Tutorial 2: Relational Modelling

Informatics 1 Data & Analysis — Tutorial Notes

Week 4, Semester 2, 2018/19

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* or at *InfBASE*.
- The Examples are there for additional study, practice, and exam preparation.
- 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 post a question on *Piazza*.

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
```

(Tutor Notes) The following are syntax errors in this table declaration, wrong in any scenario.

- There is no closing parenthesis.
- The release_year field has no comma between it and the next field declaration.
- The director name attribute is two words and so either needs quoting "director name" or linking into a single word director_name.
- The **foreign key** constraint on director name should put the field name in parentheses.
- The leading_actor field mentioned in the final constraint has not been declared at all.

Here is a version of the declaration that corrects these mistakes.

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

Even with this version, some variations might be possible. For example, it would be reasonable to also give the exact name of the keys referenced in the foreign key constraints. Thus, we might have:

```
    foreign key (leading_actor) references actor(name)
```

and similarly for director(name). In each case it is not essential — SQL will anyway correctly find the primary key in the referenced table — but it can add clarity.

Question 2: Mapping an ER Model to Relational Schemas

For this question, please use the ER model in Figure 1 on page 7, 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?

(Tutor Notes) Here is a set of SQL declarations for the ER diagram provided. The diagram does not force any particular field lengths or types, but choices ought to be sensible. SQL keywords are not case-sensitive, so although these declarations keep them lower-case that isn't essential. The names of attributes, however, are case-sensitive in some systems so should match those in the diagram.

```
create table GraphicDesigner (
    name        varchar(30),
     affilia tion varchar(30),
    posterId    integer,
    primary key (name, affiliation),
    foreign key (posterId) references Poster(id)
    )
```

```
create table Poster (
    id     integer,
    title  varchar(50),
    primary key (id)
    )
```

```
create table Judgement (
    posterId integer,
    judge    varchar(30),
    decision varchar(6),
    primary key (posterId, judge),
    foreign key (posterId) references Poster(id),
    foreign key (judge) references Judge(email)
    )
```

```
create table Judge (
    name        varchar(30),
     affilia tion varchar(30),
    email       varchar(30),
    primary key (email)
    )
```

This realisation of the ER model as SQL tables captures the key constraint that each graphic designer may be involved with at most one poster by putting the reference to that poster as a field in the GraphicDesigner table. As the ER model permits graphic designers to create no posters at all, some records in the GraphicDesigner table may have the **null** value for posterId.

An alternative approach for mapping relationships is to use a distinct table for each relationship. For this example that would give a simpler GraphicDesigner table, and a new table Creates.

```
create table GraphicDesigner (
    name        varchar(30),
     affilia tion varchar(30),
    primary key (name, affiliation)
    )
```

```
create table Creates (
    name        varchar(30),
     affilia tion varchar(30),
    posterId    integer not null,
    primary key (name, affiliation),
    foreign key (name, affiliation) references GraphicDesigner,
    foreign key (posterId) references Poster(id)
    )
```

This version avoids **null** entries, as table Creates only has tuples for graphic designers who do have posters. It also captures the key constraint that each designer can be involved with at most one poster: by choosing as primary key {name, affilia tion} no designer can appear in more than one Creates tuple.

Both versions are correct, and exactly which one to choose will depend on the wider scenario being modelled.

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?

(Tutor Notes) In most RDBMSs primary keys may not be **null**, regardless of the table declaration. This can be extended to additional fields using an explicit **not null** declaration.

Participation constraints can be expressed using **not null** declarations if they are coupled with a key constraint. However, in this case participation and key constraints do not coincide.

The final reason for using **not null** declarations is where the expected use of the database would mean certain fields should always be filled. In this scenario, for example, we might expect that every Poster should have a non-null title, and every Judge should have a name recorded.

(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?

(Tutor Notes) In the first GraphicDesigner table given above there is a foreign key constraint on the posterId field. This could be declared **on delete set null**, so that if a poster is deleted its associated designers remain on the database — just with no listed poster.

On the other hand, the foreign key references in Judgement and Creates could all be declared **on delete cascade**, because these are recording relationships (between a judge and a poster, and a graphic designer and a poster, respectively) and when the judge, poster, or graphic designer is removed then there is no need for the relationship tuple either.

(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?

(Tutor Notes) The new condition would mean total participation of the GraphicDesigner entity in the Creates relationship, denoted by a bold or double line (still being an arrow) from GraphicDesigner to Creates. In the relational model, it would mean declaring a **not null** constraint on the posterId field in the GraphicDesigner table.

```
create table GraphicDesigner (
    name        varchar(30),
    affilia tion varchar(30),
    posterId    integer not null,
    primary key (name, affiliation),
    foreign key (posterId) references Poster(id)
    )
```

In the alternative representation, with a separate Creates relation, there is no immediate way to capture the total participation constraint.

(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?

(Tutor Notes) No: without an accompanying key constraint there is no straightforward way to capture these participation constraints in the SQL declarations.

(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?

(Tutor Notes) The following declaration shows one way to (partly) capture this in the relational model.

```
create table Poster (
    id      integer,
    title   varchar(50),
    judge1 varchar(30) not null, decision1 varchar(6),
    judge2 varchar(30) not null, decision2 varchar(6),
    judge3 varchar(30) not null, decision3 varchar(6),
    primary key (id),
    foreign key (judge1) references Judge,
    foreign key (judge2) references Judge,
    foreign key (judge3) references Judge
    )
```

This lists three judges for each poster, with **not null** constraints to enforce that these be filled, and fields to record the decision by each. It also captures the total participation constraint that every poster must have judges. We drop the Judgement table completely, as the relationship is now folded into the Poster table.

However, it is still only a partial solution in that we have no way to ensure that these are three *different* judges — with the original Judgement relation, having a composite primary key of {posterid, judge} meant that no poster could be judged more than once by any judge.

To enforce that all judges must be different would require a further layer of checking. This can be done using *triggers* — code which is executed automatically whenever certain events occur in a database. However, these are beyond the curriculum of Inf1-DA.

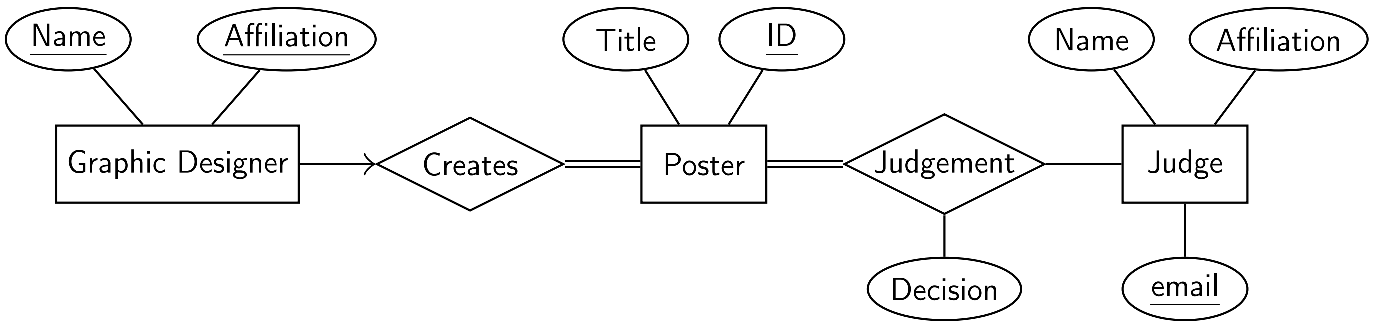


Figure 1: An ER model for part the poster exhibition scenario from Tutorial 1. There are double lines from Poster to Creates and to Judgement, and an arrow from Graphic Designer to Creates.

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?

(Tutor Notes) The relational model as shown so far is not able to implement all the constraints available in ER diagrams; although these can be captured using more advanced SQL features. Conversely, the mechanism of foreign key constraints in SQL gives a way of linking fields between tables without additional intervening relations, which is more succinctly expressive than ER modelling.

As you might expect, both modelling formalisms have their advantages and limitations. They are, however, also aimed at different application areas, with ER diagrams for conceptual modelling and SQL for precise details of final implementation.

(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?

(Tutor Notes) Although it is possible to automatically translate ER diagrams into relational declarations, it is often smarter to do so while using the original scenario to help guide detailed design questions.

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 9. 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?

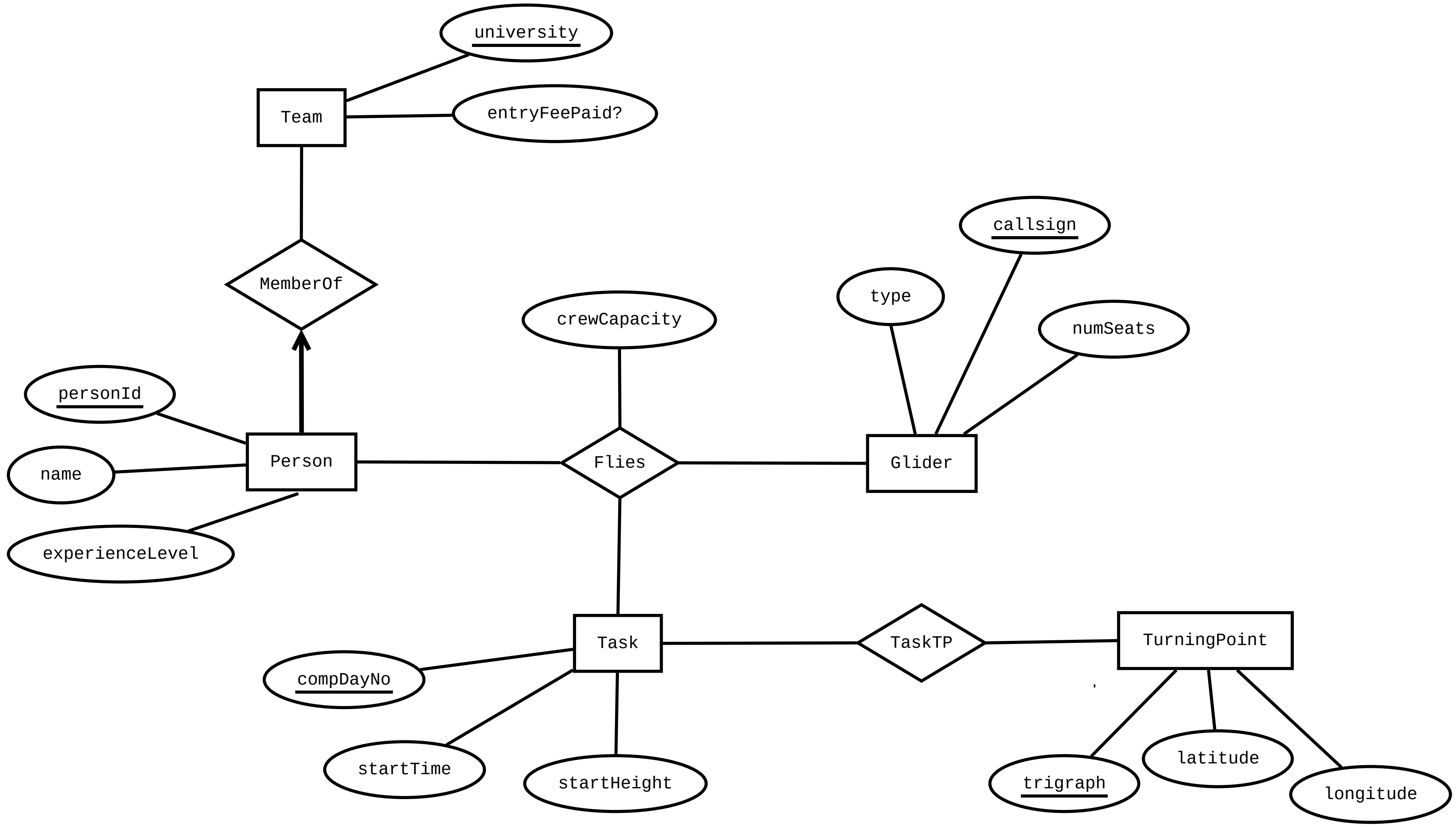
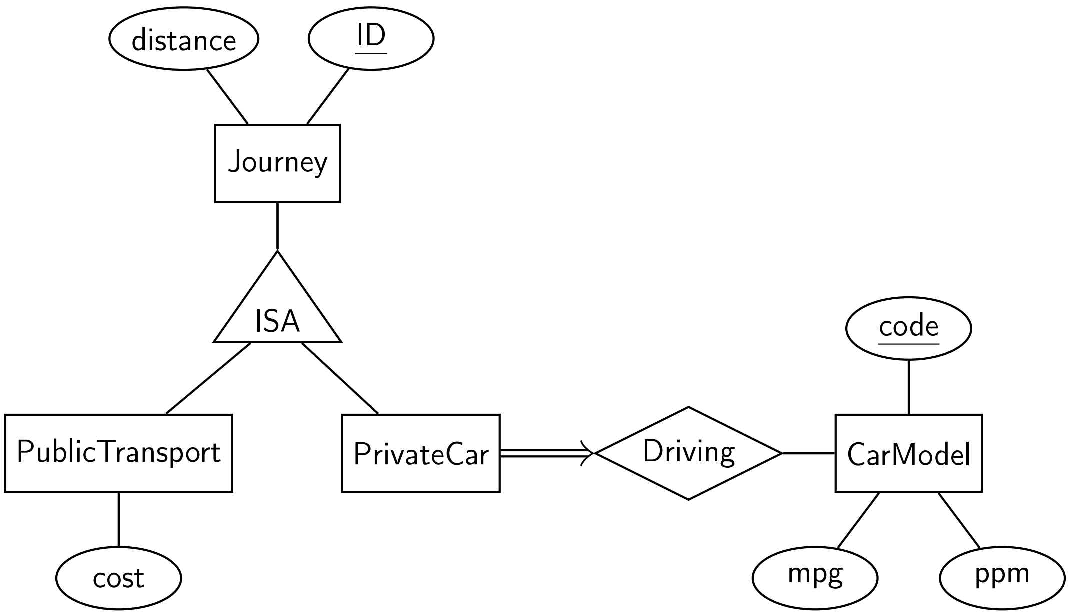


Figure 2: An Entity-Relationship model for the gliding competition database.

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.



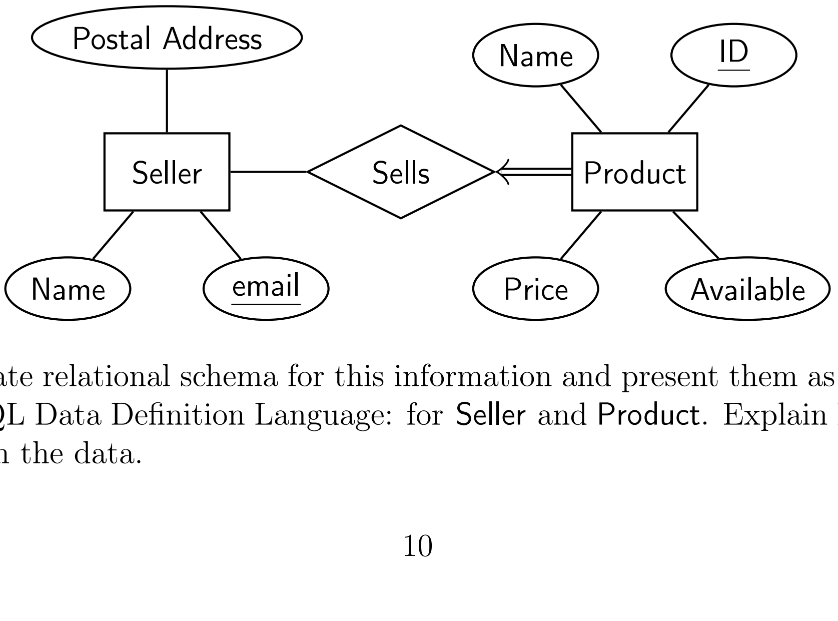
You also have the following information about the model.

- 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

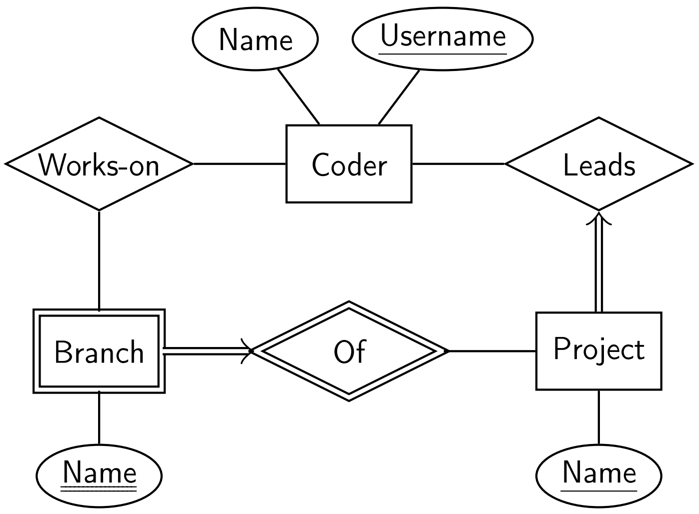
The following ER diagram appeared in Tutorial 1 as a suitable model for the *Itsy! Bitsy!* craft trading website.



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”.

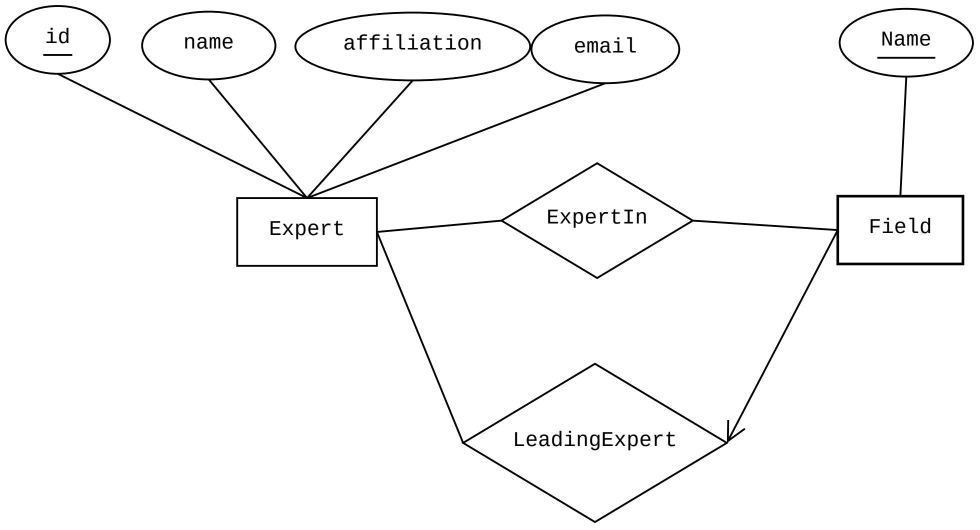


(a) What is the meaning of the double line around Branch? Why is this needed? What is the primary key for a Branch?

(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.

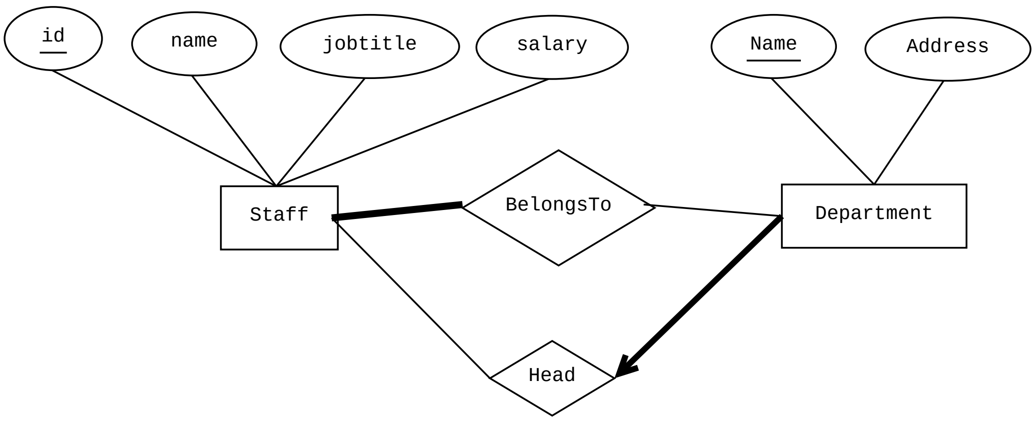


(a) Use the SQL Data Definition Language to declare relational schemas that implement this ER diagram.

(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.

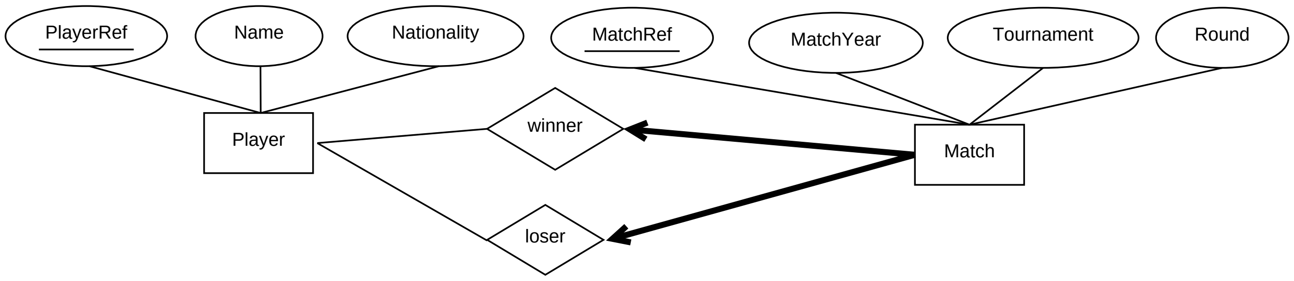


(a) Use the SQL Data Definition Language to declare relational schemas that implement this ER diagram.

(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.

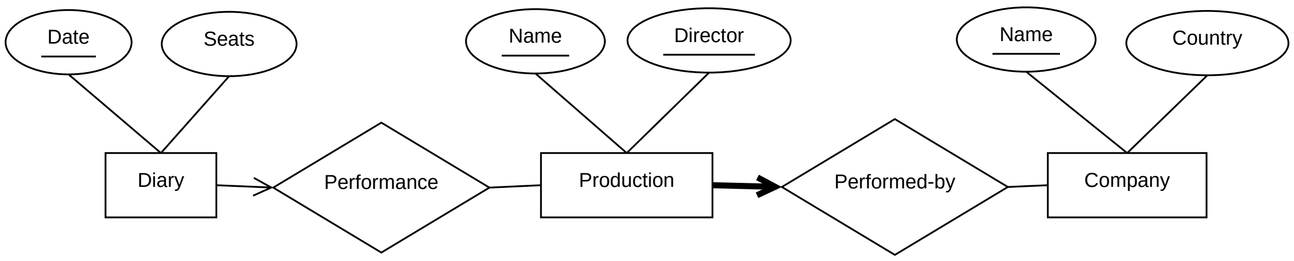


(a) Use the SQL Data Definition Language to present relational schema that implements this diagram using two tables: Player and Match.

(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 (
    personId    integer,
    university  char(50),
    primary key (personId),
    foreign key (personId)   references Person,
    foreign key (university) references Team
    )
```

```
create table Person (
    personId        integer,
    name            varchar(30),
    experienceLevel varchar(15),
    primary key (personId)
    )
```

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 (
  ID          varchar(8),
  distance    integer,
  primary key (ID)
)
```

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

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

```
create table CarModel (
   code        varchar(8),
   mpg         integer not null,
   ppm         integer not null,
   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 (
    name    varchar(30) not null,
    email   varchar(30) not null,
    address varchar(200) not null,
    primary key (email)
)
```

```
create table Product (
    name      varchar(60) not null,
    id        integer not null,
    price     integer not null,
    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),
     affilia tion 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 (
    id       varchar(6),
    name     varchar(30),
    jobtitle varchar(10),
    salary   integer,
    primary key (id)
)

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

create table Department (
    deptname varchar(20),
    address  varchar(40),
    primary key (deptname)
)

create table BelongsTo (
  id         char(6),
  deptname   char(20),
  primary key (id,deptname),
  foreign key (id) references Staff,
  foreign key (deptname) references Department)

create table HeadOfDept (
  deptname   char(20),
  id         char(6),
  primary key (deptname),
  foreign key (id) references Staff,
  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 (
    playerref   integer,
    name        varchar(20),
    nationality varchar(10),
    primary key (playerref)
)
```

```
create table Match (
    matchref    integer,
    matchyear   integer
    tournament  varchar(15),
    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 (
    date      varchar(8),
    seats     integer,
    pname     varchar(30),
    pdirector varchar(30),
    primary key  (date),
    foreign key  (pname,pdirector)
            references Production(name,director)
)

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

create table Company (
    name    varchar(30),
    country varchar(15),
    primary key (name)
)
```

(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 name and director of that performance be fields in the Diary table, pname and pdirector, 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 as a field cname in the Production table. Declaring this as **not null** enforces the total participation constraint.

Both of the **foreign key** declarations explicitly list the key fields of the table being referenced. For example, in the Production table the field cname is used for the name of the company performing so as not to clash with the name of the production itself; and then cname is declared as referring to Company(name), the name field in the Company table. The same thing happens in the Diary table when referring to the composite foreign key Production(name,director).

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.