# 7. Appendix

**Appendix A** - Schema documentation - Entities, Attributes, and Keys

| **Entity** | **Attributes & Descriptions** | **Primary Key (PK)** | **Foreign Key (FK)** |
|---|---|---|---|
| Flight | - Flight_ID: Unique flight identifier.<br>- Route_ID: Route the flight follows.<br>- Aircraft_ID: Aircraft used.<br>- Departure_Time: Scheduled time (HH:MM).<br>- Day_of_Departure: Day of operation (e.g., Monday).<br>- Seats_Booked: Number of seats booked (updated after booking).<br>- Number_of_Crew: Total crew assigned. | Flight_ID | Route_ID → Route(Route_ID),<br>Aircraft_ID → Aircraft(Aircraft_ID) |
| Airport | - Airport_ID: Unique airport identifier.<br>- Airport_Name: Official airport name.<br>- City: Airport's city.<br>- Country: Airport's country. | Airport_ID | - |
| Route | - Route_ID: Unique route identifier.<br>- Arrival_Airport_ID: Destination airport.<br>- Flight_Duration: Duration in hours. | Route_ID | Arrival_Airport_ID → Airport(Airport_ID) |

| | | | |
|---|---|---|---|
| Aircraft | - Aircraft_ID: Unique aircraft identifier.<br>- Model_Aircraft: Aircraft model (e.g., Boeing 737).<br>- Monthly_Quota: Max allowed monthly flight hours.<br>- Monthly_Limit: Maintenance limit hours.<br>- Seat_Capacity: Total seating capacity. | Aircraft_ID | - |
| Transaction | Transaction_ID: Unique booking identifier.<br>- Flight_ID: Associated flight.<br>- Payment_Amount: Amount paid by passenger.<br>- Seat_Category: Class (Economy, Business, First). | Transaction_ID | Flight_ID → Flight(Flight_ID) |
| Cost | - Cost_ID: Unique cost record identifier.<br>- Flight_ID: Associated flight.<br>- Fuel_Cost: Fuel expense.<br>- Crew_Salary: Total crew salary.<br>- Turnaround_Cost: Ground turnaround expenses. | Cost_ID | Flight_ID → Flight(Flight_ID) |

**Appendix B** - Code for Schema implementation.

```
import sqlite3

#This statement creates a connection labelled as conn.  This will be used throughout to ensure the consistency for when we start to query the database tables.
conn = sqlite3.connect('airline.db')
cursor = conn.cursor()
```

```
#ensuring that foreign keys are on for referential integrity
cursor.execute("PRAGMA foreign_keys = ON")

# Creating aircraft table
cursor.execute('''
CREATE TABLE aircrafts (
    aircraft_id VARCHAR(32) PRIMARY KEY NOT NULL,
    aircraft_model TEXT,
    aircraft_quota REAL,
    aircraft_limit REAL,
    seats_capacity INT
);
''')

# Creating airports table
cursor.execute('''
CREATE TABLE airports (
    airport_id VARCHAR(32) PRIMARY KEY NOT NULL,
    airport_name TEXT,
    airport_city TEXT,
    airport_country TEXT
);
''')

# Creating routes table
cursor.execute('''
CREATE TABLE routes (
    route_id VARCHAR(32) PRIMARY KEY NOT NULL,
    arrival_airport VARCHAR(32),
    flight_duration REAL CHECK (flight_duration > 0),
    FOREIGN KEY (arrival_airport) REFERENCES airports(airport_id)
);
''')

# Creating flights table
cursor.execute('''
```

```
CREATE TABLE flights (
    flight_id VARCHAR(32) PRIMARY KEY NOT NULL,
    route_id VARCHAR(32),
    departure_date DATETIME,
    departure_day TEXT,
    aircraft_id VARCHAR(32),
    number_of_crew INT,
    seats_booked INT,
    FOREIGN KEY (route_id) REFERENCES routes(route_id),
    FOREIGN KEY (aircraft_id) REFERENCES aircrafts(aircraft_id)
);
''')

# Creating costs table
cursor.execute('''
CREATE TABLE costs (
    cost_id VARCHAR(32) PRIMARY KEY NOT NULL,
    flight_id VARCHAR(32) UNIQUE,
    fuel_cost REAL,
    crew_salary REAL,
    turnaround_cost REAL,
    FOREIGN KEY (flight_id) REFERENCES flights(flight_id) ON DELETE SET NULL
);
''')

# Creating transactions table
cursor.execute('''
CREATE TABLE transactions (
    transaction_id VARCHAR(32) PRIMARY KEY NOT NULL,
    flight_id VARCHAR(32),
    payment_amount REAL CHECK (payment_amount > 0),
    seat_category TEXT,
    FOREIGN KEY (flight_id) REFERENCES flights(flight_id) ON DELETE SET NULL
);
''')
```

```
#This saves the chnages to the databae.  Up unitl this point the executed
SQL statement isn't stored, changes are not immediatley saved.
conn.commit()

print("Database and tables created successfully!")
```

```
# Checking tables and their columns
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

for table_name in tables:
    print(f"Table: {table_name[0]}")
    cursor.execute(f"PRAGMA table_info({table_name[0]});")
    columns = cursor.fetchall()
    for col in columns:
        print(f"   Column: {col[1]}, Type: {col[2]}, NotNull: {col[3]},
DefaultVal: {col[4]}, PrimaryKey: {col[5]}")
    print("-" * 20)
```

**Appendix C** - Code for Import Data and Data Validation

```
# Upload CSV files
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

```
# Function to import CSV into a table
import csv

def import_csv_to_table(csv_file, table_name):
    #opens the file aas read only 'r', doesn't allow the origianl csv to be
changed.
```

```
    with open(csv_file, 'r', encoding='utf-8') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)  # Skip header row if present
        for row in csv_reader:
            #? creates a placeholder for each column in the CSV file.
['?','?','?'] - Join makes it a string so it can then be inserted.
            # use of the '?' reduce risk of SQL injection
            placeholders = ', '.join(['?' for _ in row])
            #Assumes that the CSV and table have the same structure (this
could be an issue) Would have to specify column names if different.
            sql = f"INSERT INTO {table_name} VALUES ({placeholders})"
            cursor.execute(sql, row)

# Import data from CSV files into the relevant table.The import_csv_to_table
is the function, passing the two values across.
try:
    import_csv_to_table('airporttable.csv', 'airports')
    import_csv_to_table('AircraftTable.csv', 'aircrafts')
    import_csv_to_table('finalroutetable.csv', 'routes')
    import_csv_to_table('finalflighttable(1).csv', 'flights')
    import_csv_to_table('finalcosttable.csv', 'costs')
    import_csv_to_table('finalfinaltransactiontable.csv', 'transactions')
    conn.commit()
    print("Data imported successfully!")
except Exception as e:
    print(f"An error occurred: {e}")
    conn.rollback()  # Rollback changes if an error occurred
```

```
import pandas as pd
# Query all six tables and load into pandas DataFrames
flights_df = pd.read_sql_query("SELECT * FROM flights", conn)
aircrafts_df = pd.read_sql_query("SELECT * FROM aircrafts", conn)
airports_df=pd.read_sql_query("SELECT * FROM airports",conn)
routes_df=pd.read_sql_query("SELECT * FROM routes",conn)
transactions_df=pd.read_sql_query("SELECT * FROM transactions",conn)
costs_df=pd.read_sql_query("SELECT * FROM costs",conn)
```

```
# Show the first 10 lines of each DataFrame
print("\nFlights Table:")
print(flights_df.head(10))
print("\nAircrafts Table:")
print(aircrafts_df.head(10))
print("\nAirports Table:")
print(airports_df.head(10))
print("\nRoutes Table:")
print(routes_df.head(10))
print("\nTransactions Table:")
print(transactions_df.head(10))
print("\nCosts Table:")
print(costs_df.head(10))
```

```
#checking the quality of the data by ensuring no null values
print(flights_df.isnull().sum())
print(airports_df.isnull().sum())
print(aircrafts_df.isnull().sum())
print(routes_df.isnull().sum())
print(transactions_df.isnull().sum())
print(costs_df.isnull().sum())
#no null values in any of the tables
```

```
#some basic statistics
totalrevenue_df=pd.read_sql_query("SELECT      SUM(payment_amount)      AS
totalrevenue FROM transactions", conn)
print(totalrevenue_df)
totalcost_df=pd.read_sql_query("SELECT
SUM(fuel_cost+crew_salary+turnaround_cost) AS totalcost FROM costs",conn)
print(totalcost_df)
totalflights_df=pd.read_sql_query("SELECT  COUNT(*)  AS  number_of_flights
FROM flights",conn)
print(totalflights_df)
```

```
totalaircrafts_df=pd.read_sql_query("SELECT COUNT(*) AS number_of_aircraft
FROM aircrafts",conn)
print(totalaircrafts_df)
totalroutes_df=pd.read_sql_query("SELECT COUNT(*) AS number_of_routes FROM
routes",conn)
print(totalroutes_df)
totaltransaction_df=pd.read_sql_query("SELECT          COUNT(*)          AS
number_of_transactions FROM transactions",conn)
print(totaltransaction_df)
#total revenue is about 78million, with costs at about 22 million over
138000 transactions
#total 40 aircraft are used over 20 routes, with 500 flights flown over the
month
```

**Appendix D** – SQL queries for Insights

**In-demand route**

```
#next we find the highest booked flights, representing demand

flightsdemand_df = pd.read_sql_query("SELECT SUM(seats_booked), route_id
FROM flights GROUP BY route_id ORDER BY SUM(seats_booked) DESC", conn)
print(flightsdemand_df)
#getting the city names so that we can identify the cities that the top
route_ids represent
citynames_df=pd.read_sql_query("SELECT    route_id,    airport_city    FROM
airports,routes WHERE airports.airport_id=routes.arrival_airport",conn)
print(citynames_df)
#5 highest booked routes are for Tokyo, Barcelona, Paris, Turkey and Rome
in order.
```

**Seat Occupancy**

```
#now seeing the fillrate ie how much proportion of the capacity is taken up. We multiply one of the values by 1.0 to make it a float such that there is no integer division which will net us 0 for all routes.
fillrate_df = pd.read_sql_query("SELECT route_id, SUM(seats_booked)*1.0/SUM(seats_capacity) AS FillRate FROM flights,aircrafts WHERE flights.aircraft_id = aircrafts.aircraft_id GROUP BY route_id ORDER BY FillRate DESC", conn)
print(fillrate_df)
#Paris has the highest fill rate (indicating highest demand) followed by Istanbul and Philadelphia
```

**Route Revenue**

```
#next trying to find the revenue generated per route
routerevenue_df = pd.read_sql_query("SELECT route_id, SUM(payment_amount) AS TotalPayment FROM (SELECT f.route_ID, t.payment_amount FROM transactions t, flights f, routes r WHERE t.flight_id = f.flight_id AND f.route_id = r.route_id) AS SubQuery GROUP BY route_id ORDER BY TotalPayment DESC", conn)
print(routerevenue_df)
#obtaining the average revenue per flight on the route too
averagerouterevenue_df = pd.read_sql_query("SELECT route_id, SUM(payment_amount)/COUNT(DISTINCT flight_id) AS AverageRevenue FROM (SELECT f.flight_id, f.route_id, t.payment_amount FROM transactions t, flights f, routes r WHERE t.flight_id = f.flight_id AND f.route_id = r.route_id) AS SubQuery GROUP BY route_id ORDER BY AverageRevenue DESC", conn)
print(averagerouterevenue_df)
#flights to tokyo had the highest revenue, and also had the highest average revenue as well
```

**Peak Travel Day**

```
#first is to get the peak day of travel. We use seats booked for this
flightsdemandbyday_df    =    pd.read_sql_query("SELECT    SUM(seats_booked),
departure_day FROM flights GROUP BY departure_day ORDER BY SUM(seats_booked)
DESC", conn)
print(flightsdemandbyday_df)
#we see that wednesdays and thursdays have the most booked, followed by
tuesday
```

**Underutilised aircraft**

```
#checking for underutilised aircraft
aircraftutility_df                =                pd.read_sql_query("SELECT
COUNT(aircraft_id),aircraft_id FROM flights GROUP BY aircraft_id ORDER BY
COUNT(aircraft_id) ASC",conn)
print(aircraftutility_df)
#we see that there are two aircraft that only fly once in the month, checking
to see more about them
```

```
underutilised_df  =  pd.read_sql_query("SELECT  *  FROM  aircrafts  WHERE
aircraft_id='20' OR aircraft_id='28'",conn)
print(underutilised_df)
#we see that one is an A380, suited for long haul flights whereas the other
is an A320 which is suited for shorter flights. (business knowledge/context)
#we should put the a380 to japan since we found that the route to japan
generates the most revenue
#and we should put the a320 for paris since highest it has the highest
demand and fill rate
```

```
#trying to get the estimate the cost of the new flights by getting the
average of such flights from the data
tokyoA380cost_df                =                pd.read_sql_query("SELECT
SUM(turnaround_cost+crew_salary+fuel_cost)/COUNT(*)  AS  Tokyo_Average_Cost
FROM  (SELECT  *  FROM  costs  c,  flights  f,  aircrafts  a  WHERE  c.flight_id  =
f.flight_id    AND    f.route_id    =    '01JP5JPNR61PTYNAQQTCKMFKD4'    AND
```

```
f.aircraft_id=a.aircraft_id AND a.aircraft_model='A380') AS SubQuery",
conn)
print(tokyoA380cost_df)
#average cost of flying an A380 to tokyo is about 130000
parisA320cost_df = pd.read_sql_query("SELECT
SUM(turnaround_cost+crew_salary+fuel_cost)/COUNT(*) AS Paris_Average_Cost
FROM (SELECT * FROM costs c, flights f, aircrafts a WHERE c.flight_id =
f.flight_id AND f.route_id = '01JP5JPNRCP9MJ5RAHSVX45KQY' AND
f.aircraft_id=a.aircraft_id AND a.aircraft_model='A320') AS SubQuery",
conn)
print(parisA320cost_df)
#average cost of flying an A320 to paris is about 5500
```