosing a *primary key* for the table. It is imperative that we can always find a particular object (or row or record in a table).
- All records must be unique; otherwise, you can't distinguish between two that are identical.

Determining a Primary Key

- A key is a field, or combination of fields, that is guaranteed to have a unique value for every record in the table.

```
Member (name, address, phone, birth_date)
```

  - How about name for a key?
  - What about the combination (name, address)?
  - What about the combination (name, birth_date)?

- A potential key must be guaranteed to be unique for every possible record. In cases like Member table, there is not much choice but to add a new attribute or field such as member_number and then assign all members their own unique numbers so we can distinguish them.
- This is sometimes called a *surrogate* key.

- In many cases, privacy laws prevent information such as social security or tax numbers being used to identify people, so each business or organization is often compelled to provide its own personal identification number.

```
CREATE TABLE Member (
member_number INT PRIMARY KEY,
member_name VARCHAR (25)
)
```

- It is possible to get the database to automatically generate unique values for fields like **member_number**.
- You will find a field type called identity, auto_increment, autonumber, or something similar.

Concatenated Keys

- When we have a combination of fields that can uniquely identify a record, this is referred to as a *concatenated* or *composite* key.

student	course	year	grade
13887	COMP101	2011	B
17625	COMP101	2011	E
17625	COMP102	2012	A
18574	COMP102	2012	B

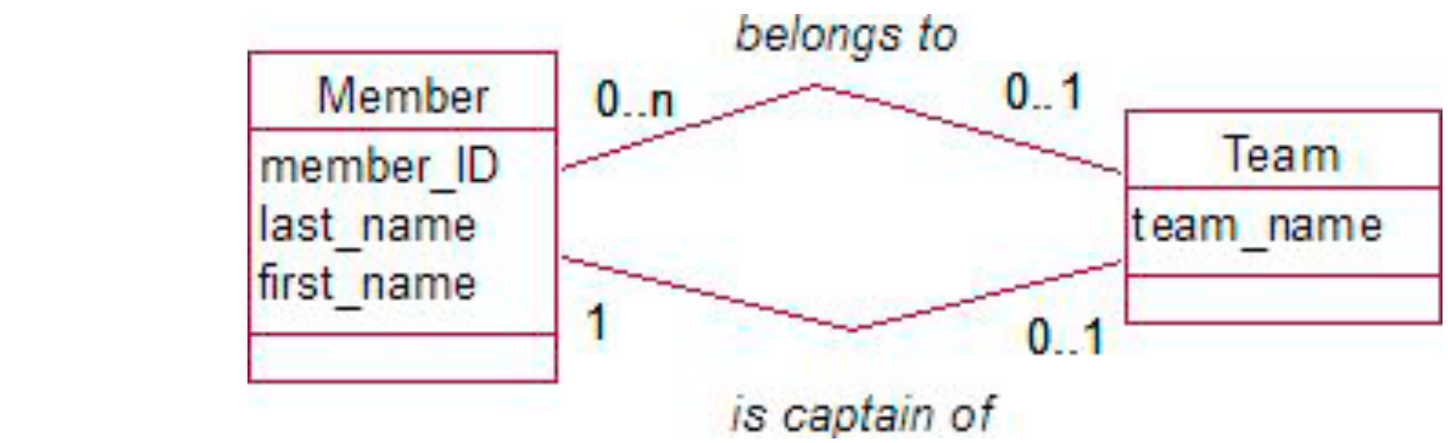
- How about **student** for a key?
- What about **course**?
- What about the combination (**student**, **course**)?
- What about the combination (**studentID**, **course**, **year**)?

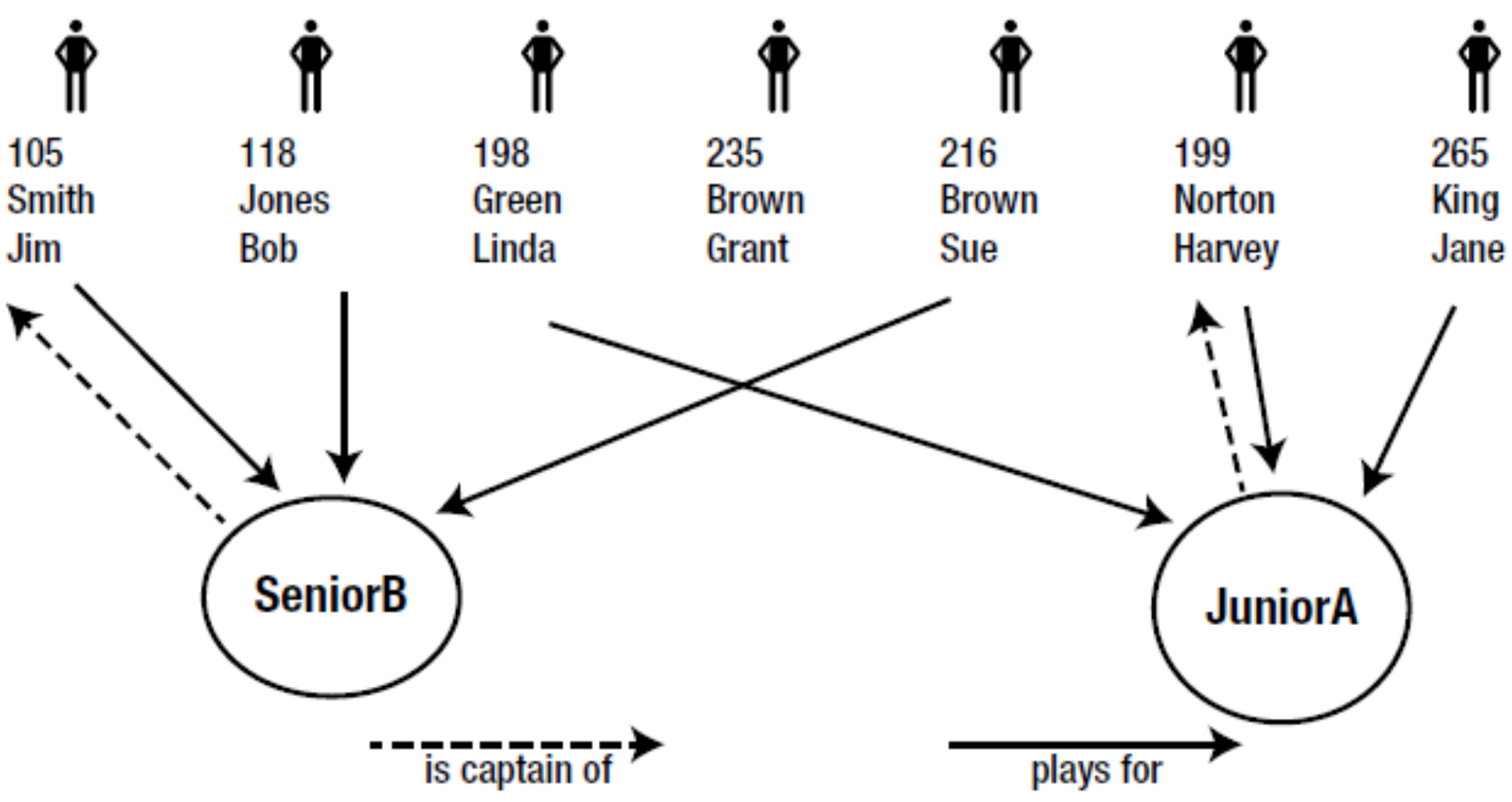
```
Enrollment (enrollment_number, student, course,
   semester, year, grade)
```

- The enrollment table now has two possible keys: enrollment_number and the combination (studentID, course, semester, year).
- What are the pros and cons of choosing one key over the other?

- An automatically generated number may be a ***sensible*** key, but it should not be included in a table because we can't be bothered thinking about alternatives.

Representing Relationships





- First we design two tables to represent the classes and choose a primary key for each.

```
Member: (member_ID, last_name, first_name)
Team: (team_name)
```

- To represent the relationships plays for and is captain of, we need a way of specifying each of the lines between the objects

Foreign Keys

member_ID	last_name	first_name
105	Smith	Jim
118	Jones	Bob
198	Green	Linda
199	Norton	Harvey
216	Brown	Sue
235	Brown	Grant
265	King	Jane

Member table

team_name	captain
JuniorA	199
SeniorB	105

Team table

- A foreign key is a field(s) (in this case **captain**) that refers to the primary key field(s) in some other table (in this case it contains a value of the key field **member_ID** from the table **Member**).

```
CREATE TABLE Team (
team_name VARCHAR(10) PRIMARY KEY,
captain INT FOREIGN KEY REFERENCES Member
)
```

- The two fields **member_ID** and **captain** will both have values from the same domain, that is, the set of MemberIDs. Formally, a foreign key and the primary key of the table it references must have the same domain.

Referential Integrity

- Arm-in-arm with the idea of a foreign key is the concept of referential integrity. This is a constraint that says that each value in a foreign key field must exist as values in the primary key field of the table being referred to.
- This prevents us putting a nonexistent member as the captain of a team.
- It also means that we cannot remove members 199 and 105 from our member table while they are captains of teams.
- As soon as you set up a foreign key, this referential integrity constraint is automatically taken care of for you.

Representing 1-Many Relationships

For a 1-Many relationship, the key field from the table representing the class at the 1 end is added as a foreign key in the table representing the class at the Many end.

member_ID	last_name	first_name	current_team
118	Jones	Bob	SeniorB
198	Green	Linda	JuniorA
199	Norton	Harvey	JuniorA
216	Brown	Sue	SeniorB
235	Brown	Grant	
265	King	Jane	JuniorA

Member table

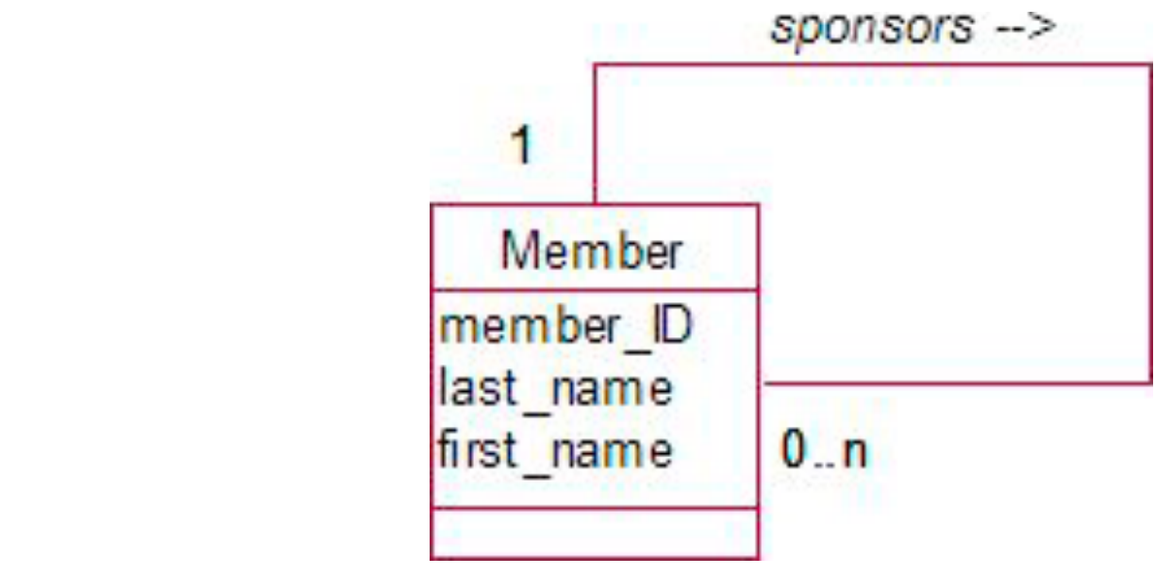
current_team is a foreign key referencing Team

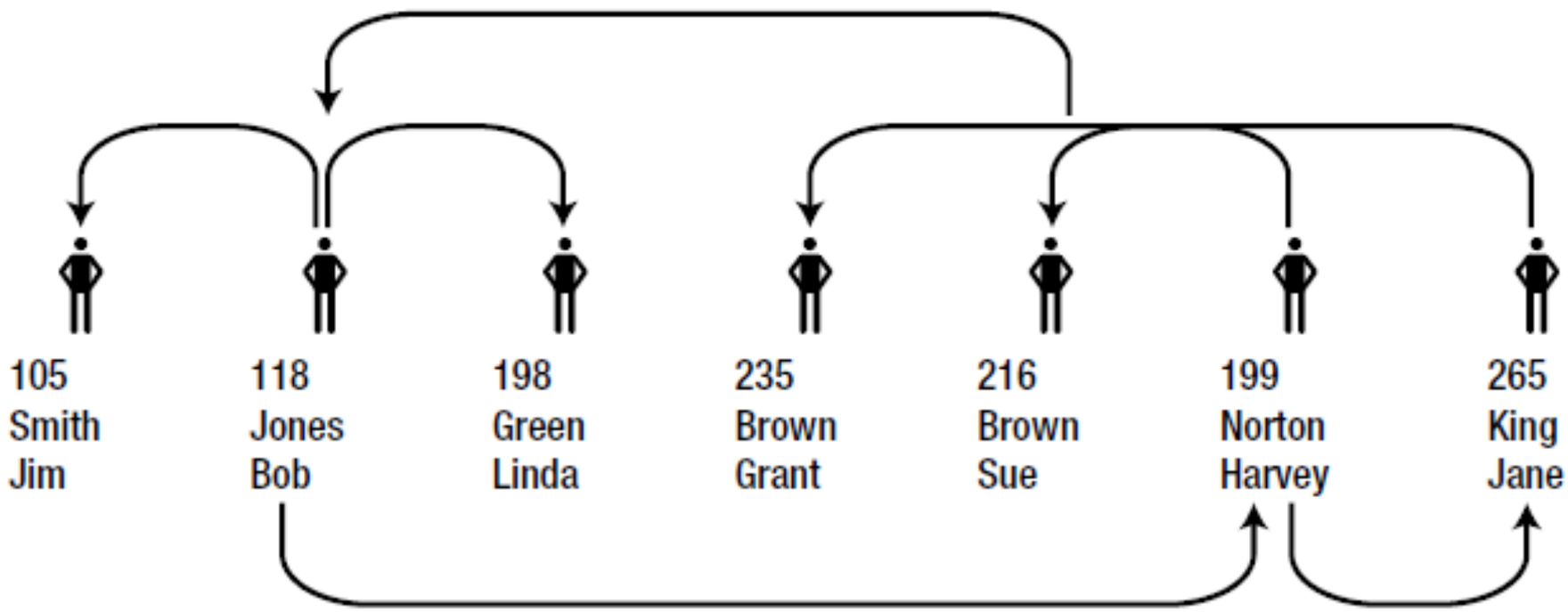
team_name	captain
JuniorA	199
SeniorB	105

Team table

captain is a foreign key referencing Member

- Referential integrity, which is a result of making the **current_team** field a foreign key, will ensure that the value entered in **current_team** can be found in the primary column (**team_name**) of the **Team** table.
- This ensures that members can only play for teams that already exist in the **Team** table.
- Note that a foreign key field can be null. This is consistent with the optionality of the plays for relationship.
- If the relationship was not optional, we would have to impose an additional constraint on the field to say that nulls were not permitted.

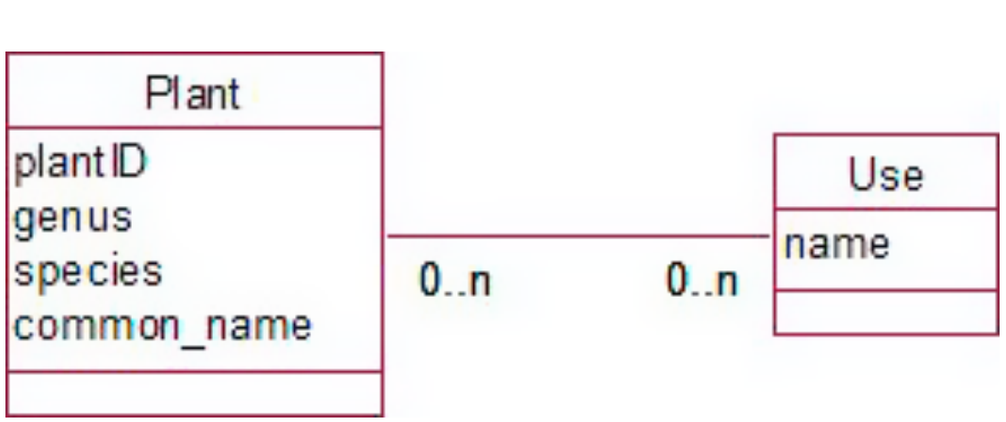

sponsors -->
1
Member
member_ID
last_name
first_name
0..n

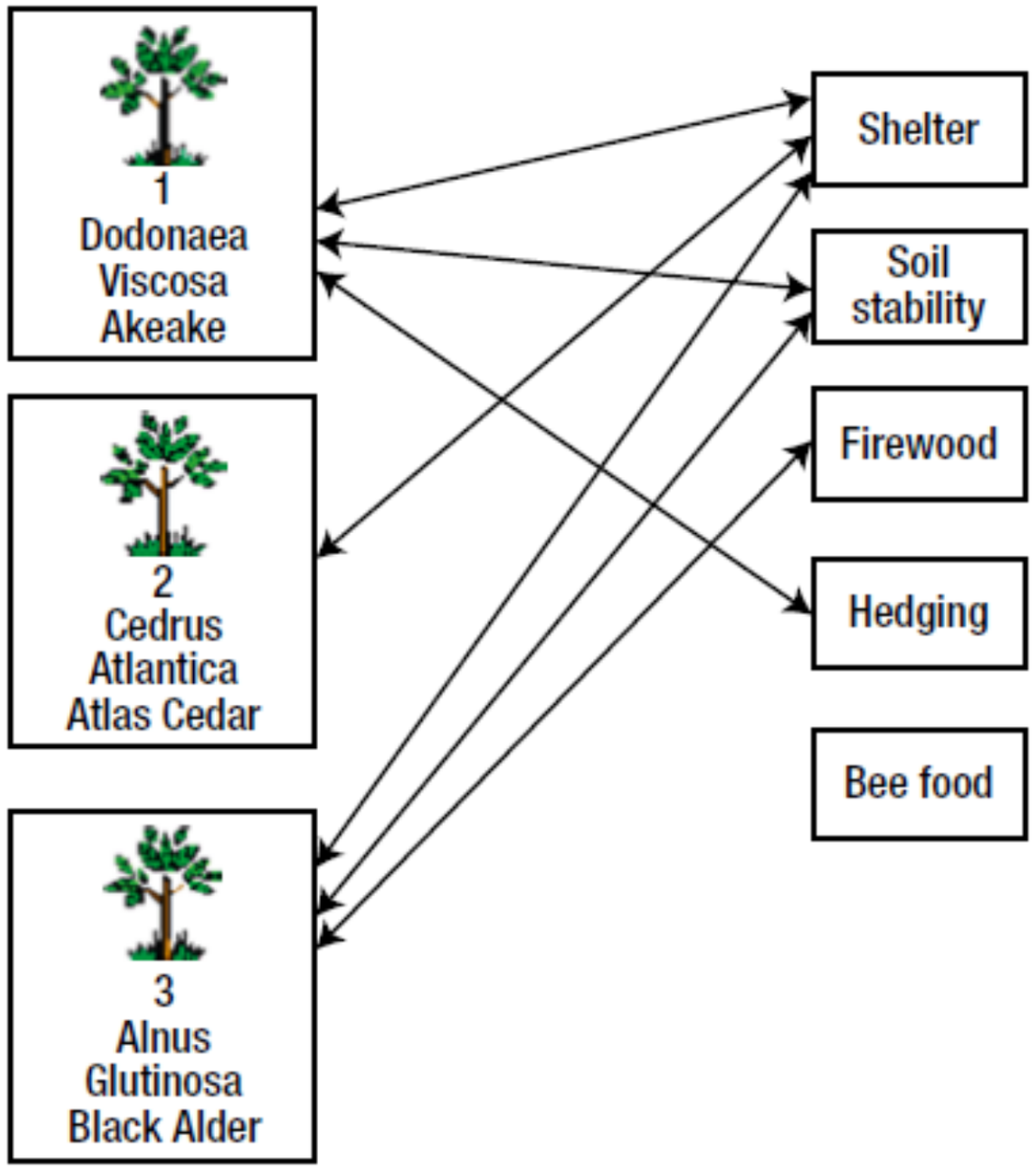

105
Smith
Jim
118
Jones
Bob
198
Green
Linda
235
Brown
Grant
216
Brown
Sue
199
Norton
Harvey
265
King
Jane

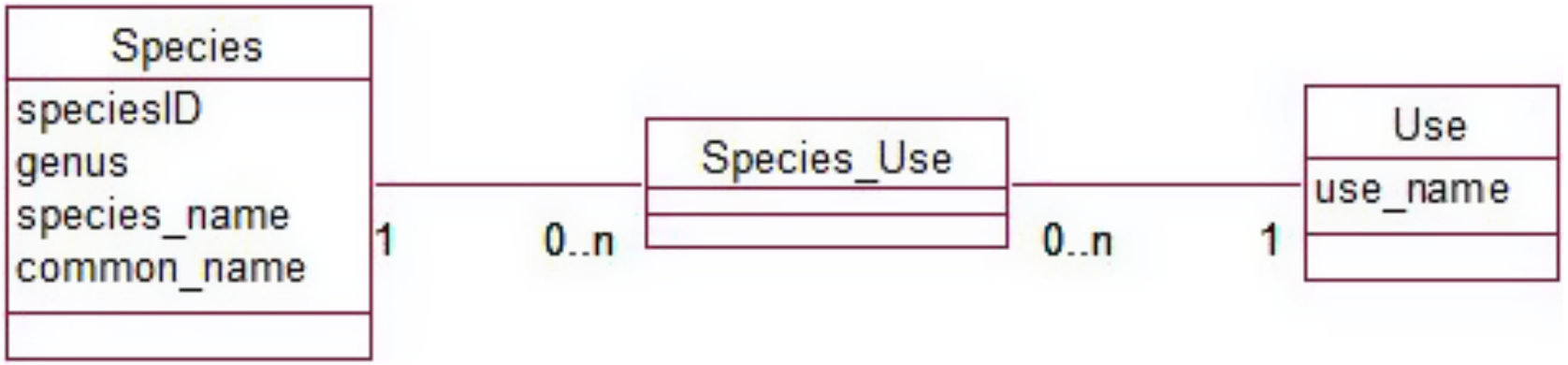
member_ID	last_name	first_name	current_team	sponsor
105	Smith	Jim	SeniorB	118
118	Jones	Bob	SeniorB	216
198	Green	Linda	JuniorA	118
199	Norton	Harvey	JuniorA	118
216	Brown	Sue	SeniorB	265
235	Brown	Grant		199
265	King	Jane	JuniorA	199

- How do you ever get the first member into the database when there is no existing member to sponsor her?
- This isn't just a database problem,

Representing Many-Many Relationships







speciesID	species_name	common_name	genus
1	viscosa	ake-ake	Dodonaea
2	atlantica	atlas cedar	Cedrus
3	nigra	black walnut	Juglans
4	melanoxylon	Tasmanian blackwood	Acacia
5	hippocastanum	Horse Chestnut	Aesculus
6	glutinosa	Black alder	Alnus

Species table

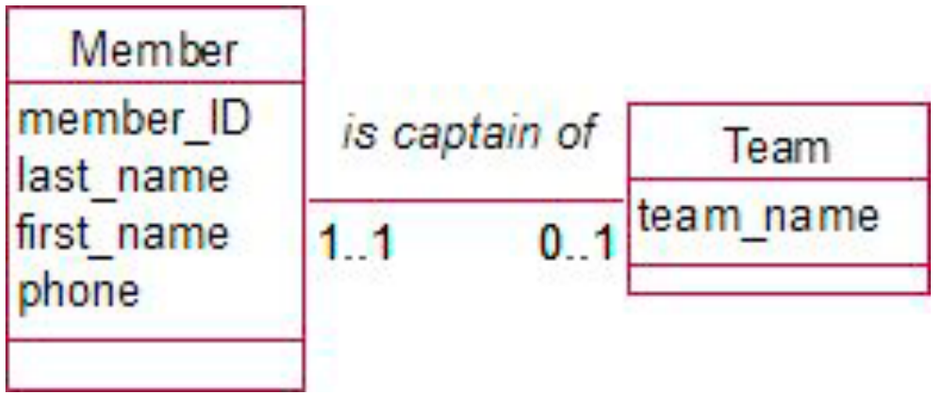
species	use
1	hedging
1	shelter
1	soil stability
2	shelter
3	firewood
3	shelter
3	soil stability

Species_Use table

use
bee food
bird food
coppicing
firewood
hedging
shelter
soil stability
timber

Use table

Representing 1-1 Relationships



member_ID	last_name	first_name	is_captain_of
105	Smith	Jim	SeniorB
118	Jones	Bob	
198	Green	Linda	
199	Norton	Harvey	JuniorA
216	Brown	Sue	
235	Brown	Grant	
265	King	Jane	

Foreign key in Member table

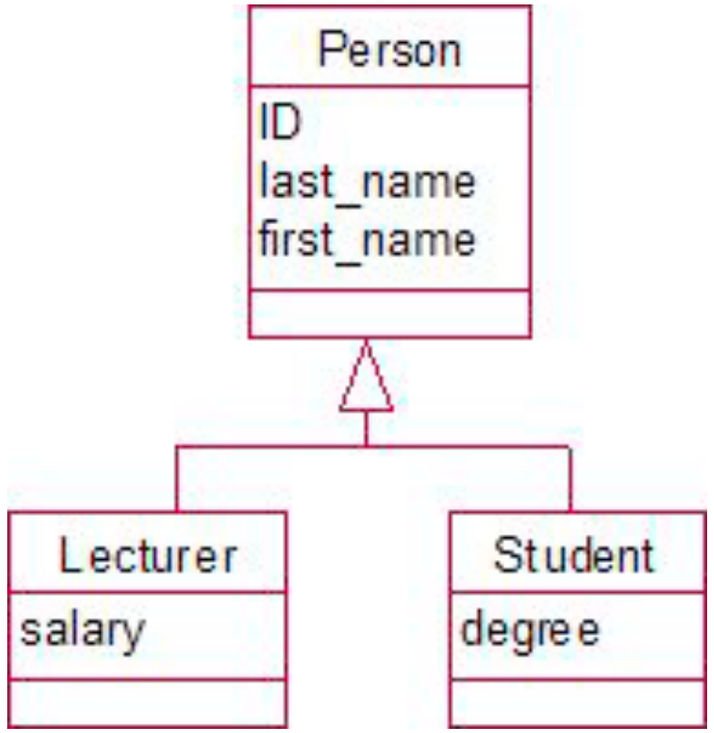
OR

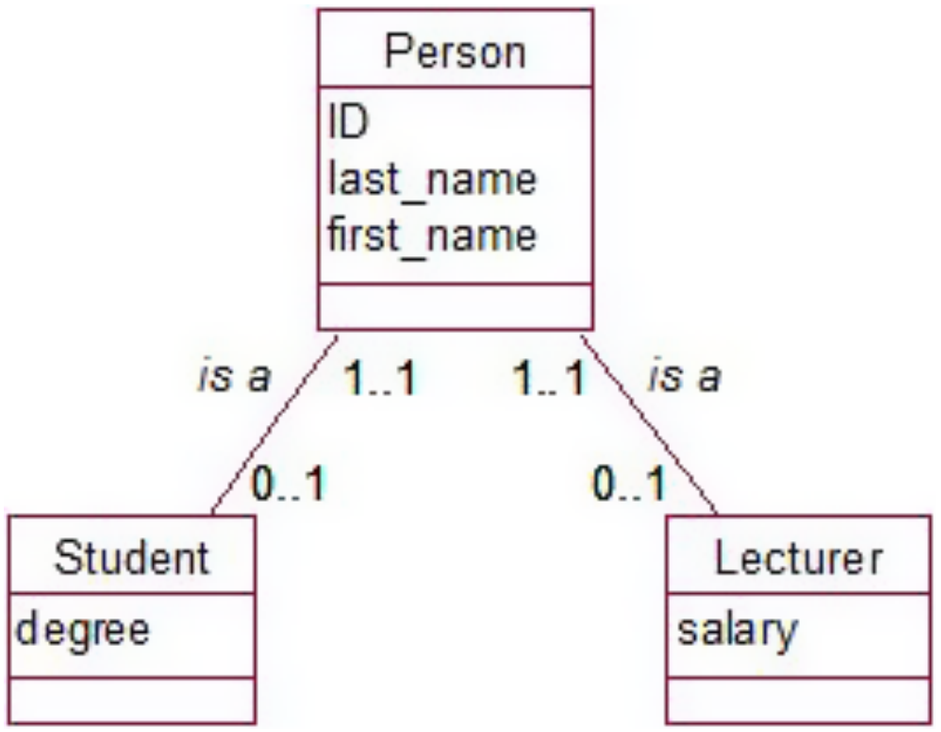
team_name	captain
JuniorA	199
SeniorB	105

Foreign key in Team table

Representing Inheritance

- Relational databases do not have the concept of inheritance built into them; however, it is possible to approximate the idea of inheritance.





ID	last_name	first_name
101	Jones	Sue
108	Brown	Lin
110	Li	Bo
112	Green	Mike

Person table

personID	salary
101	75000
108	90000
110	12000

Lecturer table

personID	degree
108	Arts
110	Science
112	Arts

Student table

member_ID	last_name	first_name	is_captain_of
105	Smith	Jim	SeniorB
118	Jones	Bob	
198	Green	Linda	
199	Norton	Harvey	JuniorA
216	Brown	Sue	
235	Brown	Grant	
265	King	Jane	

Foreign key in Member table

OR

team_name	captain
JuniorA	199
SeniorB	105

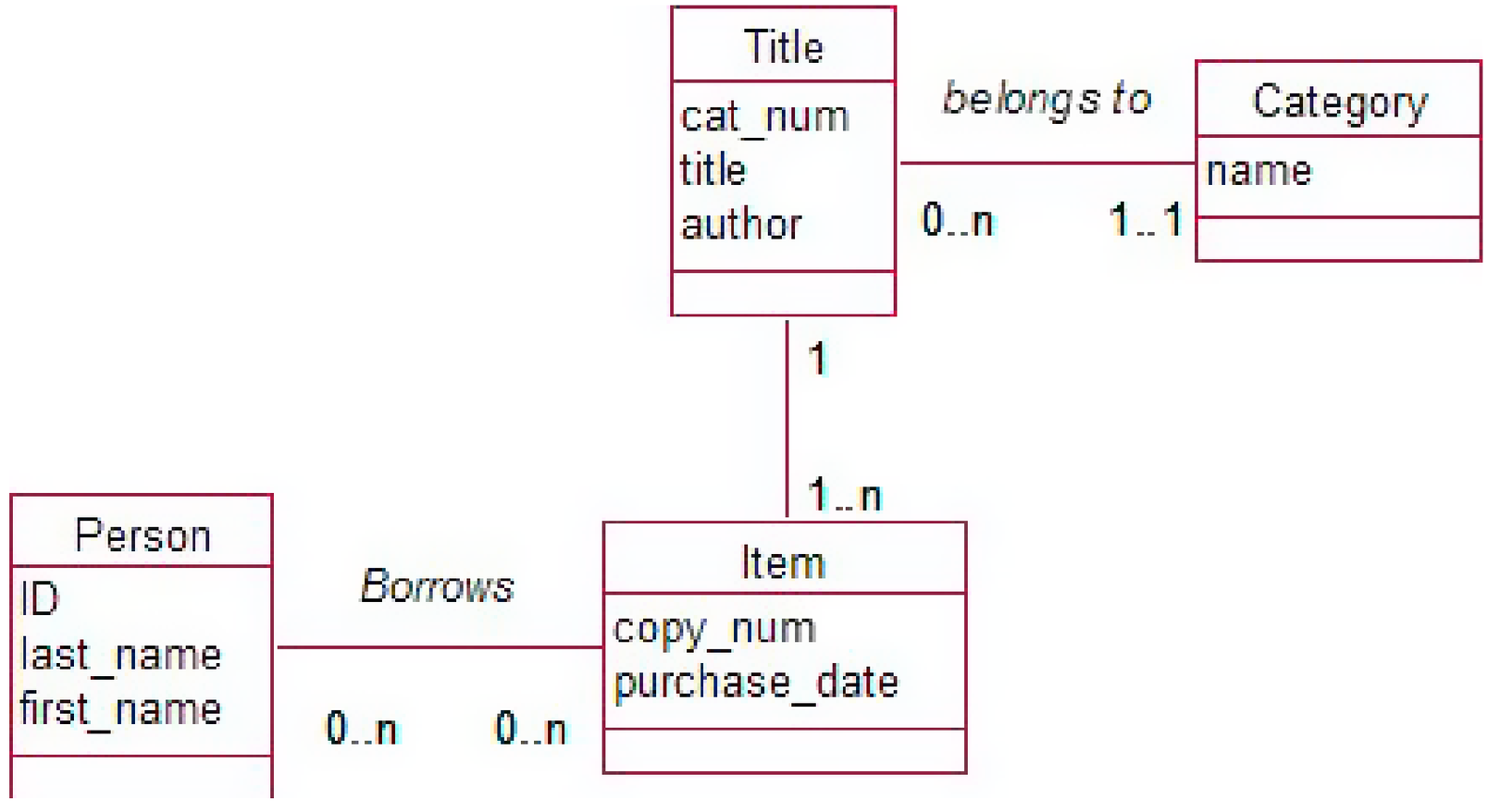
Foreign key in Team table

Exercise 10-1

The figure shows an initial data model for a small library. It is incomplete, so as you answer the questions below consider what else might need to be included.

a) *Explain to the librarian what the initial data model means.*

b) *Design tables for a relational database which would capture the information represented by the model. Include primary and foreign keys and other appropriate constraints.*

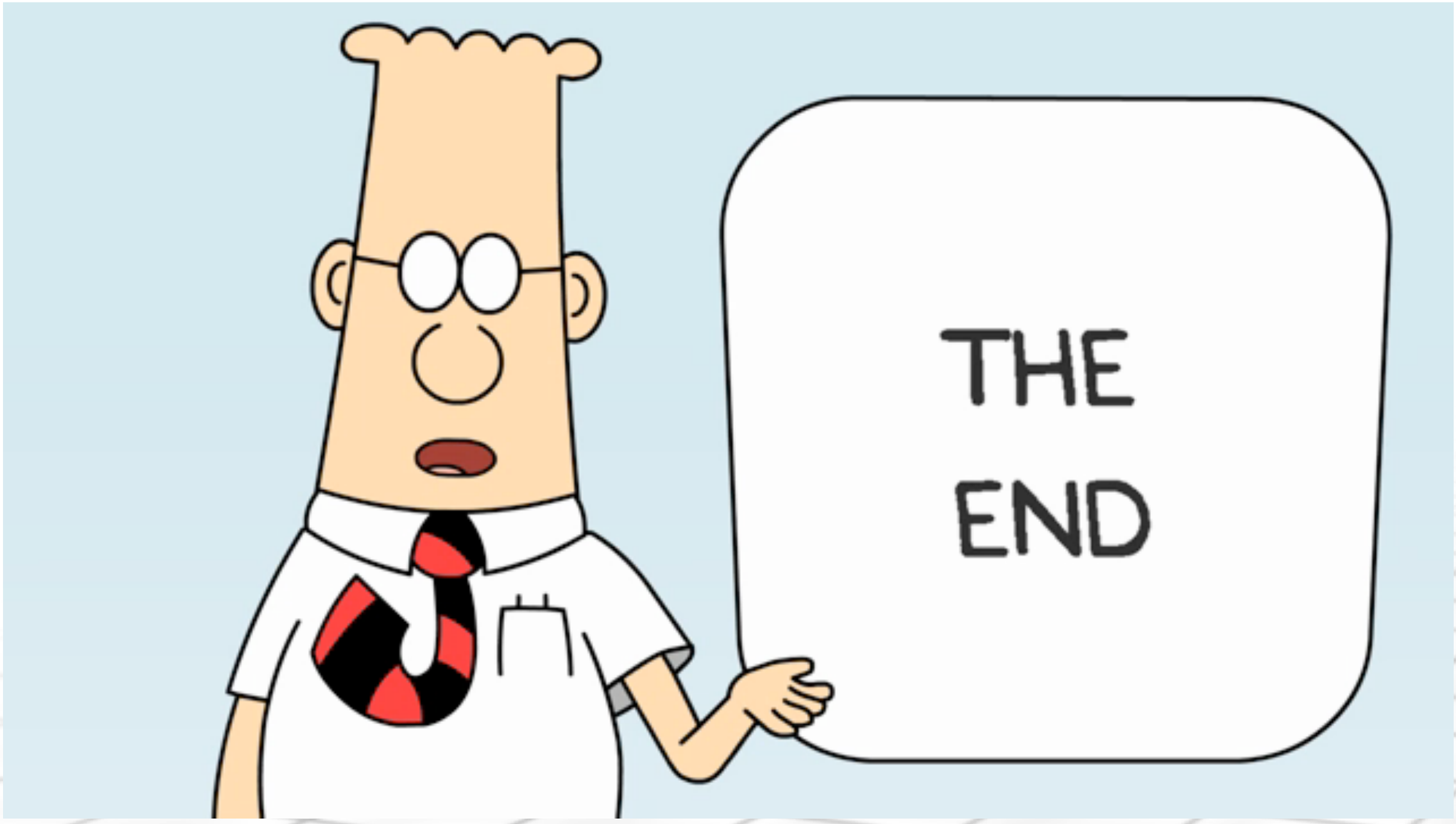


Summary

We have taken a data model and represented the main features using the functionality available in relational database products. Following is a summary of the steps:

1. For each class, create a table.
2. For each attribute, create a field and choose an appropriate data type. Consider whether some attributes (e.g., address) should be split into several fields.
3. Think about which fields should be required to have a value.
4. Consider what constraints need to be placed on the values of fields. Possibly create a new domain if your database product supports this.
5. Choose a field or combination of fields as the primary key. Ask careful questions to ensure that the key fields will always have unique values.

6. For each Many-Many relationship, insert a new intermediary class and two 1-Many relationships.
7. For each 1-Many relationship, take the primary key field(s) from the table representing the class at the 1 end and add this field(s) as a foreign key in the table representing the class at the Many end.
8. For a 1-1 relationship, put the foreign key in the table where it is most likely to have a value or where the attribute is most important.
9. For compulsory relationships, add a constraint to the foreign key fields that they must not be null.
10. For inheritance (as an approximation), alter the model to have a 1-1 is a relationship between the parent and each child class. Create tables and foreign keys as in point seven.



출처: metachannels.com

Thank you!