Peter Dinh 1835957 dbs027
Danny Phan 1609411 dbs095
Tahmeed Zaman 1625011 dbs125
DavidLeeDaniels

Database Overview

Contribution:

Danny Phan: Built the website application using the TAs code, worked on all the pages, queries, ER Diagrams and PDF.
Tahmeed Zaman: Created the video demo, worked on ER Diagram, PDF.
Peter Dinh: Worked on the employee section for the website
David Lee Daniels: Worked on table creation, resetting database, comments for sql files

Database Normalization Explanation

- There are a total of eight tables which are: employee, airport, country, flight assignment, job, office, office shift, and payment. **The only tables that are not normalized to BCNF include the lookup tables including airport and job.** This is because these have values that are dependent on each other, excluding the primary key we chose. For instance, every airport has a unique city and airport name that are in turn dependent on each other, so it breaks BCNF. However, I believe this is acceptable because an actual database has at least a few lookup tables. The same goes for job, which has some unique values such as pay rate, overtime_payrate, which can break 3NF as there is now some transitive dependencies between these columns and the other non-key columns. However, if we exclude candidate keys, these tables become BCNF.

  As for the other tables, essentially every row can have duplicates excluding the primary key. This means that transitive and dependencies nonkey -> key columns are possible, but in actual practice do not occur because for instance there can be two employees that have the same name, but live in different places with a different job. There can be flight assignments with the same departure and arrival_airport but different crew. And the same goes for officeshifts.

  Below we detail the sql for creating the tables, and the columns that come with each.

Database Creation and Tables

- The Employee table contains information for all employees including their social security number, first/last name, email, gender, street number, city, country, job, their current assigned airport code, remaining sick leave and vacation days We also fill in the sick leave and vacation days in employee according to their job in the job table. Later on you will be able to edit these sick leave and vacation days, but the initial values are there

upon database initialization (although you can't enter a value higher than the initial, for obvious reasons).

```
create table employee (
    social_security_num VARCHAR(50) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(50),
    gender VARCHAR(50),
    street_num VARCHAR(50),
    city VARCHAR(50),
    country VARCHAR(50),
    job VARCHAR(20),
    current_airport_code VARCHAR(3),
    sick_leave INT,
    vacation_days INT
);
UPDATE employee
SET sick_leave = job.sick_leave, vacation_days = job.vacation_days
FROM job
WHERE employee.job = job.job;
```

-
- The airport table contains the supported airports with connecting cities. The table contains the unique airport code assigned to each airport name. Each city has one airport.

```
create table airport (
    airport_code VARCHAR(3) PRIMARY KEY,
    city VARCHAR(50),
    airport_name VARCHAR(50)
);
```

- The country table contains the tax rate for each of the countries supported by the airport.

```
create table country (
    country VARCHAR(50) PRIMARY KEY,
    taxes DEC(10,2)
);
```

- Flight assignment table contains all information regarding booked flights. Here, each booked flight has a unique flight id, a departure/arrival airport, scheduled departure/arrival time, and the assigned flight crew (pilot, copilot, flight attendants).

```
create table flightassignment (
    flight_id VARCHAR(3) PRIMARY KEY,
    departure_airport VARCHAR (3),
    arrival_airport VARCHAR (3),
    pilot VARCHAR (50),
    copilot VARCHAR (50),
    scheduled_departure_time TIMESTAMPTZ,
    scheduled_arrival_time TIMESTAMPTZ,
    flight_attendant_1 VARCHAR (50),
    flight_attendant_2 VARCHAR (50),
    flight_attendant_3 VARCHAR (50),
    flight_attendant_4 VARCHAR (50)
);
```

- The job table details the job description and benefits of each job title. Included in the table is the job title, payrate, overtime payrate, medical/retirement benefits, travel expenses, worker compensation, sick leaves, and vacation days.

```
create table job (
    job VARCHAR(20) PRIMARY KEY,
    payrate DECIMAL (5, 2),
    overtime_payrate DECIMAL (5, 2),
    medical_benefits BOOLEAN,
    retirement_benefits BOOLEAN,
    travel_expenses BOOLEAN,
    workers_compensation BOOLEAN,
    sick_leave INT,
    vacation_days INT
);
```

- The office table contains the office number with the airport code

```
create table office (
    office_num INT PRIMARY KEY,
    airport_code VARCHAR(3)
);
```

- The office shifts table has the unique shift id, office id, shift start and end time, and the shift assigned workers

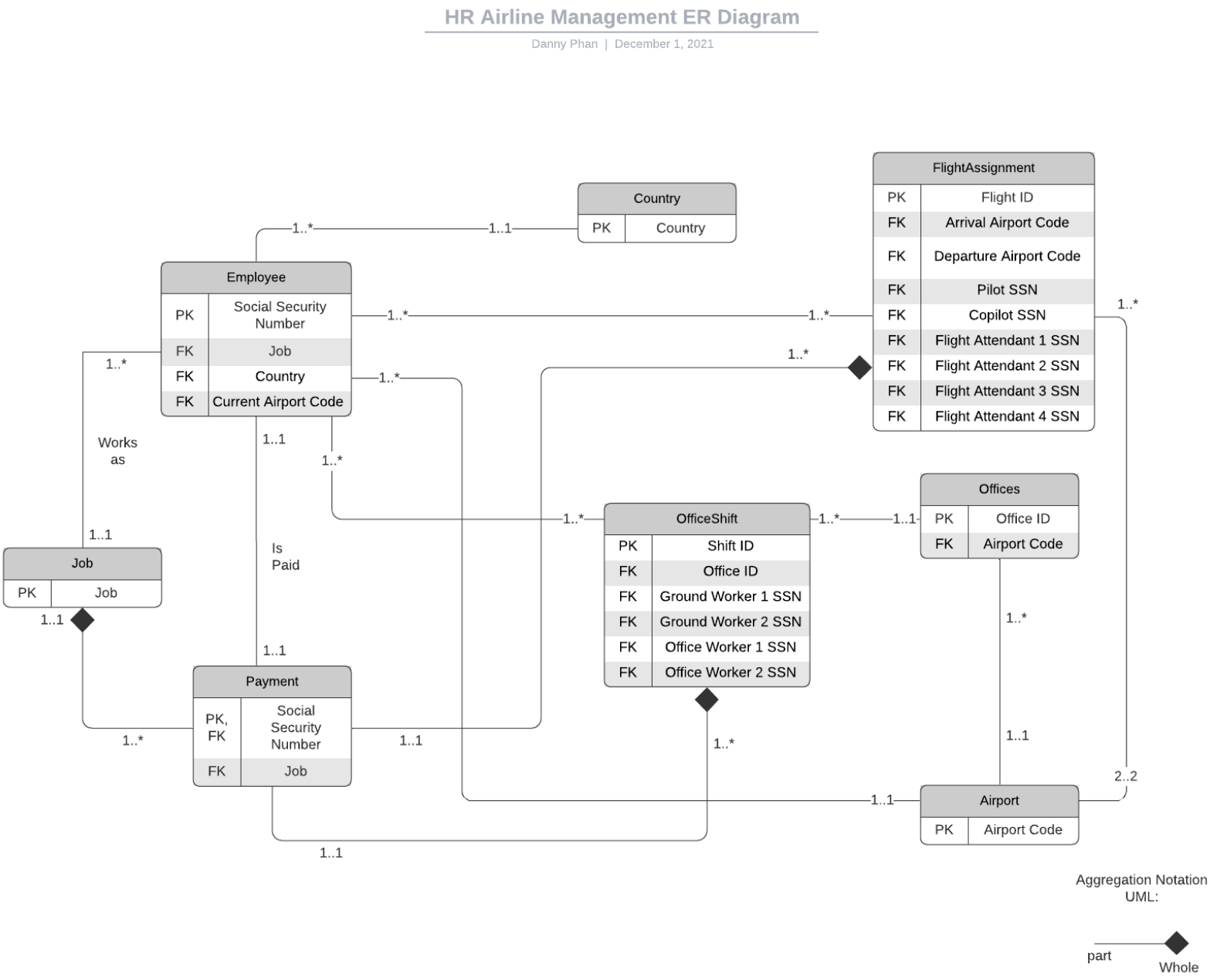
```
create table officeshift (
```

```
    shift_id SERIAL PRIMARY KEY,
    office_id INT,
    shift_start TIMESTAMPTZ,
    shift_end TIMESTAMPTZ,
    ground_worker_1 VARCHAR(50),
    ground_worker_2 VARCHAR(50),
    office_worker_1 VARCHAR(50),
    office_worker_2 VARCHAR(50)
);
```

- The payment table contains the pay of the employee based on their social security number with their job description. Moreover, it details their working hours, overtime hours, taxes, and their monthly salaries.

```
- CREATE TABLE payment (
-     social_security_num VARCHAR(50) PRIMARY KEY,
-     job VARCHAR(20),
-     normal_hours INT,
-     overtime_hours INT,
-     taxes DEC(10,2),
-     monthly_salary DEC(10,2)
- );
```

ER Diagram



I followed the exact notation used on page 226 in the Fundamental of Database Systems textbook for a UML ER diagram. On it I listed out what is used as a primary key/foreign key, and

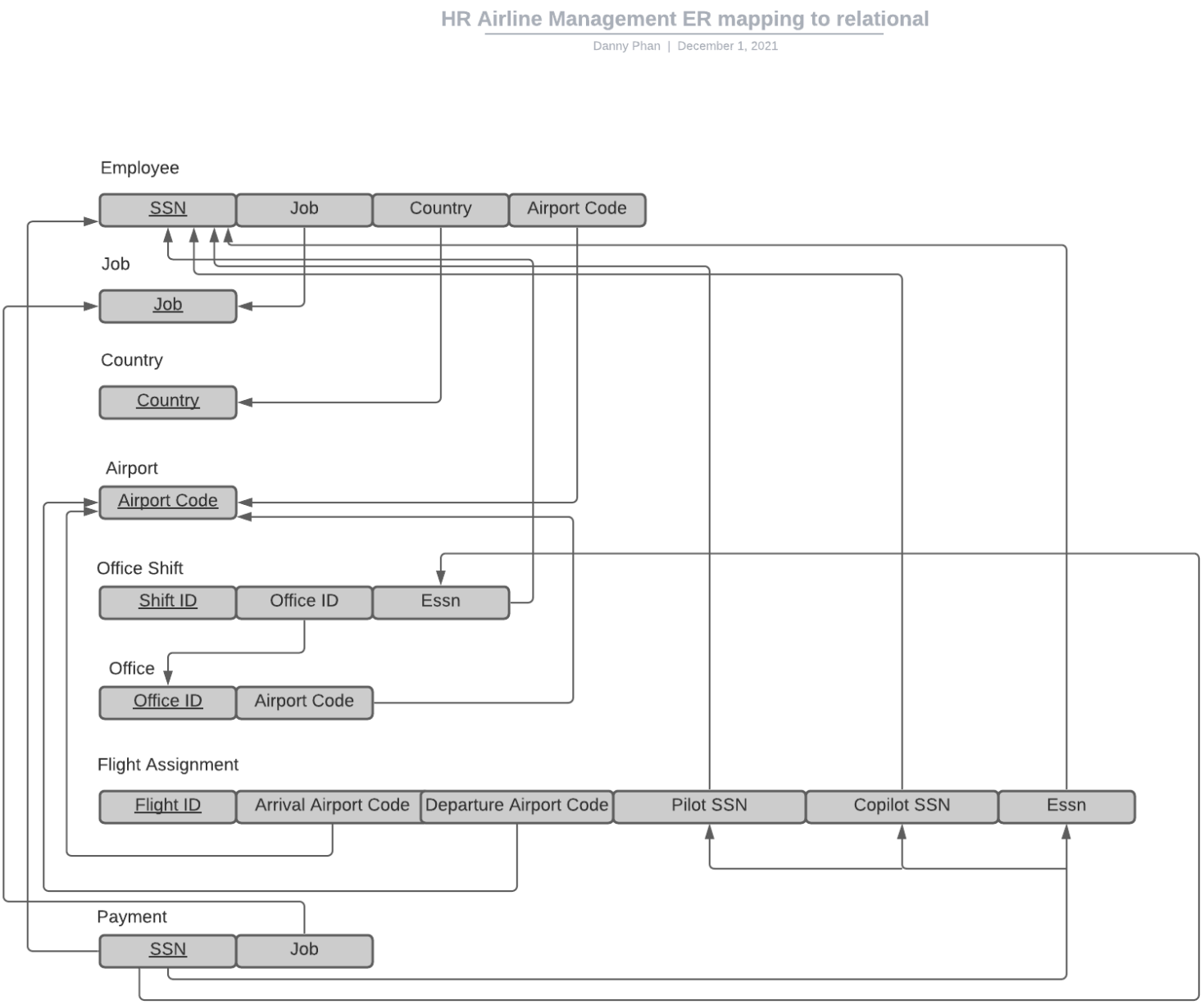
how each table is connected to one another, and their cardinality.

We start with the employee table. We have SSN as a PK, and Job, Country and current airport code as foreign keys.
As such, the relationship **Employee:Country is many to one**, in that many employees can be from one country, but an employee can only have one country. The **same goes for Employee:Airport., and Employee:Job**, as an an employee can only have one job/airport, but jobs/airports can have multiple employees each.
For **one to one relationships, we have Employee:Payment.** Every employee has only one payment, and payments are assigned to only one employee, as each can work different hours, etc
For **many to many, only the Employee:Office Shifts and Employee:Flight Assignment** exist. This is because an employee can work multiple flights and office shifts, and each flight/office shift can have multiple employees.

With the payment table, we have the primary key being an employee SSN, which also acts as a foreign key, and a foreign key job. The main thing is that this table entries are dependent on three other tables: flightassignment, officeshift, and job. We have a **many to one relationship between payment and job**, as a payment has only one job (correlating with the employee's ssn for that payment) and a job can have multiple payments associated with it. From job payment pulls the payrate to calculate its monthly salary (we have not included these fields on the ER diagram as they are neither PKs nor FKs). As we add employees to office shifts or flights, payment uses the SSNs from the workers in these tables to fill it its own columns according to its own primary SSN key. Thus, **payment:officeshifts/flights is many to many** as a payment can have many flights/office shifts (since it's based on each employee), while a flight/office shift can have many payments since row assigns four workers (multiple payments are changed).

Lastly, we have the relationships that detail the locations of flights and officeshifts. For the FlightAssignment table, we have an arrival and departure airport_code that is a FK. The relationship between **flightassignment and airport is thus many to one** (two is required). This is because a flight can have at most two airport codes since it has a arrival and departure location, while an airport obviously has multiple flights. It's a similar situation for officeshifts, only this time the Office ID acts as the FK for OfficeShift and the relationship between **officeshift and office is many to one**.

ER mapping to relational



In this diagram, basically we have the arrows pointing to the table to where the current table's foreign key points to. For instance, payment has a PK that is also a FK SSN, that points to the SSN in FlightAssignment, OfficeShift and Employee (from where it originates). Job in payment points to job in the job table. For flightassignment, arrival and departure airport code FKs point

to the PK airport code in airport. The workers SSNs are FKs (Essn is just the rest of the employees, for simplification) that point to the PK SSN from Employee. Office has airport code as an FK that points to airport code PK in airport. Office shift is similar to flight assignment in that it has the FKs Office ID that point to an office with the PK office id. Workers assigned to it have an SSN that acts as a FK that points to the PK SSN in Employee. Lastly, we have employee that has the FKs Job, Country and airport code that point to the PKs Job in Job, Country in Country, and Airport Code in airport tables respectively. This is roughly how our database works.

SQL Queries Database Explanation

Here, we'll explain the main sql queries and how they work to populate our website with information about employees, etc. Aside from main select statements that are used to display information about employees, flight assignments, and office shifts, we have some more complicated ones. In order to update and display the payments table, we have this select statement.

```
`SELECT social_security_num, job, normal_hours, CAST (payrate AS FLOAT),
overtime_hours, CAST (overtime_payrate AS FLOAT), (normal_hours * payrate
+ overtime_hours * overtime_payrate) * taxes AS taxes, (normal_hours *
payrate + overtime_hours * overtime_payrate) * (1 - taxes) AS
monthly_salary
FROM (
  SELECT t1.social_security_num, job, taxes, payrate, overtime_payrate,
      CASE
          WHEN hours_worked - 40 <= 0 THEN hours_worked
          WHEN hours_worked - 40 > 0 THEN 40
      END normal_hours,
      CASE
          WHEN hours_worked - 40 <= 0 THEN 0
          WHEN hours_worked - 40 > 0 THEN hours_worked - 40
      END overtime_hours
  FROM
      (SELECT pilot AS social_security_num, SUM(DATE_PART('day',
scheduled_arrival_time - scheduled_departure_time) * 24 +
          DATE_PART('hour', scheduled_arrival_time -
scheduled_departure_time)) AS hours_worked
      FROM flightassignment
      WHERE pilot NOT LIKE 'N/A'
      GROUP BY pilot
      UNION
```

```
      SELECT copilot AS social_security_num, SUM(DATE_PART('day',
scheduled_arrival_time - scheduled_departure_time) * 24 +
          DATE_PART('hour', scheduled_arrival_time -
scheduled_departure_time)) AS hours_worked
      FROM flightassignment
      WHERE copilot NOT LIKE 'N/A'
      GROUP BY copilot
      UNION
      SELECT flight_attendant_1 AS social_security_num,
SUM(DATE_PART('day', scheduled_arrival_time - scheduled_departure_time) *
24 +
          DATE_PART('hour', scheduled_arrival_time -
scheduled_departure_time)) AS hours_worked
      FROM flightassignment
      WHERE flight_attendant_1 NOT LIKE 'N/A'
      GROUP BY flight_attendant_1
      UNION
      SELECT flight_attendant_2 AS social_security_num,
SUM(DATE_PART('day', scheduled_arrival_time - scheduled_departure_time) *
24 +
          DATE_PART('hour', scheduled_arrival_time -
scheduled_departure_time)) AS hours_worked
      FROM flightassignment
      WHERE flight_attendant_2 NOT LIKE 'N/A'
      GROUP BY flight_attendant_2
      UNION
      SELECT flight_attendant_3 AS social_security_num,
SUM(DATE_PART('day', scheduled_arrival_time - scheduled_departure_time) *
24 +
          DATE_PART('hour', scheduled_arrival_time -
scheduled_departure_time)) AS hours_worked
      FROM flightassignment
      WHERE flight_attendant_3 NOT LIKE 'N/A'
      GROUP BY flight_attendant_3
      UNION
      SELECT flight_attendant_4 AS social_security_num,
SUM(DATE_PART('day', scheduled_arrival_time - scheduled_departure_time) *
24 +
          DATE_PART('hour', scheduled_arrival_time -
scheduled_departure_time)) AS hours_worked
```

```
      FROM flightassignment
      WHERE flight_attendant_4 NOT LIKE 'N/A'
      GROUP BY flight_attendant_4
      UNION
      SELECT ground_worker_1 AS social_security_num, SUM(DATE_PART('day',
shift_end - shift_start) * 24 +
          DATE_PART('hour', shift_end - shift_start)) AS hours_worked
      FROM officeshift
      WHERE ground_worker_1 NOT LIKE 'N/A'
      GROUP BY ground_worker_1
      UNION
      SELECT ground_worker_2 AS social_security_num, SUM(DATE_PART('day',
shift_end - shift_start) * 24 +
          DATE_PART('hour', shift_end - shift_start)) AS hours_worked
      FROM officeshift
      WHERE ground_worker_2 NOT LIKE 'N/A'
      GROUP BY ground_worker_2
      UNION
      SELECT office_worker_1 AS social_security_num, SUM(DATE_PART('day',
shift_end - shift_start) * 24 +
          DATE_PART('hour', shift_end - shift_start)) AS hours_worked
      FROM officeshift
      WHERE office_worker_1 NOT LIKE 'N/A'
      GROUP BY office_worker_1
      UNION
      SELECT office_worker_2 AS social_security_num, SUM(DATE_PART('day',
shift_end - shift_start) * 24 +
          DATE_PART('hour', shift_end - shift_start)) AS hours_worked
      FROM officeshift
      WHERE office_worker_2 NOT LIKE 'N/A'
      GROUP BY office_worker_2) AS t1
  JOIN
      (SELECT employee.job, taxes, social_security_num, payrate,
overtime_payrate
      FROM employee, job, country
      WHERE employee.job = job.job AND country.country = employee.country)
AS t2
      ON t1.social_security_num = t2.social_security_num
) AS t3;`
```

I detailed how this works in a comment inside of the query.sql when it runs. Essentially, we

```
Select ssn, hours worked, overtime hours worked, normal payrate, overtime
payrate, taxes and monthly salary from employee, flightassignmens, job and
country tables.
We find all the flights/office shifts assigned to a social security
number, get the total hours worked, place anything worked above 40 hours
as overtime, then we pull from the job table to get the payrate for the
employee and calculate their monthly salary by multiplying hours worked by
the appropriate payrate. Lastly we get the tax for the employee's country
and multiply it by the previous result, and subtract it to get the monthly
salary.
```

All of this is used to display the payment for an employee alongside their job, the normal and overtime hours they work, their normal and overtime pay rate, monthly salary, and taxes they pay.

We then run an update query like this (this example is filled with employees that currently were assigned either a flight or office shift in our database) to update the payment table so we can make sure not to lose any information. This is of course wrapped in a BEGIN COMMIT as it is an update.

```
BEGIN;
Update payment
SET normal_hours = CASE social_security_num
                     WHEN '218-24-7059' THEN 22
                     WHEN '704-09-4153' THEN 22
                   END,
    overtime_hours = CASE social_security_num
                     WHEN '218-24-7059' THEN 0
                     WHEN '704-09-4153' THEN 0
                   END,
    taxes = CASE social_security_num
                     WHEN '218-24-7059' THEN 431.20
                     WHEN '704-09-4153' THEN 431.20
                   END,
    monthly_salary = CASE social_security_num
                     WHEN '218-24-7059' THEN 1108.80
                     WHEN '704-09-4153' THEN 1108.80
                   END
WHERE social_security_num IN ('218-24-7059', '704-09-4153');
COMMIT;
```

Other transactions include updating a flightassignment to assign workers:

```
'BEGIN';
`UPDATE flightassignment SET pilot = '${pilot}', copilot = '${copilot}',
flight_attendant_1 = '${flight_attendant_1}', flight_attendant_2 =
'${flight_attendant_2}',
flight_attendant_3 = '${flight_attendant_3}', flight_attendant_4 =
'${flight_attendant_4}'
WHERE flight_id LIKE '${id}';`
'COMMIT';
```

Updating an officeshift to assign workers:

```
'BEGIN';
`UPDATE officeshift
SET ground_worker_1 = '${ground_worker_1}', ground_worker_2 =
'${ground_worker_2}', office_worker_1 = '${office_worker_1}',
office_worker_2 = '${office_worker_2}'
WHERE shift_id = ${id};`
'COMMIT';
```

Updating an employee's information:

```
'BEGIN';
`UPDATE employee
SET first_name = '${first_name}', last_name = '${last_name}', email =
'${email}', street_num = '${street_address}', city = '${city}', country =
'${country}',
sick_leave = ${sick_leave}, vacation_days = ${vacation_days}
WHERE social_security_num = '${id}';`
'COMMIT';
```

Inserting a new employee into employee table, and creating a new payment in payment method for that employee:

```
'BEGIN';
`INSERT INTO employee (social_security_num, first_name, last_name, email,
gender, street_num, city, country, job, current_airport_code, sick_leave,
vacation_days)
VALUES
('${id}','${first_name}','${last_name}','${email}','${gender}','${street_a
ddress}','${city}','${country}','${job}','${airport_code}',${sick_leave},$
{vacation_days});`
`INSERT INTO payment (social_security_num, job, normal_hours,
overtime_hours, taxes, monthly_salary)
```

```
VALUES ('${id}', '${job}', 0, 0, 0, 0);`
'COMMIT';
```