# Tutorial 2: Relational Modelling

## Informatics 1 Data & Analysis

## Week 4, Semester 2, 2016/17

This worksheet has three parts: tutorial *Questions*, followed by some *Examples* and their *Solutions*.

- Before your tutorial, work through and attempt all of the Questions in the first section. If you get stuck or need help then ask a question on *Piazza*.

- The Examples are there for additional preparation, practice, and revision.

- Use the Solutions to check your answers, and read about possible alternatives.

You must bring your answers to the main questions along to your tutorial. You will need to be able to show these to your tutor, and may be exchanging them with other students, so it is best to have them printed out on paper.

If you cannot do some questions, write down what it is that you find challenging and use this to ask your tutor in the meeting.

Tutorials will not usually cover the Examples, but if you have any questions about those then write them down and ask your tutor, or go along to InfBASE during the week.

It's important both for your learning and other students in the group that you come to tutorials properly prepared. Students who have not attempted the main tutorial questions will be sent away from the tutorial to do them elsewhere and return later.

Some exercise sheets contain material marked with a star ⋆. These are optional extensions.

Data & Analysis tutorials are not formally assessed, but the content is examinable and they are an important part of the course. If you do not do the exercises then you are unlikely to pass the exam.

Attendance at tutorials is obligatory: if you are ill or otherwise unable to attend one week then email your tutor, and if possible attend another tutorial group in the same week.

*Please send any corrections and suggestions to Ian.Stark@ed.ac.uk*

## Introduction

In the previous tutorial, you designed an ER model for a database, based on a scenario description for a poster exhibition. In this tutorial, you will design a relational model to implement such an ER model, applying the techniques described in lectures.

## Question 1: Relational Schema Notation

Below is a schema for a relational database table, written in SQL DDL. However, it has (at least) 5 errors in syntax. What are they? (Note: it's not the choice of lower or upper case — either is acceptable SQL.)

```
create table film (
    filmid          integer,
    title           varchar(50),
    release_year    integer
    director name   varchar(50) not null,
    country         varchar(30),
    primary key (filmid),
    foreign key director name references director,
    foreign key (leading_actor) references actor
```

## Question 2: Mapping an ER Model to Relational Schemas

For this question, please use the ER model in Figure 1 on page 3, which is a fragment from a solution to last week's exercise. Think about how the entities and relationships can be mapped to tables in a relational model. Then write out SQL **create table** statements to define all relations required for this database. These SQL declarations should include **primary key** and **foreign key** constraints as appropriate. The ER diagram gives a number of constraints on the model — try to capture as many as you can, remembering that it may not be possible to express all of them in relational schemas.

*Things to think about:* What different approaches are there for mapping relationship sets with key constraints? What are the advantages and disadvantages of each approach? Which approach did you choose for translating the provided ER model and why?

## Question 3: Refining the Model

The following questions are about possible refinements to your relational model.

(a) In your schemas, which fields are not allowed to take a **null** value? Are there any that you should explicitly disallow from taking a null value? What constraints on the data model can you establish by using **not null** declarations?

(b) Consider the various foreign key constraints in your relational model design. In each case, what action would it be appropriate to instruct SQL to take on delete? How would you express this in the schema declaration?

(c) Suppose each graphic designer must create exactly one poster. How would you represent this in the ER diagram? Would you be able to represent it in your relational model, and if so, how?

(d) Go back to your original relational model from Question 2. Can you implement the total participation constraint of **Poster** on the relationship sets **Creates** and **Judgement**? If yes, then how?

(e) According to the original scenario: "Each poster is judged by three different judges". Can you model this in your relational model? If yes, how?
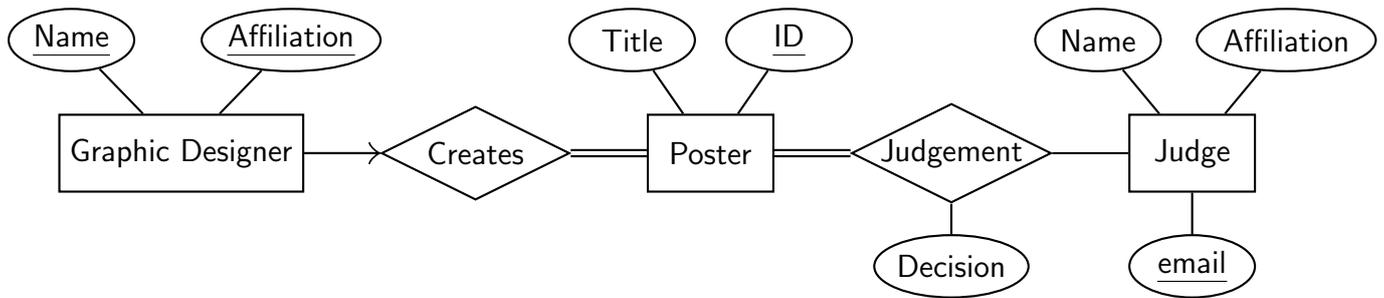
Figure 1: An ER model for part the poster exhibition scenario from Tutorial 1. There are double lines from Poster to Creates and to Judgement, and an arrow from Graphic Designer to Creates.

## Question 4: More Things to Think About

**(a)** Based on your answers so far, how do you think expressivity of the relational model compares to the ER model in this scenario?

**(b)** Based on your answers for the above questions, what strategy would you recommend for designing a relational model for a given scenario? By directly translating the corresponding ER model, by directly translating the scenario, or as a combination of the two?

# Examples

This section contains further exercises on mapping ER models into relational ones. Examples 1 and 2 are similar to the main tutorial questions; Examples 3 to 9 are based on questions in past exam papers.

Following these there is a section presenting solutions and notes on all the examples.

## An Inter-University Gliding Competition

These examples are based on the scenario presented in the Examples section of Tutorial 1: here we now map an ER model for that into relational schema.

## Example 1: Mapping an ER Model to Relational Schemas

For this example, you should use the ER model shown in Figure 2 on page 5. Think about how the entities and relationships there can be mapped to tables in a relational model. Then state appropriate SQL **create table** statements for all relations needed to create this database. These declarations should also include all appropriate *primary key* or *foreign key* constraints.

## Example 2: Discussion

Looking at the relational schema that you have created for the previous question, give your justification for the following design decisions.

(a) Look at your choice of tables, their fields and types. Are there other ways they could have been designed? Explain your preferences.

(b) In your schemas, which fields are not allowed to take a **null** value? Are there any that you should disallow from taking a null value? What constraints can you establish by preventing the fields from taking a **null** value?

(c) Did you use any **on delete cascade** declarations in your schema? Are there entries in your schema where you think it might be appropriate to do so?
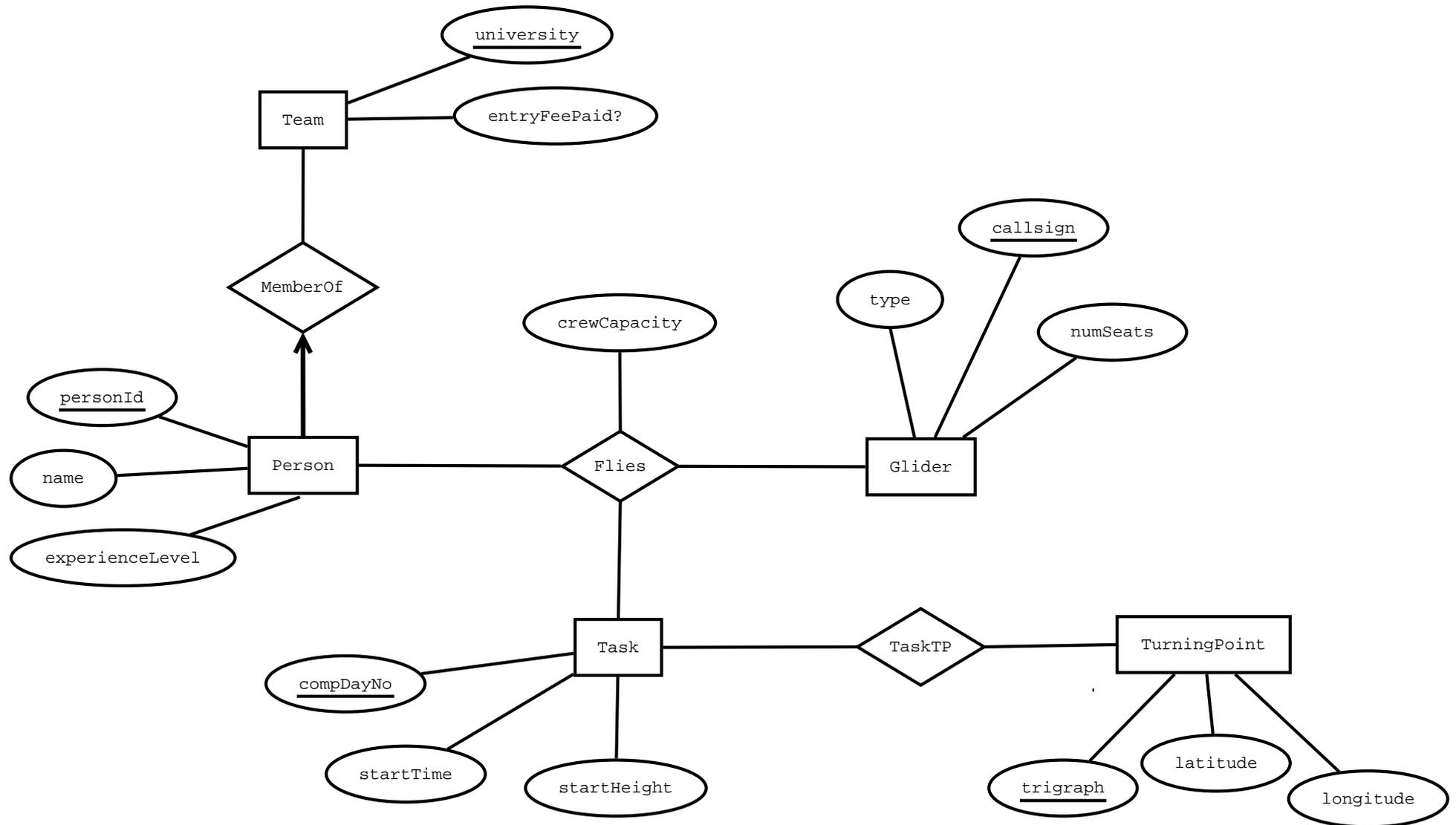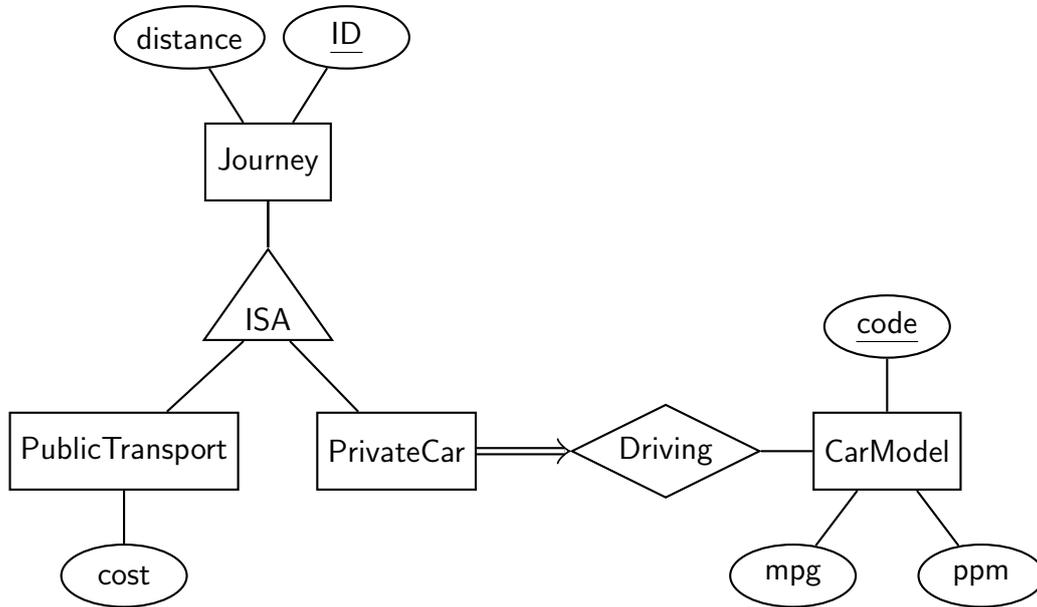
Figure 2: An Entity-Relationship model for the gliding competition database.

## Example 3

The entity-relationship diagram below shows a fragment of a proposed model for a database managing travel claims by university staff. Each journey is labelled with a unique ID, and journeys by car or by public transport have different information recorded.



You also have the following information about the model.

- Attributes distance, cost, mpg (miles per gallon fuel consumption) and ppm (pence per mile travel allowance) are represented as integers.

- All other attributes are alphanumeric values up to 8 characters long.

- Every instance of the CarModel entity must have mpg and ppm values provided.

Draw up an SQL data declaration of appropriate tables to implement this entity-relationship model. (You need not include **on delete** declarations).

## Example 4

The following ER diagram appeared in Tutorial 1 as a suitable model for the *Itsy! Bitsy!* craft trading website.
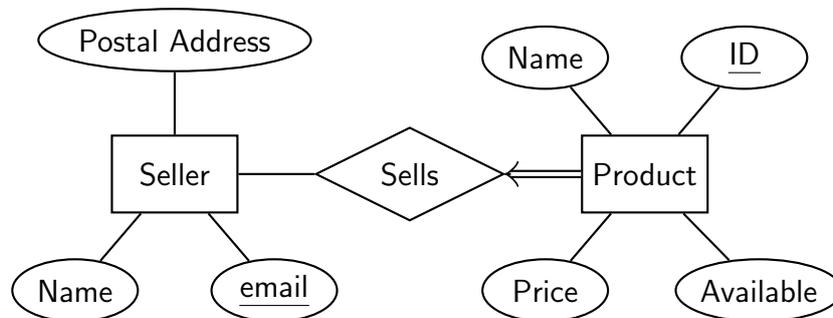


Design appropriate relational schema for this information and present them as two table declarations in the SQL Data Definition Language: for Seller and Product. Explain how you capture all constraints on the data.

# Example 5

The following entity-relationship diagram captures information about a number of *coders* who work together on a wide range of software *projects*. A project may have a number of different *branches* under active development at any one time. Branches are named, often using standard descriptions such as "master", "stable", or "unstable".
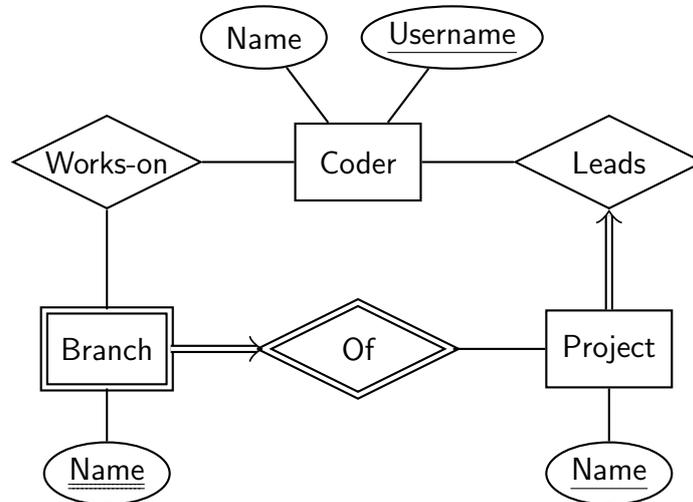


(a) What is the meaning of the double line around Branch? Why is this needed? What is the primary key for a Branch?

(b) Construct SQL data declarations for a set of tables to represent this entity-relationship diagram. Assume that usernames are limited to 64 characters and all other names fit within 200 characters. Make sure to use **not null** where necessary; however, you do not need to include **on delete** declarations.

# Example 6

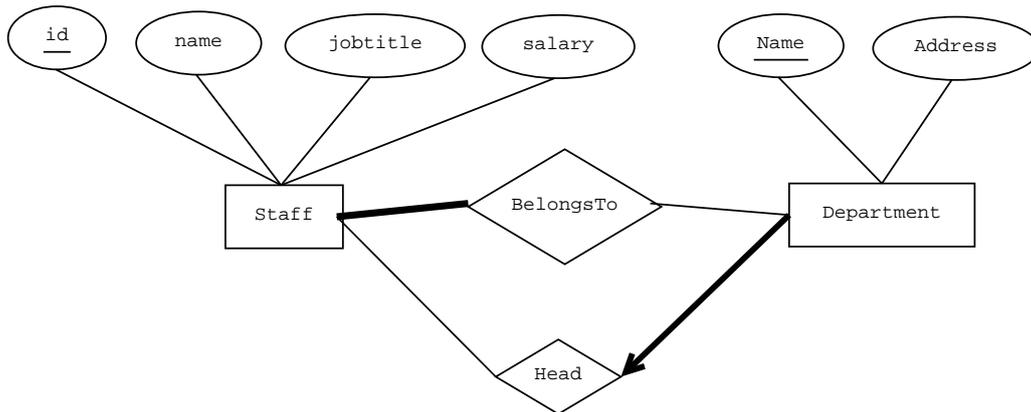This ER diagram is from Tutorial 1, as a model for a database of experts on different subjects.
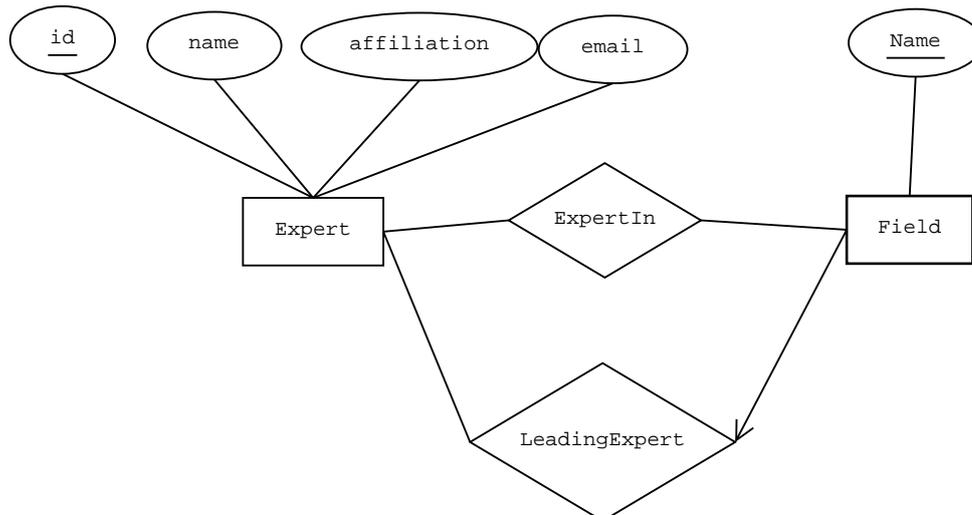


(a) Use the SQL Data Definition Language to declare relational schemas that implement this ER diagram.

(b) Explain which of the constraints in the ER diagram you have incorporated in your relational schemas and which you have not. Where you have been able to capture a constraint, explain how it appears in the SQL code.

## Example 7

This diagram from Tutorial 1 is a proposed ER model for a university staff database.



(a) Use the SQL Data Definition Language to declare relational schemas that implement this ER diagram.

(b) Explain which of the constraints in the ER diagram you have incorporated in your relational schemas and which you have not. Where you have been able to capture a constraint, explain how it appears in the SQL code.

## Example 8

This ER model captures some information about a tennis association's members and their performance in international tournaments.
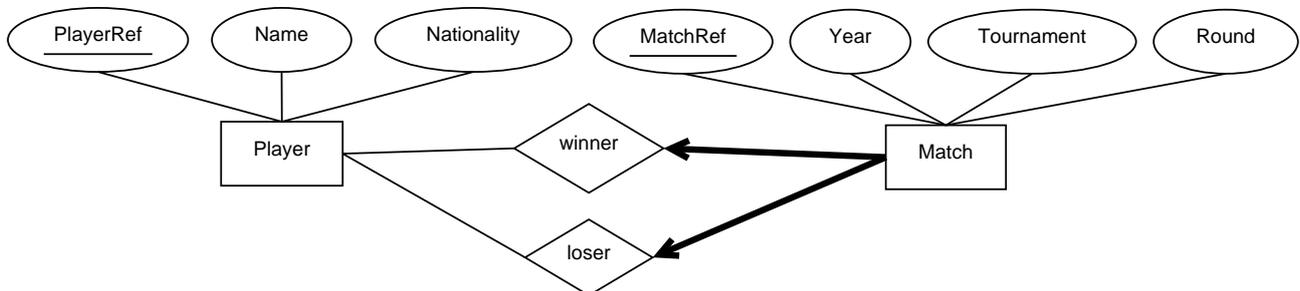


(a) Use the SQL Data Definition Language to present relational schema that implements this diagram using two tables: Player and Match.

(b) Explain how you have expressed the participation and key constraints of the ER model in your SQL declarations.

## Example 9

This ER diagram gives a conceptual model for a theatre manager's database about forthcoming shows by visiting acting companies.
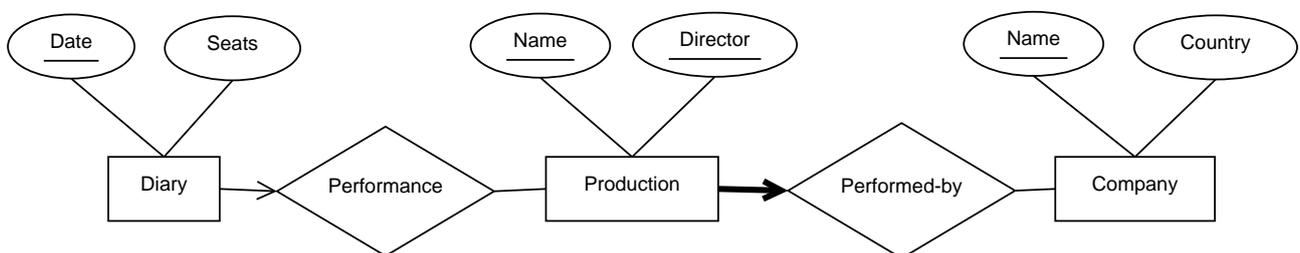
(a) Give declarations in the SQL Data Definition Language to implement this as a relational model using three tables: Diary, Production, and Company.

(b) How does your SQL code capture the total participation and key constraints shown in the ER diagram?

## Example 10

Give brief descriptions of the following terms, as they apply to tables and queries in relational databases.

(a) Cardinality

(b) Arity

(c) Foreign key

(d) Selection

(e) Projection

(f) Intersection

(g) Natural join

# Solutions to Examples

What follows are not entirely "model" answers; instead, they indicate a possible solution, together with some comments on what features are relevant and where there might be trade-offs between different alternatives.

Remember that not all of these questions will have a single "right" answer. There can be multiple appropriate ways to design a relational model for a given scenario, each with particular advantages or disadvantages. Even where these notes include more than one solution, it still cannot cover every possible correct alternative.

If you have difficulties with a particular example, or have trouble following through the solution, please raise this as a question in your tutorial.

## Solution 1

Here is a set of **create table** statements for the ER diagram provided. The particular choice of varchar(..) size is arbitrary, but should be able to sensibly contain likely values. Notice that every **foreign key** properly references the **primary key** in the linked table.

```
create table Person (
    personId       integer,
    name           varchar(30),
    experienceLevel varchar(15),
    university     varchar(50) not null,
    primary key (personId),
    foreign key (university) references Team
    )
```

```
create table Team (
    university    varchar(50),
    entryFeePaid integer,
    primary key (university)
    )
```

```
create table Glider (
    callsign     varchar(5),
    type         varchar(10),
    numSeats    integer,
    primary key (callsign)
    )
```

```
create table Task (
    compDayNo integer,
    startTime     timestamp,
    startHeight   integer,
    primary key (compDayNo)
    )
```

```
create table Flies (
    personId     integer,
    callsign     varchar(5),
    compDayNo   integer,
    crewCapacity integer,
    primary key (personId, callsign, compDayNo),
    foreign key (personId) references Person,
    foreign key (callsign)  references Glider,
    foreign key (compDayNo) references Task
    )
```

```
create table TurningPoint (
    trigraph    varchar(3),
    latitude    varchar(10),
    longitude  varchar(10),
    primary key (trigraph)
    )
```

```
create table TaskTP (
    compDayNo integer,
    trigraph      varchar(3),
    primary key (compDayNo, trigraph),
    foreign key (compDayNo) references Task,
    foreign key (trigraph) references TurningPoint
    )
```

# Solution 2

(a) The tables above capture the MemberOf relationship by bundling it into the Person table as the foreign key team. This is possible because of the key constraint between Person and MemberOf. However, the relationship could also be represented as a table itself:

```
create table MemberOf (                          create table Person (
    personId    integer,                             personId        integer,
    university  char(50),                             name            varchar(30),
    primary key (personId),                           experienceLevel varchar(15),
    foreign key (personId)   references Person,       primary key (personId)
    foreign key (university) references Team          )
)
```

Notice that the Person table has no mention of a university or reference to Team: this is all now in the new MemberOf relation. The key constraint is captured by making personId a primary key (rather than the {personId, university} composite) so that there can only be a single MemberOf record for each person.

(b) It is normal (but not universal in RDBMS) that primary keys can never be **null**. In the tables declared above there is also an explicit **not null** constraint on the university field of the Person relation. This captures the total participation constraint from the ER model: each person must have an associated university for whose team they compete.

In the alternative relational model where MemberOf is a separate relationship, there is no straightforward way to capture this participation constraint.

The **not null** declaration on university here is enforcing an integrity constraint on the data. We might also use **not null** to capture more general expectations from the scenario that certain values should always be included in the database. For example, the latitude and longitude on a TurningPoint, or the type and numSeats of a Glider.

(c) The declarations above do not include any **on delete cascade** instructions. The classic situation where they are essential is for weak entity sets in an ER model, where one entity is entirely dependent on its identifying relationship for full representation. There are none of these in the ER model here.

However, any **foreign key** is a potential site for **on delete cascade**, if the relation in which it appears loses meaning without its target. In this example, the TaskTP relation might perhaps use **on delete cascade** for the compDayNo field referencing Task — if a particular day's task is removed from the database, then the detail of which turning points belong to that task is no longer meaningful. If we have in **create table** TaskTP ( ... ) the line

> ... **foreign key** (compDayNo) **references** Task **on delete cascade**, ...

then if ever a record is erased from the Task table, every corresponding record from the TaskTP table will also be deleted.

In the alternate table declarations of solution (b), we could consider doing the same for the personId reference of the MemberOf table: if a person is removed from the database, then it is no longer relevant to record their link to a particular Team.

## Solution 3

The following tables capture the ER model given.

```
create table Journey (                    create table PublicTransport (
  ID         varchar(8),                     ID         varchar(8),
  distance   integer,                        cost       integer,
  primary key (ID)                           primary key (ID),
)                                            foreign key (ID) references Journey(ID)
                                           )

create table PrivateCar (
  ID         varchar(8),                        create table CarModel (
  driving    varchar(8) not null,                 code       varchar(8),
  primary key (ID),                               mpg        integer not null,
  foreign key (ID) references Journey(ID)          ppm        integer not null,
  foreign key (driving) references CarModel(name)  primary key (code)
)                                              )
```

The **not null** declarations here are required by the constraints given in the question; it's also acceptable to put in further **not null** declarations.

Using a standalone Driving table would not be satisfactory, as this could not capture the key and participation constraints from the ER diagram.

The PublicTransport and PrivateCar tables should not duplicate the distance field as that is already available from the parent Journey table.

## Solution 4

Here are suitable SQL data declarations for the two tables.

```
create table Seller (                    create table Product (
  name     varchar(30) not null,           name      varchar(60) not null,
  email    varchar(30) not null,           id        integer not null,
  address  varchar(200) not null,          price     integer not null,
  primary key (email)                      available  integer not null,
)                                          seller     varchar(30) not null,
                                           primary key (id),
                                           foreign key (seller) references Seller(email)
                                         )
```

The key constraint from entity Product to relationship Sells, shown by the arrowhead and which requires that every product have no more than one matching seller, is enforced by the **foreign key** declaration on field seller in table Product.

The total participation constraint from Product to Sells, shown by the thick line and which requires that every product must have some seller recorded, is enforced by the **not null** declaration on the field seller in table Product.

These declarations include **not null** statements for every field. Not all of these are necessarily essential, and they fall into three categories:

- The **not null** declaration on seller captures the total participation constraint, and so cannot be left out.

- The **not null** declarations on email and id fields can be omitted, as the **primary key** declaration implies **not null** anyway. (These fields must still not be **null**; it's just that

this does not need to be explicitly declared.)

- The **not null** declarations on the remaining fields — price, available, and name for both Seller and Product — are not essential to the integrity of the model, so whether they are included or removed depends more on the particular scenario. Here, for example, it's probably important to always have a name for a product, but the price might at times be unknown.

## Solution 5

(a) The double line indicates that Branch is a *weak entity*. This is necessary because a branch name is only unique within its project — many different projects will have their own "master" branch. The primary key for a Branch is the composite {Branch.name, Project.name}.

(b) Here are some appropriate data declarations.

```
create table Coder (
  username  varchar(64),
  name      varchar(200),
  primary key (username)
)

create table Project(
  name  varchar(200),
  leader varchar(64) not null,
  primary key (name),
  foreign key (leader) references Coder(username)
)

create table Branch(
  name    varchar(200),
  project varchar(200),
  primary key (name,project),
  foreign key (project) references Project(name)
)

create table WorksOn(
  coder   varchar(64),
  branch varchar(200),
  project varchar(200)
  primary key (coder, branch, project),
  foreign key (coder) references Coder(username),
  foreign key (branch) references Branch(name),
  foreign key (project) references Project(name)
)
```

There's quite a lot to put together here, from the very simple Coder table to the WorksOn relationship that references all three other tables.

Note that project leaders are identified with a leader field in the Project table. It is incorrect to try and do this with a project field in Coder, as that's the wrong way round — for example, the same person might lead two different projects.

Using a separate table Leads to match projects with their lead coder isn't quite enough. That can capture the key constraint (no more than one leader per project) but cannot express the total participation constraint (each project must have a designated leader). In the table above that's shown with the **not null** constraint on the Project.leader field.

The table for the weak entity Branch needs to reference the Project name, and include that as part of its primary key. Without this there would be no way to distinguish between the "main" branches of two different projects.

Because the Works-on relationship is many-to-many it requires its own table. Moreover, because Branch is a weak entity, it must also bring in the Project name to uniquely identify the branch on which a coder works. In the code above, the field WorksOn.project directly references the Project table. However, it can also be done through the Branch table, like this:

**create table** WorksOn(
  ...
  **foreign key** (branch,project) **references** Branch(name,project)
)

## Solution 6

(a) These SQL declarations express the ER model recording expertise in various fields.

```
create table Expert (
    id          varchar(6),
    name        varchar(30),
    affiliation varchar(20),
    email       varchar(20),
    primary key (id)
)

create table Field (
    fieldname   varchar(20),
    leadexpert  varchar(6),
    primary key (fieldname),
    foreign key (leadexpert) references Expert(id)
)
```

```
create table ExpertIn (
    id        varchar(6),
    fieldname varchar(20),
    primary key (id,fieldname),
    foreign key (id) references Expert,
    foreign key
(fieldname) references Field
)
```

(b) The only constraint in the diagram is a key constraint from Field to LeadingExpert, recording the requirement that there be no more than one recorded leading expert for each field of study. In the tables above, this is captured by the fact that leadingexpert is part of the Field table, with a **foreign key** constraint referencing the Expert table.

This leadingexpert field does not have a **not null** declaration, because there is no total participation constraint in the ER model: it is acceptable for a particular field to have no leading expert recorded in the database.

Notice the contrast with the ExpertIn relationship, which appears not as a single field but a whole table in the relational model, allowing for many experts in each field.

# Solution 7

(a) The following declarations map the ER diagram for a staff database into an SQL model. As in Solution 4, one relationship is captured in a single field: the Head of Department relationship is held in the hod field of the Department table.

```
create table Staff (                    create table BelongsTo (
    id       varchar(6),                    id         varchar(6),
    name     varchar(30),                   deptname varchar(20),
    jobtitle varchar(10),                   primary key (id,deptname),
    salary   integer,                       foreign key (id) references Staff,
    primary key (id)                        foreign key (deptname) references Department
)                                       )

create table Department (
    deptname varchar(20),
    address   varchar(40),
    hod       varchar(30) not null,
    primary key (deptname),
    foreign key
(hod) references Staff(id)
)
```

There is an alternative presentation using tables for all relationships, shown below. However, as will be seen in the solution to (b), this does not capture as many of the constraints in the ER model.

```
create table Staff (                    create table BelongsTo (
    id       varchar(6),                    id          char(6),
    name     varchar(30),                   deptname    char(20),
    jobtitle varchar(10),                   primary key (id,deptname),
    salary   integer,                       foreign key (id) references Staff,
    primary key (id)                        foreign key (deptname) references Department)
)
                                        create table HeadOfDept (
                                            deptname    char(20),
create table Department (                   id          char(6),
    deptname varchar(20),                   primary key (deptname),
    address   varchar(40),                  foreign key (id) references Staff,
    primary key (deptname)                  foreign key (deptname) references Department)
)
```

(b) The original ER diagram had three constraints:

- A participation constraint from Staff to BelongsTo, recording that each member of staff must belong to at least one department (but possibly more than one);

- A participation constraint from Department to Head, showing that each department must have a head; and

- A key constraint from Department to Head, showing that each department can have at most one head (and therefore exactly one).

The first set of tables in **(a)** captures the key constraint by making hod a field in Department, with a **foreign key** link to Staff: so there can be only one head of department recorded. This is then enhanced to capture the total participation constraint by declaring hod as **not null**: so there must always be some head of department. These tables do not, however, capture the total participation constraint saying that every member of staff must be in at least one department.

The alternative tables presented second in **(a)** capture none of the constraints in the original ER model.

## Solution 8

**(a)** Here are some SQL declarations for the tennis association.

```
create table Player (              create table Match (
    playerref    integer,             matchref     integer,
    name         varchar(20),         matchyear    integer
    nationality  varchar(10),         tournament   varchar(15),
    primary key (playerref)           round        varchar(15),
)                                     winner       integer not null,
                                      loser        integer not null,
                                      primary key (matchref),
                                      foreign key (winner) references Player(playerref),
                                      foreign key (loser)  references Player(playerref)
                                  )
```

It might be reasonable to declare some further fields as **not null**, depending on the application scenario: such as name of Player, or tournament, matchyear and round in Match.

**(b)** The participation and key constraints are from each Match entity to the Winner and Loser relationships. The key constraints are captured by making winner and loser both foreign key fields of the **Match** table, rather than stand-alone relations; and the participation constraints are expressed through the **not null** declarations on winner and loser.

The outcome is that every row in the Match table must have exactly one winner and loser recorded, both of which must be playerref values for members of the Player table.

# Solution 9

(a) These are suitable SQL declarations for the theatre manager.

```
create table Diary (                         create table Company (
    date     varchar(8),                         cname   varchar(30),
    seats    integer,                            country varchar(15),
    pname    varchar(30),                        primary key (cname)
    director varchar(30),                    )
    primary key (date),
    foreign key (pname,director) references Production
)


create table Production (
    pname    varchar(30),
    director varchar(30),
    cname    varchar(30) not null,
    primary key (pname,director),
    foreign key (cname) references Company
)
```

(b) There is a key constraint from Diary to Performance, indicating that on each diary date there can be at most one performance. This is expressed by making the **pname** and **director** of that performance be fields in the Diary table, declared as foreign keys referencing the Production table. Notice that these may be **null**, which would indicate a day on which there is no performance.

In the ER model there is also a key and total participation constraint from Production to Company, showing that each production must be performed by exactly one company. This is captured in the SQL again with a foreign key field, in this case putting the company name **cname** as a field in the Production table. Declaring this as **not null** enforces the total participation constraint.

# Solution 10

(a) The *cardinality* of a database table is the number of rows (records, tuples) it contains.

(b) The *arity* of a database table is the number of columns (fields, attributes) it contains.

(c) A *foreign key* in a database table is a field whose value must appear as the primary key of a record in another, specified, table.

(d) A *selection* operation on a database table picks out all rows of the table satisfying some predicate.

(e) A *projection* operation on a database table selects out some named columns of the table.

(f) The *intersection* $R \cap S$ of two tables $R$ and $S$ contains all records that appear in both $R$ and $S$.

(g) The *natural join* $R \bowtie S$ of two tables $R$ and $S$ contains all rows of $R$ combined with all rows of $S$ that agree on those named fields which the two tables have in common.