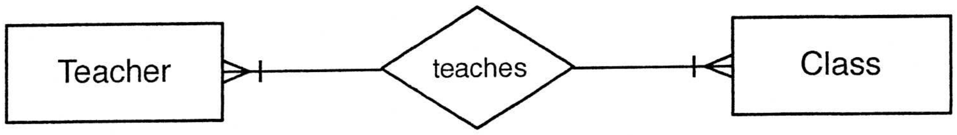
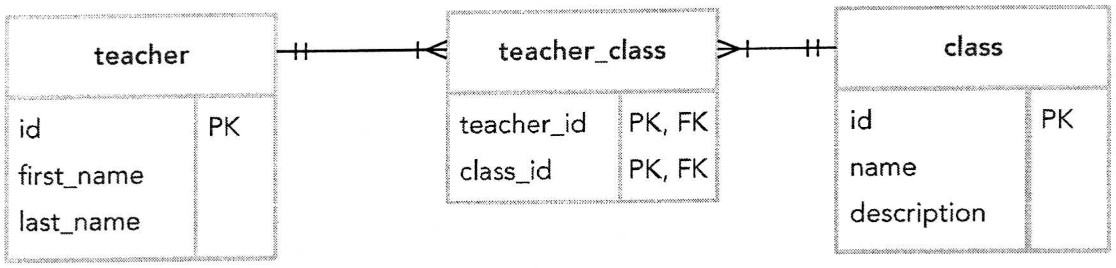
1. (30%) Consider a school's academic information system that keeps track of which teachers teach which classes. We can view the entity relationships as follows:



Notice that the diagram captures entities, their relationships, and cardinality.

Our database tables are structured to represent the relationships as follows:



Notice that the table structure captures cardinality between tables, the attribute in tables, and primary/foreign keys (PK / FK) wherever needed.

Extend the diagrams of <u>entity relationships</u> and <u>database tables</u> above for the following new requirement:

Teachers require one or more books for each of their classes.

The new information we need about books is as follows:

- The title of each book
- The date/time when a book was assigned by a teacher to a class
- The same book can be required in multiple classes, without duplication

Your model and database structure must accurately reflect which teacher has required which books for which class. Specifically, your design must help to:

- Avoid duplicate assignments of the same book to the same class.
- Be robust to changes in teacher assignment: If a teacher no longer teaches a class, we must remove the books that teacher assigned.
- Aid authorized book requirements: Teachers should only require books in classes they teach, and can only remove book requirements they assigned.

2 (10%). The following code, written in a statically typed, object-oriented language, uses a class called `LinkedList` to implement a singly linked list of instances of class `Node`. Calling `.head()` on an instance of `LinkedList` returns the first `Node` in the list.

```
LinkedList linkedList = new LinkedList();
linkedList.add( new Node("A") );
linkedList.add( new Node("B") );
linkedList.add( new Node("C") );
linkedList.add( new Node("D") );
linkedList.add( new Node("E") );

Node current = linkedList.head();
Node bar = linkedList.head();
int foo = 0;

do {
  foo++;
  if(foo%2 == 0){
    bar = bar.next();
  }
  current = current.next();
} while(current != null)

System.out.println("foo is: " + foo);
System.out.println("bar is: " + bar);
```

What is printed by this code?

(A)
```
foo is: 4
bar is: C
```
(B)
```
foo is: 5
bar is: E
```
(C)
```
foo is: 2
bar is: B
```
(D)
```
foo is: 5
bar is: C
```
(E)
```
foo is: 4
bar is: E
```
(F)
```
foo is: 2
bar is: D
```

To answer the next three questions, pick from the following data structures:

1. Hash Table
2. Array
3. Stack
4. Linked List
5. Binary Tree
6. Queue

3. (5%) Which of the above data structures do we usually use if we want to directly retrieve any element by supplying a lookup value:

(A) 2, 3, and 5
(B) 4 and 6
(C) 1, 2, and 6
(D) 3, 4, and 5
(E) 2, 3, and 5
(F) 1 and 2

4. (5%) Which of these data structures would we use to enter a list of tasks, such that we can retrieve and remove the tasks in the same order they were entered?

(A) 1　(B) 2　(C) 3　(D) 4　(E) 5　(F) 6

5. (5%) Which structure would we prefer to store the sequential order of work we have to do today, such that we can insert or remove jobs in any order?

(A) 1　(B) 2　(C) 3　(D) 4　(E) 5　(F) 6

6. (5%) Which of these data structures would we use to create an *expert system*, that asks 'yes/no' questions to find the answer to a problem. For example:

```
Do you like making your own applications (y/n)? "Y"
Do you prefer others to do the coding (y/n)?  "N"
Do you prefer to work alone (y/n)?  "N"
Would you like to have your own startup (y/n)?  "Y"
We recommend the following major: Service Science
```

(A) 1　(B) 2　(C) 3　(D) 4　(E) 5　(F) 6

7. (40%) We wish to improve the structure of our object-oriented code without changing its behavior. You must rewrite the code on this page as follows:
- Use object-orientated principles to remove repetition in the code
- Your new code must work with existing test code (see top half of *next page*)
- Follow OOP syntax of the programming language (see bottom half of *next page*)

```
class Student
  var name
  var must_teach

  def new(full_name)
    name = full_name
    must_teach = false
  end

  def notify_enroll(semester)
    puts("{{name}} must enroll for classes in {{semester}}")
  end
end

class Lecturer
  var name
  var must_teach
  var must_research

  def new(full_name)
    name = full_name
    must_teach = true
    must_research = false
  end

  def notify_grades(semester)
    puts("{{name}} must submit grades for {{semester}}")
  end
end

class Professor
  var name
  var must_teach
  var must_research

  def new(full_name)
    name = full_name
    must_teach = true
    must_research = true
  end

  def notify_grades(semester)
    puts("{{name}} must submit grades for {{semester}}")
  end

  def notify_grant(semester)
    puts("{{name}} must submit a grant in {{semester}}")
  end
end
```

Here is working test code that must still work after your refactoring:

```
# TEST CODE
lee = Student.new('Li-Hsieh Huang')
hsu = Lecturer.new('Powei Hsu')
peggy = Professor.new('Peishan Guo')

everyone = [lee, hsu, peggy]

for person in everyone
  if (not person.must_teach)
    person.notify_enroll('Spring 2016')
  end

  if (person.must_teach)
    person.notify_grades('Fall 2015')
  end

  if (person.must_teach) && (person.must_research)
    person.notify_grant('Spring 2016')
  end
end

# TEST OUTPUT
# => Li-Hsieh Huang must enroll for classes in Spring 2016
# => Powei Hsu must submit grades for Fall 2015
# => Peishan Guo must submit grades for Fall 2015
# => Peishan Guo must submit a grant in Spring 2016
```

Here is sample OOP code to help you understand the syntax of inheritance:

```
# SAMPLE OOP CODE:
class Automobile
  var brand                  # var defines a public instance variable
  var mode

  def new(brand_name)        # new is a constructor method
    brand = brand_name
    mode = 'ground'
  end
end

class Car < Automobile       # Car inherits from Automobile class
  var model
  var num_wheels

  def new(brand_name)
    super(brand_name)        # Calls the parent's constructor method
    num_wheels = 4
  end

  def describe()
    puts "The {{brand}} {{model}} has {{num_wheels}} wheels"
  end
end

my_car = Car.new('Luxgen')
my_car.model = 'U6 Turbo'
my_car.describe

# OUTPUT
# => "The Luxgen U6 Turbo has 4 wheels"
```