SQL Table Trransformation Subqueries

find flight information about flights where the origin elevation is *greater* than 2000 feet.

```
SELECT *
FROM flights
WHERE origin in (
   SELECT code
   FROM airports
   WHERE elevation > 2000);
```

As shown above, **a subquery is nested within another query** to give us our desired result.

A **non-correlated subquery** is a subquery that can be run independently of the outer query and as we saw, can be used to complete a multi-step transformation.

---

sometimes we need to aggregate in multiple steps - like taking an average of a count.
Imagine you'd like to know how many flights there are on average, for all Fridays in a given month from the flights table. First, we'd need to calculate the number of flights per day, and then we'd need to calculate the average based on the daily flight count for each day of the week. We can do this all in one step using a subquery:

```
SELECT a.dep_month,
       a.dep_day_of_week,
       AVG(a.flight_count) AS average_flights
  FROM (
        SELECT dep_month,
              dep_day_of_week,
               dep_date,
               COUNT(*) AS flight_count
          FROM flights
         GROUP BY 1,2,3
       ) a
 GROUP BY 1,2
 ORDER BY 1,2;
```

The inner query provides the count of flights by day, and the outer query uses the inner query's result set to compute the average by day of week of a given month.

Using a subquery**, find the average total distance flown by day of week and month.**
Be sure to alias the outer query as average_distance and the inner query as flight_distance.

```
   SELECT a.dep_month,
          a.dep_day_of_week,
          AVG(a.flight_distance) as average_distance
   FROM (
    SELECT dep_month,
          dep_day_of_week,
     dep_date,
     distance,
     sum(distance) AS flight_distance
       FROM flights
     GROUP BY 1,2,3
 ) a /* this is where the alias is set, in this case the alias is set to 'a' */
 GROUP BY 1,2
 ORDER BY 1,2;
```

CORRELATED QUERIES

In a **correlated subquery**, the subquery can not be run independently of the outer query. The order of operations is important in a correlated subquery:

1. A row is processed in the outer query.
2. Then, for that particular row in the outer query, the subquery is executed.

This means that for each row processed by the outer query, the subquery will also be processed for that row. In this example, we will find **the list of all flights whose distance is above average for their carrier.**

```
SELECT id
FROM flights AS f
WHERE distance > (
 SELECT AVG(distance)
 FROM flights
 WHERE carrier = f.carrier);
```

In the above query the inner query has to be re-executed for each flight.
Correlated subqueries may appear elsewhere besides the WHERE clause, they can also appear in the SELECT.

It would also be interesting to order flights by giving them a sequence number based on time, by carrier.
For instance, assuming flight_id increments with each additional flight, we could use the following query to view flights by carrier, flight id, and sequence number:

```
SELECT carrier, id,
   (SELECT COUNT(*)
FROM flights f
WHERE f.id < flights.id
AND f.carrier=flights.carrier) + 1
 AS flight_sequence_number
FROM flights;
```

write a query to view flights by origin, flight id, and sequence number. Alias the sequence number column as flight_sequence_number:

```
SELECT origin, id,
(SELECT COUNT(*)
      FROM flights f
      WHERE f.id < flights.id
      AND f.origin = flights.origin) +1
      AS flight_sequence_number
FROM flights;
```

- **Subqueries** are used to complete an SQL transformation by nesting one query within another query.
- A **non-correlated subquery** is a subquery that can be run independently of the outer query and can be used to complete a multi-step transformation.
- A **correlated subquery** is a subquery that cannot be run independently of the outer query. The order of operations in a correlated subquery is as follows:
1. A row is processed in the outer query.
2. Then, for that particular row in the outer query, the subquery is executed.