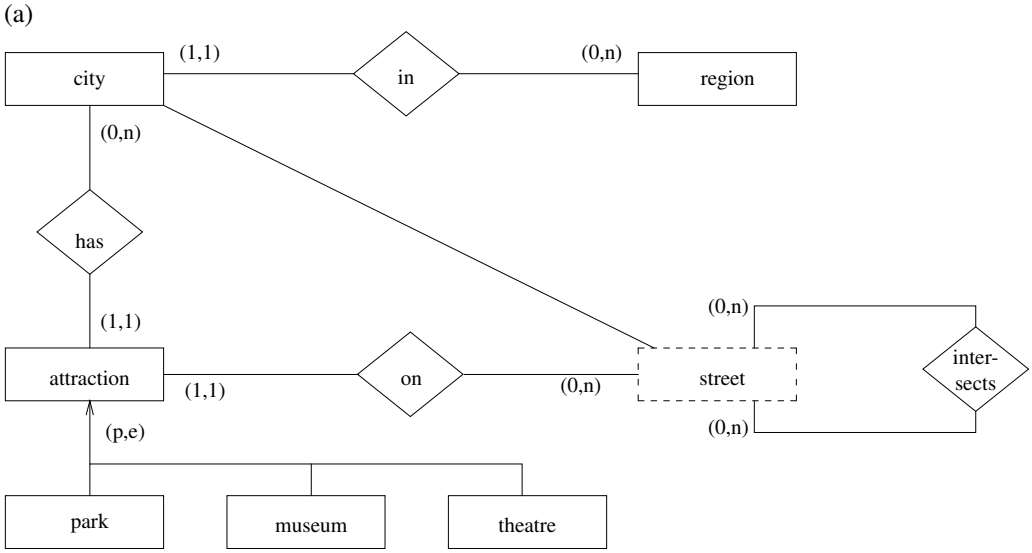
Solutions to Exercise Sheet 1

Question 1

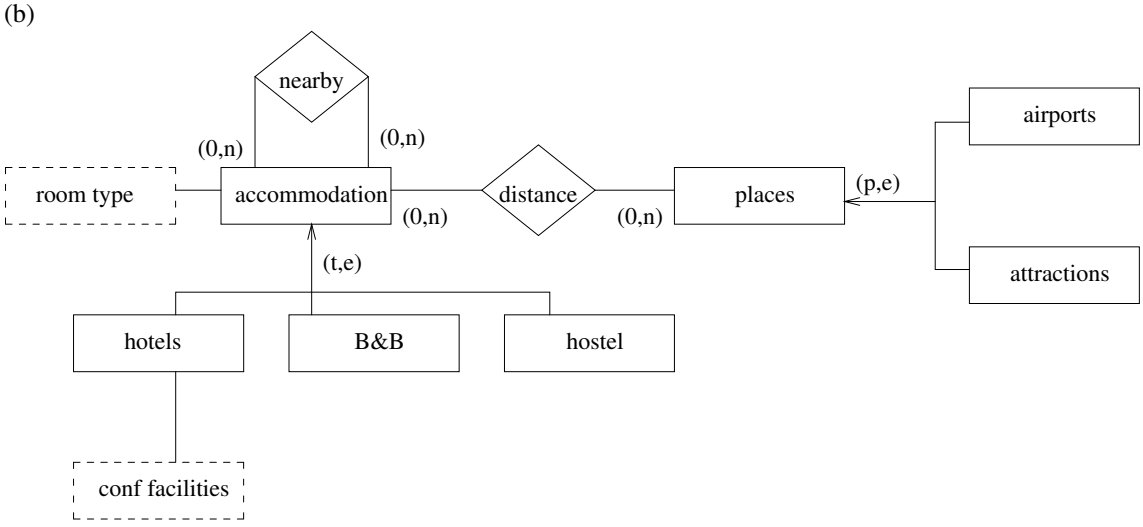
(a)



Note that "`street`" is a weak entity, depending on its containing "`city`" for its identity. This is natural because we do not expect that streets have names that are unique across the whole world (or even a whole country), but are unique within a city. (In a big city like Birmingham or London, we might need to treat each neighbourhood as a city as well.)

The three subclasses of "`attraction`" are modelled using a class hierarchy. The annotations say that the subclasses *partially* cover the whole class (because there can be other kinds of attractions, e.g., a historical or archaeological site), but the three subclasses are mutually *exclusive*.

(b)



The "`room type`" is treated as a weak entity because every accommodation decides what kinds of rooms it provides, and other attributes such as price, occupancy etc. The "`conf facilities`" are similar, except that they are specialised to hotels.

Question 2

Schemas:

a. For the tourist information system, the schemas are:

```
region(rid, name)
city(cid, name, region)
street(cid, name, main, length)
intersect(cid, street1, street2)
attraction(aid, name, cid, street)
park(aid) -- do we really need this?
museum(aid, opentime, closetime, entryfee)
theatre(aid, showtime, showprice)
```

The `in` relationship between city and region does not need to be a separate table, because a city can only be in one region. Likewise, for the relationships marked as `has` and `on` in the E-R diagram.

Note that street, being a weak entity, depends on its parent entity (city) for its identity. So its primary key consists of both the containing city and its own name. We could postulate an additional `sid` identifier for each street. This is optional. The field `main` is a boolean field to say whether the street is a main street or not.

For the class hierarchy of attractions, we are making every entity into a table. (Note that the coverage of the subclasses is partial. So `attraction` has to be a table no matter what.) The `aid` number is assigned to all attractions. Each particular kind of attraction uses the very same `aid`.

Note that parks do not seem to have any attributes other than the generic attributes common to all attractions. If so, we don't need a separate table for parks.

b. For the hotel information system, our schemas are as follows:

```
accommodation(aid, name, type)
atype(atypeid, name)
place(pid, name, type)
distance(aid, pid, dist)
nearby(aid1, aid2)
roomtype(aid, name, dailyrate, occupancy)
conference_facility(aid, name, capacity, hiringfee)
```

Here I have chosen to merge the three different kinds of accommodation into its superclass entity `accommodation`. Nothing specific needs to be stored about the subclass entities. However, we still need to be able to tell which *type* of accommodation each place is (hotel, B&B, hostel). I have chosen to store these types in a separate table `atype`, and assigned id numbers to them, which is good for efficiency. But it would also be fine to use a simple string for the "`type`" attribute.

For the room types, on the other hand, I have used the string attribute `name`, without generating additional id numbers. When we search for rooms, we will want to search based on the required `occupancy` (the number of people a room can accommodate) rather than the name that the hotel chooses to use for its rooms.

For places, again, I have combined all the subclasses into the parent entity.

CREATE TABLE statements:

a. For the tourist information system, we can use the following tables:

```
create table region(
  rid   serial          primary key,
  name  varchar(25)     not null
)
create table city(
  cid   serial          primary key,
  name  varchar(25)     not null,
  region integer        not null references region(rid) on delete no action
)
create table street(
  cid   integer         not null references city(cid) on delete cascade,
  name  varchar(15)     not null,
  main  boolean         not null,
  length integer        not null,

  primary key(cid, name)
)
create table intersect(
  cid   integer         not null references city(cid),
  street1 varchar(15)   not null,
  street2 varchar(15)   not null,

  foreign key (cid, street1) references street (cid, name) on delete cascade,
  foreign key (cid, street2) references street (cid, name) on delete cascade,
  primary key (cid, street1, street2)
)
create table attraction(
  aid   serial          primary key,
  name  varchar(25)     not null,
  cid   integer         not null references city(cid) on delete cascade,
  street varchar(15)    not null,

  foreign key (cid, street) references street (cid, name)
)
create table museum(
  aid           integer not null references attraction(aid) on delete cascade,
  opentime      time,
  closetime     time,
  entryfee      decimal(5,2) not null,

  primary key (aid)
)

create table theatre(
  aid           integer not null references attraction(aid) on delete cascade,
  showtime      time    not null,
  showprice     decimal(5,2) not null

  primary key (aid)
)
```

We have specified a street as a *weak entity*, whose identity depends on the city is contained in. I have chosen to use no new id numbers for streets, but just the name of the street along with its city. (This is for illustration purposes. It would be fine to assign id numbers to streets as well.)

Since `street` has a composite primary key, consisting of a city id and a street name, we cannot declare the foreign keys in the `intersect` table using column constraints. A special "`FOREIGN KEY`" constraint allows us to declare them as *table constraints*.

As regarding `on delete` constraints, we are conservative on references to region. (If a region is being

deleted while its cities are still in the database, somebody should pay attention!) In all other cases, we use `cascade`.

- If a city gets deleted, all its attractions should go away too.
- If an attraction gets deleted, its extra information in museum etc. should also get deleted.
- If a street gets deleted, its intersections should also get deleted.

b. For the hotel information system, the tables are as follows:

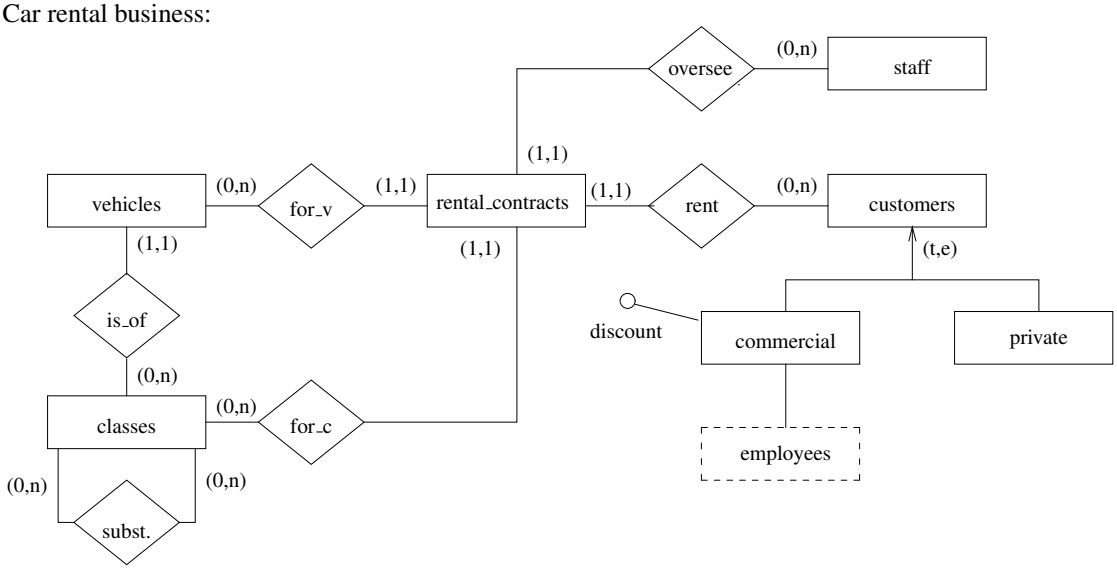
```
create table accommodation(
  aid   serial      primary key,
  name  varchar(25) not null,
  type  integer     not null references atype(atypeid) on delete no action
)
create table atype(
  atypeid serial      primary key,
  name    varchar(15) not null
)
create table place(
  pid   serial       primary key,
  name  varchar(25)  not null,
  type  varchar(10)  -- airport, attraction or null
)
create table distance(
  aid   integer     not null references accommodation(aid) on delete cascade,
  pid   integer     not null references place(pid) on delete cascade,
  dist  integer,
  primary key(aid, pid)
)
create table nearby(
  aid1  integer   not null references accommodation(aid) on delete cascade,
  aid2  integer   not null references accommodation(aid) on delete cascade,
  primary key(aid1, aid2)
)
creat table roomtype(
  aid   integer      not null references accommodation(aid) on delete cascade,
  name  varchar(10)  not null,
  dailyrate integer  not null,
  primary key (aid, name)
)
create table conference_facility(
  aid        integer  not null references accommodation(aid) on delete cascade,
  capacity   integer  not null,
  hiringfee  integer  not null
)
```

Regarding the `on delete` annotations, note that an `atype` (accommodation type) should almost never gat deleted. So, “no action” is the right thing to use. In all other cases, cascade seems right.

- If an accommodation or place gets deleted, the distance between it and something else has no more use.
- Similar consideration applies to “nearby”.
- “roomtype” and “conference_facility”, being weak entities, should get deleted if their owner strong entity gets deleted.

Questions 3

a. Car rental business:



There two tricky issues that arise in this problem.

- *How should the different types of vehicles be handled?* The first idea that occurs to mind is to treat the different kinds of vehicles as subclasses of a general `vehicle` entity. But, on a closer look, this does not seem appropriate. First of all, there is nothing different in the way the rental company deals with the different kinds of vehicles. Renting the vehicles out and processing returns are common to all the kinds, and there is not much else that needs to be done differently for the vehicle classes.

  On the other hand, the vehicle classes have attributes such as the rental rate and substitutability. Hence, we need make vehicle classes themselves as a new form of entity. Note the similarity with how we treat hotel rooms (with `roomtype` as an entity).
- We have invented a new entity called "rental_contract" even though it wasn't mentioned in the textual requirements. It is necessitated by thinking about car rentals over a period of time, and the process requirements that the customers first ask for a vehicle class (perhaps in advance) and get assigned a specific vehicle. The `for_c` and `for_v` relationships deal with a vehicle class being specified in a rental contract and a specific vehicle being assigned (which could be of a different class, depending on the circumstance).

We choose to make `employee` of commercial customers a weak entity because his/her identity can be made to depend on that of the commercial customer (which will be a company/organistaion).

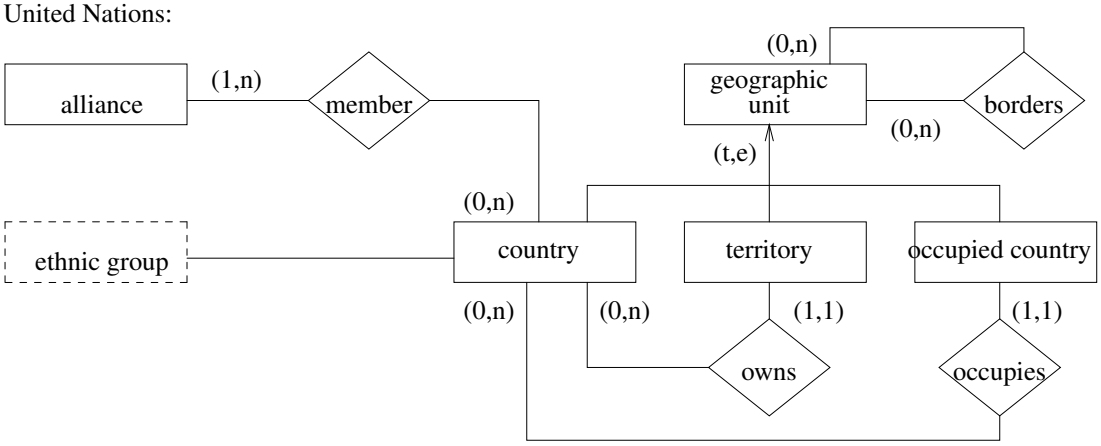
Relational table schemas: Rental contracts have many relationships where the maximum multiplicity is 1. All those relationships can be absorbed into a single table.

```
vehicle(vid, license_plate)
vehicle_class(classid, name, rental_rate)
substitutable_by(class1, class2)
staff(staffid, firstname, lastname)
customer(cid, commercial)
commercial_customer(cid, name, address, discount)
private_customer(cid, firstname, lastname, address)
employee(cid, empid, firstname, lastname)
rental_contract(contractid, cid, classid, vid°, staffid°)
```

The `vid` and `staffid` fields of `rental_contract` are allowed to be null because they are undefined until the rental event actually takes place.

An employee id (`empid`) has been added to account for the possibility that there could be multiple employees with the same name. It is expected to be specific to the employing company (the "commercial customer"). You would want this field to be a string, because we have no idea what kind of employee id's the employers would use.

b. United Nations:



Relational table schemas:

```
geographic_unit(unitid, name, continent, population, area,
                capital, independent)
borders(unitid, unitid)
ethnic_group(unitid, group, population_percentage)
alliance(allianceid, name, type)
alliance_member(unitid, allianceid)
territory(unitid, controlling_country)
occupied_country(unitid, occupying_country)
```

Note that we have added a boolean attribute `independent` to geographic units, which captures whether a unit is an independent country. We do not need a separate table for independent countries. However, territorires and occupied countries need separate tables because of the additional information required for them.