CSCI 620 Project Phase 2

Group 33

Flight Delay

Shlok Gupta, Suraj Sureshkumar, Vidit Ketan Kothari

Table of Contents

1. Schema
2. Queries
3. Functional Dependencies
4. Normalization

Schema

```
Flights :
{
	flight_number : _,
	airline:{
		name : _,
		alias : _,
		iata_code : _,
		icao_code:_,
		callsign : _,
		country:_,
	},
	source_airport:{
	name : _,
		  iata_code : _,
		 icao_code : _,
		 city: _,
		 country :  _,
		altitude : _,
		timezone : _,
		latitude : _,
		longitude : _,
		DST_zone : _
	},
	destination_airport:{
	name : _,
		  iata_code : _,
		 icao_code : _,
		 city: _,
		 country :  _,
		altitude : _,
		timezone : _,
		latitude : _,
		longitude : _,
		DST_zone : _
```

```
    },
    aircraft : {
        name : _,
        iata_code : _,
        icao_code : _
    },
    stops : _,
    codeshare : _,
}

Delays :
{
    flight_date : _,
    flight_number : _,
    origin : _,
    destination : _,
    scheduled_departure : _,
    actual_departure : _,
    departure_delay : _,
    scheduled_arrival : _,
    actual_arrival : _,
    arrival_delay : _
}
```

Comparison

The document-oriented model and the relational model use two different approaches for handling and storing data in a dataset. Here are some key differences between the two models we observed for our dataset:

The schema for a relational model needs to be defined beforehand and create according structures to insert data. This was not required on the document-oriented model.

In our relational model, relationships between tables are matched through the presence of foreign keys that link one table to another. On the other hand, we were able to create sub-documents within each document therefore not requiring the presence of an extra foreign key.

In a relational model, our data is structured into different tables that consist of rows and columns where we might require multiple joins to get a holistic data. In contrast, our document-oriented model stores data as documents and sub-documents therefore requiring less tabular structures and reducing joins.

In the relational model, we needed different tables such as aircrafts, airline, airport, routes, delays, countries and also had the necessity of few mapping tables for many-to-many relationships. This was not the case for the document-oriented model as we created just 2 documents, flights and delays.

As a side note, requirements for functional dependencies and normalization were also eliminated on the document-oriented end.

We also found that the document-oriented model due to its flexibility can be error prone where some incorrect data was also accepted and had to be cleaned separately whereas the relational model eliminated such rows.

Program:

We used Python to perform cleaning on the data and used the interface provided by MongoDBCompass to import the comma separated value files directly into collections.

Python cleaning script:

```
import pandas as pd

routes = pd.read_csv(r"Data\routes.dat",
                     names=['airline', 'airline_id', 'src_airport',
                            'src_airport_id', 'dest_airport',
                            'dest_airport_id', 'codeshare', 'stops',
                            'equipment'])
airlines = pd.read_csv(r"Data\airlines.dat",
                       names=['airline_id', 'airline.name', 'airline.alias',
                              'airline.iata_code', 'airline.icao_code',
                              'airline.callsign', 'airline.country',
                              'airline.active'])

src_airport = pd.read_csv(r"Data\airports.dat",
                          names=['source_airport.airport_id',
                                 'source_airport.name',
                                 'source_airport.city',
                                 'source_airport.country',
                                 'source_airport.iata_code',
                                 'source_airport.icao_code',
                                 'source_airport.latitude',
                                 'source_airport.longitude',
                                 'source_airport.altitude',
                                 'source_airport.timezone',
                                 'source_airport.DST_zone',
                                 'source_airport.tz', 'source_airport.type',
                                 'source_airport.source'])

dest_airport = pd.read_csv(r"Data\airports.dat",
                           names=['destination_airport.airport_id',
                                  'destination_airport.name',
                                  'destination_airport.city',
                                  'destination_airport.country',
                                  'destination_airport.iata_code',
                                  'destination_airport.icao_code',
                                  'destination_airport.latitude',
                                  'destination_airport.longitude',
                                  'destination_airport.altitude',
                                  'destination_airport.timezone',
                                  'destination_airport.DST_zone',
                                  'destination_airport.tz',
```

```
                                 'destination_airport.type',
                                 'destination_airport.source'])

planes = pd.read_csv(r"Data\planes.dat",
                     names=['aircraft.name', 'aircraft.iata_code',
                            'aircraft.icao_code'])
temp1 = pd.merge(routes, airlines, how="inner", left_on="airline",
                 right_on="airline.iata_code",
                 left_index=False, right_index=False)
temp2 = pd.merge(routes, airlines, how="inner", left_on="airline",
                 right_on="airline.icao_code",
                 left_index=False, right_index=False)
routes_airline = pd.concat([temp1, temp2], ignore_index=True)
routes_airline.drop_duplicates(inplace=True, keep='first')
temp1 = pd.merge(routes_airline, src_airport, how="inner",
                 left_on='src_airport',
                 right_on='source_airport.iata_code',
                 left_index=False, right_index=False)
temp2 = pd.merge(routes_airline, src_airport, how="inner",
                 left_on='src_airport',
                 right_on='source_airport.icao_code',
                 left_index=False, right_index=False)

routes_airline_src = pd.concat([temp1, temp2], ignore_index=True)
routes_airline_src.drop_duplicates(inplace=True, keep='first')


temp1 = pd.merge(routes_airline_src, dest_airport, how="inner",
                 left_on='dest_airport',
                 right_on='destination_airport.iata_code',
                 left_index=False, right_index=False)
temp2 = pd.merge(routes_airline, dest_airport, how="inner",
                 left_on='dest_airport',
                 right_on='destination_airport.icao_code',
                 left_index=False, right_index=False)

routes_airline_src_dest = pd.concat([temp1, temp2], ignore_index=True)
routes_airline_src_dest.drop_duplicates(inplace=True, keep='first')

routes_airline_src_dest['equipment'] = \
       routes_airline_src_dest['equipment'].apply(
           lambda x: x.split(' ') if type(x) == str else x)
routes_airline_src_dest = routes_airline_src_dest.explode('equipment',
                                                           ignore_index=True)


r_a_s_d_a = pd.merge(routes_airline_src_dest, planes,
                     how="inner", left_on='equipment',
                     right_on='aircraft.iata_code',
```

```
                    left_index=False, right_index=False)
r_a_s_d_a.drop(columns=['airline_id_x', 'src_airport_id',
                        'dest_airport_id', 'equipment',
                        'airline_id_y', 'airline.active',
                        'source_airport.airport_id', 'source_airport.tz',
                        'source_airport.type', 'source_airport.source',
                        'destination_airport.airport_id',
                        'destination_airport.tz',
                        'destination_airport.type',
                        'destination_airport.source'],
              inplace=True)
delays = pd.read_csv(r"Data\delays.csv", header=0,
                    names=['index', 'flight_date', 'airline_code',
                           'flight_number', 'source',
                           'destination', 'scheduled_departure',
                           'actual_departure', 'departure_delay',
                           'TAXI_OUT', 'WHEELS_OFF', 'WHEELS_ON', 'TAXI_IN',
                           'scheduled_arrival', 'actual_arrival',
                           'arrival_delay', 'CANCELLED',
                           'CANCELLATION_CODE', 'DIVERTED',
                           'CRS_ELAPSED_TIME', 'ACTUAL_ELAPSED_TIME',
                           'AIR_TIME', 'DISTANCE', 'CARRIER_DELAY',
                           'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY',
                           'LATE_AIRCRAFT_DELAY', 'Unnamed: 27'],
                    usecols=['airline_code', 'flight_number', 'source',
                             'destination'])

routes_delay = pd.merge(r_a_s_d_a, delays,
                       how="inner",
                       left_on=["src_airport", "dest_airport", "airline"],
                       right_on=["source", "destination", "airline_code"],
                       left_index=False, right_index=False)

routes_delay.drop(columns=["src_airport", "dest_airport", "airline",
                          'airline_code', 'source', 'destination'],
                 inplace=True)
routes_delay.drop_duplicates(inplace=True)
routes_delay.to_csv('final.csv', index=False)
```

Output:

Example output:


```
{
  "_id": {
    "$oid": "64278b1895be47bb39a28d57"
  },
  "stops": 0,
  "airline": {
    "name": "United Airlines",
    "alias": "\\N",
    "iata_code": "UA",
    "icao_code": "UAL",
    "callsign": "UNITED",
    "country": "United States"
  },
  "source_airport": {
    "name": "Chicago O'Hare International Airport",
    "city": "Chicago",
    "country": "United States",
    "iata_code": "ORD",
    "icao_code": "KORD",
    "latitude": 41.9786,
    "longitude": -87.9048,
    "altitude": 672,
    "timezone": "-6",
    "DST_zone": "A"
  },
  "destination_airport": {
    "name": "George Bush Intercontinental Houston Airport",
    "city": "Houston",
    "country": "United States",
    "iata_code": "IAH",
    "icao_code": "KIAH",
    "latitude": 29.984399795532227,
    "longitude": -95.34140014648438,
    "altitude": 97,
    "timezone": "-6",
    "DST_zone": "A"
  },
  "aircraft": {
    "name": "Airbus A319",
    "iata_code": "319",
    "icao_code": "A319"
  },
  "flight_number": 211
}
```

```
_id: ObjectId('64278b1895be47bb39a28d57')
stops: 0
▸ airline: Object
▸ source_airport: Object
▸ destination_airport: Object
▸ aircraft: Object
flight_number: 211
```

```
_id: ObjectId('64278b1895be47bb39a28d58')
stops: 0
▸ airline: Object
▸ source_airport: Object
▸ destination_airport: Object
▸ aircraft: Object
flight_number: 703
```

```
_id: ObjectId('64278b1895be47bb39a28d59')
stops: 0
▸ airline: Object
▸ source_airport: Object
▸ destination_airport: Object
▸ aircraft: Object
```

Queries:

1.Top 10 airports with the highest number of flights starting from that airport.

Query:

```
select count(al.airline_id),a.airport_name
from "Project".flights."Airport" a, "Project".flights.airlines al,
"Project".flights.routes r, "Project".flights.delays d
where al.airline_id = r.id
and a.airport_id = r.source_id
and r.source =d.source
and r.destination = d.destination
and r.airline = d.airline_code
group by a.airport_id
order by count(al.airline_id) DESC;
```

The time without indexing to execute the query is **2 mins 11sec**

The time with indexing to execute the query is **1 min 3 secs**

Output:

	count	airport_name
1	915	Hartsfield Jackson Atlanta International Airport
2	558	Chicago O'Hare International Airport
3	535	Beijing Capital International Airport
4	527	London Heathrow Airport
5	524	Charles de Gaulle International Airport
6	497	Frankfurt am Main Airport
7	492	Los Angeles International Airport
8	469	Dallas Fort Worth International Airport
9	456	John F Kennedy International Airport
10	453	Amsterdam Airport Schiphol

2.Top 10 pairs of airports which had the largest number of flights between them.
Query:

```
select count(al.airline_id), r.source, r.destination
from "Project".flights.airlines al, "Project".flights.routes r, "Project".flights.delays d
where al.airline_id = r.id
and r.source = d.source
and r.destination = d.destination
and r.airline = d.airline_code
group by r.source, r.destination
order by count(al.airline_id) DESC
limit 10;
```

The time without indexing to execute the query is **2 mins 55 sec**
The time with indexing to execute the query is **59 sec**
Output:

	count	source	destination
1	20	ORD	ATL
2	19	ATL	ORD
3	13	HKT	BKK
4	13	ORD	MSY
5	12	DOH	BAH
6	12	AUH	MCT
7	12	HKG	BKK
8	12	ATL	MIA
9	12	MIA	ATL
10	12	JFK	LHR

3.Top 10 airports with the most departure delays in minutes

Query:

```
select count(*), sum(d.departure_delay), a.airport_name
from "Project".flights."Airport" a, "Project".flights.delays d,
"Project".flights.routes r
where r.source = d.source
and r.destination = d.destination
and a.airport_id = r.source_id
and r.airline = d.airline_code
group by a.airport_name
order by count(*), sum(d.departure_delay)
limit 10;
```

The time without indexing to execute to the query is **2 mins 39sec**

The time with indexing to execute the query is **1 min 19 sec**

Output:

	count	sum	airport_name
1	1	-9	Saipan International Airport
2	6	148	Kalamazoo Battle Creek International Airport
3	9	7	Montgomery Regional (Dannelly Field) Airport
4	12	-69	Idaho Falls Regional Airport
5	26	-22	MBS International Airport
6	30	37	Gunnison Crested Butte Regional Airport
7	46	403	Montrose Regional Airport
8	48	727	South Bend Regional Airport
9	86	994	Appleton International Airport
10	86	300	Mahlon Sweet Field

4.Average delay time for each airline in minutes

Query:

```
select avg(d.departure_delay), al.airline_name
from "Project".flights.airlines al, "Project".flights.delays d,
"Project".flights."Airport" a, "Project".flights.routes r
where r.source = d.source
  and r.destination = d.destination
  and al.airline_id = r.id
  and a.airport_id = r.source_id
group by al.airline_name;
```

The time without indexing to execute to the query is **2 mins 24sec**

The time with indexing to execute the query is **1 min 2 sec**

Output:

1	12.6295054678288211	Aer Lingus
2	11.4380211749273869	Aeroflot Russian Airlines
3	8.1631501946654414	AeroMéxico
4	8.2160694247026259	Air Canada
5	9.8422413286590023	Air China
6	8.8715893729170705	Air France
7	9.0025175081246853	Air New Zealand
8	8.5976677667766777	AirTran Airways
9	7.1158575012946562	Alaska Airlines
10	1.06	Alaska Seaplane Service

5.Top 10 routes with the most delays

Query

```
select count(*), r.source, r.destination
from "Project".flights.delays d, "Project".flights.routes r
where d.source = r.source
and d.destination = r.destination
and r.airline = d.airline_code
group by  r.source, r.destination
order by count(*) DESC
limit 10;
```

The time without indexing to execute to the query is **2 min 13sec**
The time with indexing to execute the query is **1 min 24 sec**

Output:

	count	source	destination
1	62981	SFO	LAX
2	62577	LAX	SFO
3	49859	JFK	LAX
4	49833	LAX	JFK
5	49487	LGA	ORD
6	48526	OGG	HNL
7	48425	ORD	LGA
8	47304	HNL	OGG
9	45866	ATL	MCO
10	45855	MCO	ATL

Functional Dependencies:

Airline

Airline_id is the primary key where all the other columns are dependent on the primary key and uniquely identifies other columns.

Airline_id → name, alias, IATA_code, ICAO_code, callsign
ICAO_code → airline_name
IATA_code → airline_name
Airline_name → IATA_code, ICAO_code
allsign → airline_name

Airport

Airport_id is the primary key where all the other columns are dependent on the primary key and where other columns are uniquely identified by this key.

Airport_id → airport_name, city, country, IATA_code, ICAO_code, latitude, longitude, altitude, timezone, DST_zone
Latitude, longitude, altitude → IATA_code, ICAO_code

Country

Country_id is the primary key where all the other columns are dependent on the primary key where each column is uniquely identified by this. ISO_code uniquely identifies country_name.

Country_id → ISO_code
ISO_code → country_name

Aircraft

id is the primary key where all the other columns are uniquely identified by this and IATA_code and ICAO_code uniquely identify the aircraft_name.

Id → aircraft_name, IATA_code, ICAO_code
IATA_code → aircraft_name
ICAO_code → aircraft_name

Routes

Airline,source,destination is the composite key for this table and all the other columns are uniquely identified by this composite key.

airline, source, destination → stops
source_id → source
source → source_id
destination → destination_id
destination_id → destination
Airline,source, destination → equipment

Delays

Airline_code, flight_number, flight_date is the composite for this table and all the other columns are uniquely identified by this composite key.

Airline_code, flight_number → source, destination
Airline_code, flight_number, flight_date→ scheduled_departure, actual_departure, departure_delay, destination, scheduled_arrival, actual_arrival, arrival_delay

Normalization

1NF: Our data is in first normalized form as we have atomicity in each cell i.e. we have no cell with relations as elements or we don't have cells with multiple values in them.

2NF: Our data is in second normalized form as it is in 1NF and all the attributes in a table are functionally dependent on the composite primary key within the table.

3NF, 4NF, 5NF: Our data is not in third normalized form as there are functional dependencies between 2 non key attributes. Since it is not in 3NF, it is also not in 4NF and 5NF.