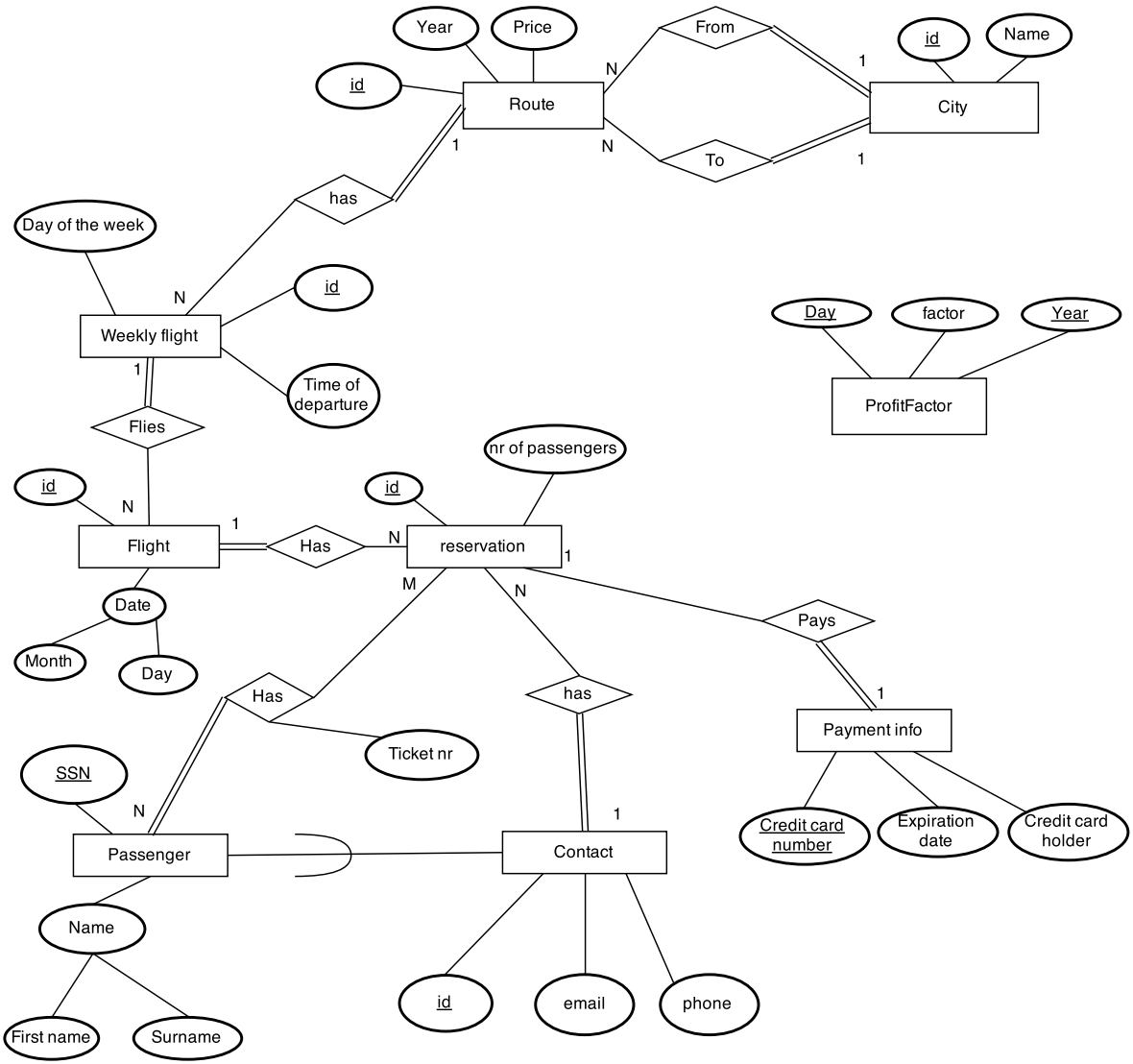
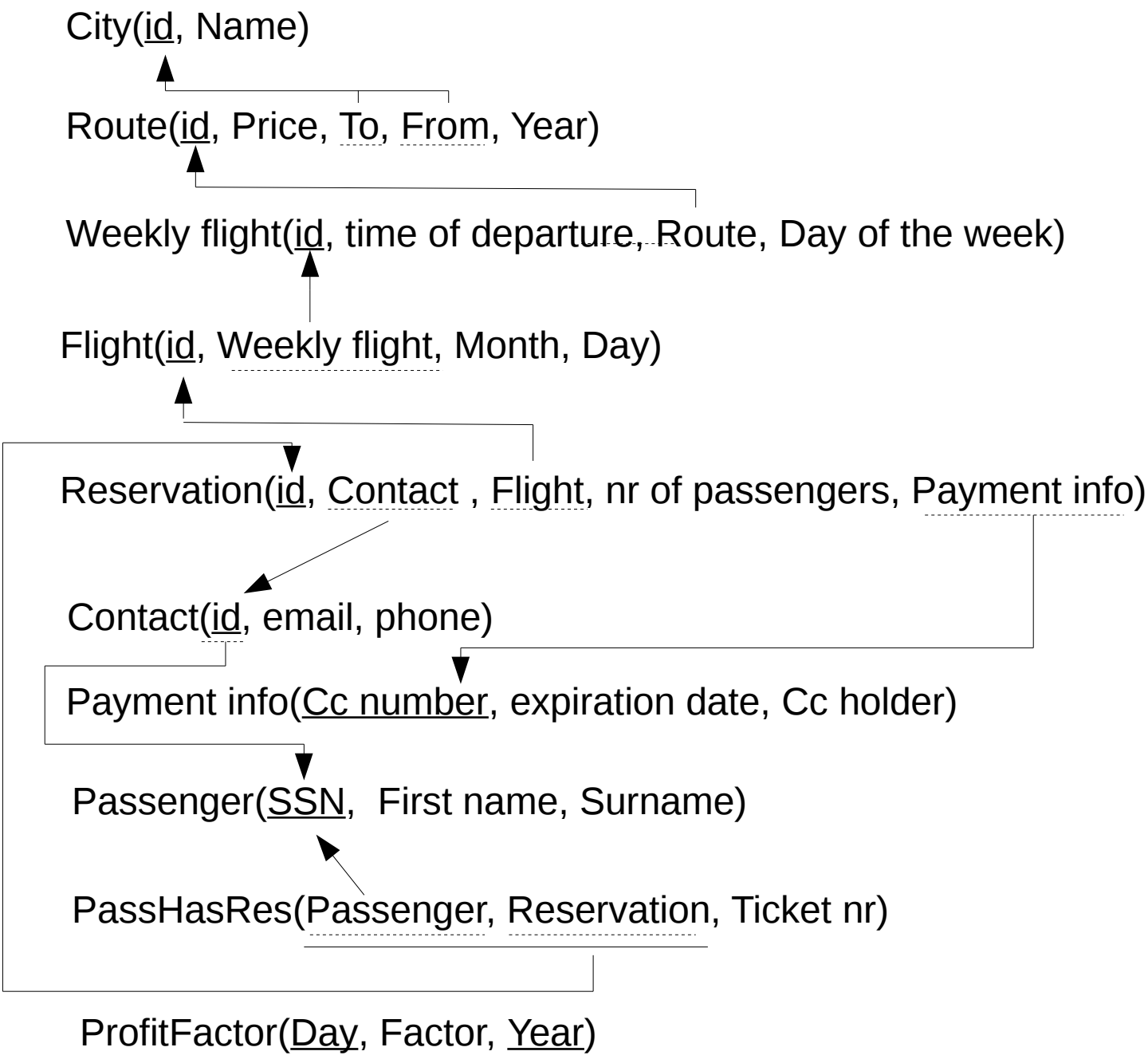

Year
Price
From
id
Name
id
Route
N
1
City
1
N
To
1
has
Day of the week
N
id
Weekly flight
Time of departure
1
Flies
Day
factor
Year
ProfitFactor
nr of passengers
id
id
N
Flight
1
Has
N
reservation
1
M
N
Date
Month
Day
Pays
1
Has
has
Payment info
Ticket nr
SSN
N
1
Credit card number
Expiration date
Credit card holder
Passenger
Contact
Name
id
email
phone
First name
Surname

Before normalisation



The purpose of every table:

City:
The city table holds a name for a city and a unique id because there's a small risk that there will be two different cities sharing the same name. We use this table so we can avoid duplicating the city name in the route table.

Candidate keys: id
Primary key: id

Route:
This table contains the different routes that flights uses, used to avoid duplicating data when different flights use the same route. Stores info about arrival and destination city aswell as the cost for the specific route. The table also has a column that tells which year the route is used.

Candidate keys: id, (to, from, year)
Primary key: id

WeeklyFlight:
Stores a connection to a route, a weekday and a time of departure. Its purpose is to contain info for all the flights that might occur every week. The table also refers to the profitfactor table.

Candidate keys: id, (route, day of week, time of departure)
Primary key: id

ProfitFactor:
This table only has 3 columns. Day, factor and year are the values stored for each row. The purpose for this table is to avoid duplicate data and it also makes it possible to have different factors for different years.

Candidate keys: (Day, Year)
Primary key: (Day, Year)

Flights:
This table references a weekly flight and will be referenced to from other tables. The table also contains a month and a day.

Candidate keys: id, (weekly flight, month, day)
Primary key: id

Reservation:
Stores information about reservations that has been made. We also store information to a contact. Also contains number of passengers and a pointer to the payment info.

Candidate keys: id
Primary key: id

PaymentInfo:
Stores information about the credit card that paid for a booking.

Candidate keys: CC_number
Primary key: CC_number

Passenger:
Stores information about all passengers, each passenger will only be present once.

Candidate keys: SSN
Primary key: SSN

PassHasRes:
Stores the relation between reservations and passengers, it's needed since each passenger can have many reservations and a reservation can have many passengers. If a reservation has been paid for it will also contain the ticket number.

Candidate keys: (Passenger, Reservation)
Primary key: (Passenger, Reservation)

Contact:
Stores contact information.

Candidate keys: id
Primary key: id

We went through the normalization process and couldn't find any changes that was needed.

Lab 4 Project Answers
Marcus Sneitz(marsn336)
Erik Sneitz(erisn687)

4c)

To protect the data we could encrypt the information so it's not stored in plain text.

4e)

- It's more secure to keep all the logic server-side.
- It's possible for different types of applications to interact with the database.
- All logic is at one place and it's easy to maintain without having to think about what framework and language the front-end uses.
- It's easier to scale in size.

9)

No, the reservation isn't visible in the other session because it's not commited to disc.

We can’t modify the reservation since we can't see it at all.

To solve this issue we can manually COMMIT the changes directly after a procedure is done, this way we will have the data on disc. If a procedure fails we can do a ROLLBACK which reverts all changes the procedure did.

10a)

If we call the create_reservation procedure at the same time from two different sessions and we check the available seats we will get an overbooking since the sessions aren't aware of each others actions. We don't know that there are in fact no more seats left since the changes arent yet commited to disc and the session can interact with the same table at the same time.

10c)

To avoid these types of problem we can LOCK all the tables that we need in a procedure before we call it and then release them afterwards, this way we don’t allow procedures to read from a table that currently is write locked. This means that one of the sessions has to wait for the other before it can proceed but it won’t be possible to create overbookings.