Class Test 2
(Model Solutions and Comments)

Q1-2. Conceptual design: ER modeling

Airport. The first decision point is what to do about the airport. Many people seem to think of the airport as an entity. However, there is only airport that the database is meant for. We postulate something as an “entity” in the E-R model only if there are several instances of it in the real world being modelled (so that we can include all those instances in a table). Since there is only one airport, there is no point in regarding the airport as a kind of entity. Even if it were made an entity, what attributes would it have?
(On the other hand, if we were developing a common database for several airports, then it would need to represent “airport” as an entity.)

Airplanes and models. There are two possible ways of modeling the airplanes and their models. One is to postulate an entity called “airplane” and several subclasses, one for each model of airplanes (DC-10 etc.) The second is to postulate the airplane model itself as an entity, and define a relationship between airplanes and models, called “instance of”. What is the right way?
We notice that each airplane model has a particular capacity and weight. If each model of airplane is made an entity, then all airplanes of those models would have the same values for these attributes. We would end up duplicating this information redundantly. Moreover, if the airport happens to purchase a new model, it would entail creating a new entity, and hence, a new table. That seems entirely needless. So, this way of modelling airplane models is wrong.

The right modeling is to use

\[
\text{airplane}(\text{regno}) \rightarrow (\text{instance_of}) \rightarrow (\text{model})
\]

with the attributes:

- \text{airplane}(\text{regno})
- \text{model}(\text{modelname}, \text{capacity}, \text{weight})
- \text{instance_of}(\text{regno}, \text{modelname})

Employees. The airport has at least two kinds of employees: technicians and air traffic controllers. There is clearly information that is generic to all employees, such as name and address. So, it is natural to define a subclass hierarchy: employee with subclasses technician and traffic controller. There could be other types of employees. So, this classification represents a partial coverage. The two known types of employees are mutually exclusive. The attributes are:

- \text{employee}(\text{empno}, \text{firstname}, \text{lastname}, \text{address}, \text{phone})
- \text{technician}(\text{empno})
- \text{traffic_controller}(\text{empno}, \text{last_medical_date})

Technicians possess expertise in particular airplane models. This is represented by a relationship:

\[
\text{technician}(\text{empno}) \rightarrow (\text{expertise}) \rightarrow (\text{model})
\]

with the schema:

- \text{expertise}(\text{empno}, \text{model})

Tests. This is the most tricky part of the question. Each type of test has an IATA number and attributes like maximum score. So, it should definitely be an entity.

\[
\text{test}(\text{IATAno}, \text{name}, \text{max_score})
\]

In addition, each type of test is conducted several times, on different airplanes, in each 3-month period. There are attributes needed for each instance of the test (technician that performs it, the score and the number of hours). There are several methods for representing this information.

Method 1: This is perhaps the easiest method. The test instances (or test events) can be represented as a relationship between three entities: test, airplane and technician.

\[
\text{test_event}(\text{IATAno}, \text{airplane}, \text{technician}, \text{date}, \text{score}, \text{hours})
\]

Note that the relationship is between three entities. It cannot be modeled as a binary relation. (The airplane attribute is expected to be the \text{regno} of an airplane and technician attribute is the \text{empno} of a technician.)
**Method 2:** This is a variant of the first method, with an additional weak entity `test_instance`, which is specific to each 3-month period.

```
test_instance(IATAno, period)
```

The `test_event` relationship is still a relationship between three entities: `test_instance`, `airplane`, and `technician`.

```
test_event(IATAno, period, airplane, date, technician, score, hours)
```

This is superior to Method 1 as it allows us to capture the information that there is at most one test event for each airplane in each period.

**Method 3:** Another variation is to postulate two binary relations for `test_instance`'s, one with airplanes and one with technicians.

```
airplane(1,n) — (performed_on) — (1,n) test_instance(1,n) — (performed_by) — (0,n) technician
```

The attributes for the entities and relationships might be as follows:

```
test_instance(IATAno, period)
performed_on(IATAno, period, airplane, date, score, hours)
performed_by(IATAno, period, technician)
```

Unfortunately, this modeling does *not work correctly*. There is a unique technician performing each test (instance) on a particular airplane. There is no uniqueness without using the airplane attribute. This representation loses this uniqueness information.

**Method 4:** A further variation on this method is represent `test_instance` as a weak entity *dependent on two entities*: a test and an airplane. This is highly unconventional. (I have never seen any text books deal with weak entities with such double dependence.) However, there is nothing logically wrong with it.

```
test_instance(1,1) — (performed_by) — (0,n) technician
```

The attributes would be as follows:

```
test_instance(IATAno, period, airplane, date, score, hours)
performed_by(IATAno, period, airplane, technician)
```