Exercises, Database Technology

These are self-study exercises with solutions.

Exercise 1 — E/R modeling

Objective: to practice E/R modeling.

1. A calendar program that allows users to browse each other's calendars and to book common appointments shall be developed. The program has a database which keeps track of the users and their calendars.

   You use the calendar to store data concerning appointments. An appointment starts and ends at a given time on a given day and is described by a text. You may specify that you wish to be reminded of an appointment. Reminders are of different kinds: a signal in the computer's loudspeaker, a pop-up window with the description of the meeting, or an e-mail containing the description. You may, for each reminder, specify how long before the appointment that you wish to be reminded.

   Develop an E/R model of the database.

2. A TV company wishes to develop a database to store data about the TV series that the company produces. The data includes information about actors who play in the series, and directors who direct the episodes of the series.

   Actors and directors are employed by the company. A TV series are divided into episodes. Each episode may be transmitted at several occasions. An actor is hired to participate in a series, but may participate in many series. Each episode of a series is directed by one of the directors, but different episodes may be directed by different directors.

   Examples of database queries:

   - Which actors play in the series Big Sister?
   - In which series does the actor Bertil Bom participate?
   - Which actors participate in more than one series?
   - How many times has the first episode of the series Wild Lies been transmitted? At what times?
   - How many directors are employed by the company?
   - Which director has directed the greatest number of episodes?

   Develop an E/R model of this system. Find attributes of the entity sets. Determine which of the attributes that can be used as primary keys.

3. Develop an E/R model of a database that is to be used in the following system:

   A homeowners association (that is, an association of people who own apartments) owns a parking lot. The parking lot has a number of parking spaces. The owners and their guests may freely use all the parking spaces, except some spaces that have electric sockets for engine heaters. Such a parking space is rented by one of the apartment owners, who has exclusive use of the space. The rent for the space is added to the apartment rent.

The spaces with electric outlets are popular, and there is a queue of apartments that wish to rent such a space. Each apartment may have at most one place in the queue. When a space becomes available, the apartment with the longest queue time may rent it. One of the apartment owners must sign the contract (an apartment may have more than one owner, and an owner may own more than one apartment).

The association sometimes has problems with scrap cars that are deposited in the parking lot. It is often a difficult procedure to get rid of these cars. Discovery of a scrap car must be registered in the database, as well as each thing that happens with the car (Parking space 43: 2009–04–26 "Discovered suspect car, license number XYZ789", 2009–05–02 "Called the police about the car", and so on).

4. A municipality needs a database containing information concerning the inhabitants of the municipality. The database will be used for the planning of schools, health care and child care.

From the database, you should be able to receive answers to queries of the following types:

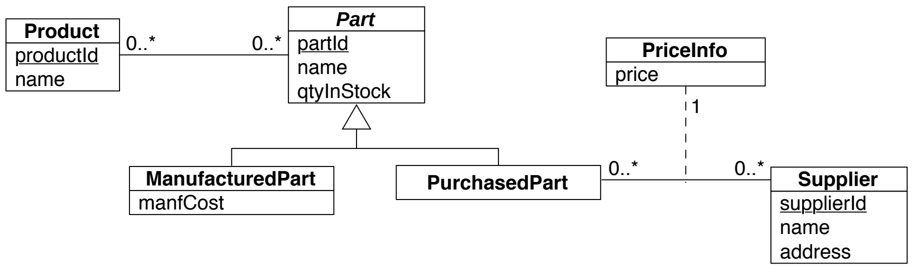
- How many boys and girls will start school during year x?
- How many people will become old-age pensioners during year x?
- How many households have more than x people?
- How many people are single parents?
- In how many households is at least one member unemployed?
- How many households have a total income that is less than the norm for receiving social benefits?

Develop an E/R model of the database, including attributes and primary keys. Carefully consider the representation of "household" and "parent".

Exercise 2 — from E/R to relations, SQL

Objective: to practice translating E/R diagrams into relations and assessing advantages and disadvantages of different translation alternatives. You will also practice more E/R modeling and SQL.

1. A company produces and sells products that are assembled from parts. A part is either manufactured internally or bought from a supplier. In the company database this is modeled as follows:



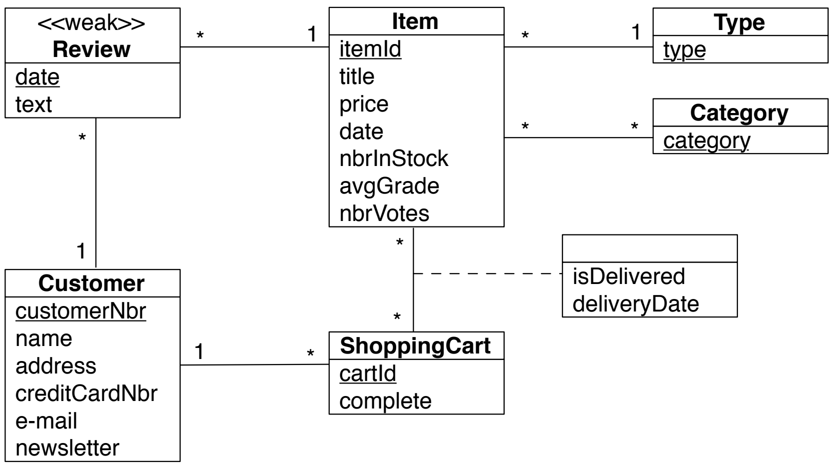
a) Translate the E/R diagram into relations. Make the translation of the hierarchical structure between the Part entity sets in three variants, according to the three methods described in the textbook.

b) Describe some typical tasks that the company wishes to solve with the help of the database, and some typical questions to which the company wishes to receive answers.

c) Which are the advantages and disadvantages with the different alternatives for the relations for the Part entity sets, when you wish to answer the questions in question b?

2. The E/R diagram below describes a web store, which sells DVD's and CD's ("items") to customers. Items may belong to several categories. Customers may write reviews of the items that they have purchased.

   Translate the E/R model into a relational model.



3. A medical health research project has a database containing data about all patients at a hospital. For each patient, data about the symptoms that the patient shows is registered: fever, headache, cough, chest pains, ... Symptoms can have different severity: low, middle, or high. A patient may show several symptoms, e.g., high fever, medium headache and some cough.

The database also contains data about diseases. Each disease is characterized by different symptoms: a patient with a cold should have fever and a cough, a malaria patient should have fever and fits of shivering, etc.

a) Describe this system in an E/R model.
b) Translate the E/R model into a relational model.
c) Write SQL statements that answer the following questions (you may define and use views, if you wish). Find the names of all patients that:
   - don't have any symptom of high severity,
   - have at least two different symptoms,
   - have at least one of the symptoms of malaria,
   - have all the symptoms of malaria.

4. Shops sell items at varying prices. Customers buy items from shops. This is described by the following relations:

   Shops(shopId, name, address)
   Items(itemId, name, description)
   Sells(*shopId, itemId*, price)
   Customers(customerId, name, address)
   Sales(saleId, *customerId, itemId, shopId*, date)

   Draw a diagram in question **??**, write SQL statements in questions **??–??** (you may create and use views).

   a) Draw an E/R diagram that describes the database.
   b) Create the table Sells. Invent suitable types for the attributes (itemId shall have the type `char(10)`) and indicate reasonable integrity constraints.
   c) Print the name and address of all customers who haven't bought any item.
   d) Print the number of shops that sell items with id's starting with `'EF'`.
   e) Print the name and address of the shop(s) that sell the item with id = `'EF123-A'` at the lowest price.
   f) For all customers that have bought at least one item: print the customer id and the total sum of all purchases.

Exercise 3 — FD's and normalization

Objective: to practice finding functional dependencies, finding keys and normalizing relations.

1. The relation R(A, B, C, D, E) has the following functional dependencies:

   FD1. $A \to C$
   FD2. $B \to D$
   FD3. $AC \to D$
   FD4. $CD \to E$
   FD5. $E \to A$

   Determine all the keys of the relation.

2. The relation R(A, B, C, D) has the following functional dependencies:

   FD1. $D \to AC$
   FD2. $A \to B$
   FD3. $B \to C$

   a) What are the keys of the relation?
   b) Show that the relation is not in BCNF and not in 3NF.
   c) Decompose the relation into smaller relations that are in BCNF.

3. A company has several employees, all with different names, who perform interviews with job applicants (one applicant is interviewed by one employee). The job applicants also have different names. The interviewer makes appointments for interviews with the applicants. Each applicant may be interviewed at several occasions, possibly by different interviewers, but in that case the interviews take place during different days.

   The company has special interview rooms. Each interviewer uses the same room for all interviews during a day. A room may, however, be used by different interviewers during a day, as long as the interviews don't collide in time.

   The reservation of interview appointments is to be computerized. The database developer has decided to use a single relation for all data, with the following schema:

   Interviews(interviewer, applicant, day, time, room)

   a) From the text, find functional dependencies in the relation.
   b) Find the keys of the relation.
   c) Show that the relation is in 3NF but not in BCNF.
   d) Decompose the relation in relations that are in BCNF.
   e) Extra assignment: draw an E/R diagram that describes the system. Try to incorporate all dependencies from the text in the diagram (not easy).

4. We wish to develop a database to keep track of persons, their children and their cars. For this purpose, we will use the following relation:

   PersonData(pNbr, pName, pAddress, cNbr, cName, cAddress, aLic, aMake)

   pNbr, pName, pAddress is the person number, name and address of a person. cNbr, cName, cAddress is the corresponding information for a child. Each person has exactly one address. aLic, aMake is the license number and make of a car. A car may be owned by more than one person.

   a) What are the functional dependencies in this relation?

b) Find the keys of the relation and show that the relation is not in BCNF.

c) Decompose the relation into relations that are in BCNF.

d) Show that one of the resulting relations violates the 4NF condition.

e) Decompose this relation into relations that are in 4NF.

f) Draw an E/R diagram that describes the problem and use this diagram to create relations. Persons and children are considered to be different entity sets. Are the resulting relations the same as the ones that you created in questions **??** and **??**?

Exercise 4 — database design

Objective: to practice formulating simple queries in relational algebra. Then, you will summarize your database knowledge by designing two databases.

1. A company organizes its activities in projects. Products that are used in the projects are bought from suppliers. This is described in a database with the following schema:

   Projects(projNbr, name, city)
   Products(prodNbr, name, color)
   Suppliers(supplNbr, name, city)
   Deliveries(*supplNbr, prodNbr, projNbr*, number)

   Write relational algebra expressions that give the following information:

   a) All information about all projects.
   b) All information about all projects in London.
   c) The supplier numbers of the suppliers that deliver to project number 123.
   d) The product numbers of products that are delivered by suppliers in London.
   e) All pairs of product numbers such that at least one supplier delivers both products.

2. Another company also organizes its activities in projects. Each project has a name and is run at a specific location. Unfortunately, there are several projects with the same name — it may even be the case that two projects with the same name are run at the same location. Each project has a telephone, which is shared if there is more than one project at the same location. Each project has a boss, who is one of the employees of the company.

   Each company employee has a person number, address and telephone number. Many of the employees have several telephone numbers (office telephone, cellular 1, cellular 2, . . .). The employees work in the projects, most of them in more than one project.[1] Each such "project employment" starts at a specific date and ends at another date (the end date of current project employments is unknown).

   a) Describe the structure of the company in an E/R model.
   b) Convert the E/R model to a relational model. Indicate attributes that are primary keys and foreign keys.
   c) Prove that the relations are in BCNF.

3. In a botanical survey, an inventory is made of the Swedish flora, i.e., it is investigated where different plants grow. Plants are identified by their Latin names: *Anemone nemorosa*, *Ranunculus ficaria*, etc.

   The survey is made at different sites. A site is described by its name ("The Midsummer Meadow in Stolphult"), its type ("meadow"), and its coordinates in the coordinate system Swedish Grid ("153100E, 670300N"). At a site, investigations are performed in 1×1 m squares. Each square also has coordinates, which are measured relative to the site coordinates. For each plant that occurs in a square, the degree of coverage (in percent) is recorded.

   Chemical analyses of different chemical properties are performed in some of the squares. Which analyses that are performed may vary, but common measurements are pH and the content of different heavy metals. The results of the measurements are given in different units: no unit, ppm, etc.

[1] However, there are employees who have worked in the same project during all their time in the company. Furthermore, there are employees who do not participate in a project, e.g., new employees.

The survey involves a lot of people. Each person has a person number, name, and address. Each square is investigated by one person.

a) Develop an E/R model that describes the survey.
b) Translate the E/R model into a relational model. All relations must be in BCNF (prove, or at least motivate carefully, that this is the case). Indicate primary keys and foreign keys in all relations.
c) Using your relations in question **??**, write an SQL statement that gives the names of all plants that occur in squares that have a pH value less than 4.

Exercise 1 — Solutions

1. We start by identifying candidates for entity sets, for example by writing down the nouns and noun phrases in the requirements specification. We obtain the following list:

   Calendar program, user, calendar, appointment, program, database, track, data, time, day, text, reminder, kind, signal, computer, loudspeaker, pop-up window, description, meeting, e-mail.

   Not all of these candidate are suitable entity sets. We discard most of the candidates, for the following reasons:

Irrelevant	Calendar program, program, database, track, kind, computer, loudspeaker
Attributes	time, day, text (of appointment), time (of reminder)
Operations	—
Redundant	data (same as attributes of appointment), description (of appointment, same as text), meeting (same as appointment)
Misleading names	—

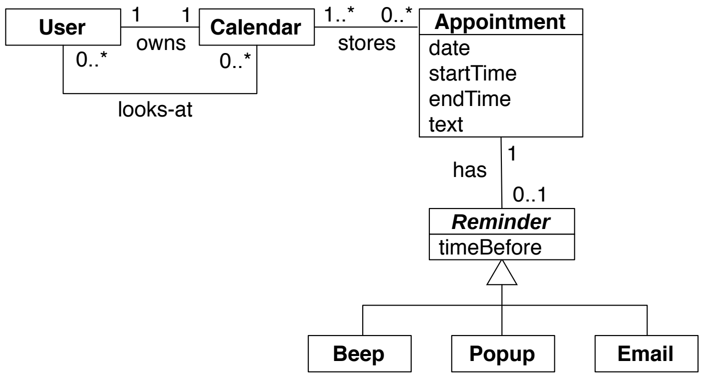
Remaining candidates: user, calendar, appointment, reminder, signal/pop-up window/e-mail.

Relationships from the requirements specification:

- users browse each other's calendars
- users have calendars
- calendars store appointments
- appointments may be in several calendars
- appointments may have reminders

An obvious generalization is that there are three kinds of reminders: signal, pop-up window, e-mail.

This results in the following E/R diagram:



The relationship "has" between Appointment and Reminder seems to express what we wish, but note that it says that an appointment has the same reminder in all calendars. If an appointment could have different reminders in different calendars, we can make Reminder an association class on the relationship "stores" between Calendar and Appointment.

Note that this is a conceptual E/R model, not a database model. Before we can transform the E/R model into a relational model we must study it closer and modify it quite a lot. For instance, the relationship "looks-at" will only be necessary if it has attributes, e.g., if users only have access to certain other calendars. The entity sets User and Calendar are not complete — they don't have any attributes, and we cannot form a key for the entity sets. Probably, this doesn't mean that the entity sets are unnecessary, but rather that we haven't yet found any suitable attributes. Appointment also needs an attribute that can be used as a key. — Finally, we can note that Reminder, at least in its present form, is a weak entity set.

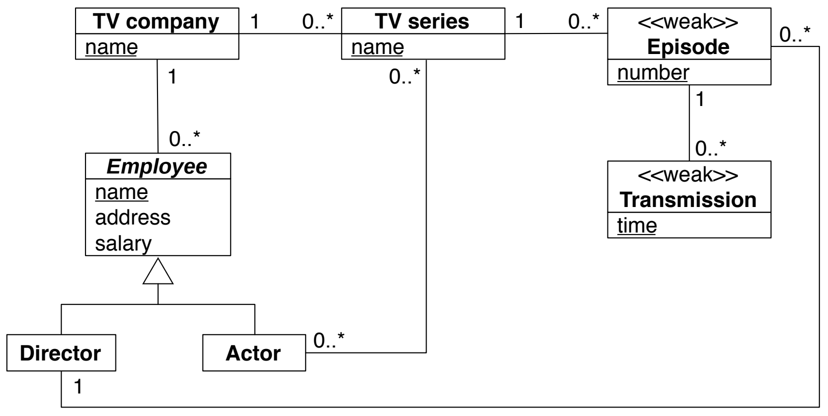
2. We use the same procedure as in question **??**. First, candidates for entity sets:
TV company, database, data, TV series, company, information, actor, series, director, episode, occasion.
Discarded candidates:

Irrelevant	database, data, information
Attributes	occasion (the time when an episode is shown)
Operations	—
Redundant	company (same as TV company, series (same as TV series)
Misleading names	—

Since each episode may be transmitted at several occasions we need an entity set to hold the transmission time. This entity set we call Transmission.

We also need an entity set that describes the common features of actors and directors, namely that they are employed by a TV company. We call this entity set Employee. Entity sets Actor and Director will be subentities to Employee.

E/R diagram:



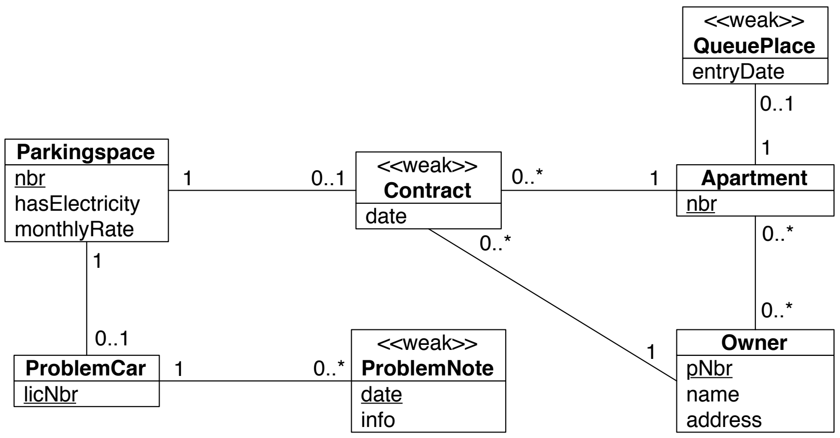
From the requirements specification we only obtain the attribute "time" in Transmission. Other attributes have been invented.

Entity sets Episode and Transmission are weak. To form the key for Episode, "number" alone does not suffice: you need to know the name of the TV series as well. Similarly for Transmission: the time, episode number, and TV series name together form the key.

We can check that it is possible to answer the database queries using only the E/R diagram. To check that we can answer the query "Which actors play in the series Big Sister?" we check that there is a relationship between the involved entity sets, here Actor and TV series.

We can also check more complex queries, for instance "At what times can we see the actor Bertil Bom?". By starting in Actor and following relationships we first reach the TV series in which Bertil Bom participate, then the episodes of this series, then the transmissions of the episodes, and there we find the times. (In the database this will correspond to select statements with several relations, "joins".)

3. E/R diagram:

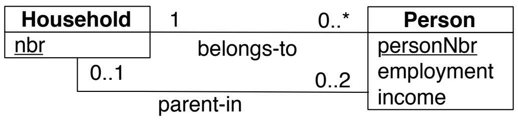


4. Suggested entity sets: person, household.

Reasons for this choice: Inhabitant means the same as Person, which is a better name. Boy and girl are not entity sets — the gender of a person is coded into the person number, which is an attribute. Pensioner is not an entity set — the year of birth of a person is coded into the person number. Parent is not an entity set — the parenthood is introduced when a person has children, and this will be described by a relationship.

Attributes of the entity set Person (minimal set): person number (containing year of birth and gender), employed/unemployed, income. Household attributes: household number (or something else that uniquely identifies a household).

E/R diagram:



Note especially the relationship "parent-in". This relationship does not describe a biological parenthood, only that the person is considered to be a parent in a household. This will make it easy to handle people that are divorced and later form a new household with a new set of children.

Exercise 2 — Solutions

1. We start with the entity sets. There are two entity sets that must be converted into relations, and where we don't have any alternatives for the implementation (primary keys are underlined):

   Products(productId, name)
   Suppliers(supplierId, name, address)

   The many-many relationships must also be converted into relations (foreign keys are in italics):

   ProductParts(*productId, partId*)
   PartSuppliers(*partId, supplierId*, unitPrice)

   When we are to translate the hierarchical structure, we have three alternatives: E/R, OO and NULLS (see the course book). We get the following relations for the three alternatives:

E/R:	Parts(partId, name, qtyInStock)
	ManufacturedParts(*partId*, manfCost)
	PurchasedParts(*partId*)
OO:	Parts(partId)
	ManufacturedParts(*partId*, name, qtyInStock, manfCost)
	PurchasedParts(*partId*, name, qtyInStock)
	(The relation Parts is necessary because the foreign key partId in the relation ProductParts must lead to *one* relation.)
NULLS:	Parts(partId, typeCode, name, qtyInStock, manfCost)
	(The type code specifies whether the part is manufactured or purchased. If there had been any attributes in PurchasedParts, all of them would have become part of the relation.)

   Typical tasks that have to be solved:

   - Change a product so it contains a new manufactured part:

E/R:	Insert a new tuple in Parts and one in ManufacturedParts.
OO:	Insert a new tuple in Parts and one in ManufacturedParts.
NULLS:	Insert a new tuple with the corresponding type code in Parts.

     Regardless of which alternative we use, we also must insert a new tuple in ProductParts.

   - Update the stock (qtyInStock) when a purchased part is delivered:

E/R:	Change a tuple in Parts.
OO:	Change a tuple in PurchasedParts.
NULLS:	Change a tuple in Parts.

- Find the total cost to produce a product:

E/R:	Find all parts from ManufacturedParts that are part of the product, sum all manfCosts. Do the same for PurchasedParts, but sum all the corresponding unitPrices from PartSuppliers.
OO:	Same as E/R.
NULLS:	Find all parts from Parts with typecode 'manufactured', sum the manf-Costs. Then sum the unitPrices for the parts with type code 'purchased'.

In this model, it is difficult to find types of questions that demand that you, starting from a subclass, need to find the superclass to get information or vice versa. In such questions, the E/R variant requires that you make a join on the subclass table and the superclass table. (An invented example of such a question: Given the name of a manufactured part, find the price of that part.)

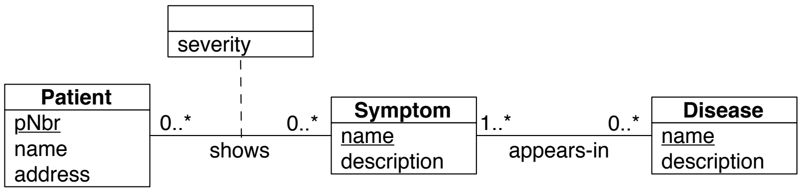
An advantage of hierarchical modeling should be that it is easy to introduce new subclasses. In this case it will probably not be relevant — it is hard to imagine a third category of parts that you do not either manufacture or purchase. If you do introduce a third category, it will lead to substantial changes in the database schema or in the SQL questions, regardless of which variant of translation you have used.

2. We choose to implement the many-one relationships by inserting the key from the "one side" into the relation created from the entity set on the "many side". Review is weak, but the key {customerNbr, itemId, date} is too long, so we add a key reviewId.

   Relations:

   Customers(customerNbr, name, address, creditCardNbr, e-mail, newsletter)
   Items(itemId, *itemType*, title, price, date, nbrInStock, avgGrade, nbrVotes)
   ShoppingCarts(cartId, complete, *customerNbr*)
   Types(type)
   Categories(category)
   Reviews(reviewId, date, text, *customerNbr*, *itemId*)
   ItemsInShoppingCart(*itemId*, *cartId*, isDelivered, deliveryDate)
   ItemsInCategories(*itemId*, *category*)

3. E/R diagram:



We have two many-many relationships, so if we merge all entity sets into one relation the resulting relation will not be in 4NF. But we, of course, use the ordinary method to convert the E/R diagram into relations, and we get the following relations:

Patients(pNbr, name, address)
Symptoms(name, description)
Diseases(name, description)
PatientSymptoms(*patientNbr*, *symptomName*, severity)
DiseaseSymptoms(*diseaseName*, *symptomName*)

All these relations are in 4NF — the only functional dependencies are the key dependencies and there are no multivalued dependencies.

Patients that have no symptoms with high severity:

```
select name
from Patients
where pNbr not in
    (select patientNbr
    from PatientSymptoms
    where severity = ‘high’);
```

Patients that have at least two different symptoms:

```
create view SymptomCount as
select patientNbr, count(*) nbrSymptoms
from PatientSymptoms
group by patientNbr;

select name
from Patients, SymptomCount
where pNbr = patientNbr and
      nbrSymptoms >= 2;
```

Patients that have at least one of the symptoms of malaria:

```
create view MalariaSymptoms as
select symptomName
from DiseaseSymptoms
where diseaseName = ‘Malaria’;

select patientNbr
from PatientSymptoms, MalariaSymptoms
where PatientSymptoms.symptomName = MalariaSymptoms.symptomName
group by patientNbr
having count(*) >= 1;
```

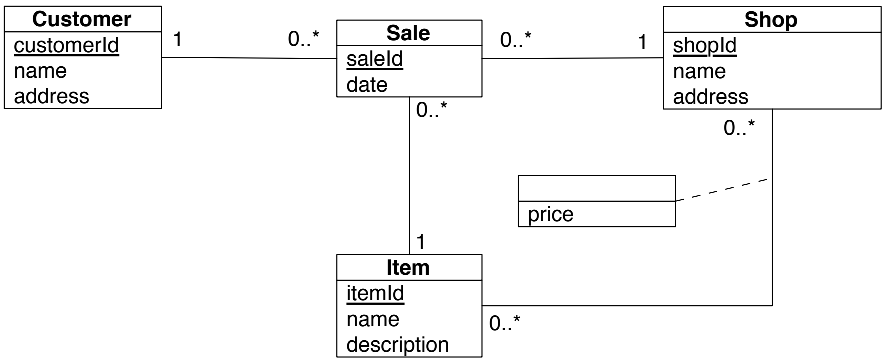
Patients that have all symptoms of malaria (the view MalariaSymptoms is used here, too):

```
select patientNbr
from PatientSymptoms, MalariaSymptoms
where PatientSymptoms.symptomName = MalariaSymptoms.symptomName
group by patientNbr
having count(*) = (select count(*) from MalariaSymptoms);
```

The last two queries give us person numbers instead of names. To find the names, you can use the query as a subquery, e.g., like this:

```
select name
from Patients
where pNbr in
    (select patientNbr
    from PatientSymptoms, MalariaSymptoms
    where PatientSymptoms.symptomName = MalariaSymptoms.symptomName
    group by patientNbr
    having count(*) = (select count(*) from MalariaSymptoms));
```

4. E/R diagram:



```
create table Sells (
    shopId char(10),
    itemId char(10),
    price  double not null,
    primary key (shopId, itemId),
    foreign key (shopId) references Shops(shopId),
    foreign key (itemId) references Items(itemId)
);

select name, address
from Customers
where customerId not in (select customerId from Sales);

select count(distinct shopId)
from Sells
where itemId like 'EF%';

select name, address
from Shops natural join Sells
where itemId = 'EF123-A' and
      price = (select min(price) from Sells where itemId = 'EF123-A');

select customerId, sum(price)
from Sales natural join Sells
group by customerId;
```

Exercise 3 — Solutions

1. To find the keys we must calculate the closures of all subsets of all combinations of attributes, i.e., $\{A\}^+$, $\{B\}^+, \ldots, \{AB\}^+, \{AC\}^+, \ldots$ If the closure of a subset contains all attributes of the relation, the subset is a key.

   Observation that makes things easier in the future: since B does not appear on the right-hand side of any of the functional dependencies, all keys must contain B. Thus, we only need to consider subsets that contain B.

   The functional dependencies were:

   FD1. $A \to C$
   FD2. $B \to D$
   FD3. $AC \to D$
   FD4. $CD \to E$
   FD5. $E \to A$

   Calculation of closures, subsets with one attribute that contain B:

   $$\{B\}^+ \Rightarrow \{B\} \overset{FD2}{\Rightarrow} \{BD\}$$

   Subsets with two attributes, containing B:

   $$\{AB\}^+ \Rightarrow \{AB\} \overset{FD1}{\Rightarrow} \{ABC\} \overset{FD2}{\Rightarrow} \{ABCD\} \overset{FD4}{\Rightarrow} \{ABCDE\}$$
   $$\{BC\}^+ \Rightarrow \{BC\} \overset{FD2}{\Rightarrow} \{BCD\} \overset{FD4}{\Rightarrow} \{BCDE\} \overset{FD5}{\Rightarrow} \{ABCDE\}$$
   $$\{BD\}^+ \Rightarrow \{BD\}$$
   $$\{BE\}^+ \Rightarrow \{BE\} \overset{FD2}{\Rightarrow} \{BDE\} \overset{FD5}{\Rightarrow} \{ABDE\} \overset{FD1}{\Rightarrow} \{ABCDE\}$$

   $\{AB\}$, $\{BC\}$ and $\{BE\}$ are keys.

   Subsets with three attributes that contain B: $\{ABC\}$, $\{ABD\}$, $\{ABE\}$ — all of these contain $\{AB\}$, so they do not need to be studied (they are superkeys). $\{BCD\}$, $\{BCE\}$ contain $\{BC\}$. $\{BDE\}$ contains $\{BE\}$. Similar reasoning shows that we don't need to study larger subsets.

2. To find the keys we do as in the previous assignment, i.e., calculate the closures of all subsets of the set of attributes. But the attribute D doesn't appear on the right-hand side of any of the dependencies, so it must be included in the key.

   FD1. $D \to AC$
   FD2. $A \to B$
   FD3. $B \to C$

   Closure of the subset $\{D\}$:

   $$\{D\}^+ \overset{FD1}{\Rightarrow} \{DAC\} \overset{FD2}{\Rightarrow} \{DACB\}$$

   The closure contains all attributes so $\{D\}$ is a key. It is the also the only key: all sets of two or more attributes that contain D are superkeys.

   FD2 and FD3 violate the BCNF condition, since the left-hand sides (A, B) aren't superkeys. The relation isn't in 3NF, since the right-hand sides of FD2 and FD3 (B, C) aren't part of the key.

We decompose starting from FD2 and get:

R1($\underline{\text{A}}$, B)
R2(A, C, $\underline{\text{D}}$)

R1 is in BCNF, since it has two attributes. R2 isn't in BCNF, since it has the same key as R and the dependency $A \rightarrow C$ (obtained from FD2 and FD3 and the transitive rule). We split R2 using $A \rightarrow C$:

R2a($\underline{\text{A}}$, C)
R2b(A, $\underline{\text{D}}$)

Both these relations have only two attributes and are in BCNF.

3. Functional dependencies:

FD1. applicant day $\rightarrow$ interviewer time room
(Each day an applicant is interviewed, the interview is performed by a certain interviewer at a certain time in a certain room.)
FD2. day time room $\rightarrow$ interviewer applicant
(An interview room is booked for a certain time by an interviewer, for an interview with a certain applicant.)
FD3. interviewer day $\rightarrow$ room
(An interviewer uses the same room for all interviews during one day.)

FD1 and FD2 contain all attributes so {applicant, day} and {day, time, room} are keys. (You must also check that no subsets of these sets are keys. This is easy to do.) There is one more key — by computing closures you obtain:

FD4. interviewer day time $\rightarrow$ applicant room
(At any time, an interviewer only interviews one applicant.)

The relation is not in BCNF because of FD3: {interviewer, day} is not a superkey. But the relation is in 3NF, since the right-hand side of FD3, room, is part of a key.

Conversion to BCNF: take one of the functional dependencies that violate the BCNF conditions. Here we have only one such dependency, FD3.

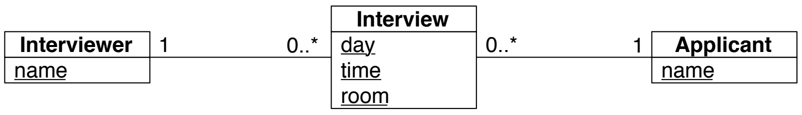
Create a new relation with the attributes that appear in FD3, and another relation with all attributes except those on the right-hand side of FD3:

R1(interviewer, day, room)
R2(interviewer, day, time, applicant)

R1 states that an interviewer books one room for all interviews during one day and should be called RoomBookings, for example. R2 states that an interviewer books times for interviews with the applicants and should be called Interviews, for example.

RoomBookings has the key {interviewer, day} (from FD3). Interviews has the keys {interviewer, day, time} (from FD4) and {applicant, day} (from FD1). Both relations are in BCNF.

First draft of an E/R diagram:

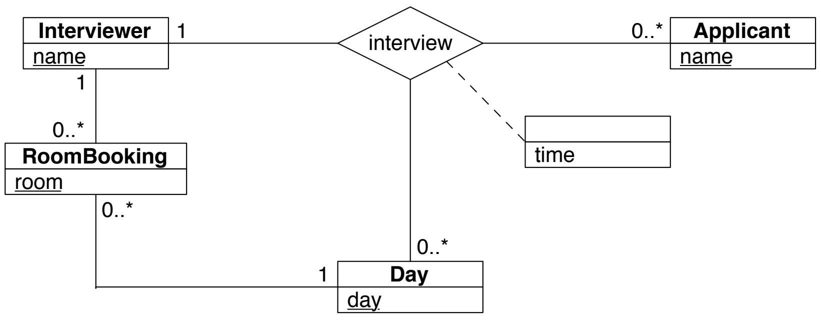


Note that the only FD that is visible in the diagram is FD2, day time room → interviewer applicant. If you, according to the rules, transform the E/R diagram into relations, you obtain the following relations:

Interviewer(name)
Applicant(name)
Interviews(interviewerName, applicantName, day, time, room)

That is, the same unnormalized relation Interviews as you started with! This shows that even a good E/R model can result in unnormalized relations, if there are conditions (in this case the functional dependencies) that are difficult to express in the E/R model.

You can, after a lot of thought, develop an E/R model where most of the functional dependencies are expressed. Days and rooms must be entity sets, and it may look like this:



From this model you get the normalized relations:

RoomBookings(interviewerName, day, room)
Interviews(interviewerName, day, time, applicantName)

The question is if it was worth the trouble. *I* don't think so: the first E/R diagram is easy to understand, the second is very detailed and hard to understand. It is better that the E/R model is easy to understand than that it is complete, especially since it is easy to normalize relations afterwards.

4. We start with an example that shows that the relation contains redundant information: Nils (person number 111) and Anna (person number 222) live in Lund and have a daughter, Eva, and a car, ABC123:

pNbr	pName	pAddress	cNbr	cName	cAddress	aLic	aMake
111	Nils	Lund	333	Eva	Lund	ABC123	Toyota
222	Anna	Lund	333	Eva	Lund	ABC123	Toyota

If Nils and Anna get a son, Johan, and also buy a new car, DEF456, the relation will look like this:

pNbr	pName	pAddress	cNbr	cName	cAddress	aLic	aMake
111	Nils	Lund	333	Eva	Lund	ABC123	Toyota
222	Anna	Lund	333	Eva	Lund	ABC123	Toyota
111	Nils	Lund	444	Johan	Lund	ABC123	Toyota
222	Anna	Lund	444	Johan	Lund	ABC123	Toyota
111	Nils	Lund	333	Eva	Lund	DEF456	Ford
222	Anna	Lund	333	Eva	Lund	DEF456	Ford
111	Nils	Lund	444	Johan	Lund	DEF456	Ford
222	Anna	Lund	444	Johan	Lund	DEF456	Ford

Here the same information is repeated several times, so the relation obviously isn't normalized. It remains to prove this. We start by looking at functional dependencies. The person number is used to identify persons, and the license number is used to identify cars, so the following functional dependencies hold:

FD1. pNbr → pName pAddress
FD2. cNbr → cName cAddress
FD3. aLic → aMake

The relation has the key {pNbr, cNbr, aLic}, so all functional dependencies violate the BCNF rules. If we decompose the relation to get BCNF relations we obtain, after several steps:

Persons(<u>pNbr</u>, pName, pAddress)
Children(<u>cNbr</u>, cName, cAddress)
Automobiles(<u>aLic</u>, aMake)
PersChildAuto(<u>*pNbr, cNbr, aLic*</u>)

The first three relations contain no redundant information, but the relation PersChildAuto does. With the same data as in the example above, the relation instance looks like this:

pNbr	cNbr	aLic
111	333	ABC123
222	333	ABC123
111	444	ABC123
222	444	ABC123
111	333	DEF456
222	333	DEF456
111	444	DEF456
222	444	DEF456

As an example of redundancy, it is stated twice that person 111 has a child 333. This must be so, since 111 owns two cars.

The relation PersChildAuto has no functional dependencies at all, so it is in BCNF. But each person has a well-defined *set* of children and a well-defined *set* of cars, and these sets are independent of each other. This is called a *multivalued dependency*.

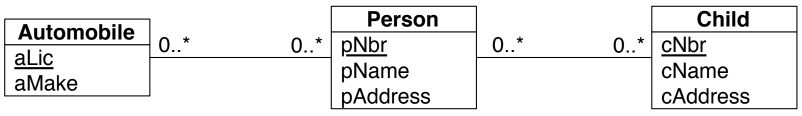
The following multivalued dependencies hold (multivalued dependencies always occur in pairs):

MVD1. pNbr $\twoheadrightarrow$ cNbr
MVD2. pNbr $\twoheadrightarrow$ aLic

PersChildAuto is not in 4NF, since pNbr is not a superkey. We can decompose the relation using one of the multivalued dependencies, in the same way as when we decompose relations into BCNF. We get:

HasChildren(*pNbr, cNbr*)
OwnsAutos(*pNbr, aLic*)

If we develop an E/R model starting from the original formulation of the problem we get, without much trouble, the following diagram:



From this, we get the same normalized relations as above.

Exercise 4 — Solutions

1. Expressions in relational algebra:

   a) $Projects$

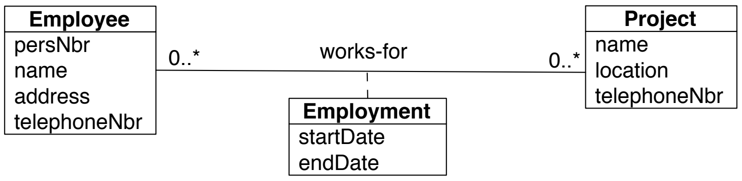
   b) $\sigma_{city='London'}(Projects)$

   c) $\pi_{supplNbr}(\sigma_{projNbr=123}(Deliveries))$

   d) $\pi_{prodNbr}(\sigma_{city='London'}(Deliveries \bowtie Suppliers))$

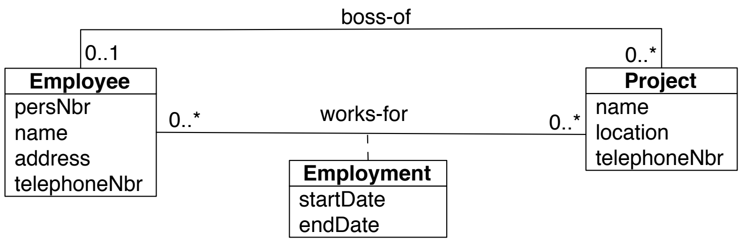
   e) $\pi_{L1.prodNbr,L2.prodNbr}$
   $(\sigma_{L1.prodNbr<L2.prodNbr}(\rho_{L1}(Deliveries) \underset{L1.supplNbr=L2.supplNbr}{\bowtie} \rho_{L2}(Deliveries))$
   ρ is the renaming operator: $\rho_{L1}(Deliveries)$ renames $Deliveries$ to $L1$.

2. We develop this solution in several steps. We start by describing that there are employees and projects, and that employees work in projects. Employments have start and end dates. We have tried to include most of the attributes:



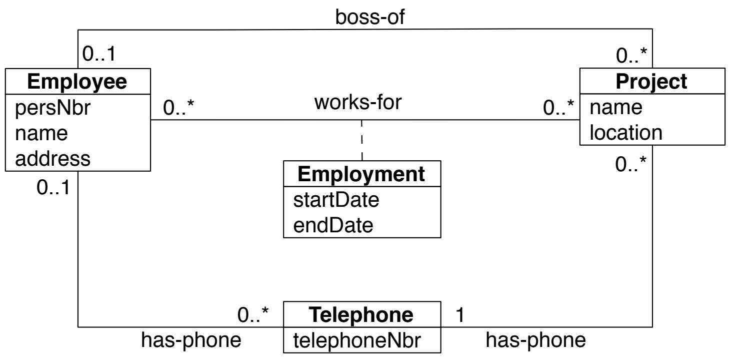
We haven't expressed that each project has a boss. This is best expressed by a relationship between Project and Employee. We don't need an entity set for the bosses — according to the text, they don't have additional properties compared to other employees.

With this relationship, the diagram looks like this:



We allow a boss to manage more than one project, and we also allow projects without a boss. The text says that each project should have a boss, but it seems reasonable that a project, at least temporarily, stands without a boss.

The entity set Employee has telephoneNbr as an attribute. All attributes must be "simple", i.e., this model specifies that an employee has one telephone number. This wasn't the case, according to the specification, and we must be able to describe that an employee has "many" telephone numbers. The only way to achieve this is to introduce a new entity set and a relationship with multiplicity "many". We have done this in the following diagram (since we now have an entity set describing telephones we also have removed the telephone number from the project — it is better to describe a thing in only one place):



By using "0..1" and "0..*" as multiplicities on the relationships from Telephone we have tried to express that a telephone is owned by an employee or a project (or several projects). It mustn't happen that the multiplicity is non-zero for both relationships at the same time. (UML has notation to express such restrictions, but we haven't learned this.)

Before translating the E/R model into relations we study the keys of the entity sets. Employee has the key persNbr and Telephone has the key telephoneNbr. We cannot form a key for Project, since several projects may have the same name. This could indicate that Project is a weak entity set and that we should fetch the key from another entity set, but that doesn't seem to be the case here — we cannot reasonably use a person number or a telephone number to identify a project. The only way out is to invent a key for projects. We call the key projNbr, but we don't show the E/R diagram again.

The entity sets are translated into the following relations:

Employees(<u>persNbr</u>, name, address)
Projects(<u>projNbr</u>, name, location)
Telephones(<u>telephoneNbr</u>)

The relationship "works-for" has multiplicity many-many, so this must become a relation:

Employments(<u>*projNbr, persNbr*</u>, startDate, endDate)

For the remaining relationships, with multiplicity 1 or 0..1, we have several options. We start by looking at "boss-of". One option is to create a relation for this relationship:

BossOf(<u>*projNbr*</u>, *persNbr*)

But since each project has exactly one boss, or maybe no boss, we can instead insert the corresponding key into the relation on the "many" side, i.e., in Projects:

Projects(<u>projNbr</u>, name, location, *bossPersNbr*)

In a project without a boss, bossPersNbr is null.

The employee telephones can be treated in the same fashion. Then, the relation Telephones will look like this:

Telephones(*persNbr*, <u>telephoneNbr</u>)

We cannot put the project telephones in this relation, and it seems unnecessary to create a separate relation for the relationship between project and telephone. Since each project only has one telephone we can insert the telephone number in the project relation:

Projects(projNbr, name, location, *bossPersNbr*, *telephoneNbr*)

To summarize, we have created the following relations:

Employees(persNbr, name, address)
Telephones(*persNbr*, telephoneNbr)
Projects(projNbr, name, location, *bossPersNbr*, *telephoneNbr*)
Employments(*projNbr, persNbr*, startDate, endDate)

The relation Telephones now only contains data concerning employee telephones, and we should change the name of the relation to, for example, EmployeeTelephones.

Are these relations normalized? Yes, the relations that only have key dependencies are, but this is not the case for all of the relations. The relation:

Projects(projNbr, name, location, *bossPersNbr*, *telephoneNbr*)

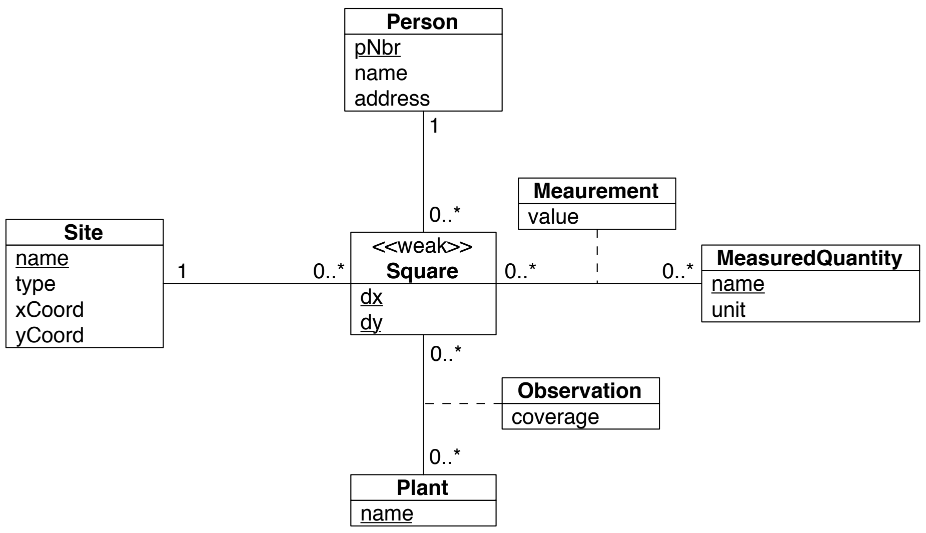
has a functional dependency in addition to the key dependency. The text says that "...has a telephone, which is shared if there is more than one project at the same location." This means that the location owns the telephone, not the project. In other words, we have the functional dependency location $\rightarrow$ telephoneNbr. Location is not a superkey, and the relation isn't in BCNF. We normalize the relation by splitting it in two:

LocationTelephones(location, telephoneNbr)
Projects(projNbr, name, *location*, *bossPersNbr*)

Resulting relations:

Employees(persNbr, name, address)
EmployeeTelephones(*persNbr*, telephoneNbr)
Projects(projNbr, name, *location*, *bossPersNbr*)
LocationTelephones(location, telephoneNbr)
Employments(*projNbr, persNbr*, startDate, endDate)

3. E/R diagram:



Square is a weak entity set, since the coordinates are relative to the coordinates of the site. Relations from the entity sets (will be modified later):

Sites(name, type, xCoord, yCoord)
Squares(siteName, dx, dy)
Plants(name)
MeasuredQuantities(name, unit)
Persons(pNbr, name, address)

Relations from the relationships:

Observations(*plantName, siteName, dx, dy*, coverage)
Measurements(*measuredName, siteName, dx, dy*, value)
InventoryResponsible(*siteName, dx, dy*, *pNbr*)

The key in Squares is too complex — it is difficult to handle when it is to be used as a foreign key in other relations. Therefore, we choose to identify a square by a unique number, squareId. We also delete the relation InventoryResponsible, and instead introduce the person number of the responsible person as a foreign key in Square. Resulting relations:

Sites(name, type, xCoord, yCoord)
Squares(squareId, siteName, dx, dy, *responsiblePNbr*)
Plants(name)
MeasuredQuantities(name, unit)
Persons(pNbr, name, address)
Observations(*plantName, squareId*, coverage)
Measurements(*measuredName, squareId*, value)

In addition to the indicated primary keys, {xCoord, yCoord} is a key in Sites, as is {siteName, dx, dy} in Squares. There are no functional dependencies beside the key dependencies, so the relations are in BCNF.

All plants that occur in squares that have a pH value less than 4:

```
select distinct plantName
from Measurements, Observations
where measuredName = 'pH' and
      value < 4 and
      Measurements.squareId = Observations.squareId;
```