**Team 11 - Web App Report**

**Program Instructions:**

The web app does use libraries and have dependencies. To ensure a functional use, please run the following commands:

npm i [this installs the dependencies]

npm run build [this is the compiler, and builds the app]

npm run start [this runs the webapp]

The installed node.js must be at version 12.22 at least to run this. Additionally, there may be another install requested upon npm run build, but this install is enough to allow it to run. This extra install only will appear with the correct node.js version.

More information regarding these dependencies, if necessary, can be found in README.MD.

**Contribution:**

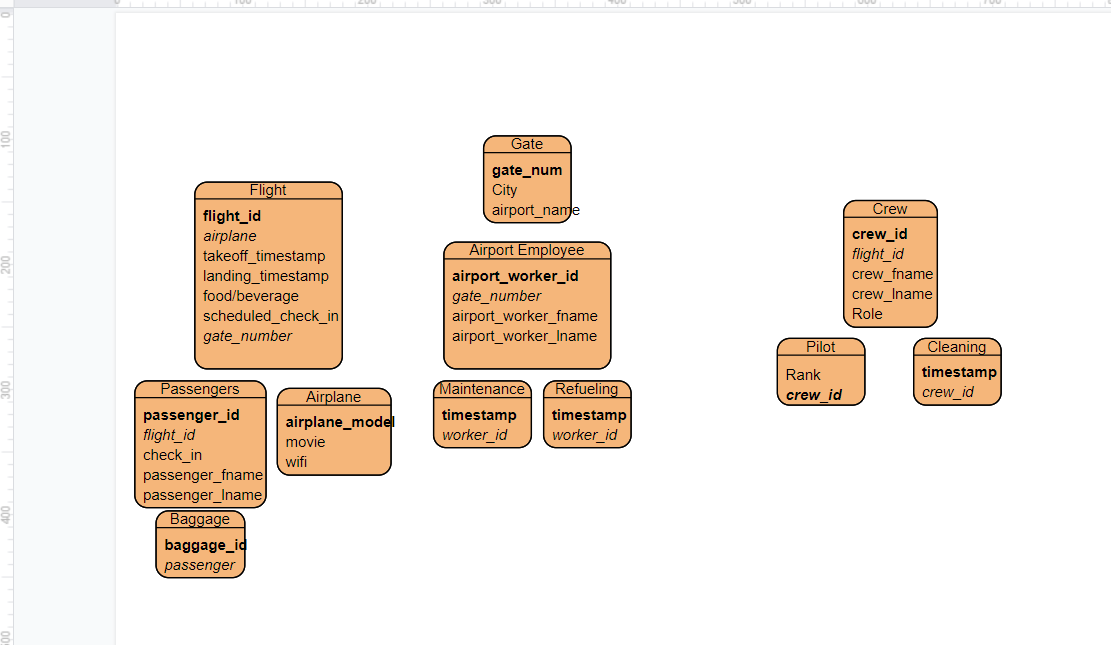
Zachary Robinson - SQL Transaction Programming, Normalization Process, and ER Design

Hugh Hoang - ER Design, Instancing Database on Server, SQL Transaction Programming

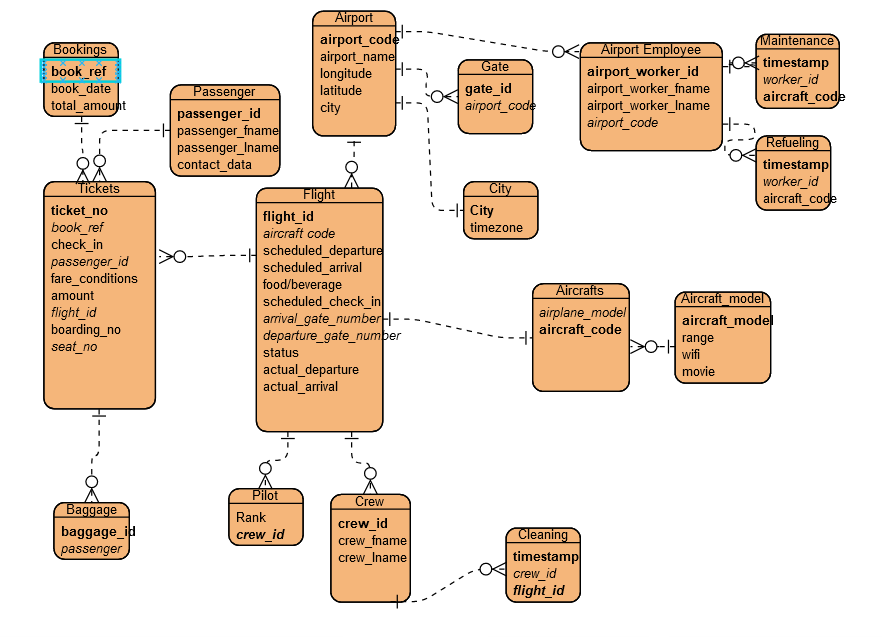
Jana Tahan - Web App Development, JavaScript Implementation

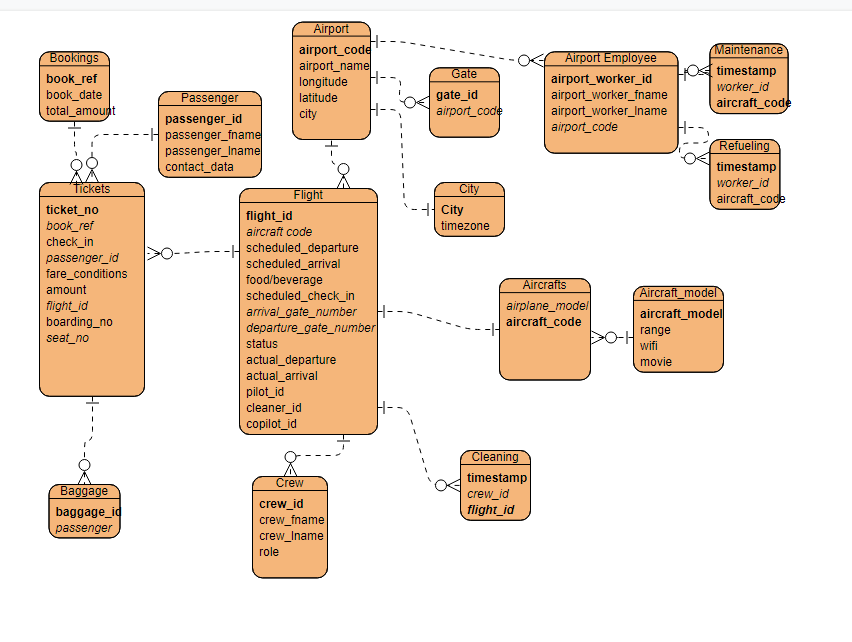
**ER Design Evolution:**

Rough Er Diagram



Intermediate ER Design



Final Er Diagram

The above three ER diagrams show the progress made while designing the database. In the first, rough iteration, the tables were simple and based exclusively on the target assignment requirements. The second, intermediate iteration, was based on the Russian Airline schema, incorporating the target assignment requirements into the base Russian Airline. Additionally, normalization occurred here as some tables like “seat” and “ticket\_flight” from the Russian airline were found to be unnecessary for our case and also needlessly violating BCNF. With this new implementation, almost all tables are in BCNF, except Airport, which is actually in 2NF. The violation came from longitude and latitude determining the city, but it seemed impractical to have a table exclusively for longitude and latitude as they are not important to the operation by themselves but in context of the airport, and having a table with a combination of all relevant longitude and latitudes connected to cities seemed redundant with the City table existing connecting cities to timezones. In the final version, the main changes revolved around the Crew and Flight tables. Pilot was removed and incorporated into Crew, and now each flight has its own pilot, co-pilot, and cleaner.

Main changes between the Russian Airline database and the one implemented here is the use of Gate, which stores each gate so that flights do not conflict with each other, City, which holds the names of relevant cities and their timezone, Baggage, which keeps track of all baggage for the passengers, and then Crew and Airport Employee, who are workers for the airline. Crew consists of pilots, co-pilots, and cleaners who have specific cleaning times in Cleaning, and members of Airport Employee sign up for times in Refuelling and Maintenance.

**Web App Development:**

The Web App part of the project in JavaScript was developed using frameworks and libraries such as Next.js and pg.promise. pg.promise is an API for JavaScript that allows minimum SQL alterations when using JavaScript, by passing in parameters directly with a file of SQL code with the QueryFile class. The API developed for this web application uses this for all interactions with the database. The frontend aspect of the app sends this API all user-inputed data to be used as parameters for pg.promise. Queries are used to verify the transaction is committed successfully for the user. Forms are implemented with a form switcher of sorts that makes it much easier for the user to use the app efficiently.

**Transactions:**

The web application has 8 main operations for users. Firstly, it can input new flights, inserting no flights that have the same destination or arrival gate at the same time. Secondly, this new flight will be assigned an aircraft from the available pool at a given time. Similarly, the flight will generate a pilot, co-pilot, and cleaner for the flight from the available crew members. For cleaners, they can schedule specific times to clean the aircraft on their flight, while airport workers can also schedule specific times to perform refueling and maintenance between flights. The site is able to check passengers in upon arriving at the gate of their flight, as well as check in all of the necessary baggage and remove baggage when retrieved by the passenger. The actual takeoff and arrival times can also be recorded so that it can be compared with the scheduled times. There are additional queries as well such as filtering whether certain aircrafts have a specific type of entertainment (wifi/movie). Transactions/Queries for the above defined operations can be found below.

The transactions for the above operations can be found below:

* **Input new flights and ensure no conflicts with gate numbers at a certain time, assigning an aircraft and crew members**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**possible\_departure\_gate VARCHAR(6) := 'CLTJKE';**

**possible\_arrival\_gate VARCHAR(6) := 'DFWQWE';**

**possible\_departure\_time TIMESTAMP := '2021-08-18 13:57:40';**

**possible\_arrival\_time TIMESTAMP := '2021-08-18 16:12:00';**

**possible\_food VARCHAR(20) := 'CHICKEN';**

**possible\_check\_in TIMESTAMP := '2021-08-18 12:00:00';**

**currentFlightID VARCHAR(5) := 'AT128';**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**possiblePilot INTEGER := 635;**

**possibleCopilot INTEGER := 917;**

**possibleCleaner INTEGER := 459;**

**p\_status VARCHAR(10) := 'ONTIME';**

**flightAircraft VARCHAR(4) := '-';**

**Taken\_Count INTEGER := -1;**

**validOrder BOOLEAN := True;**

**validCheckIn BOOLEAN := True;**

**begin**

**/\*This checks and ensures the departure is before arrival\*/**

**IF (possible\_departure\_time - possible\_arrival\_time) > '0 seconds' THEN validOrder := False; END IF;**

**/\*This checks that the check in time is an acceptable distance from the planned departure\*/**

**IF (possible\_check\_in - possible\_departure\_time) < '-4 hours' OR (possible\_check\_in - possible\_departure\_time) > '-1 hour' THEN validCheckIN := False; END IF;**

**/\*This searches through aircrafts to find one that is available for this flight\*/**

**SELECT aircraft\_code INTO flightAircraft**

**FROM Aircraft**

**EXCEPT**

**SELECT aircraft\_code**

**FROM( SELECT aircraft\_code, scheduled\_arrival**

**FROM Flight) as aircraftTest**

**WHERE (scheduled\_arrival - possible\_departure\_time) < '1 day' AND (scheduled\_arrival - possible\_departure\_time) > '-1 day'**

**LIMIT 1;**

**/\*This searches through crew members who are pilots to find one that is available for this flight\*/**

**SELECT crew\_id INTO possiblePilot**

**FROM Crew**

**WHERE role = 'Pilot'**

**EXCEPT**

**SELECT pilot\_id**

**FROM ( SELECT pilot\_id, scheduled\_arrival**

**FROM Flight) as pilotTest**

**WHERE (scheduled\_arrival - possible\_departure\_time) < '1 day' AND (scheduled\_arrival - possible\_departure\_time) > '-1 day'**

**LIMIT 1;**

**/\*This searches through crew members who are copilots to find one that is available for this flight\*/**

**SELECT crew\_id INTO possibleCopilot**

**FROM Crew**

**WHERE role = 'Copilot'**

**EXCEPT**

**SELECT copilot\_id**

**FROM( SELECT copilot\_id, scheduled\_arrival**

**FROM FLight) as copilotTest**

**WHERE (scheduled\_arrival - possible\_departure\_time) < '1 day' AND (scheduled\_arrival - possible\_departure\_time) > '-1 day'**

**LIMIT 1;**

**/\*This searches through crew members who are cleaners to find one that is available for this flight\*/**

**SELECT crew\_id INTO possibleCleaner**

**FROM CREW**

**WHERE role = 'Cleaner'**

**EXCEPT**

**SELECT cleaner\_id**

**FROM( SELECT cleaner\_id, scheduled\_arrival**

**FROM Flight) as cleanerTest**

**WHERE (scheduled\_arrival - possible\_departure\_time) < '1 day' AND (scheduled\_arrival - possible\_departure\_time) > '-1 day'**

**LIMIT 1;**

**/\*This searches through all flights to ensure there is no other conflicting flights at the proposed gates at the given times\*/**

**SELECT COUNT(\*) into Taken\_Count**

**FROM (SELECT flight\_id**

**FROM Flight**

**WHERE (departure\_gate\_number = possible\_departure\_gate**

**AND ((possible\_departure\_time - scheduled\_departure) <= '1 hour' AND (possible\_departure\_time - scheduled\_departure) >= '-1 hour'))**

**OR (arrival\_gate\_number = possible\_arrival\_gate**

**AND ((possible\_arrival\_time - scheduled\_arrival) <= '1 hour' AND (possible\_arrival\_time - scheduled\_arrival) >= '-1 hour'))) AS invalid\_flight;**

**/\*Only insert the flight if all the checks were true\*/**

**IF Taken\_Count = 0 AND validOrder = TRUE AND validCheckIn = True THEN**

**INSERT INTO Flight**

**VALUES (currentFlightID, flightAircraft, possible\_departure\_time, possible\_arrival\_time, possible\_food, possible\_check\_in, possible\_arrival\_gate, possible\_departure\_gate, p\_status, NULL, NULL, possiblePilot, possibleCopilot, possibleCleaner);**

**END IF;**

**end;**

**$$;**

**COMMIT;**

* **Schedule cleaning, refueling, and maintenance for crew attendants and workers**

**/\*Assume an employee is signing up for a specific cleaning time, check and see if that time is not already taken...**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**given\_flight\_id VARCHAR(5) := 'AC512';**

**possible\_cleaning TIMESTAMP := '2021-01-09 12:40:20';**

**Taken\_Count INTEGER := -1;**

**validTime BOOLEAN := True;**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**beginTime TIMESTAMP := '1900-01-01 12:00:00';**

**endTime TIMESTAMP := '1900-01-01 12:00:00';**

**begin**

**/\*find the given flight’s scheduled\_departure\*/**

**SELECT scheduled\_departure INTO beginTime**

**FROM Flight**

**WHERE flight\_id = given\_flight\_id;**

**/\*find the given flight’s scheduled\_arrival\*/**

**SELECT scheduled\_arrival INTO endTime**

**FROM Flight**

**WHERE flight\_id = given\_flight\_id;**

**/\*check and ensure the given cleaning time is actually during the scheduled time of the flight\*/**

**IF (possible\_cleaning < beginTime) OR (possible\_cleaning > endTime) THEN validTime := False; END IF;**

**/\*count and ensure this is not too close to a previously inserted cleaning time\*/**

**SELECT COUNT(\*) INTO Taken\_Count**

**FROM (SELECT cleaning\_time, flight\_id**

**FROM Cleaning**

**WHERE (((possible\_cleaning - cleaning\_time) <= '30 minutes' AND (possible\_cleaning - cleaning\_time) >= '-30 minutes') AND given\_flight\_id = flight\_id)) AS T;**

**/\*if the following checks come out true, the insert the proposed cleaning shift\*/**

**IF Taken\_Count = 0 AND validTime = True THEN**

**INSERT INTO Cleaning**

**VALUES (possible\_cleaning, given\_flight\_id);**

**END IF;**

**end; $$;**

**COMMIT;**

**------------------------------------------------------------------------------------------------------------**

**This is a separate transaction for maintenance:**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**given\_aircraft\_code VARCHAR(4) := 'B421';**

**given\_maintenance\_time TIMESTAMP := '2021-09-01 3:00:00';**

**given\_worker\_id INTEGER := '123';**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**previous\_time TIMESTAMP := '1990-01-01 12:00:00';**

**next\_time TIMESTAMP := '2999-01-01 12:00:00';**

**TakenCount INTEGER := -1;**

**busyCounter INTEGER := -1;**

**begin**

**/\*this checks if the proposed airport worker is busy with another job\*/**

**SELECT COUNT(\*) INTO busyCounter**

**FROM Refueling**

**WHERE (given\_maintenance\_time - refuel\_time) <= '20 minutes' AND (given\_maintenance\_time - refuel\_time) >= '-20 minutes' AND worker\_id = given\_worker\_id;**

**/\*this finds the most recent flight with the given aircraft/\***

**SELECT MAX(scheduled\_arrival) INTO previous\_time**

**FROM( SELECT flight\_id, scheduled\_arrival**

**FROM Flight**

**WHERE scheduled\_arrival < given\_maintenance\_time) AS T1;**

**/\*this finds the next flight with this give aircraft/\***

**SELECT MIN(scheduled\_departure) INTO next\_time**

**FROM( SELECT flight\_id, scheduled\_departure**

**FROM Flight**

**WHERE scheduled\_departure > given\_maintenance\_time) AS T2;**

**/\*if either of these are NULL, put a placeholder for the tests to run effectively\*/**

**IF next\_time IS NULL**

**THEN next\_time := '2999-01-01 12:00:00';**

**END IF;**

**IF previous\_time IS NULL**

**THEN previous\_time := '1990-01-01 12:00:00';**

**END IF;**

**/\*check there is not already a maintenance job between the aircrafts’ flights\*/**

**SELECT COUNT(\*) INTO TakenCount**

**FROM Maintenance**

**WHERE maintenance\_time >= previous\_time AND maintenance\_time <= next\_time AND given\_aircraft\_code = aircraft\_code;**

**/\*insert the shift as long as the above tests succeed\*/**

**IF TakenCount = 0 AND busyCounter = 0**

**THEN INSERT INTO Maintenance**

**VALUES (given\_maintenance\_time,given\_worker\_id, given\_aircraft\_code);**

**END IF;**

**end; $$;**

**COMMIT;**

**------------------------------------------------------------------------------------------------------------**

**This is a separate transaction for refuelling:**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**given\_aircraft\_code VARCHAR(4) := 'B512';**

**given\_refuel\_time TIMESTAMP := '2021-09-01 3:00:00';**

**given\_worker\_id INTEGER := '123';**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**previous\_time TIMESTAMP := '1990-01-01 12:00:00';**

**next\_time TIMESTAMP := '2999-01-01 12:00:00';**

**TakenCount INTEGER := -1;**

**busyCounter INTEGER := -1;**

**begin**

**/\*check and ensure the given airport worker is not busy with another job\*/**

**SELECT COUNT(\*) INTO busyCounter**

**FROM Maintenance**

**WHERE (given\_refuel\_time - maintenance\_time) <= '20 minutes' AND (given\_refuel\_time - maintenance\_time) >= '-20 minutes' AND worker\_id = given\_worker\_id;**

**/\*find the previous flight with this aircraft\*/**

**SELECT MAX(scheduled\_arrival) INTO previous\_time**

**FROM( SELECT flight\_id, scheduled\_arrival**

**FROM Flight**

**WHERE scheduled\_arrival < given\_refuel\_time) AS T1;**

**/\*find the next flight with this aircraft\*/**

**SELECT MIN(scheduled\_departure) INTO next\_time**

**FROM( SELECT flight\_id, scheduled\_departure**

**FROM Flight**

**WHERE scheduled\_departure > given\_refuel\_time) AS T2;**

**/\*if either of these are NULL, put a placeholder for the tests to run effectively\*/**

**IF next\_time IS NULL**

**THEN next\_time := '2999-01-01 12:00:00';**

**END IF;**

**IF previous\_time IS NULL**

**THEN previous\_time := '1990-01-01 12:00:00';**

**END IF;**

**/\*check there is not already a refuel job between the aircrafts’ flights\*/**

**SELECT COUNT(\*) INTO TakenCount**

**FROM Refueling**

**WHERE refuel\_time >= previous\_time AND refuel\_time <= next\_time AND given\_aircraft\_code = aircraft\_code;**

**/\*insert the shift as long as the above tests succeed\*/**

**IF TakenCount = 0 AND busyCounter = 0**

**THEN INSERT INTO Refueling**

**VALUES(given\_refuel\_time, given\_worker\_id, given\_aircraft\_code);**

**END IF;**

**end; $$;**

**COMMIT;**

* **Check in customers when they arrive at their flight**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**given\_ticket\_no INTEGER := 100001;**

**given\_timestamp TIMESTAMP := '2020-08-09 12:00:00';**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**begin**

**/\*update the check\_in time based on the given timestamp\*/**

**UPDATE Tickets SET check\_in = given\_timestamp WHERE ticket\_no = given\_ticket\_no;**

**end; $$;**

**COMMIT;**

* **Manage baggage for each customer**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**given\_baggage\_id INTEGER := 399214;**

**given\_passenger\_id VARCHAR(6) := 'HCF99K';**

**check\_in\_or\_out BOOLEAN := 'False'; /\*false means check out, true means check in\*/**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**TakenCount INTEGER := -1;**

**Begin**

**/\*check the amount of baggage present that matches the ID given; this will only be ever 0 or 1\*/**

**SELECT COUNT(\*) INTO TakenCount**

**FROM (SELECT baggage\_id**

**FROM Baggage**

**WHERE (baggage\_id = given\_baggage\_id)) AS invalid\_baggage;**

**/\*if there is no matching bag and the user wants to insert, then insert and if**

**if there is a matching bag and the user wants to remove, then remove\*/**

**IF TakenCount = 0 AND check\_in\_or\_out = True**

**THEN INSERT INTO Baggage**

**VALUES(given\_baggage\_id, given\_passenger\_id);**

**ELSIF TakenCount = 1 AND check\_in\_or\_out = False**

**THEN DELETE FROM Baggage WHERE given\_baggage\_id = baggage\_id;**

**END IF;**

**end; $$;**

**COMMIT;**

* **Record actual takeoff and arrival times**

**Departure:**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**given\_flight\_id VARCHAR(5) := 'AT128';**

**given\_timestamp TIMESTAMP := '2021-08-18 14:00:00';**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**begin**

**/\*assuming the times are NULL, then update the flight with the actual departure\*/**

**UPDATE Flight SET actual\_departure = given\_timestamp WHERE flight\_id = given\_flight\_id AND actual\_departure IS NULL;**

**end; $$;**

**COMMIT;**

**---------------------------------------------------------------------------------------------------------------------**

**Arrival:**

**BEGIN;**

**SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**

**do $$**

**DECLARE**

**given\_flight\_id VARCHAR(5) := 'AT128';**

**given\_timestamp TIMESTAMP := '2021-08-18 16:17:31';**

**/\*the above variables should be inserted by the website user, these are hardcoded in for readability and debugging in SQL\*/**

**begin**

**/\*assuming the times are NULL, then update the flight with the actual arrival\*/**

**UPDATE Flight SET actual\_arrival = given\_timestamp WHERE flight\_id = given\_flight\_id AND actual\_arrival IS NULL;**

**end; $$;**

**COMMIT;**

* **Filter whether aircrafts have entertainment**

**/\*$1 refers to the the users’ preference on movie (true if movies must be present) and $2 refers to the users’ preference of wifi (true if wifi must be present)\*/**

**SELECT \* FROM Aircraft JOIN Aircraft\_Model ON Aircraft\_Model.aircraft\_model = Aircraft.Aircraft\_Model**

**WHERE (($1 = true AND wifi = true) OR $1 = false) AND (($2 = true AND movie = true) OR $2 = false);**

There are also queries used to check the results of the transactions above. These test for the primary key (or what was changed in the case of the UPDATE transactions). These include the following checks:

* **new\_flight\_transaction\_q**

**/\*currentFlightID is the value given by the user; this should result with 1\*/**

**SELECT COUNT (\*)**

**FROM Flight**

**WHERE currentFlightID = flight\_id**

* **assign\_clean\_shift\_q**

**/\*possible\_cleaning and given\_flight\_id are values given by the user; this should result with 1\*/**

**SELECT COUNT(\*)**

**FROM Cleaning**

**WHERE possible\_cleaning = cleaning\_time AND given\_flight\_id = flight\_id**

* **assign\_maintenance\_shift\_q**

**/\*given\_maintenance\_time and given\_aircraft\_code are values given by the user; this should result with 1\*/**

**SELECT COUNT(\*)**

**FROM Maintenance**

**WHERE given\_maintenance\_time = maintenance\_time AND given\_aircraft\_code = aircraft\_code**

* **assign\_refuel\_shift\_q**

**/\*given\_refuel\_time and given\_aircraft\_code are given by the user; this should result with 1\*/**

**SELECT COUNT(\*)**

**FROM Refueling**

**WHERE given\_refuel\_time = refuel\_time AND given\_aircraft\_code = aircraft\_code**

* **baggage\_manager\_q**

**/\*given\_baggage\_id is the value inserted by the user; this should result with 1 if checked in baggage, and 0 if checked out baggage\*/**

**SELECT COUNT(\*)**

**FROM Baggage**

**WHERE given\_baggage\_id = baggage\_id**

* **ticket\_checkin\_q**

**/\*given\_ticket\_no and given\_timestamp are values inserted by the user; this should result with 1\*/**

**SELECT COUNT (\*)**

**FROM Tickets**

**WHERE given\_ticket\_no = ticket\_no AND given\_timestamp= check\_in**

* **record\_departure\_q**

**/\*given\_flight\_id and given\_timestamp are values inserted by the user; this should result with 1\*/**

**SELECT COUNT(\*)**

**FROM Flight**

**WHERE given\_flight\_id= flight\_id AND given\_timestamp = actual\_departure**

* **record\_arrival\_q**

**/\*given\_flight\_id and given\_timestamp are values inserted by the user; this should result with 1\*/**

**SELECT COUNT(\*)**

**FROM Flight**

**WHERE given\_flight\_id = flight\_id AND actual\_arrival = given\_timestamp**

Demo video

<https://www.dropbox.com/sh/267dlln90er7c0y/AACYNLHEVHOh9Hm_S4PEbnvCa?dl=0>