FAMOUS Salami

salamifamous@gmail.com

In [1]:
```
import psycopg2
import pandas as pd
```

Defining PostgreSQL Connection class

In [2]:
```
class PostgreSQLConnection:
    def __init__(self, host, dbname, user, password):
        self.host = host
        self.dbname = dbname
        self.user = user
        self.password = password
        self.connection = None

    def connect(self):
        try:
            if self.connection:
                return ('Connection already established')
            else:
                self.connection = psycopg2.connect(
                    host=self.host,
                    dbname=self.dbname,
                    user=self.user,
                    password=self.password
                )
                if self.connection:
                    cursor_ = self.connection.cursor()
                    return cursor_, self.connection
                #print('Connected to PostgreSQL')
        except Exception as e:
            print(f'Error {e} occurred')

    def close(self):
        if self.cursor:
            self.cursor.close()
        if self.connection:
            self.connection.close()
            return ('PostgreSQL connection closed')

    def create_cursor(self):
        if self.connection:
            cursor_ = self.connection.cursor()
            if dbcursor:
                print ('Cursor creation succeeded.')
                return cursor_
            else:
                return None

    def set_autocommit(self, boolValue):
        if self.connection:
            self.connection.autocommit = boolValue
            print(f'Autocommit set to {boolValue}.')
        else:
            return None
```

In [3]:
```
class DatabaseSQLHandler:
    def __init__(self, dbcon, dbcursor):
        self.dbcon = dbcon
        self.dbcursor = dbcursor

    def create_database(self, dbase_name):
        create_database_query = f"""CREATE DATABASE {dbase_name};"""
        try:
            self.dbcursor.execute(create_database_query)
            #self.dbcon.commit()
            self.dbcursor.execute("commit")
            print (f'Database "{dbase_name}" creation succeeded.')
        except Exception as e:
            print(f"Error: {e}")

    def create_table(self, table_name, columns):
        create_table_query = f"""CREATE TABLE IF NOT EXISTS {table_name} ({columns});"""
        try:
            self.dbcursor.execute(create_table_query)
            self.dbcursor.execute("commit")
            print (f'Table "{table_name}" creation succeeded.')
        except Exception as e:
            print(f"Error: {e}")

    def insert_data(self, table_name, columns, data):
        insert_query = f"""INSERT INTO {table_name} ({columns}) VALUES ({', '.join(['%s' for _ in range(len(data))])
        try:
            self.dbcursor.execute(insert_query, data)
            self.dbcursor.execute("commit")
            #print (f'Data insertion succeeded.')
        except Exception as e:
            print(f"Error: {e}")

    def run_query(self, query, num_rows=0):
        #select_query = f"""[query]"""
        try:
            self.dbcursor.execute(query)
            if num_rows == 0:
                rows = self.dbcursor.fetchall()
            elif num_rows > 0:
                rows = self.dbcursor.fetchmany(num_rows)
            return rows
            #print (f'Data insertion succeeded.')
        except Exception as e:
            print(f"Error: {e}")
```

Establishing a PostgreSQL Connection

In [4]:
```
pgconstr = PostgreSQLConnection(
    host='127.0.0.1',
    dbname='postgres',
    user='postgres',
    password='sirlammy'
)
```

Creating PostgreSQL connection and cursor for Queries

In [5]:
```
#if not pgcon:
pgcursor, pgcon = pgconstr.connect()

if pgcursor:
    print ('Cursor creation succeeded.')
```

```
Cursor creation succeeded.
```

In [ ]:

In [6]:
```
# initialize connection
pgcursor.execute("rollback")
pg_handler = DatabaseSQLHandler(pgcon, pgcursor)

# create 'flights' database
pg_handler.create_database("flights")
```

```
Database "flights" creation succeeded.
```

Connecting to the "flights" database

In [7]:
```
dbconstr = PostgreSQLConnection(
    host='127.0.0.1',
    dbname='flights',
    user='postgres',
    password='sirlammy'
)

# create connection and cursor
#if not dbcon:
dbcursor, dbcon = dbconstr.connect()

if dbcursor:
    print ('Cursor creation succeeded.')
```

```
Cursor creation succeeded.
```

Brief Analysis of the datasets

- Flights dataset analysis

In [8]:
```
# Flights data analysis
flight_df = pd.read_csv("./flight_bookings.csv")

flight_df.info()
print('\n')
flight_df.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271888 entries, 0 to 271887
Data columns (total 10 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   travelCode  271888 non-null  int64
 1   userCode    271888 non-null  int64
 2   from        271888 non-null  object
 3   to          271888 non-null  object
 4   flightType  271888 non-null  object
 5   price       271888 non-null  float64
 6   time        271888 non-null  float64
 7   distance    271888 non-null  float64
 8   agency      271888 non-null  object
 9   date        271888 non-null  object
dtypes: float64(3), int64(2), object(5)
memory usage: 20.7+ MB
```

Out[8]:

	travelCode	userCode	from	to	flightType	price	time	distance	agency	date
0	0	0	Recife (PE)	Florianopolis (SC)	firstClass	1434.38	1.76	676.53	FlyingDrops	09/26/2019
1	0	0	Florianopolis (SC)	Recife (PE)	firstClass	1292.29	1.76	676.53	FlyingDrops	09/30/2019
2	1	0	Brasilia (DF)	Florianopolis (SC)	firstClass	1487.52	1.66	637.56	CloudFy	10/03/2019
3	1	0	Florianopolis (SC)	Brasilia (DF)	firstClass	1127.36	1.66	637.56	CloudFy	10/04/2019
4	2	0	Aracaju (SE)	Salvador (BH)	firstClass	1684.05	2.16	830.86	CloudFy	10/10/2019
5	2	0	Salvador (BH)	Aracaju (SE)	firstClass	1531.92	2.16	830.86	CloudFy	10/12/2019
6	3	0	Aracaju (SE)	Campo Grande (MS)	economic	743.54	1.69	650.10	Rainbow	10/17/2019
7	3	0	Campo Grande (MS)	Aracaju (SE)	economic	877.56	1.69	650.10	Rainbow	10/20/2019
8	4	0	Recife (PE)	Florianopolis (SC)	economic	803.39	1.76	676.53	Rainbow	10/24/2019
9	4	0	Florianopolis (SC)	Recife (PE)	economic	695.30	1.76	676.53	Rainbow	10/26/2019

In [9]:
```
# flight_df.groupby('agency').size().count()

flight_df.groupby('agency').size().sort_values(ascending=False)
```

Out[9]:
```
agency
Rainbow        116752
CloudFy        116378
FlyingDrops     38758
dtype: int64
```

In [10]:
```
flight_df.groupby('flightType').size().sort_values(ascending=False)
```

Out[10]:
```
flightType
firstClass    116418
premium        78004
economic       77466
dtype: int64
```

- Hotel dataset analysis

In [11]:
```
# Hotel lodges analysis
hotel_df = pd.read_csv("./hotels.csv")

hotel_df.info()

print('\n')
hotel_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40552 entries, 0 to 40551
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   travelCode  40552 non-null  int64
 1   userCode    40552 non-null  int64
 2   name        40552 non-null  object
 3   place       40552 non-null  object
 4   days        40552 non-null  int64
 5   price       40552 non-null  float64
 6   total       40552 non-null  float64
 7   date        40552 non-null  object
dtypes: float64(2), int64(3), object(3)
memory usage: 2.5+ MB
```

Out[11]:

	travelCode	userCode	name	place	days	price	total	date
0	0	0	Hotel A	Florianopolis (SC)	4	313.02	1252.08	09/26/2019
1	2	0	Hotel K	Salvador (BH)	2	263.41	526.82	10/10/2019
2	7	0	Hotel K	Salvador (BH)	3	263.41	790.23	11/14/2019
3	11	0	Hotel K	Salvador (BH)	4	263.41	1053.64	12/12/2019
4	13	0	Hotel A	Florianopolis (SC)	1	313.02	313.02	12/26/2019

In [12]:
```
# hotel_df.groupby('name').size().count()

hotel_df.groupby('name').size().sort_values(ascending=False)
```

Out[12]:
```
name
Hotel K     5094
Hotel CB    5029
Hotel BD    4829
Hotel AF    4828
Hotel AU    4467
Hotel BP    4437
Hotel BW    4333
Hotel Z     4205
Hotel A     3330
dtype: int64
```

- Users dataset analysis

In [13]:
```
# Users
users_df = pd.read_csv("./users.csv")

users_df.info()
print('\n')
users_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1340 entries, 0 to 1339
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   code     1340 non-null   int64 
 1   company  1340 non-null   object
 2   name     1340 non-null   object
 3   gender   1340 non-null   object
 4   age      1340 non-null   int64 
dtypes: int64(2), object(3)
memory usage: 52.5+ KB
```

Out[13]:

	code	company	name	gender	age
0	0	4You	Roy Braun	male	21
1	1	4You	Joseph Holsten	male	37
2	2	4You	Wilma Mcinnis	female	48
3	3	4You	Paula Daniel	female	23
4	4	4You	Patricia Carson	female	44

In [14]:
```
# users_df.groupby('company').size().count()
users_df.groupby('company').size().sort_values(ascending=False)
```

Out[14]:
```
company
4You              453
Acme Factory      261
Wonka Company     237
Monsters CYA      195
Umbrella LTDA     194
dtype: int64
```

General Insights

- Flight records between 2019 and 2022
- All three datasets contain complete data with no null entry value.
- Has 271,888 flight records; 40,552 hotel lodges by 1,340 users across 5 different companies
- 5 Companies are represented with '4You' having highest number of travellers, 453
- 9 hotels are represented with 'Hotel K' having highest lodge of 5,094 times
- 1,340 users comprising of males, females and undeclared genders
- All flight trips were booked with 3 agencies (Rainbow, CloudFy and FlyingDrops) with 'Rainow' recording highest travels of 116,752
- Users boarded firstclass, premium and economic. Most boarded class was 'Firstclass', boarded 116,418 times

Some useful functions

In [15]:
```
def convert_dt_to_postgrtype(datatype):
    if datatype == "object":
        return "varchar"
    elif datatype == "int64":
        return "int"
    elif datatype == "float64":
        return "float"

def strip_keywords(attribute):
    if attribute == "from":
        return "flightFrom"
    elif attribute == "to":
         return "flightTo"
    else:
        return attribute

def get_columns_for_table(df):
    try:
        attributes = df.columns
        result = []
        for attribute in attributes:
            dtype = convert_dt_to_postgrtype(df[attribute].dtypes)
            attribute = strip_keywords(attribute)
            result.append(f'{attribute} {dtype}')
            joined_result = ', '.join(result)
        return joined_result
    except Exception as e:
        print(f'Error: {str(e)}')

def get_columns_for_insertion(df):
    try:
        attributes = df.columns
        result = []
        for attribute in attributes:
            attribute = strip_keywords(attribute)
            result.append(f'{attribute}')
            joined_result = ', '.join(result)
        return joined_result
    except Exception as e:
        print(f'Error: {str(e)}')

def convert_row_to_frame(data,columns,col1=None ,col2=None, col3=None):
    df = pd.DataFrame(rowCount, columns=columns)

    # Format specified columns
    if (col1 != None):
        df[col1] = df[col1].apply(lambda x: "${:,.2f}".format(x))
    if (col2 != None):
        df[col2] = df[col2].apply(lambda x: "${:,.2f}".format(x))
    if (col3 != None):
        df[col3] = df[col3].apply(lambda x: "{:,}".format(x))

    return df

def line_break(sep_key, key_val):
    if sep_key == '':
        sep_key = key_val
    if sep_key != key_val:
        print('')
        sep_key = ''
    return sep_key
```

In [ ]:

In [16]:
```
# Create an instance of DatabaseSQLHandler
db_handler = DatabaseSQLHandler(dbcon, dbcursor)
```

In [17]:
```
# Define the columns for the table based on Dataset
table_columns = get_columns_for_table(flight_df)

# Create the table
db_handler.create_table('flight_bookings', table_columns)
```

```
Table "flight_bookings" creation succeeded.
```

In [18]:
```
# Define the columns for the table based on Dataset
table_columns = get_columns_for_table(hotel_df)

# Create the table
db_handler.create_table('hotel_bookings', table_columns)
```

```
Table "hotel_bookings" creation succeeded.
```

In [19]:
```
# Define the columns for the table based on Dataset
table_columns = get_columns_for_table(users_df)
```

```
# Create the table
db_handler.create_table('users', table_columns)
```

```
Table "users" creation succeeded.
```

Inserting data into the tables

In [20]:
```
# Inserting flights data
flight_table_name = 'flight_bookings'
insert_columns = get_columns_for_insertion(flight_df)
for i, row in flight_df.iterrows():
    data = tuple(row)
    db_handler.insert_data(flight_table_name, insert_columns, data)
print (f'Data insertion succeeded.')
```

```
Data insertion succeeded.
```

In [21]:
```
# Inserting hotels data
hotel_table_name = 'hotel_bookings'
insert_columns = get_columns_for_insertion(hotel_df)
for i, row in hotel_df.iterrows():
    data = tuple(row)
    db_handler.insert_data(hotel_table_name, insert_columns, data)
print (f'Data insertion succeeded.')
```

```
Data insertion succeeded.
```

In [22]:
```
# Inserting users data
user_table_name = 'users'
insert_columns = get_columns_for_insertion(users_df)
for i, row in users_df.iterrows():
    data = tuple(row)
    db_handler.insert_data(user_table_name, insert_columns, data)
print (f'Data insertion succeeded.')
```

```
Data insertion succeeded.
```

In [ ]:

Data Transformation and Analysis with SQL

Fetching top 5 rows from flight bookings table

In [23]:
```
# Fetching top 5 rows from flights table
rowCount = db_handler.run_query(f"select * from {flight_table_name} LIMIT 5;")
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : (0, 0, 'Recife (PE)', 'Florianopolis (SC)', 'firstClass', 1434.38, 1.76, 676.53, 'FlyingDrops', '09/26/2019')
2 : (0, 0, 'Florianopolis (SC)', 'Recife (PE)', 'firstClass', 1292.29, 1.76, 676.53, 'FlyingDrops', '09/30/2019')
3 : (1, 0, 'Brasilia (DF)', 'Florianopolis (SC)', 'firstClass', 1487.52, 1.66, 637.56, 'CloudFy', '10/03/2019')
4 : (1, 0, 'Florianopolis (SC)', 'Brasilia (DF)', 'firstClass', 1127.36, 1.66, 637.56, 'CloudFy', '10/04/2019')
5 : (2, 0, 'Aracaju (SE)', 'Salvador (BH)', 'firstClass', 1684.05, 2.16, 830.86, 'CloudFy', '10/10/2019')
```

Fetching top 5 rows from hotel bookings table

In [24]:
```
# Fetching top 5 rows from hotels table

rowCount = db_handler.run_query(f"select * from {hotel_table_name} LIMIT 5;")
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : (0, 0, 'Hotel A', 'Florianopolis (SC)', 4, 313.02, 1252.08, '09/26/2019')
2 : (2, 0, 'Hotel K', 'Salvador (BH)', 2, 263.41, 526.82, '10/10/2019')
3 : (7, 0, 'Hotel K', 'Salvador (BH)', 3, 263.41, 790.23, '11/14/2019')
4 : (11, 0, 'Hotel K', 'Salvador (BH)', 4, 263.41, 1053.64, '12/12/2019')
5 : (13, 0, 'Hotel A', 'Florianopolis (SC)', 1, 313.02, 313.02, '12/26/2019')
```

Fetching top 5 rows from users table

In [25]:
```
# Fetching top 5 rows from users table

rowCount = db_handler.run_query(f"select * from {user_table_name} LIMIT 5;")
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : (0, '4You', 'Roy Braun', 'male', 21)
2 : (1, '4You', 'Joseph Holsten', 'male', 37)
3 : (2, '4You', 'Wilma Mcinnis', 'female', 48)
4 : (3, '4You', 'Paula Daniel', 'female', 23)
5 : (4, '4You', 'Patricia Carson', 'female', 44)
```

In [26]:
```
# Fetching flights table description/structure for analysis

rowCount = db_handler.run_query(f"SELECT column_name, data_type FROM information_schema.columns WHERE table_name = '
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : ('travelcode', 'integer')
2 : ('usercode', 'integer')
3 : ('price', 'double precision')
4 : ('time', 'double precision')
5 : ('distance', 'double precision')
6 : ('date', 'character varying')
7 : ('agency', 'character varying')
8 : ('flightfrom', 'character varying')
9 : ('flightto', 'character varying')
10 : ('flighttype', 'character varying')
```

Some transformations on "flight_bookings" table

In [27]:
```
# We want to add new table columns 'flightMonth', 'MonthName', 'monthYear' and 'flightYear' for detailed insights
# during analysis and visualization in Tableau

db_handler.run_query(f"ALTER table {flight_table_name} ADD flightMonth int;")

db_handler.run_query(f"ALTER table {flight_table_name} ADD MonthName varchar(10);")

db_handler.run_query(f"ALTER table {flight_table_name} ADD flightYear int;")

db_handler.run_query(f"ALTER table {flight_table_name} ADD monthYear varchar(7);")
```

```
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
```

In [28]:
```
# Fetching table description/structure to confirm new columns were successfully created

rowCount = db_handler.run_query(f"SELECT column_name, data_type FROM information_schema.columns WHERE table_name = '
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : ('travelcode', 'integer')
2 : ('usercode', 'integer')
3 : ('price', 'double precision')
4 : ('time', 'double precision')
5 : ('distance', 'double precision')
6 : ('flightmonth', 'integer')
7 : ('flightyear', 'integer')
8 : ('monthname', 'character varying')
9 : ('agency', 'character varying')
10 : ('flightfrom', 'character varying')
11 : ('flightto', 'character varying')
12 : ('flighttype', 'character varying')
13 : ('date', 'character varying')
14 : ('monthyear', 'character varying')
```

In [29]:
```
# To add values to the newly created columns

updateQuery = db_handler.run_query(f"UPDATE {flight_table_name} SET flightYear = (EXTRACT(YEAR FROM TO_DATE(date,'MM/

updateQuery = db_handler.run_query(f"UPDATE {flight_table_name} SET flightMonth = (EXTRACT(MONTH FROM TO_DATE(date,'M

updateQuery = db_handler.run_query(f"UPDATE {flight_table_name} SET MonthName = TRIM(TO_CHAR(TO_DATE(date, 'MM/DD/YY

updateQuery = db_handler.run_query(f"UPDATE {flight_table_name} SET MonthYear = CONCAT(flightYear,'-',flightMonth);")
```

```
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
```

In [ ]:

In [30]:
```
# Fetching top 5 rows to confirm updates
rowCount = db_handler.run_query(f"select * from {flight_table_name};",5)
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : (2671, 23, 'Recife (PE)', 'Natal (RN)', 'premium', 474.6, 0.58, 222.67, 'CloudFy', '06/25/2020', 6, 'June', 2020,
'2020-6')
2 : (8828, 89, 'Recife (PE)', 'Natal (RN)', 'economic', 301.51, 0.58, 222.67, 'CloudFy', '06/11/2020', 6, 'June', 202
0, '2020-6')
3 : (8949, 90, 'Recife (PE)', 'Natal (RN)', 'economic', 360.22, 0.58, 222.67, 'Rainbow', '01/06/2022', 1, 'January',
2022, '2022-1')
4 : (11512, 112, 'Recife (PE)', 'Natal (RN)', 'premium', 488.73, 0.58, 222.67, 'Rainbow', '02/02/2023', 2, 'Februar
y', 2023, '2023-2')
5 : (17474, 174, 'Recife (PE)', 'Natal (RN)', 'premium', 488.73, 0.58, 222.67, 'Rainbow', '11/10/2022', 11, 'Novembe
r', 2022, '2022-11')
```

Some transformations on "hotel_bookings" table

In [31]:
```
# Fetching hotel table description/structure for analysis

rowCount = db_handler.run_query(f"SELECT column_name, data_type FROM information_schema.columns WHERE table_name = '
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : ('travelcode', 'integer')
2 : ('usercode', 'integer')
3 : ('days', 'integer')
4 : ('price', 'double precision')
5 : ('total', 'double precision')
6 : ('name', 'character varying')
7 : ('place', 'character varying')
8 : ('date', 'character varying')
```

In [32]:
```
# We want to add new columns 'lodgeMonth', 'MonthName', 'monthYear' and 'lodgeYear' for detailed insights...
# during analysis and visualization in Tableau

alterQuery = db_handler.run_query(f"ALTER table {hotel_table_name} ADD lodgeMonth int;")

alterQuery = db_handler.run_query(f"ALTER table {hotel_table_name} ADD MonthName varchar(10);")

alterQuery = db_handler.run_query(f"ALTER table {hotel_table_name} ADD lodgeYear int;")

alterQuery = db_handler.run_query(f"ALTER table {hotel_table_name} ADD monthYear varchar(7);")

print(alterQuery)
```

```
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
None
```

In [33]:
```
# Fetching table description/structure to confirm new columns were successfully created

rowCount = db_handler.run_query(f"SELECT column_name, data_type FROM information_schema.columns WHERE table_name = '
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : ('travelcode', 'integer')
2 : ('usercode', 'integer')
3 : ('days', 'integer')
4 : ('price', 'double precision')
5 : ('total', 'double precision')
6 : ('lodgemonth', 'integer')
7 : ('lodgeyear', 'integer')
8 : ('name', 'character varying')
9 : ('place', 'character varying')
10 : ('date', 'character varying')
11 : ('monthyear', 'character varying')
12 : ('monthname', 'character varying')
```

In [34]:
```
db_handler.run_query(f"UPDATE {hotel_table_name} SET lodgeYear = (EXTRACT(YEAR FROM TO_DATE(date,'MM/DD/YYYY')));")

db_handler.run_query(f"UPDATE {hotel_table_name} SET lodgeMonth = (EXTRACT(MONTH FROM TO_DATE(date,'MM/DD/YYYY')));"

db_handler.run_query(f"UPDATE {hotel_table_name} SET MonthName = TRIM(TO_CHAR(TO_DATE(date, 'MM/DD/YYYY'), 'Month'));

db_handler.run_query(f"UPDATE {hotel_table_name} SET MonthYear = CONCAT(lodgeYear,'-',lodgeMonth);")
```

```
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
Error: no results to fetch
```

In [35]:
```
# Fetching top 5 rows to confirm updates

rowCount = db_handler.run_query(f"select * from {hotel_table_name};",5)
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : (0, 0, 'Hotel A', 'Florianopolis (SC)', 4, 313.02, 1252.08, '09/26/2019', 9, 'September', 2019, '2019-9')
2 : (2, 0, 'Hotel K', 'Salvador (BH)', 2, 263.41, 526.82, '10/10/2019', 10, 'October', 2019, '2019-10')
3 : (7, 0, 'Hotel K', 'Salvador (BH)', 3, 263.41, 790.23, '11/14/2019', 11, 'November', 2019, '2019-11')
4 : (11, 0, 'Hotel K', 'Salvador (BH)', 4, 263.41, 1053.64, '12/12/2019', 12, 'December', 2019, '2019-12')
5 : (13, 0, 'Hotel A', 'Florianopolis (SC)', 1, 313.02, 313.02, '12/26/2019', 12, 'December', 2019, '2019-12')
```

Some more EDA With PosgreSQL

Most visited destination between 2019-2023

In [37]:
```
# most visited destination
rowCount = db_handler.run_query(f"select flightto, count(flightto) as totalVisits from {flight_table_name} \
GROUP BY flightto ORDER BY totalVisits DESC;",0)
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : ('Florianopolis (SC)', 57317)
2 : ('Aracaju (SE)', 37224)
3 : ('Campo Grande (MS)', 34748)
4 : ('Brasilia (DF)', 30779)
5 : ('Recife (PE)', 30480)
6 : ('Natal (RN)', 23796)
7 : ('Sao Paulo (SP)', 23625)
8 : ('Salvador (BH)', 17104)
9 : ('Rio de Janeiro (RJ)', 16815)
```

Most visited destination each year

In [38]:
```python
# most visited destination each year
yr = ''
rowCount = db_handler.run_query(f"select flightyear, flightto, count(flightto) as totalVisits from {flight_table_name
GROUP BY flightto, flightyear ORDER BY flightyear, totalVisits DESC;",0)
for i, row in enumerate(rowCount):
    yr = (line_break(yr,row[0]))
    print(f'{i+1} : {row}')
```

```
1 : (2019, 'Florianopolis (SC)', 7490)
2 : (2019, 'Aracaju (SE)', 4864)
3 : (2019, 'Campo Grande (MS)', 4529)
4 : (2019, 'Brasilia (DF)', 4093)
5 : (2019, 'Recife (PE)', 4000)
6 : (2019, 'Natal (RN)', 3204)
7 : (2019, 'Sao Paulo (SP)', 3192)
8 : (2019, 'Rio de Janeiro (RJ)', 2276)
9 : (2019, 'Salvador (BH)', 2178)

10 : (2020, 'Florianopolis (SC)', 23501)
11 : (2020, 'Aracaju (SE)', 15505)
12 : (2020, 'Campo Grande (MS)', 14298)
13 : (2020, 'Brasilia (DF)', 12618)
14 : (2020, 'Recife (PE)', 12545)
15 : (2020, 'Natal (RN)', 10017)
16 : (2020, 'Sao Paulo (SP)', 9813)
17 : (2020, 'Salvador (BH)', 7268)
18 : (2020, 'Rio de Janeiro (RJ)', 7006)

19 : (2021, 'Florianopolis (SC)', 15782)
20 : (2021, 'Aracaju (SE)', 10320)
21 : (2021, 'Campo Grande (MS)', 9727)
22 : (2021, 'Brasilia (DF)', 8594)
23 : (2021, 'Recife (PE)', 8472)
24 : (2021, 'Sao Paulo (SP)', 6621)
25 : (2021, 'Natal (RN)', 6535)
26 : (2021, 'Salvador (BH)', 4679)
27 : (2021, 'Rio de Janeiro (RJ)', 4633)

28 : (2022, 'Florianopolis (SC)', 9184)
29 : (2022, 'Aracaju (SE)', 5631)
30 : (2022, 'Campo Grande (MS)', 5373)
31 : (2022, 'Recife (PE)', 4708)
32 : (2022, 'Brasilia (DF)', 4700)
33 : (2022, 'Natal (RN)', 3543)
34 : (2022, 'Sao Paulo (SP)', 3491)
35 : (2022, 'Salvador (BH)', 2610)
36 : (2022, 'Rio de Janeiro (RJ)', 2521)

37 : (2023, 'Florianopolis (SC)', 1360)
38 : (2023, 'Aracaju (SE)', 904)
39 : (2023, 'Campo Grande (MS)', 821)
40 : (2023, 'Brasilia (DF)', 774)
41 : (2023, 'Recife (PE)', 755)
42 : (2023, 'Sao Paulo (SP)', 508)
43 : (2023, 'Natal (RN)', 497)
44 : (2023, 'Rio de Janeiro (RJ)', 379)
45 : (2023, 'Salvador (BH)', 369)
```

Year with highest travel and hotel expenses

In [39]:
```python
# The join was too expensive to run, so I skipped it but the code runs well
yr = ''
rowCount = db_handler.run_query(f"select f.flightyear, sum(f.price) as totalFlight, sum(h.price) as totalHotel from {
JOIN {hotel_table_name} h ON f.flightyear = h.lodgeyear GROUP BY f.flightyear ORDER BY f.flightyear DESC;",0)
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row[0], "${:,.2f}".format(row[1]), "${:,.2f}".format(row[2])}')

print('\n')

# Display in a dataframe for nice viewing
totalSpendings = convert_row_to_frame(rowCount,['Flight Year', 'Travel Spendings', 'Hotel Spendings'],'Travel Spendir
totalSpendings.head()
```

```
1 : (2023, '$5,762,476,252.80', '$1,280,599,803.61')
2 : (2022, '$247,792,862,993.80', '$55,565,250,576.39')
3 : (2021, '$807,508,895,462.95', '$180,635,425,394.94')
4 : (2020, '$1,823,253,030,514.52', '$408,343,021,058.95')
5 : (2019, '$181,745,920,754.20', '$40,840,581,346.21')
```

Out[39]:

	Flight Year	Travel Spendings	Hotel Spendings
0	2023	$5,762,476,252.80	$1,280,599,803.61
1	2022	$247,792,862,993.80	$55,565,250,576.39
2	2021	$807,508,895,462.95	$180,635,425,394.94
3	2020	$1,823,253,030,514.52	$408,343,021,058.95
4	2019	$181,745,920,754.20	$40,840,581,346.21

In [40]:
```python
print('\n')

# Display in a dataframe for nice viewing
totalSpendings = convert_row_to_frame(rowCount,['Flight Year', 'Travel Spendings', 'Hotel Spendings'],'Travel Spendir
(totalSpendings.head())
```

Out[40]:

	Flight Year	Travel Spendings	Hotel Spendings
0	2023	$5,762,476,252.80	$1,280,599,803.61
1	2022	$247,792,862,993.80	$55,565,250,576.39
2	2021	$807,508,895,462.95	$180,635,425,394.94
3	2020	$1,823,253,030,514.52	$408,343,021,058.95
4	2019	$181,745,920,754.20	$40,840,581,346.21

Year with highest travel expenses

In [41]:
```python
rowCount = db_handler.run_query(f"select flightyear, sum(price) as totalPrice from {flight_table_name} \
GROUP BY flightyear ORDER BY totalPrice DESC;",0)
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row[0], "${:,.2f}".format(row[1])}')
```

```
1 : (2020, '$107,699,984.05')
2 : (2021, '$72,137,653.69')
3 : (2022, '$40,219,584.97')
4 : (2019, '$34,124,281.03')
5 : (2023, '$6,117,278.40')
```

Total monthly travel from high to low between 2019 and 2023

In [44]:
```python
# Peak travel month in each year
yr = ''
rowCount = db_handler.run_query(f"SELECT flightyear, monthname, MAX(totalFlights) \
FROM ( \
    SELECT flightyear, monthname, COUNT(flightto) as totalFlights \
    FROM {flight_table_name} \
    GROUP BY flightyear, monthname \
)  \
GROUP BY flightyear, monthname \
ORDER BY flightyear, MAX(totalFlights) DESC;",0)
for i, row in enumerate(rowCount):
    yr = (line_break(yr,row[0]))
    print(f'{i+1} : {row}')
```

```
1 : (2019, 'October', 11762)
2 : (2019, 'November', 10782)
3 : (2019, 'December', 10612)
4 : (2019, 'September', 2670)

5 : (2020, 'January', 11326)
6 : (2020, 'April', 10321)
7 : (2020, 'March', 10028)
8 : (2020, 'February', 9902)
9 : (2020, 'July', 9847)
10 : (2020, 'May', 9793)
11 : (2020, 'October', 9131)
12 : (2020, 'August', 9071)
13 : (2020, 'June', 9050)
14 : (2020, 'December', 8201)
15 : (2020, 'September', 7954)
16 : (2020, 'November', 7947)

17 : (2021, 'January', 7756)
18 : (2021, 'April', 7313)
19 : (2021, 'February', 6869)
20 : (2021, 'March', 6824)
21 : (2021, 'May', 6749)
22 : (2021, 'July', 6703)
23 : (2021, 'June', 5874)
24 : (2021, 'September', 5771)
25 : (2021, 'August', 5741)
26 : (2021, 'October', 5476)
27 : (2021, 'December', 5344)
28 : (2021, 'November', 4943)

29 : (2022, 'January', 4804)
30 : (2022, 'March', 4593)
31 : (2022, 'February', 4238)
32 : (2022, 'April', 4083)
33 : (2022, 'May', 3834)
34 : (2022, 'June', 3756)
35 : (2022, 'July', 3434)
36 : (2022, 'September', 3073)
37 : (2022, 'August', 3054)
38 : (2022, 'October', 2611)
39 : (2022, 'December', 2189)
40 : (2022, 'November', 2092)

41 : (2023, 'January', 1701)
42 : (2023, 'February', 1378)
43 : (2023, 'March', 1296)
44 : (2023, 'April', 890)
45 : (2023, 'May', 592)
46 : (2023, 'June', 381)
47 : (2023, 'July', 129)
```

Peak travel month in each year

In [45]:
```
# Peak travel month in each year
rowCount = db_handler.run_query(f"SELECT flightyear, monthname, totalFlights \
FROM ( \
    SELECT flightyear, monthname, COUNT(flightto) AS totalFlights, \
        RANK() OVER (PARTITION BY flightyear ORDER BY COUNT(flightto) DESC) AS ranking \
    FROM {flight_table_name} GROUP BY flightyear, monthname \
) AS RankedFlights \
WHERE ranking = 1 \
ORDER BY flightyear;",0)

for i, row in enumerate(rowCount):
    print(f'{i+1} : {row}')
```

```
1 : (2019, 'October', 11762)
2 : (2020, 'January', 11326)
3 : (2021, 'January', 7756)
4 : (2022, 'January', 4804)
5 : (2023, 'January', 1701)
```

Total Flights and cost handled by agencies in each year

In [46]:
```
yr = ''
rowCount = db_handler.run_query(f"select agency, flightyear, sum(price) as totalPrice, count(flightto) as totalFlight
GROUP BY agency, flightyear ORDER BY flightyear, totalPrice DESC;",0)
for i, row in enumerate(rowCount):
    print(f'{i+1} : {row[0], row[1], "${:,.2f}".format(row[2]), "{:,}".format(row[3])}')

print('\n')

# Display in a dataframe for nice viewing
totalFlights = convert_row_to_frame(rowCount,['Agency', 'flightyear', 'totalPrice', 'totalFlights'],'totalPrice' ,Nor
(totalFlights.head(16))
```

```
1 : ('Rainbow', 2019, '$14,159,650.37', '15,446')
2 : ('CloudFy', 2019, '$13,934,756.48', '15,280')
3 : ('FlyingDrops', 2019, '$6,029,874.18', '5,100')
4 : ('CloudFy', 2020, '$44,495,417.02', '48,404')
5 : ('Rainbow', 2020, '$44,278,986.11', '48,206')
6 : ('FlyingDrops', 2020, '$18,925,580.92', '15,961')
7 : ('Rainbow', 2021, '$30,005,116.22', '32,599')
8 : ('CloudFy', 2021, '$29,542,565.53', '32,127')
9 : ('FlyingDrops', 2021, '$12,589,971.94', '10,637')
10 : ('Rainbow', 2022, '$16,470,063.88', '17,791')
11 : ('CloudFy', 2022, '$16,433,617.82', '17,832')
12 : ('FlyingDrops', 2022, '$7,315,903.27', '6,138')
13 : ('CloudFy', 2023, '$2,532,978.04', '2,735')
14 : ('Rainbow', 2023, '$2,472,426.74', '2,710')
15 : ('FlyingDrops', 2023, '$1,111,873.62', '922')
```

Out[46]:

	Agency	flightyear	totalPrice	totalFlights
0	Rainbow	2019	$14,159,650.37	15,446
1	CloudFy	2019	$13,934,756.48	15,280
2	FlyingDrops	2019	$6,029,874.18	5,100
3	CloudFy	2020	$44,495,417.02	48,404
4	Rainbow	2020	$44,278,986.11	48,206
5	FlyingDrops	2020	$18,925,580.92	15,961
6	Rainbow	2021	$30,005,116.22	32,599
7	CloudFy	2021	$29,542,565.53	32,127
8	FlyingDrops	2021	$12,589,971.94	10,637
9	Rainbow	2022	$16,470,063.88	17,791
10	CloudFy	2022	$16,433,617.82	17,832
11	FlyingDrops	2022	$7,315,903.27	6,138
12	CloudFy	2023	$2,532,978.04	2,735
13	Rainbow	2023	$2,472,426.74	2,710
14	FlyingDrops	2023	$1,111,873.62	922

Unified table showing users bookings with return tickets

In [47]:

```python
#dbcursor.execute("rollback")
rowCount = db_handler.run_query(f"SELECT travelCode, userCode, \
    MAX(CASE WHEN sequence = 1 THEN flightType END) AS flightType, \
    MAX(CASE WHEN sequence = 1 THEN price END) AS flightPrice, \
    MAX(CASE WHEN sequence = 1 THEN date END) AS flightDate, \
    MAX(CASE WHEN sequence = 2 THEN price END) AS returnPrice, \
    MAX(CASE WHEN sequence = 2 THEN date END) AS returnDate, \
    MAX(CASE WHEN sequence = 1 THEN agency END) AS agency, \
    MAX(CASE WHEN sequence = 1 THEN flightfrom END) AS flightFrom \
FROM ( \
    SELECT travelCode, userCode, agency, flightfrom, flightType, price, date, \
        ROW_NUMBER() OVER (PARTITION BY travelCode, userCode, flightType ORDER BY date) AS sequence \
    FROM {flight_table_name} \
) GROUP BY travelCode, userCode;",0)
for i, row in enumerate(rowCount):
    if i < 10:
        print(f'{i+1} : {row}')

print('\n')

# Display in a dataframe for nice viewing
unified_df = convert_row_to_frame(rowCount,['travelCode', 'userCode', 'flightType', 'flightPrice', 'flightDate', \
            'returnPrice', 'returnDate', 'Agency', 'flightFrom'],'flightPrice' ,'returnPrice', None)
unified_df.head(20)
```

```
1 : (0, 0, 'firstClass', 1434.38, '09/26/2019', 1292.29, '09/30/2019', 'FlyingDrops', 'Recife (PE)')
2 : (1, 0, 'firstClass', 1487.52, '10/03/2019', 1127.36, '10/04/2019', 'CloudFy', 'Brasilia (DF)')
3 : (2, 0, 'firstClass', 1684.05, '10/10/2019', 1531.92, '10/12/2019', 'CloudFy', 'Aracaju (SE)')
4 : (3, 0, 'economic', 743.54, '10/17/2019', 877.56, '10/20/2019', 'Rainbow', 'Aracaju (SE)')
5 : (4, 0, 'economic', 803.39, '10/24/2019', 695.3, '10/26/2019', 'Rainbow', 'Recife (PE)')
6 : (5, 0, 'firstClass', 1287.52, '10/31/2019', 898.04, '11/01/2019', 'FlyingDrops', 'Brasilia (DF)')
7 : (6, 0, 'premium', 1070.54, '11/07/2019', 1013.4, '11/10/2019', 'Rainbow', 'Recife (PE)')
8 : (7, 0, 'economic', 964.83, '11/14/2019', 811.73, '11/17/2019', 'CloudFy', 'Aracaju (SE)')
9 : (8, 0, 'economic', 513.06, '11/21/2019', 829.91, '11/24/2019', 'CloudFy', 'Recife (PE)')
10 : (9, 0, 'economic', 583.6, '11/28/2019', 506.56, '11/30/2019', 'CloudFy', 'Brasilia (DF)')
```

Out[47]:

	travelCode	userCode	flightType	flightPrice	flightDate	returnPrice	returnDate	Agency	flightFrom
0	0	0	firstClass	$1,434.38	09/26/2019	$1,292.29	09/30/2019	FlyingDrops	Recife (PE)
1	1	0	firstClass	$1,487.52	10/03/2019	$1,127.36	10/04/2019	CloudFy	Brasilia (DF)
2	2	0	firstClass	$1,684.05	10/10/2019	$1,531.92	10/12/2019	CloudFy	Aracaju (SE)
3	3	0	economic	$743.54	10/17/2019	$877.56	10/20/2019	Rainbow	Aracaju (SE)
4	4	0	economic	$803.39	10/24/2019	$695.30	10/26/2019	Rainbow	Recife (PE)
5	5	0	firstClass	$1,287.52	10/31/2019	$898.04	11/01/2019	FlyingDrops	Brasilia (DF)
6	6	0	premium	$1,070.54	11/07/2019	$1,013.40	11/10/2019	Rainbow	Recife (PE)
7	7	0	economic	$964.83	11/14/2019	$811.73	11/17/2019	CloudFy	Aracaju (SE)
8	8	0	economic	$513.06	11/21/2019	$829.91	11/24/2019	CloudFy	Recife (PE)
9	9	0	economic	$583.60	11/28/2019	$506.56	11/30/2019	CloudFy	Brasilia (DF)
10	10	0	firstClass	$992.17	12/05/2019	$824.31	12/06/2019	Rainbow	Brasilia (DF)
11	11	0	premium	$1,268.97	12/12/2019	$882.86	12/16/2019	Rainbow	Brasilia (DF)
12	12	0	premium	$965.62	12/19/2019	$706.36	12/20/2019	Rainbow	Brasilia (DF)
13	13	0	firstClass	$1,434.38	12/26/2019	$1,292.29	12/27/2019	FlyingDrops	Recife (PE)
14	14	0	firstClass	$893.65	01/02/2020	$742.94	01/04/2020	CloudFy	Brasilia (DF)
15	15	0	premium	$474.60	01/09/2020	$563.23	01/11/2020	CloudFy	Recife (PE)
16	16	0	premium	$1,021.53	01/16/2020	$1,215.45	01/18/2020	CloudFy	Aracaju (SE)
17	17	0	economic	$301.51	01/23/2020	$429.77	01/24/2020	CloudFy	Recife (PE)
18	18	0	economic	$791.66	01/30/2020	$697.51	01/31/2020	CloudFy	Brasilia (DF)
19	19	0	firstClass	$1,596.61	02/06/2020	$1,348.04	02/09/2020	FlyingDrops	Recife (PE)

In [ ]:

In [48]:

```
# Close connection
pgcon.close()
dbcon.close()
```