

# COMP421 - Project Deliverable 2

Eric Cheng, Gordon Ng, Samuel Ren, Greta Zu

February 23, 2024

## 1 Relational Schema

### 1.1 Entities:

Users (user ID, name, email, phone number, address, credit card information)

- NOT NULL: name, email, phone number, address, credit card information

Registered (user ID, username, password, language)

- Foreign Key: user ID references Users
- NOT NULL: username, password, language

Cities (city name, country) DepartureCity (departure city, departure country, airport departure name)

- Foreign Key: departure city references Cities (city name), departure country references Cities (country)
- NOT NULL: airport departure name

ArrivalCity (arrival city, arrival country, airport arrival name)

- Foreign Key: arrival city references Cities (city name), arrival country references Cities (country)
- NOT NULL: airport arrival name

Flights (flight number, departure date time, airline policy, airline, economy seats, premium economy seats, business seats, first class seats, economy cost, premium economy cost, business cost, first class cost, departure city, departure country, arrival city, arrival country, departure date time, arrival date time, flight duration)

- Foreign Key: departure city and arrival city reference Cities (city name); departure country and arrival country reference Cities (country) (all related through *route*)
- NOT NULL: airline policy, airline, airplane model, economy seats, economy cost, departure city, departure country, arrival city, arrival country, arrival date time, flight duration

Car (car license plate, number of seats, car rental agency, transmission type, car model, car engine type, car daily cost, city name, country, company policy, AC, carplay)

- Foreign Key: city name and country reference Cities (related through *belongs to*)
- NOT NULL: number of seats, car rental agency, transmission type, car model, car engine type, car daily cost, city name, country, company policy, AC, carplay

Hotel (brand affiliation, hotel address, hotel policy, airport shuttle, business facilities, restaurants, listing name, fitness center, on-site parking, pet allowance, pool, city name, country)

- Foreign Key: city name and country reference Cities (related through *located in*)
- NOT NULL: hotel policy, listing name, city name, country

Flight Booking (flight reference number, user ID, passenger names, flight number, departure date time, flight total cost, fare class, seat numbers, plane ticket cost, plane ticket surcharge, plane ticket tax, flight booking fees, flight booking date)

- Foreign Key: user ID references Users (related through *flight reservation*), flight number and departure date time references Flights (related through *flight booked*)
- NOT NULL: user ID, passenger names, flight number, departure date time, flight total cost, fare class, seat numbers, plane ticket cost, plane ticket surcharge, plane ticket tax, flight booking fees, flight booking date

Hotel Booking (hotel reference number, user ID, room number, brand affiliation, hotel address, checkin date, checkout date, hotel total cost, hotel tax, hotel booking fees, hotel booking date)

- Foreign Key: user ID references Users (related through *hotel reservation*), room number references Room (related through *room booked*), brand affiliation and hotel address reference Hotel (weak entity relation)
- NOT NULL: room cost, hotel tax, hotel booking date, hotel booking fees, hotel total cost, user ID, room number, brand affiliation, hotel address

Car Rental Booking (car rental reference number, user ID, car license plate, pickup location, return location, pickup date time, return date time, car rental booking date, car rental cost, car rental tax, car rental booking fees, car rental total cost, insurance)

- Foreign Key: car license plate references Car (related through *car booked*), user ID references Users (related through *car rental reservation*)
- NOT NULL: user ID, car license plate, pickup location, return location, pickup date time, return date time, car rental booking date, car rental cost, car rental tax, car rental booking fees, car rental total cost, insurance

## 1.2 Weak Entity:

Room (hotel address, brand affiliation, room number, room name, room capacity, beds, room price, size, free wifi, view, minibar, private bathroom, smoking)

- Foreign Key: brand affiliation and hotel address reference Hotel (related through *part of*)
- NOT NULL: room name, room capacity, beds, room price, size, view, minibar, private bathroom, smoking

## 1.3 Relationships:

Route (departure city, departure country, arrival city, arrival country)

- Foreign Key: departure city and departure country reference DepartureCity, arrival city and arrival country reference ArrivalCity

## 2 Pending Constraints

- Flight Booking:
  - Our model does not check if there are any seats available before booking. By extension seats assignment do not check their availabilities (and thus two people could book the same seat).
  - The input fare class might not exist. Our model does not make the check.
  - Our model does not make sure that each passenger in the booking is assigned a seat.
  - The ticket pricing might not match the listed prices of the price. All the pricing calculations are manually done and there are no constraints to ensure that the prices are accurate.
- Hotel Booking:
  - Our model does not check if there are any rooms available before the booking and thus, two users could potentially book the same room.
  - The booking price might not match the listing price.
  - Same as the flight booking, the pricing might not match.
- Car Rental Booking
  - Our model does not check for the car availabilities, therefore, two users could book the same car.
  - Similarly, our model does not constrain the prices and so the amounts might not match with the listed prices nor the calculated prices.

## 3 SQL Queries

### 3.1 Query 1

Find the total costs for each type of bookings (hotel, flight and car rental) for a certain user (user 4 in this example).

```
SELECT
    COALESCE(flight.user_id, hotel.user_id, car.user_id) AS user_id,
    flight.flight_total_cost AS flight_total_cost,
    hotel.hotel_total_cost AS hotel_total_cost,
    car.car_rental_total_cost AS car_rental_total_cost
FROM
    (SELECT
        user_id,
        SUM(flight_total_cost) AS flight_total_cost
    FROM
        FlightBooking
    WHERE
        user_id = 4
    GROUP BY
        user_id) AS flight
FULL OUTER JOIN
    (SELECT
        user_id,
        SUM(hotel_total_cost) AS hotel_total_cost
    FROM
        HotelBooking
    WHERE
        user_id = 4
    GROUP BY
        user_id) AS hotel ON flight.user_id = hotel.user_id
FULL OUTER JOIN
    (SELECT
        user_id,
        SUM(car_rental_total_cost) AS car_rental_total_cost
    FROM
        CarRentalBooking
    WHERE
        user_id = 4
    GROUP BY
        user_id) AS car ON flight.user_id = car.user_id;
```

Here is a screenshot of the statement being executed in DB2 using the command line utility:

```

db2 => SELECT
  COALESCE(flight.user_id, hotel.user_id, car.user_id) AS user_id,
  flight.flight_total_cost AS flight_total_cost,
  hotel.hotel_total_cost AS hotel_total_cost,
  car.car_rental_total_cost AS car_rental_total_cost
FROM
  (SELECT
    user_id,
    SUM(flight_total_cost) AS flight_total_cost
  FROM
    FlightBooking
  WHERE
    user_id = 4
  GROUP BY
    user_id) AS flight
FULL OUTER JOIN
  (SELECT
    user_id,
    SUM(hotel_total_cost) AS hotel_total_cost
  FROM
    HotelBooking
  WHERE
    user_id = 4
  GROUP BY
    user_id) AS hotel ON flight.user_id = hotel.user_id
FULL OUTER JOIN
  (SELECT
    user_id,
    SUM(car_rental_total_cost) AS car_rental_total_cost
  FROM
    CarRentalBooking
  WHERE
    user_id = 4
  GROUP BY
    user_id) AS car ON flight.user_id = car.user_id;db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>

USER_ID      FLIGHT_TOTAL_COST      HOTEL_TOTAL_COST      CAR_RENTAL_TOTAL_COST
-----
          4              1070.00              1055.00              3595.00

1 record(s) selected.

db2 => █

```

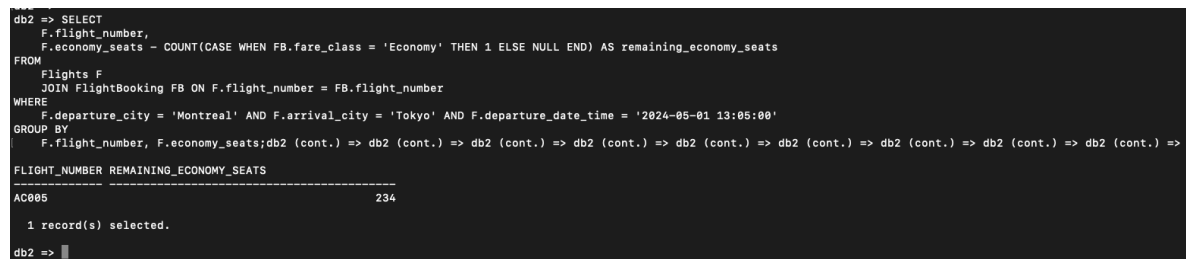
Figure 1: Input and output of Query 1 execution.

### 3.2 Query 2

Find the number of economy seats remaining on the flight from Montreal to Tokyo on May 1, 2024.

```
SELECT
    F.flight_number ,
    F.economy_seats - COUNT(CASE WHEN FB.fare_class = 'Economy' THEN 1 ELSE
        NULL END) AS remaining_economy_seats
FROM
    Flights F
    JOIN FlightBooking FB ON F.flight_number = FB.flight_number
WHERE
    F.departure_city = 'Montreal' AND F.arrival_city = 'Tokyo' AND F.
        departure_date_time = '2024-05-01 13:05:00'
GROUP BY
    F.flight_number , F.economy_seats;
```

Here is a screenshot of the statement being executed in DB2 using the command line utility:



```
db2 => SELECT
    F.flight_number,
    F.economy_seats - COUNT(CASE WHEN FB.fare_class = 'Economy' THEN 1 ELSE NULL END) AS remaining_economy_seats
FROM
    Flights F
    JOIN FlightBooking FB ON F.flight_number = FB.flight_number
WHERE
    F.departure_city = 'Montreal' AND F.arrival_city = 'Tokyo' AND F.departure_date_time = '2024-05-01 13:05:00'
GROUP BY
    F.flight_number, F.economy_seats;db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
FLIGHT_NUMBER REMAINING_ECONOMY_SEATS
-----
AC005                                234
1 record(s) selected.
db2 =>
```

Figure 2: Input and output of Query 2 execution.

### 3.3 Query 3

Find all users who have booked both a flight and a hotel.

```
SELECT
    U.user_id,
    U.name,
    COUNT(DISTINCT FB.flight_number) AS total_flights_booked,
    COUNT(DISTINCT HB.hotel_reference_number) AS total_hotels_booked
FROM
    Users U
    JOIN FlightBooking FB ON U.user_id = FB.user_id
    JOIN HotelBooking HB ON U.user_id = HB.user_id
GROUP BY
    U.user_id, U.name
HAVING
    COUNT(DISTINCT FB.flight_number) > 0 AND COUNT(DISTINCT HB.
        hotel_reference_number) > 0;
```

Here is a screenshot of the statement being executed in DB2 using the command line utility:

```

db2 => SELECT U.user_id, U.name,
db2 (cont.) => COUNT(DISTINCT FB.flight_number) AS total_flights_booked,
db2 (cont.) => COUNT(DISTINCT HB.hotel_reference_number) AS total_hotels_booked
db2 (cont.) => FROM Users U
db2 (cont.) => JOIN FlightBooking FB ON U.user_id = FB.user_id
db2 (cont.) => JOIN HotelBooking HB ON U.user_id = HB.user_id
db2 (cont.) => GROUP BY U.user_id, U.name
db2 (cont.) => HAVING
db2 (cont.) => COUNT(DISTINCT FB.flight_number) > 0 AND COUNT(DISTINCT HB.hotel_reference_number) > 0;

USER_ID      NAME                TOTAL_FLIGHTS_BOOKED TOTAL_HOTELS_BOOKED
-----
1 John Doe                1                1
3 John Smith             2                2
4 Jane Smith             2                2

3 record(s) selected.

db2 => _

```

Figure 3: Input and output of Query 3 execution.

### 3.4 Query 4

Find all users who have booked a round trip (a flight to a city and another flight back).

```

SELECT u.user_id, u.name, u.email, u.phone_number
FROM Users u
WHERE EXISTS (
    SELECT 1 FROM FlightBooking fb1
    JOIN FlightBooking fb2 ON fb1.user_id = fb2.user_id
    JOIN Flights f1 ON fb1.flight_number = f1.flight_number
    JOIN Flights f2 ON fb2.flight_number = f2.flight_number
    WHERE fb1.user_id = u.user_id
    AND fb1.departure_date_time < fb2.departure_date_time
    AND fb1.flight_reference_number != fb2.flight_reference_number
    AND f1.arrival_city = f2.departure_city
    AND f2.arrival_city = f1.departure_city
);

```

Here is a screenshot of the statement being executed in DB2 using the command line utility:

```

db2 => SELECT u.user_id, u.name, u.email, u.phone_number
FROM Users u
WHERE EXISTS (
    SELECT 1 FROM FlightBooking fb1
    JOIN FlightBooking fb2 ON fb1.user_id = fb2.user_id
    JOIN Flights f1 ON fb1.flight_number = f1.flight_number
    JOIN Flights f2 ON fb2.flight_number = f2.flight_number
    WHERE fb1.user_id = u.user_id
    AND fb1.departure_date_time < fb2.departure_date_time
    AND fb1.flight_reference_number != fb2.flight_reference_number
    AND f1.arrival_city = f2.departure_city
    AND f2.arrival_city = f1.departure_city
);
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
USER_ID      NAME                EMAIL                PHONE_NUMBER
-----
3 John Smith             jsmith@yahoo.com      +1(514)345-6789
4 Jane Smith             janes@yahoo.com        +1(438)345-6789

2 record(s) selected.

db2 => █

```

Figure 4: Input and output of Query 4 execution.

### 3.5 Query 5

List all cities that have a flight going there from Montreal with hotels and car rentals offered.

```
SELECT DISTINCT f.arrival_city
FROM Flights f
WHERE f.departure_city = 'Montreal'
AND f.arrival_city IN (SELECT h.city_name FROM Hotel h)
AND f.arrival_city IN (SELECT c.city_name FROM Car c);
```

Here is a screenshot of the statement being executed in DB2 using the command line utility:

```
db2 => SELECT DISTINCT f.arrival_city
FROM Flights f
WHERE f.departure_city = 'Montreal'
AND f.arrival_city IN (SELECT h.city_name FROM Hotel h)
AND f.arrival_city IN (SELECT c.city_name FROM Car c);db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>

ARRIVAL_CITY
-----
Tokyo

      1 record(s) selected.

db2 => █
```

Figure 5: Input and output of Query 5 execution.



## 4 SQL Modifications

### 4.1 Mod 1

Insert new user with ID one greater than the largest existing ID. (Inserting the result of a query)

```
INSERT INTO Users (user_id, name, email, phone_number, address,
credit_card_information)
VALUES (
(SELECT COALESCE(MAX(user_id), 0) + 1 FROM Users),
'NewUser', -- Example user name
'newuser@example.com', -- Example email
'1234567890', -- Example phone number
'New Address', -- Example address
'1234-5678-9012-3456' -- Example credit card information
);
```

```
db2 =>
db2 => INSERT INTO Users (user_id, name, email, phone_number, address, credit_card_information)
VALUES (
(SELECT COALESCE(MAX(user_id), 0) + 1 FROM Users),
'NewUser', -- Example user name
'newuser@example.com', -- Example email
'1234567890', -- Example phone number
'New Address', -- Example address
'1234-5678-9012-3456' -- Example credit card information
);db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
DB20000I The SQL command completed successfully.
db2 => SELECT * FROM Users;
```

USER_ID	NAME	EMAIL	PHONE_NUMBER	ADDRESS	CREDIT_CARD_INFORMATION
1	John Doe	jdoe@gmail.com	+1(514)123-4567	1234 Rue Saint-Catherine, Montreal, QC, Canada	1234123412341234, 12/23, 123
2	Jane Doe	jane@gmail.com	+1(438)123-4567	1234 Rue Saint-Catherine, Montreal, QC, Canada	4321432143214321, 11/25, 321
3	John Smith	jsmith@yahoo.com	+1(514)345-6789	321 Blvd Saint-Jean, Pointe-Claire, QC, Canada	1234432112344321, 06/26, 456
4	Jane Smith	janes@yahoo.com	+1(438)345-6789	321 Blvd Saint-Jean, Pointe-Claire, QC, Canada	4321123443211234, 07/24, 789
5	John Johnson	jj@gmail.com	+1(514)567-8901	1111 Blvd Saint-Charles, Montreal, QC, Canada	1234123443214321, 06/27, 555
6	Jane Johnson	janej@gmail.com	+1(438)567-8901	1111 Blvd Saint-Charles, Montreal, QC, Canada	4321123412344321, 07/25, 666
7	NewUser	newuser@example.com	1234567890	New Address	1234-5678-9012-3456

```
7 record(s) selected.
db2 =>
```

Figure 6: Input and output of Mod 1 execution with the Users Table after Mod 1 execution.

### 4.2 Mod 2

Update room prices for rooms in hotels with an average room price below a threshold. In this case, all room prices under \$150.00 are increased.

```
-- Update room prices for rooms in hotels with an average room price above a
threshold
UPDATE Room
SET room_price = room_price * 1.15 -- Increase room prices by 15%
WHERE hotel_address IN (
SELECT H.hotel_address
FROM Hotel H
JOIN Room R ON H.hotel_address = R.hotel_address
GROUP BY H.hotel_address
HAVING AVG(R.room_price) < 150.00 -- Example threshold
);
```

```
db2 =>
db2 => SELECT * FROM Room;
```

HOTEL_ADDRESS	BRAND_AFFILIATION	ROOM_NUMBER	ROOM_NAME	ROOM_CAPACITY	BEDS	ROOM_PRICE	SIZE	FREE_WIFI	VIEW	MINIBAR	PRIVATE_BATHROOM	SMOKING
1234 Rue Sherbrooke Ouest	Hilton	101	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	102	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	103	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	104	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	105	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	106	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	107	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	108	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	101	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	102	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	103	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	104	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	105	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	106	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	107	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	101	Single Room	1	1 Double Bed	80.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	102	Single Room	1	1 Double Bed	80.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	103	Single Room	1	1 Double Bed	80.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	104	Single Room	1	1 Double Bed	80.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	201	Double Room	2	2 Double Bed	130.00	400 sq ft	Y	City view	Y	Y	N

28 record(s) selected.

```
db2 => -- Update room prices for rooms in hotels with an average room price above a threshold
UPDATE Room
SET room_price = room_price * 1.15 -- Increase room prices by 15%
WHERE hotel_address IN (
  SELECT H.hotel_address
  FROM Hotel H
  JOIN Room R ON H.hotel_address = R.hotel_address
  GROUP BY H.hotel_address
  HAVING AVG(R.room_price) < 150.00 -- Example threshold
);db2 => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
DB200001 The SQL command completed successfully.
db2 => SELECT * FROM Room;
```

HOTEL_ADDRESS	BRAND_AFFILIATION	ROOM_NUMBER	ROOM_NAME	ROOM_CAPACITY	BEDS	ROOM_PRICE	SIZE	FREE_WIFI	VIEW	MINIBAR	PRIVATE_BATHROOM	SMOKING
1234 Rue Sherbrooke Ouest	Hilton	101	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	102	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	103	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	104	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	105	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	106	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	107	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue Sherbrooke Ouest	Hilton	108	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	101	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	102	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	103	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	104	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	105	Standard Room	2	1 Queen Bed	150.00	300 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	106	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
1234 Rue René-Lévesque Ouest	Marriott	107	Family Room	4	2 Queen Bed	250.00	600 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	101	Single Room	1	1 Double Bed	92.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	102	Single Room	1	1 Double Bed	92.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	103	Single Room	1	1 Double Bed	92.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	104	Single Room	1	1 Double Bed	92.00	200 sq ft	Y	City view	Y	Y	N
18-2 Nihonbashi-Kakigara-cho, Chuo-ku	APA Hotel	201	Double Room	2	2 Double Bed	149.50	400 sq ft	Y	City view	Y	Y	N

28 record(s) selected.

```
db2 => █
```

Figure 7: Room Table before Mod 2 execution followed by the input and output of Mod 2 execution and the Room Table after Mod 2 execution.

## 5 Views

### 5.1 View 1

The first view represents the contact information of all users irrespective of registration, including both email and phone number. It pulls its information solely from the Users table and has a simple constraint where the user is also a registered user.

```
CREATE VIEW contact_information AS
SELECT user_id, name, email, phone_number
FROM Users
WHERE user_id IN (SELECT user_id FROM Registered);
```

#### 5.1.1 Screenshots

```
db2 => CREATE VIEW contact_information AS
SELECT user_id, name, email, phone_number
FROM Users
WHERE user_id IN (SELECT user_id FROM Registered);db2 (cont.) => db2 (cont.) => db2 (cont.) =>
DB200001 The SQL command completed successfully.
db2 => SELECT * from contact_information;
```

USER_ID	NAME	EMAIL	PHONE_NUMBER
1	John Doe	jdoe@gmail.com	+1(514)123-4567
2	Jane Doe	janed@gmail.com	+1(438)123-4567
3	John Smith	jsmith@yahoo.com	+1(514)345-6789
4	Jane Smith	janes@yahoo.com	+1(438)345-6789
5	John Johnson	jj@gmail.com	+1(514)567-8901
6	Jane Johnson	janej@gmail.com	+1(438)567-8901

6 record(s) selected.

```
db2 => █
```

Figure 8: Input and output of view creation execution.

```
db2 => SELECT * from contact_information LIMIT 5;
```

USER_ID	NAME	EMAIL	PHONE_NUMBER
1	John Doe	jdoe@gmail.com	+1(514)123-4567
2	Jane Doe	janed@gmail.com	+1(438)123-4567
3	John Smith	jsmith@yahoo.com	+1(514)345-6789
4	Jane Smith	janes@yahoo.com	+1(438)345-6789
5	John Johnson	jj@gmail.com	+1(514)567-8901

```

5 record(s) selected.

db2 =>

```

Figure 9: View selection output truncated to 5 records.

```

db2 => INSERT INTO user_contact_information (name, email, phone_number)
db2 (cont.) => VALUES ('Johnson John', 'jj@yahoo.com', '+1(514)765-9810');
DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL0407N Assignment of a NULL value to a NOT NULL column "TBSPACEID=2,
TABLEID=7686, COLNO=0" is not allowed. SQLSTATE=23502
db2 =>

```

Figure 10: Failed insertion attempt into the view.

## 5.2 View 2

The second view represents a view of the statistics of the expenses for each user, depending on the type of booking.

```

CREATE VIEW TotalCost AS
SELECT 'Flight' AS booking_type, user_id, SUM(flight_total_cost) AS total_cost
FROM FlightBooking
GROUP BY user_id
UNION ALL
SELECT 'Hotel' AS booking_type, user_id, SUM(hotel_total_cost) AS total_cost
FROM HotelBooking
GROUP BY user_id
UNION ALL
SELECT 'Car Rental' AS booking_type, user_id, SUM(car_rental_total_cost) AS
    total_cost
FROM CarRentalBooking
GROUP BY user_id;

```

### 5.2.1 Screenshots

```

db2 => CREATE VIEW TotalCost AS
SELECT 'Flight' AS booking_type, user_id, SUM(flight_total_cost) AS total_cost
FROM FlightBooking
GROUP BY user_id
UNION ALL
SELECT 'Hotel' AS booking_type, user_id, SUM(hotel_total_cost) AS total_cost
FROM HotelBooking
GROUP BY user_id
UNION ALL
SELECT 'Car Rental' AS booking_type, user_id, SUM(car_rental_total_cost) AS total_cost
FROM CarRentalBooking
GROUP BY user_id;
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => DB20000I The SQL command completed successfully.
db2 =>
db2 => SELECT * FROM TotalCost;

```

BOOKING_TYPE	USER_ID	TOTAL_COST
Flight	1	2220.00
Flight	3	2220.00
Flight	4	1070.00
Hotel	1	2727.00
Hotel	3	2209.00
Hotel	4	1000.00
Car Rental	1	3010.00
Car Rental	3	0000.00
Car Rental	4	3090.00

```

9 record(s) selected.

db2 =>

```

Figure 11: Input and output of view creation execution.

```
db2 => SELECT * FROM TotalCost Limit 5;
```

BOOKING_TYPE	USER_ID	TOTAL_COST
Flight	1	2220.00
Flight	3	2220.00
Flight	4	1070.00
Hotel	1	2727.00
Hotel	3	2209.50

```

5 record(s) selected.

db2 => █

```

Figure 12: View selection output truncated to 5 records.

```
db2 =>
db2 => INSERT INTO TotalCost (user_id, total_cost) Values (2, 100.00);
DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL0150N The target fullselect, view, typed table, materialized query table,
range-clustered table, or staging table in the INSERT, DELETE, UPDATE, MERGE,
or TRUNCATE statement is a target for which the requested operation is not
permitted. SQLSTATE=42807
db2 => █

```

Figure 13: Failed insertion attempt into the view.

### 5.3 Error Explanation

Given the error "DB21034E" in both View 1 and View 2, it indicates an error in the syntax of the command line statement provided. This is due to the fact that the views themselves are not tables and should not be treated as if they are. Such is highlighted in the DB2 manual for this error code: "An error occurs when the CLP passes the text to the database manager to process as an SQL statement. When this scenario happens, this message is returned, followed by the error message that was returned by the database manager."

## 6 Check Constraints

### 6.1 Check 1

This constraint serves to check that no flights will be registered at a negative price. This includes all fares for all classes from economy to business.

```
ALTER TABLE Flights
ADD CONSTRAINT check_not_negative
CHECK (
    economy_cost > 0 AND
    premium_economy_cost > 0 AND
    business_cost > 0 AND
    first_class_cost > 0
);
```

```
db2 (cont.) => ADD CONSTRAINT check_not_negative
db2 (cont.) => CHECK (
db2 (cont.) => economy_cost > 0 AND
db2 (cont.) => premium_economy_cost > 0 AND
db2 (cont.) => business_cost > 0 AND
db2 (cont.) => first_class_cost > 0
db2 (cont.) => );
DB20000I The SQL command completed successfully.
```

Figure 14: Altering table with check.

```
db2 => INSERT INTO Flights (flight_number, airline_policy, airline, airplane_model, economy_seats, premium_economy_seats,
business_seats, first_class_seats, economy_cost, premium_economy_cost, business_cost, first_class_cost, departure_city, de
parture_country, arrival_city, arrival_country, departure_date_time, arrival_date_time, flight_duration)
VALUES ('AC113', '2 checked luggages included, 23kg each', 'Air Canada', 'Boeing 777-300ER', 236, 24, 40, 0, 900.00, -1200
.00, 4000.00, 0, 'Vancouver', 'Canada', 'Tokyo', 'Japan', '2024-05-01-13.00.00.000000', '2024-05-02-16.05.00.000000', '11:
30:00');
db2 (cont.) => DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL0545N The requested operation is not allowed because a row does not
satisfy the check constraint "CS4216118.FLIGHTS.CHECK_NOT_NEGATIVE".
SQLSTATE=23513
db2 =>
```

Figure 15: Insertion attempt with an erroneous negative flight price.

### 6.2 Check 2

This constraint serves to check that the daily cost, as well as the seat number of a car will always be positive.

```
ALTER TABLE Car
ADD CONSTRAINT check_car_validity
CHECK (
    car_daily_cost > 0 AND
    number_of_seats > 0
);
```

```

db2 => ALTER TABLE Car
db2 (cont.) => ADD CONSTRAINT check_car_validity
db2 (cont.) => CHECK(
db2 (cont.) => car_daily_cost > 0 AND
db2 (cont.) => number_of_seats > 0
db2 (cont.) => );
DB20000I The SQL command completed successfully.
db2 =>

```

Figure 16: Altering table with check.

```

db2 => INSERT INTO Car (CAR_LICENSE_PLATE, NUMBER_OF_SEATS, CAR_RENTAL_AGENCY, TRANSMISSION_TYPE, CAR_MODEL, CAR_ENGINE_TY
PE, CAR_DAILY_COST, CITY_NAME, COUNTRY, COMPANY_POLICY, AC, CARPLAY)
VALUES ('ABC322', 5, 'Enterprise', 'Automatic', 'Toyota Corolla', 'Gasoline', -50.00, 'Montreal', 'Canada', 'No smoking, P
ets allowed, Insurance Included', 'Y', 'Y');
db2 (cont.) => DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL0545N The requested operation is not allowed because a row does not
satisfy the check constraint "CS421G118.CAR.CHECK_CAR_VALIDITY".
SQLSTATE=23513
db2 =>

```

Figure 17: Insertion attempt with an erroneous negative daily cost.

## 7 Creativity - Real data sets

The meaningful data was inserted into the DepartureCity and the ArrivalCity tables (they are essentially the same). We created these two tables by simply using Google to search up city names across the world and their corresponding airport names. We managed to generate a pretty large quantity of data in this way. *References used: Google Search Engine.*

## 8 Work Process

We split the work into the different parts: Greta did question 1-3, Eric did 4 and 9, Samuel did 5-6 and Gordon did 7-8. To split the work, we first had an online meeting and then as we worked, we would have smaller group discussions for each part to help each other out. Finally, Eric and Greta had a meeting online to go over all the work and confirm that everything was right and working. In total, we had 3 full group meetings and around 5 smaller meetings.