Claire Getz, Jacob Lasalle, and Jiazheng Huang
CSC460 Project 4
December 6, 2021

i) A final ER diagram is shown on the last page.

ii) Logical Database Design

After converting out ER diagram to tables and doing the normalization, we have the following schema:

> **Appointment**: {apptID, apptTime, apptSuccessful, apptType, customerID, employeeID},
> **Customer** {customerID, firstName, lastName, address, dateOfBirth},
> **Employee** {employeeID, firstName, lastName, jobID, deptID},
> **Job** {jobID, title, salary},
> **Department**{deptID, name, fee},
> **StateID**{stateIDNo, issueDate, expireDate, customerID, deptID},
> **Permit**{permitNo, issueDate, expireDate, class, customerID, deptID},
> **License**{stateIDNo, issueDate, expireDate, class, customerID, deptID},
> **Registration**{stateIDNo, issueDate, expireDate, customerID, deptID},

where customerID, employeeID are foreign keys in Appointment, deptID and jobID are foreign keys in Employee, and customerID and deptID are foreign keys in StateID, Permit, License and Registration.

iii) Normalization Analysis

Appointment (from draft): {apptID, apptTime, apptSuccessful, apptType, fee, deptID, customerID, employeeID}

Functional dependencies:

> {apptID} —> {apptTime, apptSuccessful, apptType, fee, deptID, customerID, employeeID}
> {deptID} —> {fee}
> {employeeID, fee} —> {deptID}

ApptID is the primary key, so apptID functionally determines everything else. If two relations agree on their deptID's they must also agree on their fees, because fees belong to the department and departments only have a single service that costs a fixed amount. Also, because employees only belong to a single department, and the four fees for the departments happen to be unique, employeeID and fee both functionally determine departmentID. That fee functionally determines deptID is somewhat coincidental, because if one service changed their fee to be the same as another, the FD wouldn't be true anymore.

Closures:

> {apptID}+ = {apptID, apptTime, apptSuccessful, apptType, fee, deptID, customerID, employeeID}
> {apptTime}+ = {apptTime}
> {apptSuccessful}+ = {apptSuccessful}
> {apptType}+ = {apptType}
> {fee}+ = {fee, deptID}

{deptID}+ = {deptID, fee}
{customerID}+ = {customerID}
{employeeID}+ ={employeeID, deptID, fee}

After closure I found that emplyoeeID also functionally determines fee.

Data Normalization
None of the attributes are set-valued so this is in 1NF.
The non-prime attributes here are apptTime, apptSuccessful, apptType, fee, deptID, customerID, and employeeID, of those deptID and fee are only partially dependent on apptID, deptID should be removed and fee should go into the department table, that information can be found using employeeID:

Appointment: {apptID, apptTime, apptSuccessful, apptType, customerID, employeeID}
{apptID}+ = {apptID, apptTime, apptSuccessful, apptType, fee, deptID, customerID, employeeID}
{apptTime}+ = {apptTime}
{apptSuccessful}+ = {apptSuccessful}
{apptType}+ = {apptType}
{customerID}+ = {customerID}
{employeeID}+ ={employeeID}

Now all the FDs aside from apptID are trivial or the superkey apptID, so this is in BCNF.

Customer {customerID, firstName, lastName, address, dateOfBirth}
Functional dependencies:
{customerID} —> {firstName, lastName, address, dateOfBirth}

CustomerID is the primary key, so it functionally determines everything else. Here, there are no other functional dependencies because people can have the same first but not last name, and vice versa, multiple different people may live at the same address, and not everyone in the database named ‘John’ would live at the same address. If you had George Foreman in your database you could even have people with the same first and last names and address in the database but not the same DOB.

Closures:
{customerID}+ = {customerID, firstName, lastName, address, dateOfBirth}
{firstName}+ = {firstName}
{lastName}+ = {lastName}
{address}+ = {address}
{dateOfBirth}+ = {dateOfBirth}

Data Normalization
None of the attributes are set-valued so this is in 1NF. The only non-trivial attribute in Customer is the primary key, which is a superkey. That means this is in BCNF and therefore in 2NF as well.

Employee {employeeID, firstName, lastName, jobID, deptID}
Functional dependencies:

{employeeID} —> { firstName, lastName, jobID, deptID}

EmployeeID is the primary key, so it functionally determines everything else. Here, there are no other functional dependencies because people can have the same first but not last name, and vice versa, multiple different people may have the same job or work in the same department, and not everyone in the database named 'John' would do the same job or be in the same department. If you had George Foreman in your database you could even have people with the same first and last names and jobID in the database but not the same department, or vice versa.

Closures:

{employeeID}+ = {firstName, lastName, jobID, deptID }
{firstName}+ = {firstName}
{lastName}+ = {lastName}
{jobID }+ = {jobID}
{deptID }+ = {deptID}

Data Normalization

None of the attributes are set-valued so this is in 1NF. The only non-trivial attribute in Employee is the primary key, which is a superkey. That means this is in BCNF and therefore in 2NF as well.

Job {jobID, title, salary}

Functional dependencies:

{jobID} —> { title, salary}

JobID is the primary key, so it functionally determines everything else. Here, there are no other functional dependencies because two jobs can have the same salary. As the rubric was laid out, it seemed that a job title always had the same salary, but a situation to have this not be true could be one where raises are given out but adding new salaries to the same title (desk clerk at 40,000 might be starting salary but perhaps it could go up to 48,000).

Closures:

{jobID}+ = {jobID, title, salary }
{title}+ = {title}
{lastName}+ = {lastName}
{jobID }+ = {jobID}
{deptID }+ = {deptID}

Data Normalization

None of the attributes are set-valued so this is in 1NF. The only non-trivial attribute in Employee is the primary key, which is a superkey. That means this is in BCNF and therefore in 2NF as well.

Department{deptID, name, fee}

Functional dependencies:

{deptID} → { name, fee }

{name} → {fee}

DeptID is the primary key, so it functionally determines everything else. The same department name would also not be used twice in the DMV, so name is a candidate key in this relation.

Closures:

{deptID, name}+ = { deptID, name, fee }

Data Normalization

None of the attributes are set-valued so this is in 1NF. The two non-trivial attributes in Department are both superkeys because both deptID and name are candidate keys. That means this is in BCNF and therefore in 2NF as well.

StateID{stateIDNo, issueDate, expireDate, customerID, deptID}

Functional dependencies:

{stateIDNo} → { issueDate, expireDate, customerID, deptID }

{customerID} → { stateIDNo, issueDate, expireDate, deptID }

stateIDNo is the primary key, so it functionally determines everything else. Since each one will only have one StateID, so customerID functional determines the stateIDNo and it's issueDate and expireDate.

Closures:

{ stateIDNo }+ = { issueDate, expireDate, customerID, deptID}

{ customerID }+ = { stateIDNo, issueDate, expireDate, deptID }

{ issueDate }+ = { issueDate }

{ expireDate }+ = { expireDate }

{deptID }+ = {deptID}

Data Normalization

None of the attributes are set-valued so this is in 1NF. The only non-trivial attribute in StateID is the primary key, which is a superkey. That means this is in BCNF and therefore in 2NF as well.

Permit{permitNo, issueDate, expireDate, class, customerID, deptID}

Functional dependencies:

{ permitNo } → { issueDate, expireDate, class, customerID, deptID }

{customerID} → { permitNo, issueDate, expireDate, class, deptID }

permitNo is the primary key, so it functionally determines everything else. Since each one will only have one permitNo, so customerID functional determines the permitNo, class and it's issueDate and expireDate.

Closures:

{ permitNo }+ = { issueDate, expireDate, class, customerID, deptID}

{ customerID }+ = { permitNo, issueDate, expireDate, class, deptID }

{ issueDate }+ = { issueDate }

{ expireDate }+ = { expireDate }
{class}+ = {class}
{deptID }+ = {deptID}

Data Normalization
None of the attributes are set-valued so this is in 1NF. The only non-trivial attribute in permitNo is the primary key, which is a superkey. That means this is in BCNF and therefore in 2NF as well.

License{licenseNo, issueDate, expireDate, class, customerID, deptID}
Functional dependencies:
{ licenseNo } → { issueDate, expireDate, class, customerID, deptID }
{customerID} → { licenseNo, issueDate, expireDate, class, deptID }
licenseNo is the primary key, so it functionally determines everything else. Since each one will only have one licenseNo, so customerID functional determines the licenseNo, class and it's issueDate and expireDate.

Closures:
{ licenseNo }+ = { issueDate, expireDate, class, customerID, deptID}
{ customerID }+ = { licenseNo, issueDate, expireDate, class, deptID }
{ issueDate }+ = { issueDate }
{ expireDate }+ = { expireDate }
{class}+ = {class}
{deptID }+ = {deptID}

Data Normalization
None of the attributes are set-valued so this is in 1NF. The only non-trivial attribute in licenseNo is the primary key, which is a superkey. That means this is in BCNF and therefore in 2NF as well.

Registration{licensePlateNo, issueDate, expireDate, VIN, customerID, deptID}
Functional dependencies:
{ licensePlateNo } → { issueDate, expireDate, VIN, customerID, deptID }
{ VIN } → { licensePlateNo, issueDate, expireDate, customerID, deptID }
licensePlateNo is the primary key, so it functionally determines everything else. Since every registration will only have one VIN number, so VIN functional determines everything else. In here, since one person might have multiple registration, so customerID does not functional determines any of them. For example, one person have three cars and one boat, there will be 4 registration on the record.

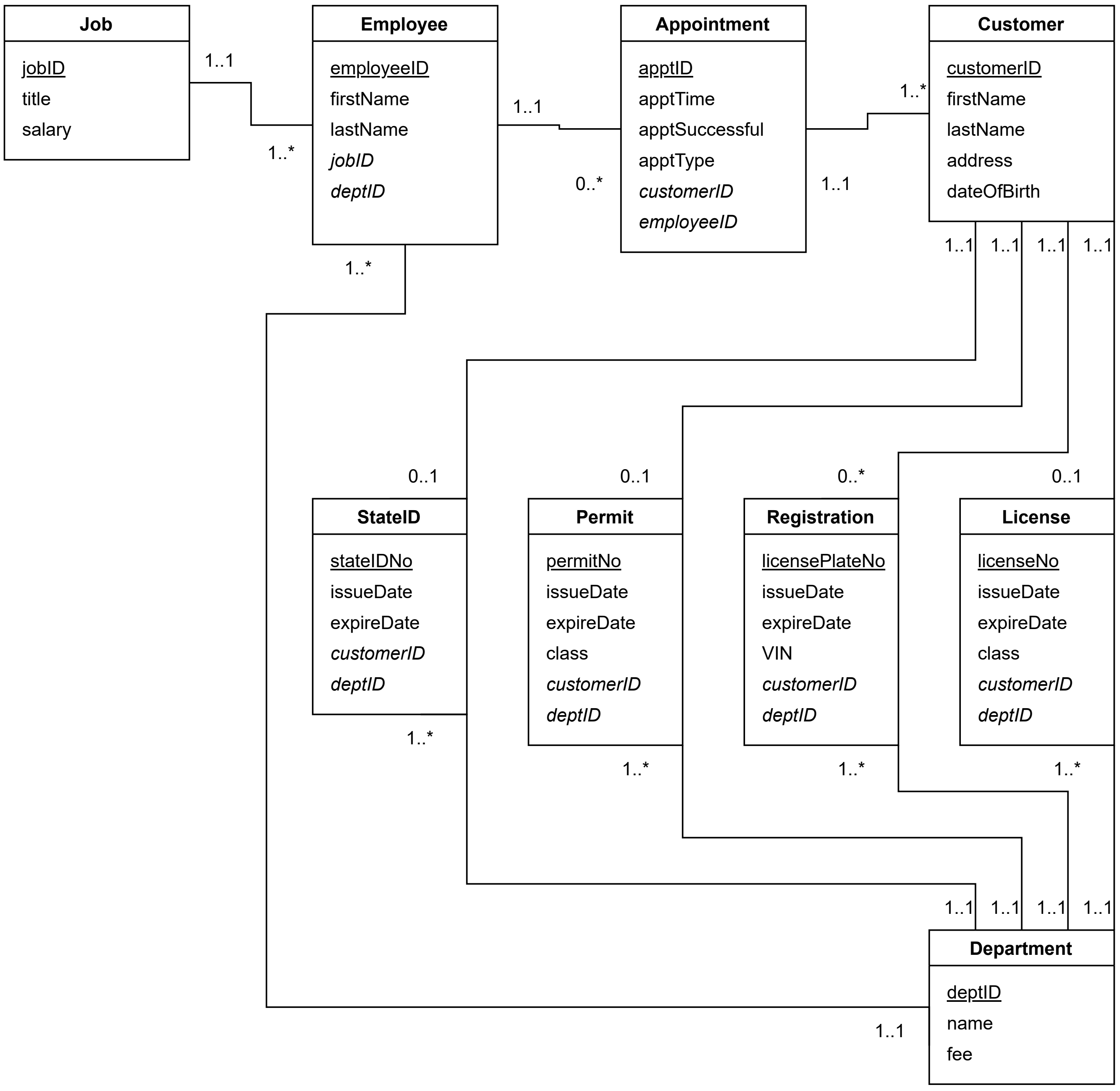
Closures:

{ licensePlateNo }+ = { issueDate, expireDate, class, customerID, deptID}
{ VIN }+ = { licensePlateNo, issueDate, expireDate, class, customerID, deptID }
{ issueDate }+ = { issueDate }
{ expireDate }+ = { expireDate }
{class}+ = {class}
{ customerID }+ = { customerID }
{deptID }+ = {deptID}

Data Normalization
None of the attributes are set-valued so this is in 1NF. The two non-trivial attributes in Registration are both superkeys because both licensePlateNo and VIN are candidate keys. That means this is in BCNF and therefore in 2NF as well.

iv) Query Description
Our query 4 displays all of the job titles and salaries for a given department. This query could be helpful for HR, in terms of seeing what other employees are making, or in terms of seeing where staffing shortages may exist within the DMV.

Claire Getz
Jacob LaSalle
Jiazheng Huang



Job
jobID
title
salary
Employee
employeeID
firstName
lastName
jobID
deptID
Appointment
apptID
apptTime
apptSuccessful
apptType
customerID
employeeID
Customer
customerID
firstName
lastName
address
dateOfBirth
StateID
stateIDNo
issueDate
expireDate
customerID
deptID
Permit
permitNo
issueDate
expireDate
class
customerID
deptID
Registration
licensePlateNo
issueDate
expireDate
VIN
customerID
deptID
License
licenseNo
issueDate
expireDate
class
customerID
deptID
Department
deptID
name
fee
1..1
1..*
1..1
0..*
1..*
1..1
1..*
1..1
1..1
1..1
1..1
0..1
0..1
0..*
0..1
1..*
1..*
1..*
1..*
1..1
1..1
1..1
1..1
1..1