Databases - Juicd

Gordon Royle

School of Mathematics & Statistics
University of Western Australia

ERD for Juicd

We have to identify the *entities* and *relationships* and then connect them appropriately.

Some things are clearly entities, or clearly relationships, while others can be modelled as either, and which you choose depends on which makes it most natural to meet the requirements.

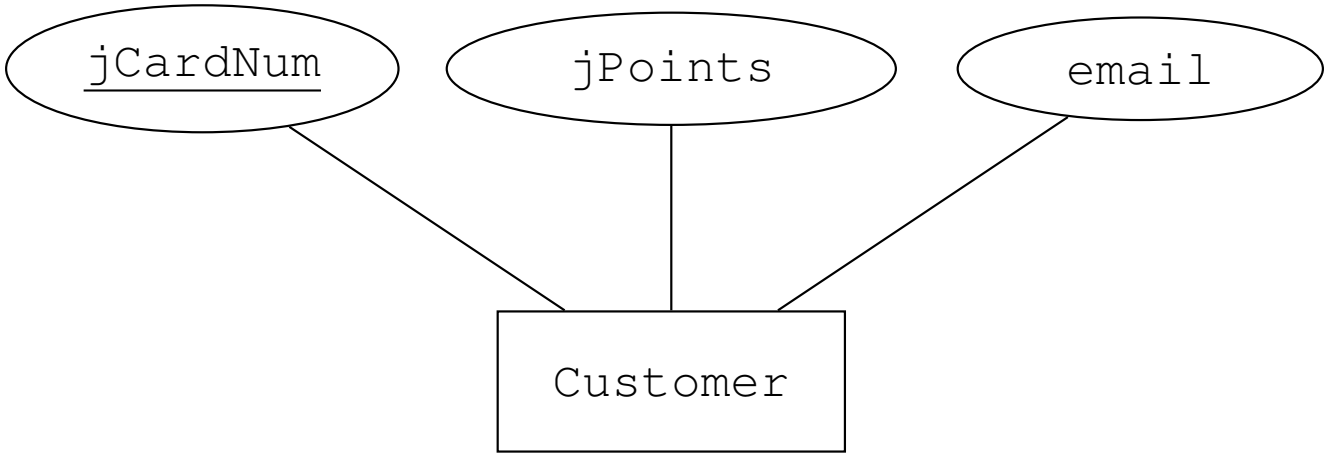
Due to the small page size, the diagrams only show the attributes the first time entity set is described, and then they are omitted.

Entities

Some things are clearly entities:

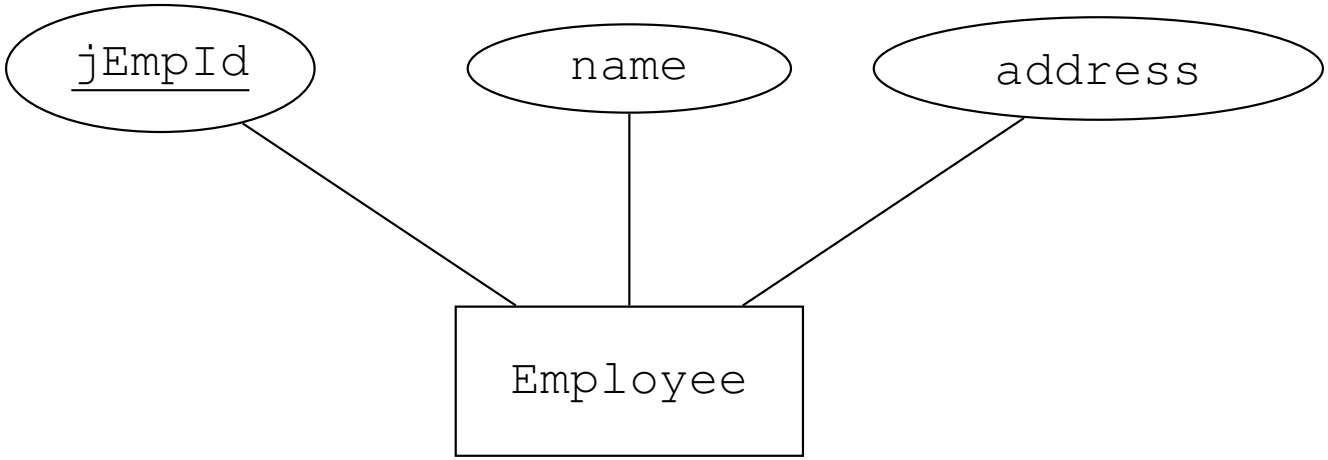
- Customer
- Employee
- Outlet
- Juice Ingredient
- Non Juice Product

Customers



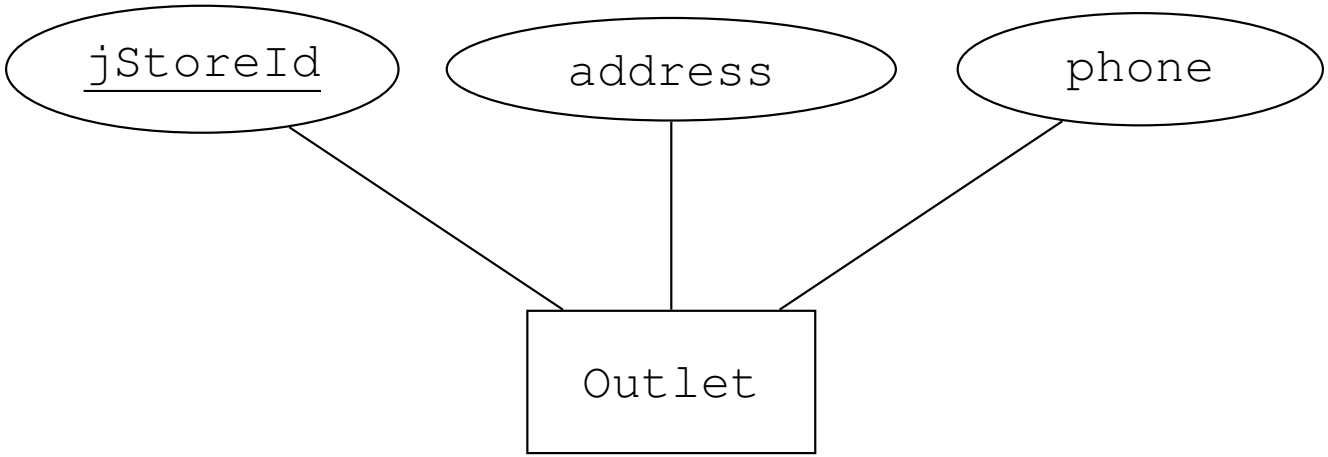
We need `jPoints` and `email` to accumulate Juicd points and send the email newsletter, and the specs said that we could assume that everyone has a Juicd card, and hence a unique number to identify them.

Employees



Employees just need something to identify them – an employee number – and probably we'd store their name and address as well.

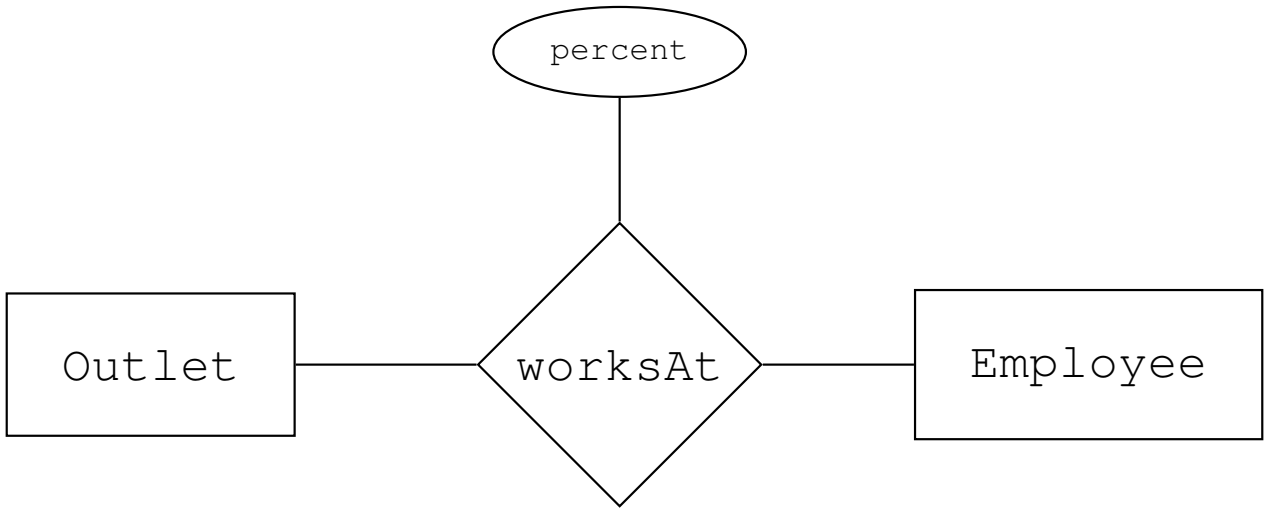
Outlets



Outlets just need any unique identifier — in this case we're just using a made up identifer.

Working

Employees work at outlets for a percentage of their time, so this can only really be represented by a *relationship*, with "percentage" as a relationship attribute.



We could add participation constraints to indicate that every employee works somewhere or that every outlet has some employees working there, but at the moment we have no information on that.

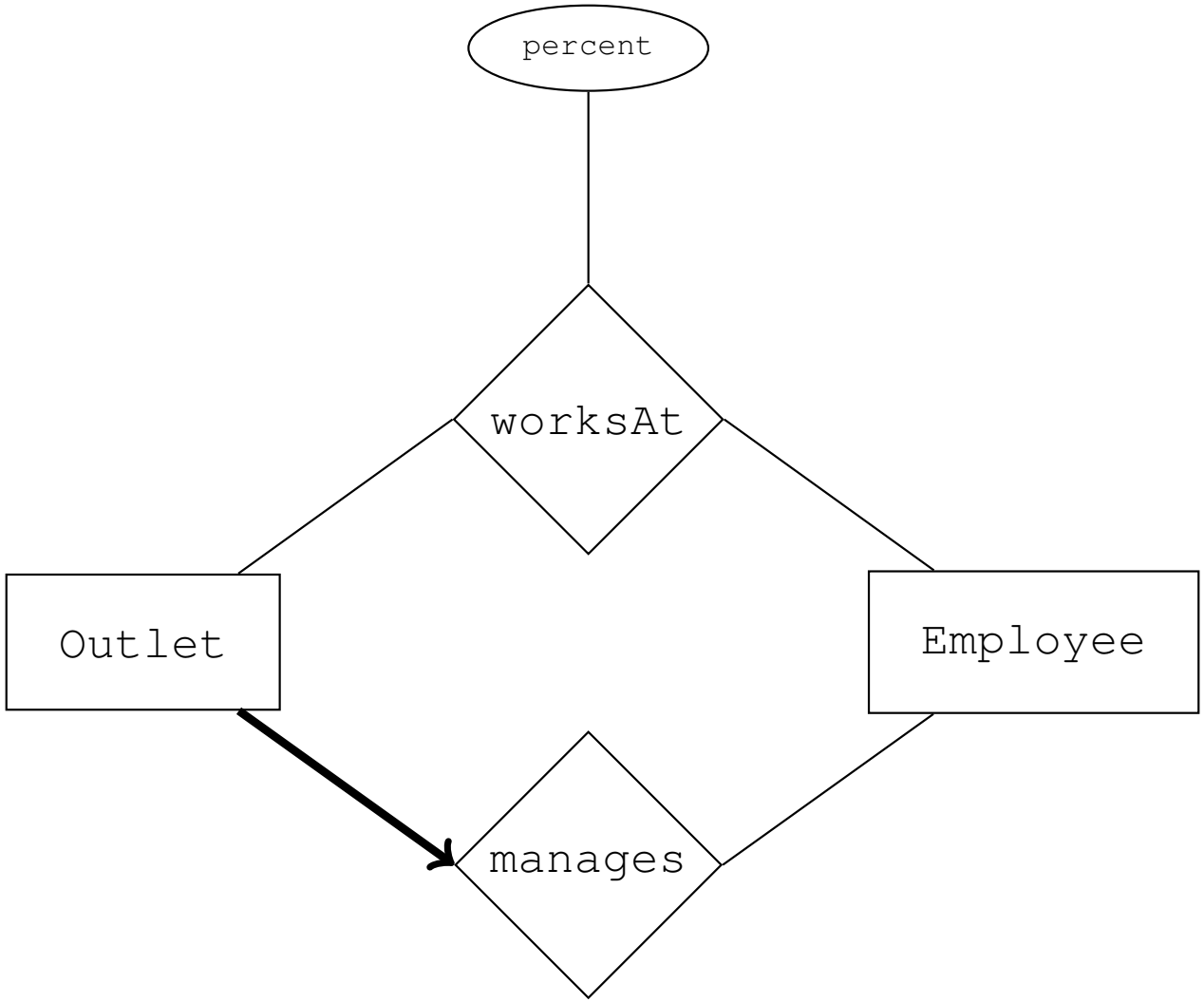
Managing

Managers can be modelled as a separate entity set, or as a relationship between employees and departments.

From the examples in R&G, it seems that the relationship set is preferred.

Here we have some constraints, namely that each outlet has *exactly one* manager, not 0, not more than 1.

Managing

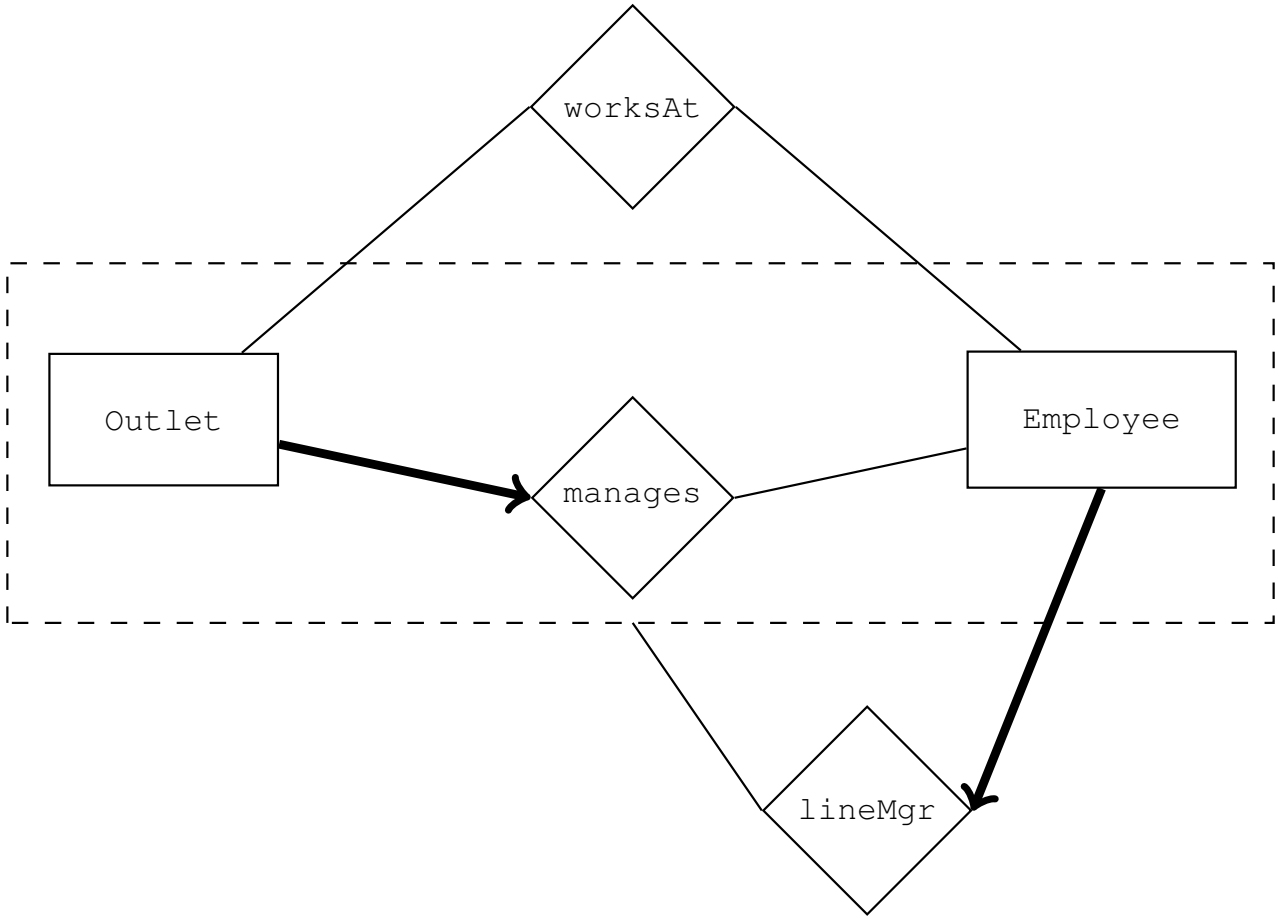


LineManager

If management is expressed as a *relationship*, then how can we insist that each employee has a unique manager?

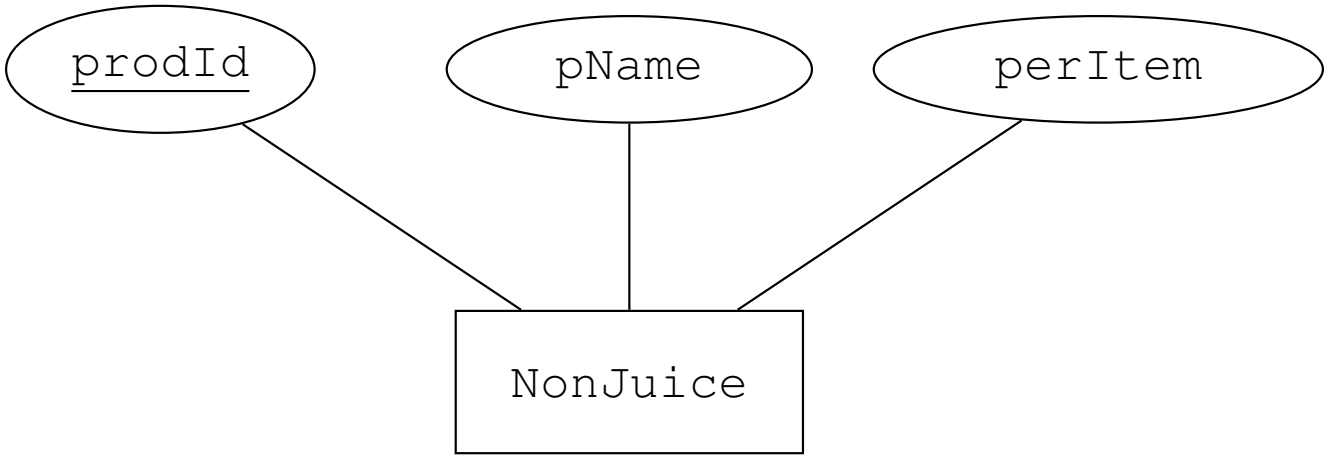
We'll use an *aggregate entity* to represent a manager, namely an *employee managing an outlet* is a manager.

This structure allows us to enforce the requirements in a natural way.


worksAt
Outlet
manages
Employee
lineMgr

Non-juices

Non-juices are an entity set.

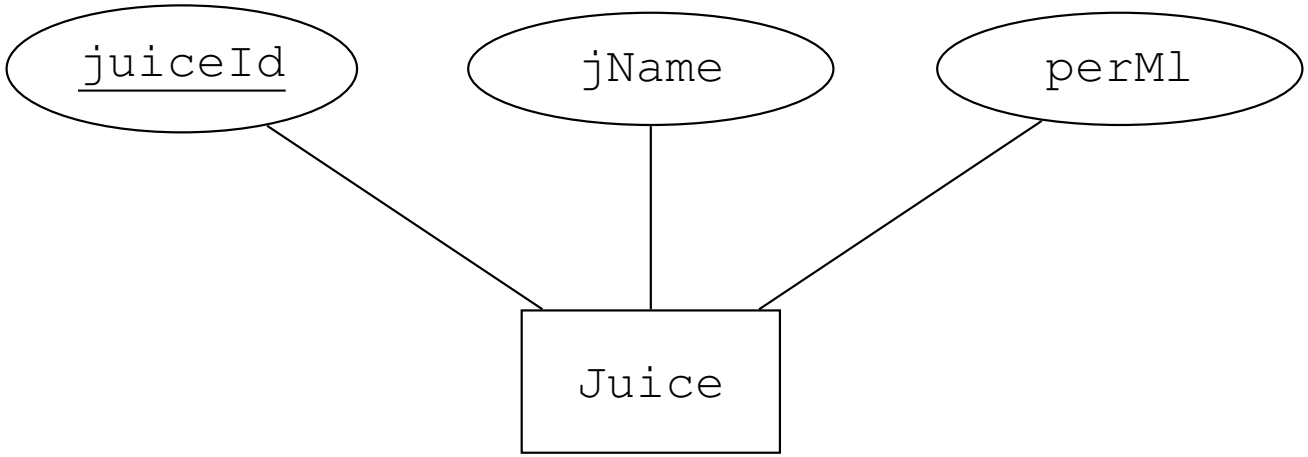


Non-juices have a product id, a name, a per-item price. A typical row is:

```
(12, "Protein Ball", 3.00)
```

Juices

Juices – or rather, the raw ingredients – form another entity set.

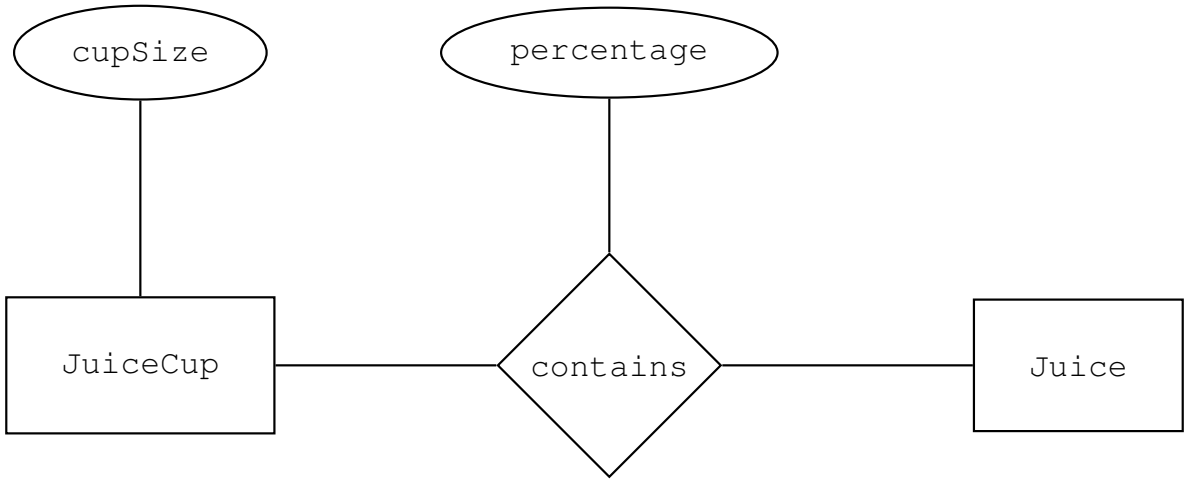


A typical row might be

```
(10, "Watermelon", 0.03)
```

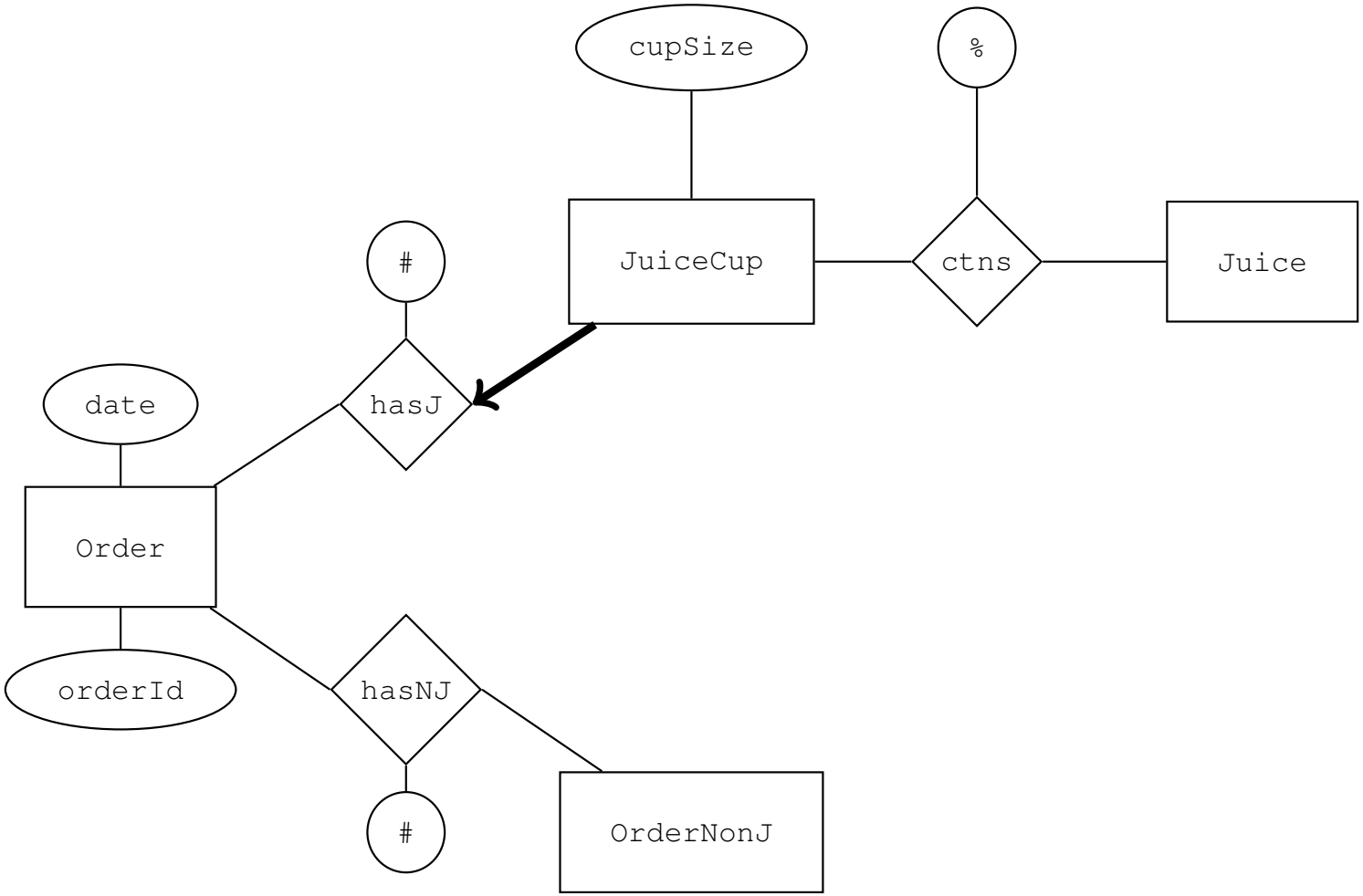
Juice Cups

A *cup of juice* consists of various juices, but also has its own attribute, namely the size of the cup.



Order

An *customer order* has non-juice items and various juice-cups



The order process

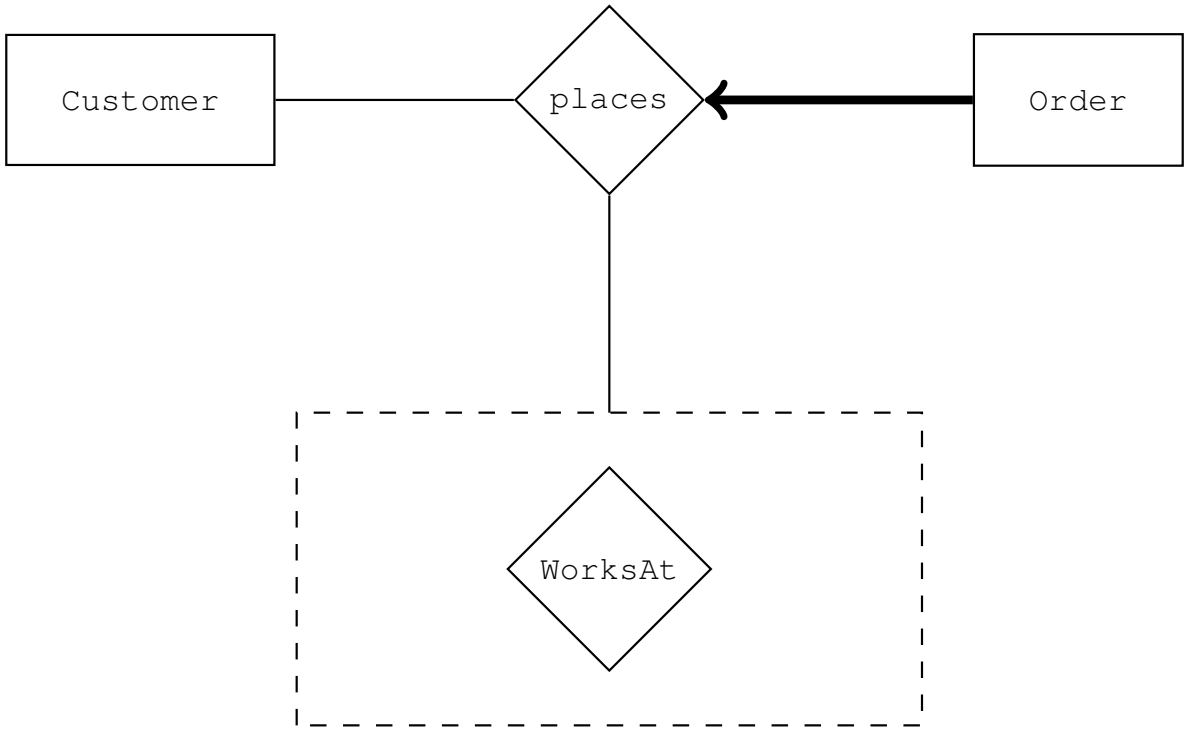
What is an order? It is when a *customer*, at an *outlet*, places an *order* with an *employee*.

So ideally all of these pieces of information should be stored in a single tuple with four components — this means that the order is really a *quaternary relation*.

However two of the four components are actually related already, because we can't just use *any employee* at *any outlet*, but rather only an employee who actually works at the outlet. So this means we really want to relate a customer, an order and an "employee-working-at-an-outlet".

The order

Luckily, this is *exactly* the purpose of an *aggregate entity* — this is dicussed in detail in Section 2.5.4 of Ramakrishnan & Gehrke.



Implementation

Implementation is when the ERD is translated to SQL; normally every entity becomes a table, and most of the relationships become tables, although relationships with key constraints and participation constraints can often be represented via attributes.

For example, the `manages` relation might be implemented by having a `mgrId` attribute in `Outlet` rather than having a separate table; this however is an implementation detail and not part of the ERD.

Integrity constraints

```
CREATE TABLE CustomerOrder(
  orderID INT PRIMARY KEY,
  jCardNum INT,
  employeeID INT,
  outletID INT,
  date DATE,
  FOREIGN KEY (jCardNum)
     REFERENCES Customer(jCardNum),
  FOREIGN KEY (employeeID, outletID)
     REFERENCES worksAt(jEmpId, jStoreID));
```

Notice that the second foreign key constraint uses a *compound key*, namely a key with 2 components, rather than a single attribute.