Note: The solutions in this document are only indicative. They may be incomplete or even contain errors.

Practicum Solutions Data Modelling

Practicum/Homework Questions

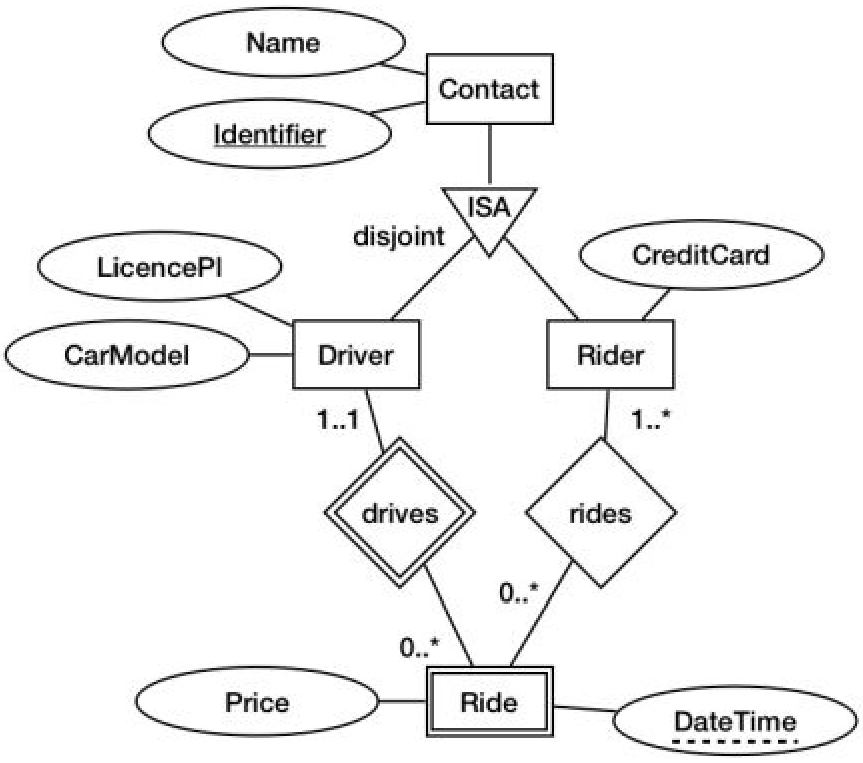
1. Provide a conceptual database model in the form of a E/R Diagram. Include Entities, Relationships, Attributes, Cardinalities, Primary Keys, Weak Entities (if present) etc. Explain the important design decisions when facing ambiguity and document relevant assumptions.
2. Give the associated relational schema, indicating relations R(A1,..) with their attributes Ai, primary keys underlined, and foreign key relationships (→). Separately also comment on NULLable attributes (if any).

Note: If you are drawing the diagrams on your computer, you may want to use a specialised editor such as yEd (https://www.yworks.com/yed-live/), or, maybe easier to insert in Google Drive, io.draw.

Practicum Scenario 1

You are asked to design the database for a ride sharing company, "HUBRIS". The business model is simple, there is an app that connects drivers with people in need of transport (riders). Riders pay for rides through the app which allows the company to charge a service fee on the paid amount. All the riders and drivers have a known name and a unique identifier **=> superclass "contact"**. For drivers, the model of car and their license plate is also known to the company. For riders, a credit card number is sometimes known **=> subclasses**. A person cannot be both driver and rider **=> disjoint (and probably total)**. For rides a date and time is known. Rides also track the **specific (=> ride-specific => weak entity)** driver and the (possibly multiple) riders that participated and the ride price when known.

Solution:

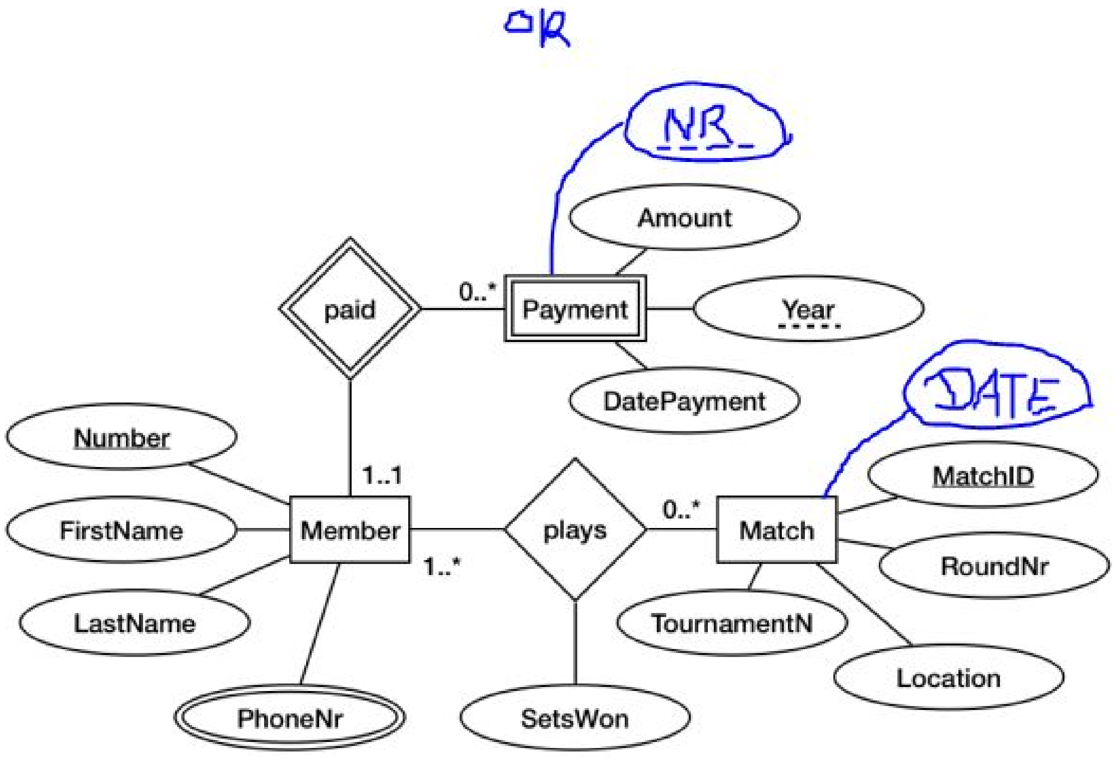


Contact(<u>Identifier</u>, Name)
Driver(<u>Identifier→Contact</u>, LicencePl, CarModel)
Rider(<u>Identifier→Contact</u>, CreditCard)
Ride(<u>Identifier→Driver</u>, <u>DateTime</u>, Price)
Rides(<u>(Identifier, DateTime)→Ride</u>, <u>Identifier→Rider</u>)

Practicum Scenario 2

You are asked to develop a database for a tennis club "SLAMMERS". The membership list contains basic information about members: member number (assigned by the club administrator), first name, last name, possibly several telephone number(s) **=> multi-valued attribute**. Substantial fees are due every calendar year **=> unclear whether installments allowed or once per year**. For fee payments, the amount, the paying member **=> weak**, the membership year and the bank transfer date are tracked. For tennis matches played at the club the database needs to record the **<u>players</u>** involved, the date and the result per **<u>player/match combination</u>** **=> match not weak on players because 1…***. Matches are **always played in the context of some club tournament** **=> strong reason for weak match depending on tournament** and thus have a tournament name and round number. Sometimes a location for matches is also known. **=> Nullable statement**

Solution:

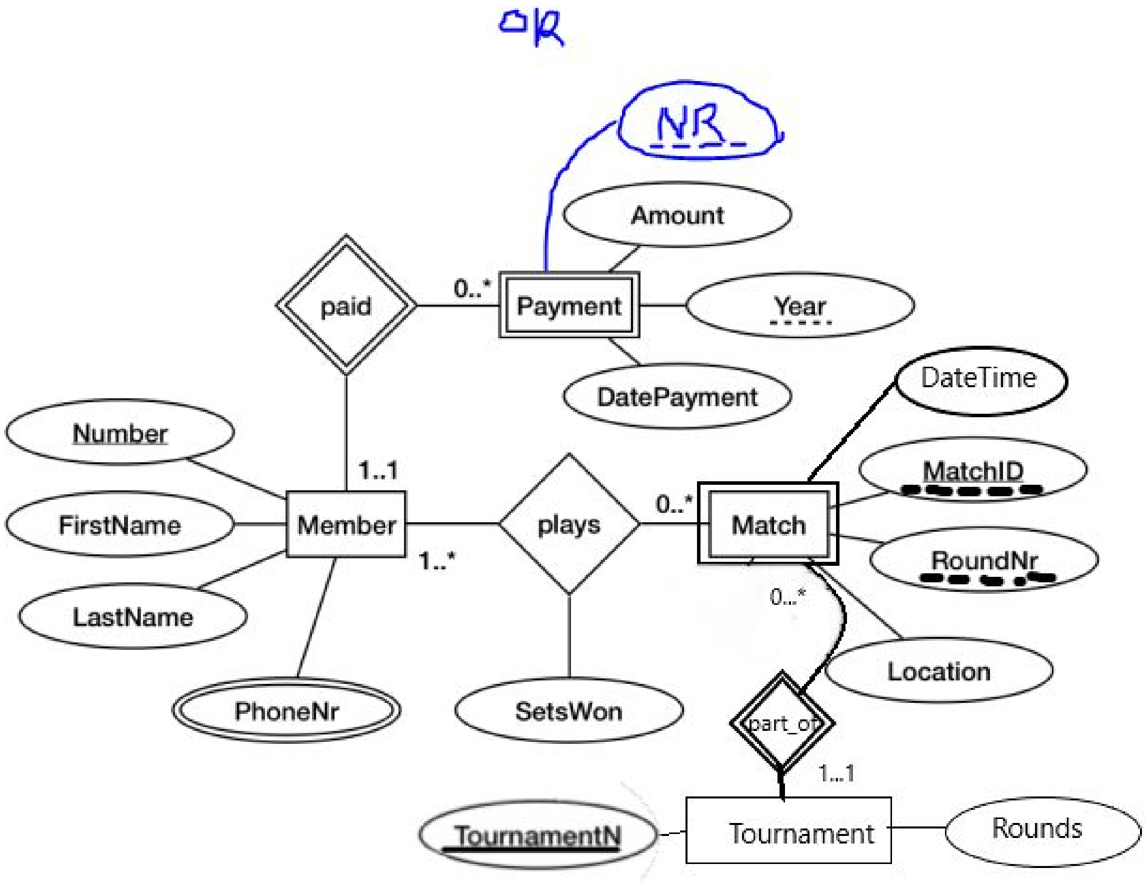


Note: the text allows for the interpretation that yearly membership fees are paid at once for that year. It could also, however, be argued that paying in installments is possible. In this case, a payment_number attribute in combination with year as part of the key is a good solution.

Member(<u>Number</u>, FirstName, LastName)
Phone(<u>Number→Member</u>, <u>PhoneNr</u>)
Payment(<u>Number→Member</u>, *<u>payment_nr</u>*, <u>Year</u>, amount, DatePayment)
Match(<u>MatchID</u>, RoundNr, TournamentN, Date, Location) *=>Location could be composite*
Plays(<u>Number->Member</u>, <u>MatchID→Match</u>, SetsWon)

Location is NULLABLE. It “is sometimes” known.

Alternative solution with tournament entity:



Note that one might also add a tournament date or date range. One could also add a relationship between member and tournament if we want to explicitly store which members will participate in a tournament beforehand.

Member(Number, FirstName, LastName)
Phone(Number→Member, PhoneNr)
Payment(Number→Member, *payment_nr*, Year, amount, DatePayment)
Match(TournamentName -> Tournament, MatchID, RoundNr, DateTime, Location)
=>Location could still be composite
Plays(Number->Member, (TournamentName, MatchID, RoundNr)→Match, SetsWon)

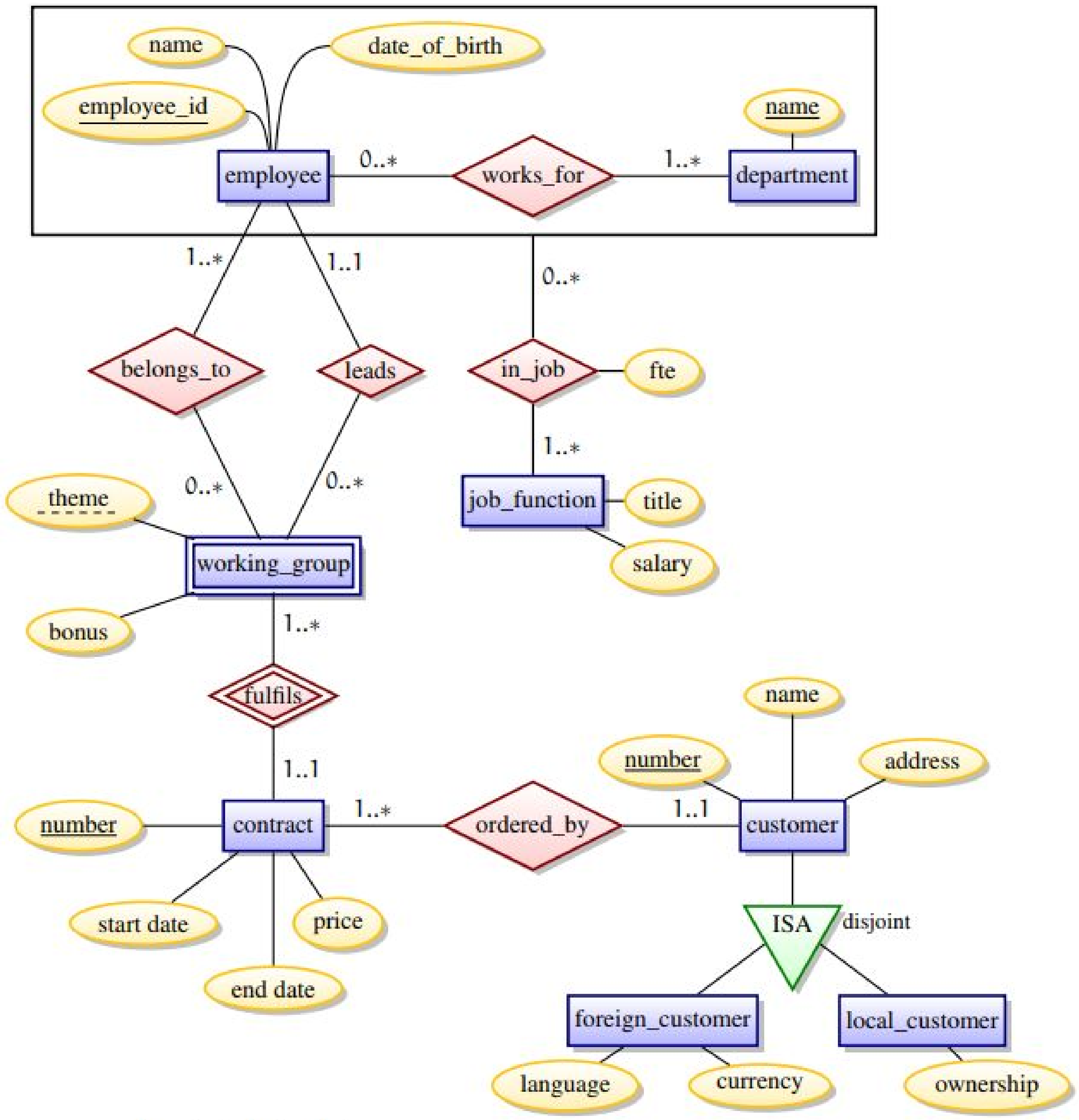
Practicum Scenario 3

You are asked to develop a database for small company. The company wants to have a database keeping track of their employees and contracts. Each employee has a unique employee identification number, and we have to store the name and date of birth. The company has different departments which are identified by their name. Every employee works for at least one department, but due to restructuring a department does not necessarily have any employees. If an employee works for a department, he/she has one or more job titles (software engineer, manager, accountant, cleaner, etc.) **associated with this work (that is, the job titles can depend on employee and department)**. This association should also carry the information on the fte (that the employee works on a particular job title for a department). The salary depends on the job title.

The company has contracts which have a unique contract number, and every contract has a start date, an end date and the price that the customer pays on fulfilment of the contract. We assign each customer a unique customer number, and store the name and address. We also store what contract was ordered by what customer. For the customers, we distinguish foreign customers and local customers. For the local customers it is important to know whether the customer belongs to the public or private sector since different taxation rates may apply. For foreign customers we store information on the language of communication and currency in which payments are made.

For every contract, we create one or more working groups. Each working group is assigned to precisely one contract. The working groups have a theme which is unique among the working groups for a particular contract. Every working group has precisely one leader assigned to it, and consists of one or more employees. Every employee can lead and can be part of an arbitrary number of working groups. Each working group is assigned a bonus (percentage of the contract payment) which will be paid on fulfilment of the contract.

Solution:



- Every entity should have a key, every weak entity should have a discriminator.
- The working groups are created especially for a contract and have no key themselves; hence they are a weak entity that with a identifying relationship to the contract.
- An employee can belong to and/or lead multiple working groups. Every working group has at least one leader. The easiest way to model this is two relationship sets between working group and employee.

- The job titles are associated to a particular ‘works for‘ relationship between employee and department. In particular, and employee can work for different departments on different job titles. We of course want that there is a job title associated only if there exists a ‘works for‘ relationship, therefore the best way to model this is aggregation; see also the manages relationship on the slides.
- **Note that when making design decisions:**
    - Aggregation alternative: Fte could have also been an attribute of the job, if in_job (the relation) was a weak entity depending on the three entity sets employee, department, and job_function. A weak entity set does not need to depend on one entity set - multiple is possible. Aggregation practically lifts works_for to become an abstract entity set in translation.
    - Contract could have been weak - but it shouldn’t be - identified by the customer IF the text didn’t clearly provide a unique key for contract. An entity is weak only if it depends on another entity and itself does not have a key.

Translation to Relational Model for Scenario 3
Basic Translation

Here, we only eliminate tables of identifying relationships of weak entities.

employee (employee_id, name, date_of_birth)

department (name)

contract (number, start_date, end_date, price)

working_group (number -> contract, theme, bonus)

customer (number, name, address)

foreign_customer (number -> customer, language, currency)

local_customer (number -> customer, ownership)

job_function (title, salary)

works_for (employee_id -> employee, name -> department)

belong_to (employee_id -> employee, (number, theme) -> working_group)

leads (employee_id -> employee, (number, theme) -> working_group)

ordered_by (contract_number -> contract, customer_number -> customer)

in_job ((employee_id,name) -> works_for, title -> job_function, fte)

Nullable and uniqueness constraints:

- Basically all attributes are NOT NULLABLE.
- The following attributes must be declared unique:
    - number of foreign_customer

  - number of local_customer
    since we do not want several foreign/local customers being the same customer.

Moreover, we can model the cardinality constraints ‘at most one’ using the following uniqueness constraints:
– contract_number of ordered_by
– (number, theme) of leads

Optimized translation

The optimised translation has two advantages:

1. we eliminate unnecessary tables
2. we can model ‘precisely 1’ constraints

We eliminate the following tables:

- **leads** by extending **working_group** with the key of **employee**
  we use the constraint NOT NULLABLE to model that every working group has 1..1 leaders
- **ordered_by** by extending **contract** with the key of **customer**
  we use the constraint NOT NULLABLE to model that every contract has 1..1 customers

employee (employee_id, name, date_of_birth)
department (name)
contract (number, start_date, end_date, price, ordered_by -> customer)
working_group (number -> contract, theme, bonus, leader -> employee)
customer (number, name, address)
foreign_customer (number -> customer, language, currency)
local_customer (number -> customer, ownership)
job_function (title, salary)

works_for (employee_id -> employee, name -> department)
belong_to (employee_id -> employee, (number, theme) -> working_group)
in_job ((employee_id,name) -> works_for, title -> job_function, fte)