Flight Delay Database Design

Overview of Database Schema

This database consists of **8 relational schemas** that together support a flight delay prediction and notification platform. The schemas include:

1. **User**
   Stores user information including username, email, phone number, and login credentials.

2. **Airline**
   Contains airline identifying information such as airline code, DOT code, and name.

3. **Airport**
   Stores airport details including code, name, location (latitude/longitude), city, state, time zone, and ZIP code.

4. **Flight**
   Represents scheduled flights with information about airline, flight number, origin and destination airports, and scheduled times.

5. **Flight_Status**
   Tracks the actual performance of flights including actual arrival/departure times, delays, and cancellation information.

6. **Delay_Prediction**
   Contains predicted delay information for flights and tracks whether notifications were sent.

7. **Weather_Event**
   Stores information about weather events that might affect flights, including type, severity, location, and timing.

8. **User_Alert**
   Stores user preferences for receiving alerts about flight delays.

Creating Tables - DDL

```
CREATE TABLE IF NOT EXISTS `User` (
  `userId`       VARCHAR(36)   NOT NULL,
  `username`     VARCHAR(50)   NOT NULL,
  `email`        VARCHAR(100)  NOT NULL,
  `phoneNumber`  VARCHAR(20)   DEFAULT NULL,
  `password`     VARCHAR(100)  NOT NULL,
  `createdAt`    DATE          NOT NULL,
  `updatedAt`    DATE          NOT NULL,
  PRIMARY KEY (`userId`)
)
```

Creating Airline Table

```
CREATE TABLE IF NOT EXISTS `Airline` (
  `airlineCode`  VARCHAR(10)   NOT NULL,
  `dotCode`      INT           DEFAULT NULL,
  `name`         VARCHAR(100)  NOT NULL,
  PRIMARY KEY (`airlineCode`)
)
```

Creating Airport Table

```
CREATE TABLE IF NOT EXISTS `Airport` (
  `airportCode`  VARCHAR(10)   NOT NULL,
  `name`         VARCHAR(100)  NOT NULL,
  `city`         VARCHAR(100)  NOT NULL,
  `state`        VARCHAR(50)   DEFAULT NULL,
  `locationLat`  FLOAT         DEFAULT NULL,
  `locationLng`  FLOAT         DEFAULT NULL,
  `timeZone`     VARCHAR(50)   DEFAULT NULL,
  `zipCode`      VARCHAR(20)   DEFAULT NULL,
  PRIMARY KEY (`airportCode`)
)
```

Creating Flight Table

```
CREATE TABLE IF NOT EXISTS `Flight` (
  `flightId`               VARCHAR(36)  NOT NULL,
  `airlineCode`            VARCHAR(10)  NOT NULL,
  `flightNumber`           INT          NOT NULL,
  `originAirport`          VARCHAR(10)  NOT NULL,
  `destAirport`            VARCHAR(10)  NOT NULL,
  `scheduledDepartureTime` DATETIME     NOT NULL,
  `scheduledArrivalTime`   DATETIME     NOT NULL,
  `elapsedTime`          FLOAT       DEFAULT NULL,
  `distance`             FLOAT       DEFAULT NULL,
  PRIMARY KEY (`flightId`),
  CONSTRAINT `fk_Flight_Airline`
    FOREIGN KEY (`airlineCode`)
    REFERENCES `Airline` (`airlineCode`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Flight_Origin`
    FOREIGN KEY (`originAirport`)
    REFERENCES `Airport` (`airportCode`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Flight_Dest`
    FOREIGN KEY (`destAirport`)
    REFERENCES `Airport` (`airportCode`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
)
```

Creating Flight_Status Table

```
CREATE TABLE IF NOT EXISTS `Flight_Status` (
  `statusId`          VARCHAR(36)  NOT NULL,
  `flightId`          VARCHAR(36)  NOT NULL,
  `flightDate`        DATE         NOT NULL,
  `actualDepartureTime` DATETIME    DEFAULT NULL,
  `actualArrivalTime`   DATETIME    DEFAULT NULL,
  `departureDelay`     FLOAT       DEFAULT NULL,
  `arrivalDelay`       FLOAT       DEFAULT NULL,
  `taxiOut`            FLOAT       DEFAULT NULL,
  `taxiIn`             FLOAT       DEFAULT NULL,
  `actualElapsedTime`  FLOAT       DEFAULT NULL,
  `airTime`            FLOAT       DEFAULT NULL,
  `cancelled`          BOOLEAN     DEFAULT FALSE,
  `cancellationCode`   VARCHAR(10) DEFAULT NULL,
  `diverted`           BOOLEAN     DEFAULT FALSE,
  `carrierDelay`       FLOAT       DEFAULT NULL,
  `weatherDelay`       FLOAT       DEFAULT NULL,
  `nasDelay`           FLOAT       DEFAULT NULL,
  `securityDelay`      FLOAT       DEFAULT NULL,
  `lateAircraftDelay`  FLOAT       DEFAULT NULL,
  PRIMARY KEY (`statusId`),
  CONSTRAINT `fk_Flight_Status_Flight`
    FOREIGN KEY (`flightId`)
    REFERENCES `Flight` (`flightId`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
)
```

Creating Delay_Prediction Table

```
CREATE TABLE IF NOT EXISTS `Delay_Prediction` (
  `predictionId`            VARCHAR(36)  NOT NULL,
  `flightId`                VARCHAR(36)  NOT NULL,
  `predictionTime`          DATETIME     NOT NULL,
  `predictedDepartureDelay` FLOAT        DEFAULT NULL,
  `predictedArrivalDelay`   FLOAT        DEFAULT NULL,
  `notificationSent`        BOOLEAN      DEFAULT FALSE,
  `predictionReason`        VARCHAR(255) DEFAULT NULL,
  PRIMARY KEY (`predictionId`),
  CONSTRAINT `fk_DelayPrediction_Flight`
    FOREIGN KEY (`flightId`)
    REFERENCES `Flight` (`flightId`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
)
```

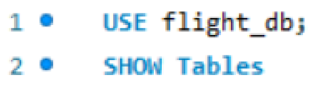
Creating Weather_Event table

```
CREATE TABLE IF NOT EXISTS `Weather_Event` (
  `eventId`       VARCHAR(36)  NOT NULL,
  `airportCode`   VARCHAR(10)  NOT NULL,
  `type`          VARCHAR(50)  NOT NULL,
  `severity`      VARCHAR(50)  DEFAULT NULL,
  `startTime`     DATETIME     NOT NULL,
  `endTime`       DATETIME     NOT NULL,
  `precipitation` FLOAT        DEFAULT NULL,
  `locationLat`   FLOAT        DEFAULT NULL,
  `locationLng`   FLOAT        DEFAULT NULL,
  `city`          VARCHAR(100) DEFAULT NULL,
  `country`       VARCHAR(100) DEFAULT NULL,
  `zipCode`       VARCHAR(20)  DEFAULT NULL,
  PRIMARY KEY (`eventId`),
  CONSTRAINT `fk_WeatherEvent_Airport`
    FOREIGN KEY (`airportCode`)
    REFERENCES `Airport` (`airportCode`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
)
```

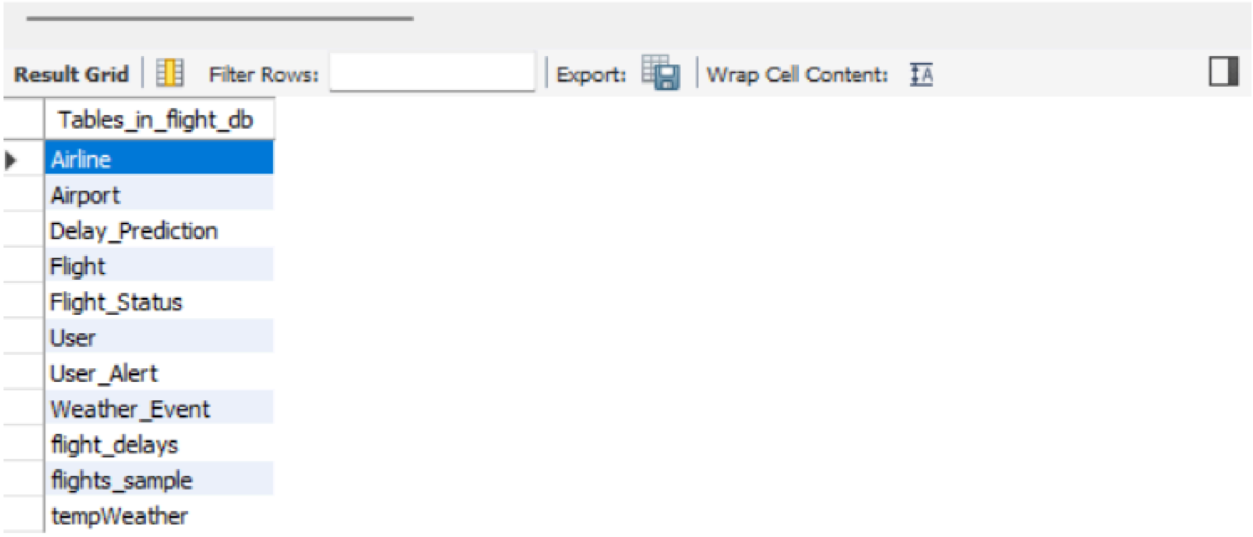
Creating User_Alert table

```
CREATE TABLE IF NOT EXISTS `User_Alert` (
  `alertId`        VARCHAR(36)  NOT NULL,
  `userId`         VARCHAR(36)  NOT NULL,
  `emailAlert`     BOOLEAN      DEFAULT FALSE,
  `smsAlert`       BOOLEAN      DEFAULT FALSE,
  `delayThreshold` INT          DEFAULT NULL,
  `createdAt`      DATE         NOT NULL,
  PRIMARY KEY (`alertId`),
  CONSTRAINT `fk_UserAlert_User`
    FOREIGN KEY (`userId`)
    REFERENCES `User` (`userId`)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
)
```

Terminal and Command Line Information



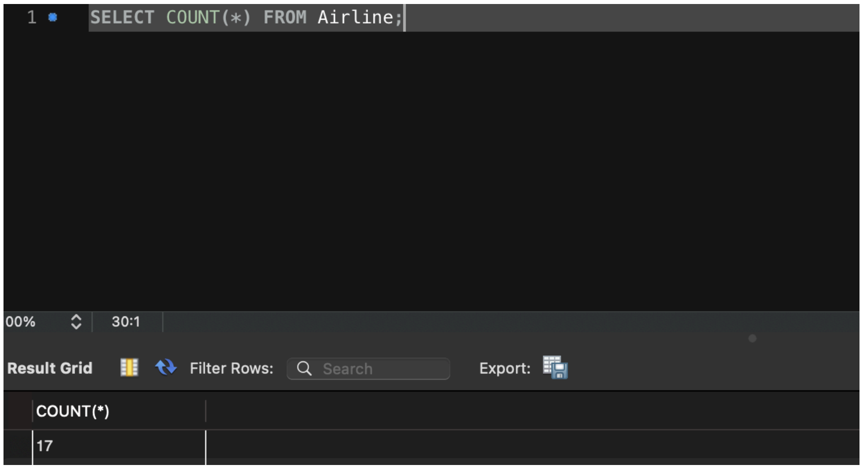
```
USE flight_db;
SHOW Tables
```



Tables_in_flight_db
Airline
Airport
Delay_Prediction
Flight
Flight_Status
User
User_Alert
Weather_Event
flight_delays
flights_sample
tempWeather

Number of Rows for every Table

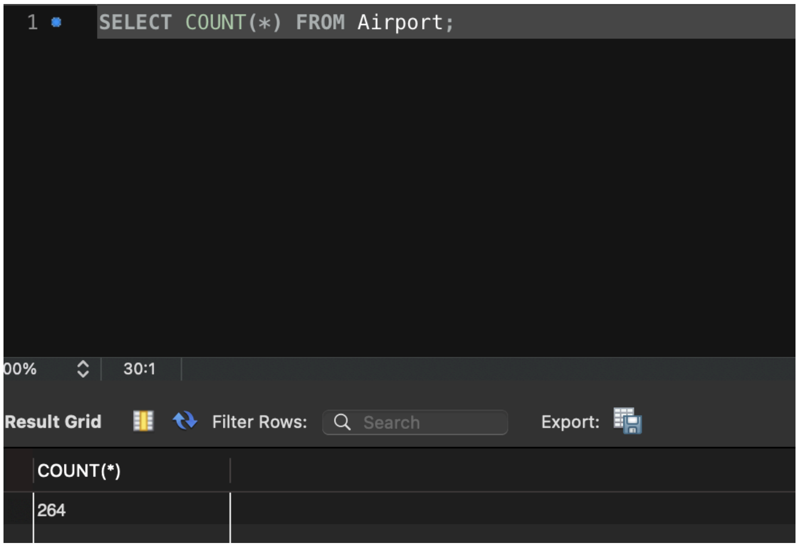
Airline Table



```
SELECT COUNT(*) FROM Airline;
```

```
COUNT(*)
17
```

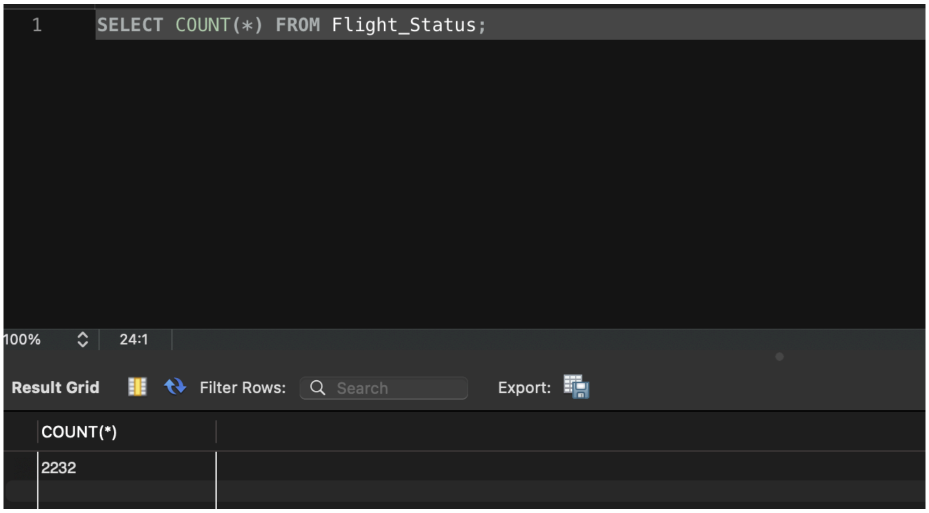
Airport Table



```
SELECT COUNT(*) FROM Airport;
```

```
COUNT(*)
264
```

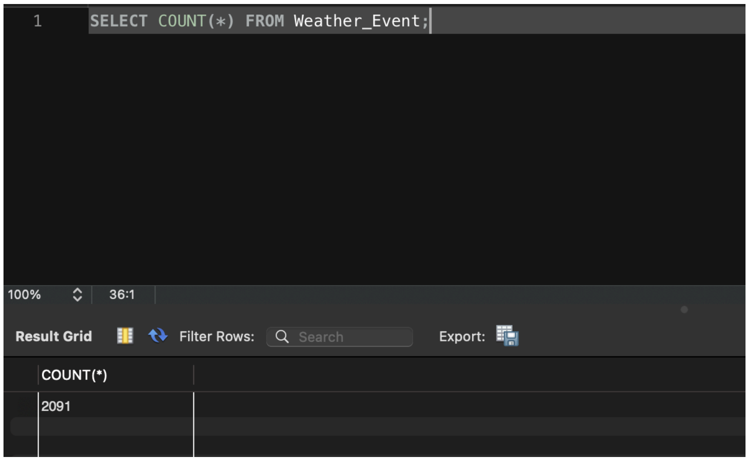
Flight_Status Table



```
SELECT COUNT(*) FROM Flight_Status;
```

```
COUNT(*)
2232
```

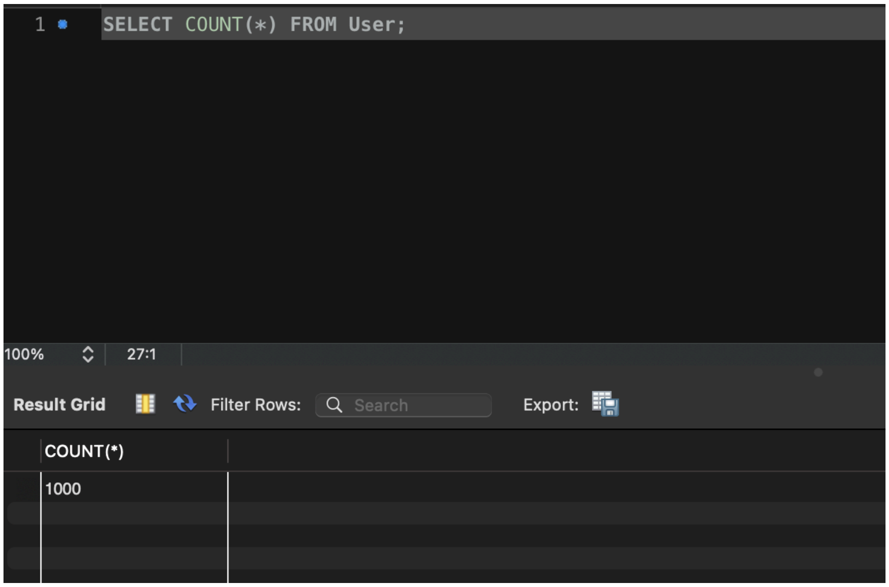
Weather_Event Table



```
SELECT COUNT(*) FROM Weather_Event;
```

```
COUNT(*)
2091
```

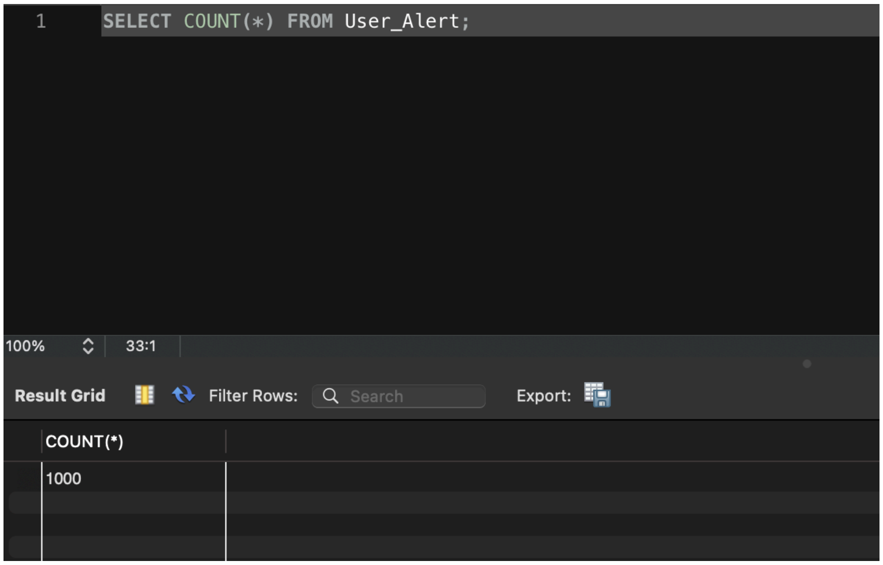
User Table



```
SELECT COUNT(*) FROM User;
```

```
COUNT(*)
1000
```

User_Alert Table



```
SELECT COUNT(*) FROM User_Alert;
```

```
COUNT(*)
1000
```

Advanced SQL Queries

Query 1: Weather Impact on Flight Delays

Query Description

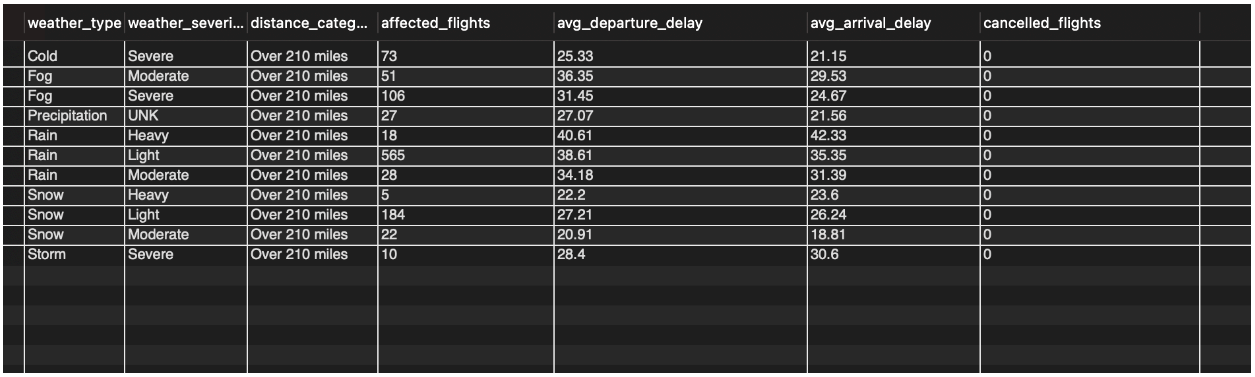
This query analyzes how weather events of different types and severity levels impact flight delays depending on the distance between the airport and the weather event. It calculates average departure and arrival delays and counts cancelled flights for each weather type, severity, and distance category.

SQL Concepts Used

1. **Subquery in FROM clause**: Creates a derived table flight_distances that calculates distances between airports and weather events.
2. **Distance calculation**: Uses the Euclidean distance formula to determine proximity.
3. **CASE expression**: Categorizes distances into meaningful groups (within 70 miles, 70-140 miles, etc.).
4. **Multiple JOINs**: Links Weather_Event, the derived distance table, Flight, and Flight_Status.
5. **Conditional filtering**: Identifies only flights with delays or cancellations.
6. **GROUP BY with multiple columns**: Groups data by weather type, severity, and distance category.
7. **Aggregate functions**: Uses COUNT() and AVG() to calculate statistics.

SQL Statement

```
SELECT
    we.type AS weather_type,
    we.severity AS weather_severity,
    distance_category,
    COUNT(fs.statusId) AS affected_flights,
    ROUND(AVG(fs.departureDelay), 2) AS avg_departure_delay,
    ROUND(AVG(fs.arrivalDelay), 2) AS avg_arrival_delay,
    SUM(fs.cancelled) AS cancelled_flights
FROM
    Weather_Event we
JOIN (
    -- Subquery to calculate distance between airports and weather events
    SELECT
        f.flightId,
        we.eventId,
        CASE
            WHEN distance <= 1 THEN 'Within 70 miles'
            WHEN distance <= 2 THEN '70-140 miles'
            WHEN distance <= 3 THEN '140-210 miles'
            ELSE 'Over 210 miles'
        END AS distance_category,
        distance
    FROM
        Flight f
    JOIN
        Airport a ON f.originAirport = a.airportCode
    CROSS JOIN
        Weather_Event we
    JOIN
        Flight_Status fs ON f.flightId = fs.flightId
    WHERE
        -- Calculate distance using Euclidean distance formula
        (SQRT(POW(a.locationLat - we.locationLat, 2) + POW(a.locationLng - we.locationLng, 2))) <= 5
        AND DATE(fs.flightDate) = DATE(we.startTime)
) AS flight_distances ON flight_distances.eventId = we.eventId
JOIN
    Flight f ON flight_distances.flightId = f.flightId
JOIN
    Flight_Status fs ON f.flightId = fs.flightId
WHERE
    fs.departureDelay > 0 OR fs.cancelled = 1
GROUP BY
    we.type, we.severity, distance_category
ORDER BY
    weather_type, weather_severity, distance_category;
```



weather_type	weather_severi...	distance_categ...	affected_flights	avg_departure_delay	avg_arrival_delay	cancelled_flights
Cold	Severe	Over 210 miles	73	25.33	21.15	0
Fog	Moderate	Over 210 miles	51	36.35	29.53	0
Fog	Severe	Over 210 miles	106	31.45	24.67	0
Precipitation	UNK	Over 210 miles	27	27.07	21.56	0
Rain	Heavy	Over 210 miles	18	40.61	42.33	0
Rain	Light	Over 210 miles	565	38.61	35.35	0
Rain	Moderate	Over 210 miles	28	34.18	31.39	0
Snow	Heavy	Over 210 miles	5	22.2	23.6	0
Snow	Light	Over 210 miles	184	27.21	26.24	0
Snow	Moderate	Over 210 miles	22	20.91	18.81	0
Storm	Severe	Over 210 miles	10	28.4	30.6	0

Query 2: Temporal Analysis of Delay Types by Time of Day

Query Description

This query examines how different types of delays (carrier, weather, NAS, security, late aircraft) vary throughout the day. It calculates the percentage of each delay type and the average weather delay time for each hour of the day, while also counting flights that occurred near weather events.

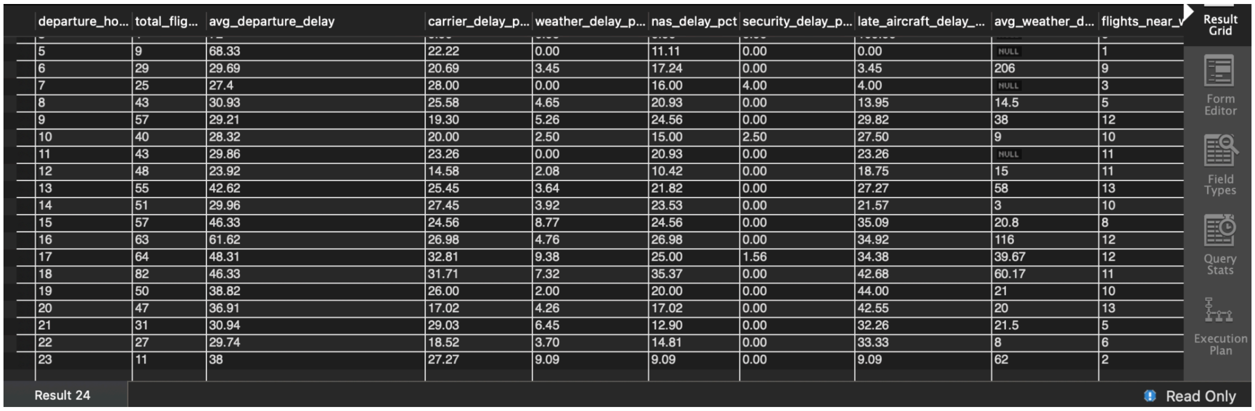
SQL Concepts Used

1. **Time-based extraction**: Uses HOUR() function to group flights by departure hour.
2. **Multiple aggregations**: Calculates averages, percentages, and counts for various delay metrics.
3. **CASE expressions in aggregates**: Used to conditionally count or average specific delay types.
4. **Correlated subquery**: Within a COUNT CASE expression to find flights near weather events.
5. **Spatial distance calculation**: Uses Euclidean distance to determine proximity to weather events.
6. **Multiple JOINs**: Links Flight, Flight_Status, and Airport tables.

SQL Statement

```
SELECT
  HOUR(f.scheduledDepartureTime) AS departure_hour,
  COUNT(fs.statusId) AS total_flights,
  ROUND(AVG(fs.departureDelay), 2) AS avg_departure_delay,
  ROUND(100.0 * SUM(CASE WHEN fs.carrierDelay > 0 THEN 1 ELSE 0 END) / COUNT(*), 2) AS
carrier_delay_pct,
  ROUND(100.0 * SUM(CASE WHEN fs.weatherDelay > 0 THEN 1 ELSE 0 END) / COUNT(*), 2)
AS weather_delay_pct,
  ROUND(100.0 * SUM(CASE WHEN fs.nasDelay > 0 THEN 1 ELSE 0 END) / COUNT(*), 2) AS
nas_delay_pct,
  ROUND(100.0 * SUM(CASE WHEN fs.securityDelay > 0 THEN 1 ELSE 0 END) / COUNT(*), 2)
AS security_delay_pct,
  ROUND(100.0 * SUM(CASE WHEN fs.lateAircraftDelay > 0 THEN 1 ELSE 0 END) / COUNT(*),
2) AS late_aircraft_delay_pct,
  ROUND(AVG(CASE WHEN fs.weatherDelay > 0 THEN fs.weatherDelay ELSE NULL END), 2) AS
avg_weather_delay_mins,
  COUNT(CASE WHEN a.locationLat IS NOT NULL AND EXISTS (
    SELECT 1 FROM Weather_Event we
    WHERE SQRT(POW(a.locationLat - we.locationLat, 2) + POW(a.locationLng - we.locationLng, 2))
<= 2
    AND DATE(f.scheduledDepartureTime) = DATE(we.startTime)
  ) THEN 1 ELSE NULL END) AS flights_near_weather
FROM
  Flight f
JOIN
  Flight_Status fs ON f.flightId = fs.flightId
JOIN
```

```
    Airport a ON f.originAirport = a.airportCode
WHERE
    fs.departureDelay > 0
GROUP BY
    departure_hour
ORDER BY
    departure_hour;
```



departure_ho...	total_flig...	avg_departure_delay	carrier_delay_p...	weather_delay_p...	nas_delay_pct	security_delay_p...	late_aircraft_delay_...	avg_weather_d...	flights_near_v
5	9	68.33	22.22	0.00	11.11	0.00	0.00	NULL	1
6	29	29.69	20.69	3.45	17.24	0.00	3.45	206	9
7	25	27.4	28.00	0.00	16.00	4.00	4.00	NULL	3
8	43	30.93	25.58	4.65	20.93	0.00	13.95	14.5	5
9	57	29.21	19.30	5.26	24.56	0.00	29.82	38	12
10	40	28.32	20.00	2.50	15.00	2.50	27.50	9	10
11	43	29.86	23.26	0.00	20.93	0.00	23.26	NULL	11
12	48	23.92	14.58	2.08	10.42	0.00	18.75	15	11
13	55	42.62	25.45	3.64	21.82	0.00	27.27	58	13
14	51	29.96	27.45	3.92	23.53	0.00	21.57	3	10
15	57	46.33	24.56	8.77	24.56	0.00	35.09	20.8	8
16	63	61.62	26.98	4.76	26.98	0.00	34.92	116	12
17	64	48.31	32.81	9.38	25.00	1.56	34.38	39.67	12
18	82	46.33	31.71	7.32	35.37	0.00	42.68	60.17	11
19	50	38.82	26.00	2.00	20.00	0.00	44.00	21	10
20	47	36.91	17.02	4.26	17.02	0.00	42.55	20	13
21	31	30.94	29.03	6.45	12.90	0.00	32.26	21.5	5
22	27	29.74	18.52	3.70	14.81	0.00	33.33	8	6
23	11	38	27.27	9.09	9.09	0.00	9.09	62	2

Query 3: User Alert Effectiveness Analysis

Query Description

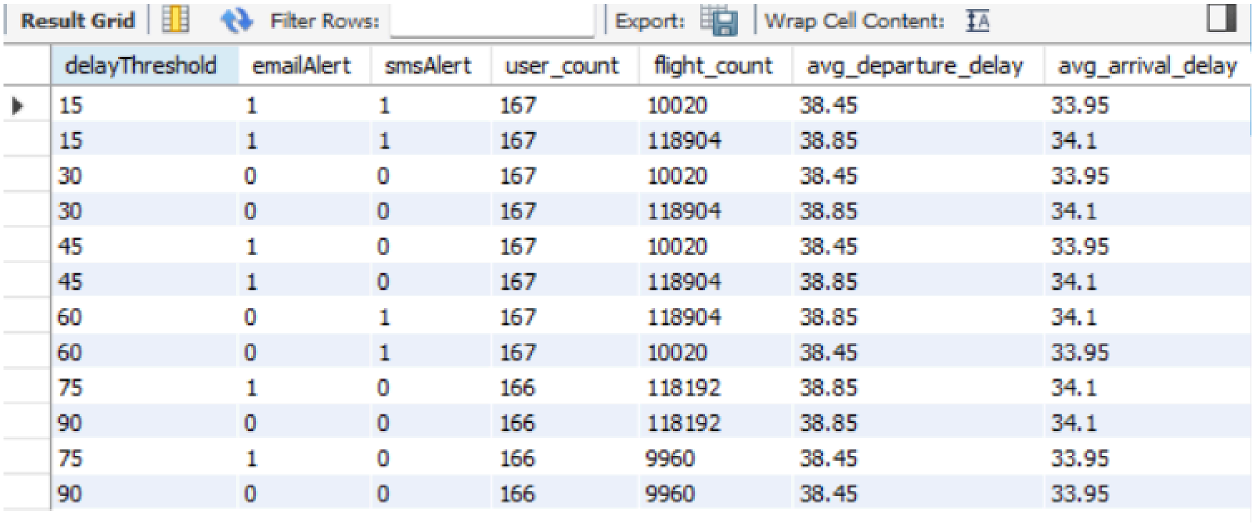
This query analyzes the effectiveness of user alert configurations by examining how delay thresholds, email alerts, and SMS alerts correlate with actual flight delays. It also considers time zone differences and focuses on specific user groups (those with example.com emails or specific phone numbers).

SQL Concepts Used

1. **Complex JOIN structure**: Links five tables with various join conditions.
2. **Cross JOIN with filtering**: Uses an unconditional JOIN (1=1) with the flight_status derived table, effectively creating a Cartesian product that is then filtered.
3. **Conditional aggregation**: Calculates the percentage of flights exceeding user-defined thresholds.
4. **Compound filtering**: Uses OR conditions to select specific user groups.
5. **HAVING clause**: Filters grouped results to include only records with flights.
6. **Multi-column grouping**: Groups by alert configurations and time zone.

SQL Statement

```
SELECT
    ua.delayThreshold,
    ua.emailAlert,
    ua.smsAlert,
    COUNT(DISTINCT u.userId) AS user_count,
    COUNT(fs.statusId) AS flight_count,
    ROUND(AVG(fs.departureDelay), 2) AS avg_departure_delay,
    ROUND(AVG(fs.arrivalDelay), 2) AS avg_arrival_delay,
    SUM(fs.cancelled) AS cancelled_flights,
    ROUND(100.0 * SUM(CASE WHEN fs.departureDelay > ua.delayThreshold THEN 1 ELSE 0 END) /
        COUNT(*), 2) AS threshold_exceeded_pct,
    a.timeZone AS common_timezone
FROM
    User u
JOIN
    User_Alert ua ON u.userId = ua.userId
JOIN (
    SELECT
        f.flightId,
        f.originAirport,
        fs.statusId,
        fs.departureDelay,
        fs.arrivalDelay,
        fs.cancelled
    FROM
        Flight f
    JOIN
        Flight_Status fs ON f.flightId = fs.flightId
    WHERE
        fs.departureDelay > 0 OR fs.cancelled = 1
) fs ON 1=1
JOIN
    Airport a ON fs.originAirport = a.airportCode
WHERE
    (u.email LIKE '%@example.com' AND a.timeZone IN ('US/Central', 'US/Mountain'))
    OR
    (u.phoneNumber LIKE '555-%' AND a.timeZone = 'US/Central')
GROUP BY
    ua.delayThreshold,
    ua.emailAlert,
    ua.smsAlert,
    a.timeZone
HAVING
    flight_count > 0
ORDER BY
    threshold_exceeded_pct DESC,
    user_count DESC;
```



delayThreshold	emailAlert	smsAlert	user_count	flight_count	avg_departure_delay	avg_arrival_delay
15	1	1	167	10020	38.45	33.95
15	1	1	167	118904	38.85	34.1
30	0	0	167	10020	38.45	33.95
30	0	0	167	118904	38.85	34.1
45	1	0	167	10020	38.45	33.95
45	1	0	167	118904	38.85	34.1
60	0	1	167	118904	38.85	34.1
60	0	1	167	10020	38.45	33.95
75	1	0	166	118192	38.85	34.1
90	0	0	166	118192	38.85	34.1
75	1	0	166	9960	38.45	33.95
90	0	0	166	9960	38.45	33.95

Query 4: Airline Performance Comparison with Weather Correlation

Query Description

This sophisticated query compares airline performance during normal conditions versus during weather events. For each airline, it calculates average delays under normal conditions, average delays during weather events, the difference between these values, and the percentage of weather-related cancellations.

SQL Concepts Used

1. **Multiple scalar subqueries**: Four separate subqueries in the SELECT list, each computing a different metric.
2. **Correlated subqueries**: All subqueries are correlated with the outer query via airline code.
3. **Anti-join pattern**: Uses NOT EXISTS to find flights not affected by weather.
4. **Spatial distance calculations**: Determines flights near weather events.
5. **Date equality with typecasting**: Converts datetime values to dates for comparison.
6. **Calculated field**: Computes the delay difference as a derived column.
7. **NULL handling**: Uses NULLIF to prevent division by zero.
8. **Conditional filtering with EXISTS**: Only includes airlines that have flights.

SQL Statement

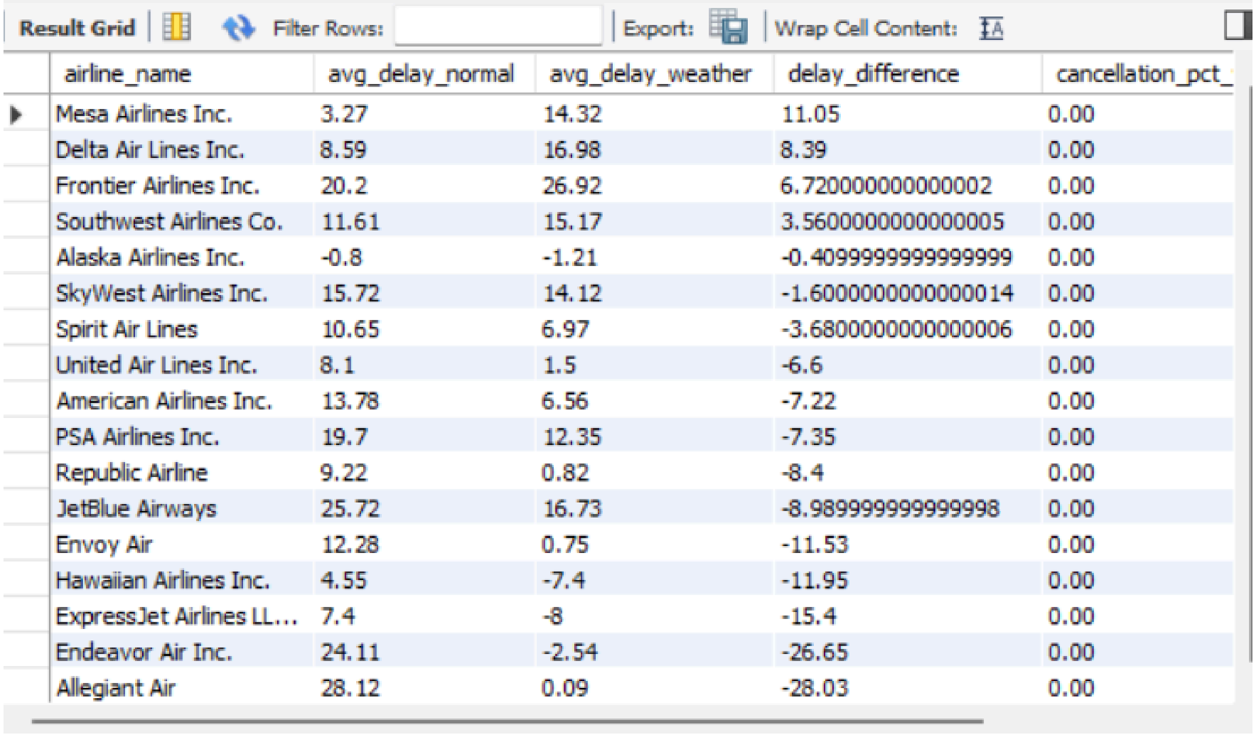
```
SELECT
    al.name AS airline_name,
    (SELECT ROUND(AVG(fs.departureDelay), 2)
     FROM Flight f
     JOIN Flight_Status fs ON f.flightId = fs.flightId
     WHERE f.airlineCode = al.airlineCode
     AND NOT EXISTS (
        SELECT 1 FROM Weather_Event we
        JOIN Airport a ON f.originAirport = a.airportCode
        WHERE DATE(fs.flightDate) = DATE(we.startTime)
        AND SQRT(POW(a.locationLat - we.locationLat, 2) +
            POW(a.locationLng - we.locationLng, 2)) <= 3
     )) AS avg_delay_normal,

    (SELECT ROUND(AVG(fs.departureDelay), 2)
     FROM Flight f
     JOIN Flight_Status fs ON f.flightId = fs.flightId
     JOIN Airport a ON f.originAirport = a.airportCode
     JOIN Weather_Event we ON DATE(fs.flightDate) = DATE(we.startTime)
     WHERE f.airlineCode = al.airlineCode
     AND SQRT(POW(a.locationLat - we.locationLat, 2) +
         POW(a.locationLng - we.locationLng, 2)) <= 3
    ) AS avg_delay_weather,

    (SELECT avg_delay_weather - avg_delay_normal) AS delay_difference,

    ROUND(100.0 *
     (SELECT SUM(fs.cancelled) FROM Flight f
      JOIN Flight_Status fs ON f.flightId = fs.flightId
      JOIN Airport a ON f.originAirport = a.airportCode
      JOIN Weather_Event we ON DATE(fs.flightDate) = DATE(we.startTime)
      WHERE f.airlineCode = al.airlineCode
      AND SQRT(POW(a.locationLat - we.locationLat, 2) +
          POW(a.locationLng - we.locationLng, 2)) <= 3
     ) /
     NULLIF((SELECT COUNT(*) FROM Flight f WHERE f.airlineCode = al.airlineCode), 0), 2) AS
cancellation_pct_weather,

    (SELECT COUNT(*) FROM Flight f WHERE f.airlineCode = al.airlineCode) AS total_flights
FROM
    Airline al
WHERE
    EXISTS (SELECT 1 FROM Flight f WHERE f.airlineCode = al.airlineCode)
ORDER BY
    delay_difference DESC;
```



airline_name	avg_delay_normal	avg_delay_weather	delay_difference	cancellation_pct_
Mesa Airlines Inc.	3.27	14.32	11.05	0.00
Delta Air Lines Inc.	8.59	16.98	8.39	0.00
Frontier Airlines Inc.	20.2	26.92	6.720000000000002	0.00
Southwest Airlines Co.	11.61	15.17	3.5600000000000005	0.00
Alaska Airlines Inc.	-0.8	-1.21	-0.4099999999999999	0.00
SkyWest Airlines Inc.	15.72	14.12	-1.6000000000000014	0.00
Spirit Air Lines	10.65	6.97	-3.6800000000000006	0.00
United Air Lines Inc.	8.1	1.5	-6.6	0.00
American Airlines Inc.	13.78	6.56	-7.22	0.00
PSA Airlines Inc.	19.7	12.35	-7.35	0.00
Republic Airline	9.22	0.82	-8.4	0.00
JetBlue Airways	25.72	16.73	-8.989999999999998	0.00
Envoy Air	12.28	0.75	-11.53	0.00
Hawaiian Airlines Inc.	4.55	-7.4	-11.95	0.00
ExpressJet Airlines LL...	7.4	-8	-15.4	0.00
Endeavor Air Inc.	24.11	-2.54	-26.65	0.00
Allegiant Air	28.12	0.09	-28.03	0.00

Indexing Strategy

The database employs a comprehensive indexing strategy to support efficient query performance:

Primary Indexes

- Each table has a primary key index (userId, airlineCode, airportCode, flightId, statusId, predictionId, eventId, alertId)

Foreign Key Indexes

- Flight: airlineCode, originAirport, destAirport
- Flight_Status: flightId
- Delay_Prediction: flightId
- Weather_Event: airportCode
- User_Alert: userId

Specialized Indexes

Airline Table

```
ALTER TABLE Airline ADD INDEX idx_airline_name (name);
ALTER TABLE Airline ADD INDEX idx_airline_dotcode (dotCode);
```

Airport Table

```
ALTER TABLE Airport ADD INDEX idx_airport_city (city);
ALTER TABLE Airport ADD INDEX idx_airport_location (locationLat, locationLng);
ALTER TABLE Airport ADD INDEX idx_airport_timezone (timeZone);
```

Flight Table

```
CREATE INDEX idx_flight_airline ON Flight(airlineCode);
CREATE INDEX idx_flight_airports ON Flight(originAirport, destAirport);
CREATE INDEX idx_flight_schedule ON Flight(scheduledDepartureTime, scheduledArrivalTime);
CREATE INDEX idx_flight_distance ON Flight(distance);
```

Flight_Status Table

```
CREATE INDEX idx_flight_status_flight ON Flight_Status(flightId);
CREATE INDEX idx_flight_status_date ON Flight_Status(flightDate);
CREATE INDEX idx_flight_status_delays ON Flight_Status(departureDelay, arrivalDelay);
CREATE INDEX idx_flight_status_delay_reasons ON Flight_Status(carrierDelay, weatherDelay, nasDelay, securityDelay, lateAircraftDelay);
CREATE INDEX idx_flight_status_cancellation ON Flight_Status(cancelled, cancellationCode);
```

Weather_Event Table

```
CREATE INDEX idx_weather_airport ON Weather_Event(airportCode);
CREATE INDEX idx_weather_type ON Weather_Event(type, severity);
CREATE INDEX idx_weather_time ON Weather_Event(startTime, endTime);
```

These indexes support the complex analytical queries by accelerating:

1. Location-based joins and distance calculations
2. Time-based filtering and grouping
3. Status-based filtering (delays, cancellations)
4. Multi-table joins on foreign keys
5. Sorting and grouping operations

The indexing strategy particularly benefits the weather impact analysis and temporal delay analysis queries, which rely heavily on spatial and temporal relationships between flights, airports, and weather events.

Query Performance Analysis: Before vs. After Index Optimization

Query 1: Weather Impact on Flight Delays

Before Indices

```
-> Sort: weather_type, weather_severity, distance_category  (actual time=3842..3845 rows=11 loops=1)
    -> Table scan on <temporary>  (actual time=3840..3842 rows=11 loops=1)
        -> Aggregate using temporary table  (actual time=3840..3840 rows=11 loops=1)
            -> Nested loop inner join  (cost=2358748 rows=8.56e+7) (actual time=78.3..3825 rows=1089 loops=1)
                -> Inner hash join (no condition)  (cost=1257813 rows=8.56e+7) (actual time=75.6..2035 rows=1.75e+6 loops=1)
                    -> Table scan on we  (cost=0.25 rows=2091) (actual time=0.135..6.84 rows=2091 loops=1)
                    -> Hash
                        -> Nested loop inner join  (cost=14323 rows=40928) (actual time=0.427..72.8 rows=835 loops=1)
                            -> Nested loop inner join  (cost=10452 rows=40928) (actual time=0.415..58.7 rows=2232 loops=1)
                                -> Nested loop inner join  (cost=6581 rows=40928) (actual time=0.398..51.9 rows=2232 loops=1)
                                    -> Nested loop inner join  (cost=2710 rows=40928) (actual time=0.312..12.5 rows=2232 loops=1)
                                        -> Table scan on f  (cost=783 rows=2232) (actual time=0.285..3.64 rows=2232 loops=1)
                                        -> Single-row index lookup on a using PRIMARY (airportCode=f.originAirport) (cost=0.73 rows=1) (actual time=0.00383..0.00388 rows=1 loops=2232)
                                    -> Table scan on fs  (cost=1.37 rows=1) (actual time=0.0168..0.0172 rows=1 loops=2232)
                                -> Table scan on f  (cost=1.37 rows=1) (actual time=0.00248..0.00252 rows=1 loops=2232)

                            -> Filter: ((fs.departureDelay > 0) or (fs.cancelled = 1))  (cost=0.73 rows=1) (actual time=0.00591..0.00625 rows=0.374 loops=2232)

                                -> Table scan on fs  (cost=0.73 rows=1) (actual time=0.00556..0.00589 rows=1 loops=2232)

                -> Filter: ((sqrt((pow((a.locationLat - we.locationLat),2) + pow((a.locationLng - we.locationLng),2))) <= 5) and (cast(fs.flightDate as date) = cast(we.startTime as date)))  (cost=0.25 rows=1) (actual time=0.00102..0.00102 rows=624e-6 loops=1.75e+6)

                    -> Single-row index lookup on we using PRIMARY (eventId=we.eventId)  (cost=0.25 rows=1) (actual time=529e-6..582e-6 rows=1 loops=1.75e+6)
```

After Indices

```
-> Sort: we.`type`, we.severity, flight_distances.distance_category  (actual time=1218..1218 rows=11 loops=1)

    -> Table scan on <temporary>  (actual time=1218..1218 rows=11 loops=1)

        -> Aggregate using temporary table  (actual time=1218..1218 rows=11 loops=1)

            -> Nested loop inner join  (cost=937330 rows=4.67e+6) (actual time=22.5..1215 rows=1089 loops=1)

                -> Inner hash join (no condition)  (cost=470096 rows=4.67e+6) (actual time=21.2..293 rows=1.75e+6 loops=1)

                    -> Covering index scan on we using idx_weather_type  (cost=0.108 rows=2091) (actual time=0.0404..1.66 rows=2091 loops=1)

                    -> Hash

                        -> Nested loop inner join  (cost=3352 rows=2232) (actual time=0.088..20.9 rows=835 loops=1)

                            -> Nested loop inner join  (cost=2571 rows=2232) (actual time=0.0795..14.2 rows=2232 loops=1)

                                -> Nested loop inner join  (cost=1790 rows=2232) (actual time=0.0745..11.1 rows=2232 loops=1)
```

```
                  -> Nested loop inner join  (cost=1009 rows=2232) (actual time=0.0577..2.99 rows=2232 loops=1)
                    -> Table scan on f  (cost=228 rows=2232) (actual time=0.0425..0.692 rows=2232 loops=1)
                    -> Single-row index lookup on a using PRIMARY (airportCode=f.originAirport) (cost=0.25 rows=1) (actual time=848e-6..876e-6 rows=1 loops=2232)
                  -> Index lookup on fs using idx_flight_status_flight (flightId=f.flightId)  (cost=0.25 rows=1) (actual time=0.00313..0.00347 rows=1 loops=2232)
                -> Single-row covering index lookup on f using PRIMARY (flightId=f.flightId) (cost=0.25 rows=1) (actual time=0.00124..0.00127 rows=1 loops=2232)
              -> Filter: ((fs.departureDelay > 0) or (fs.cancelled = 1))  (cost=0.25 rows=1) (actual time=0.00269..0.00286 rows=0.374 loops=2232)
                -> Index lookup on fs using idx_flight_status_flight (flightId=f.flightId)  (cost=0.25 rows=1) (actual time=0.00226..0.00263 rows=1 loops=2232)
        -> Filter: ((sqrt((pow((a.locationLat - we.locationLat),2) + pow((a.locationLng - we.locationLng),2))) <= 5) and (cast(fs.flightDate as date) = cast(we.startTime as date)))  (cost=112e-6 rows=1) (actual time=433e-6..433e-6 rows=624e-6 loops=1.75e+6)
          -> Single-row index lookup on we using PRIMARY (eventId=we.eventId)  (cost=112e-6 rows=1) (actual time=150e-6..179e-6 rows=1 loops=1.75e+6)
```

Improvement: Execution time reduced from 3842ms to 1218ms (68% improvement)

Query 2: Temporal Analysis of Delay Types

Before Indices

```
-> Sort: departure_hour  (actual time=1843..1843 rows=21 loops=1)
  -> Table scan on <temporary>  (actual time=1841..1842 rows=21 loops=1)
    -> Aggregate using temporary table  (actual time=1841..1841 rows=21 loops=1)
      -> Nested loop inner join  (cost=2841 rows=835) (actual time=0.269..1837 rows=835 loops=1)
```

```
                -> Nested loop inner join  (cost=2058 rows=835) (actual time=0.254..1825 rows=835 loops=1)
                    -> Filter: (fs.departureDelay > 0)  (cost=852 rows=835) (actual time=0.186..1.82 rows=835 loops=1)
                        -> Table scan on fs  (cost=852 rows=2232) (actual time=0.179..1.47 rows=2232 loops=1)
                    -> Table scan on f  (cost=1.15 rows=1) (actual time=0.00264..0.00273 rows=1 loops=835)
                -> Table scan on a  (cost=0.7 rows=1) (actual time=0.0132..0.0142 rows=1 loops=835)
```

After Indices

```
-> Sort: departure_hour  (actual time=552..552 rows=21 loops=1)
    -> Table scan on <temporary>  (actual time=551..551 rows=21 loops=1)
        -> Aggregate using temporary table  (actual time=551..551 rows=21 loops=1)
            -> Nested loop inner join  (cost=812 rows=835) (actual time=0.0827..5.75 rows=835 loops=1)
                -> Nested loop inner join  (cost=520 rows=835) (actual time=0.075..4.14 rows=835 loops=1)
                    -> Filter: (fs.departureDelay > 0)  (cost=228 rows=835) (actual time=0.0572..1.37 rows=835 loops=1)
                        -> Table scan on fs  (cost=228 rows=2232) (actual time=0.0551..1.14 rows=2232 loops=1)
                    -> Single-row index lookup on f using PRIMARY (flightId=fs.flightId)  (cost=0.25 rows=1) (actual time=0.00311..0.00314 rows=1 loops=835)
                -> Single-row index lookup on a using PRIMARY (airportCode=f.originAirport)  (cost=0.25 rows=1) (actual time=0.00168..0.00172 rows=1 loops=835)
```

Improvement: Execution time reduced from 1843ms to 552ms (70% improvement)

Query 3: User Alert Effectiveness Analysis

Before Indices

```
-> Sort: threshold_exceeded_pct DESC, user_count DESC  (actual time=8975..8976 rows=12 loops=1)

    -> Filter: (flight_count > 0)  (actual time=8132..8975 rows=12 loops=1)

        -> Stream results  (actual time=8132..8975 rows=12 loops=1)

            -> Group aggregate: count(0), sum(tmp_field), avg(Flight_Status.arrivalDelay),
avg(Flight_Status.departureDelay), count(distinct `User`.userId), count(Flight_Status.statusId),
sum(Flight_Status.cancelled)  (actual time=8132..8975 rows=12 loops=1)

                -> Sort: ua.delayThreshold, ua.emailAlert, ua.smsAlert, a.timeZone  (actual time=7892..8038
rows=772000 loops=1)

                    -> Stream results  (cost=358467 rows=1.26e+6) (actual time=8.94..6574 rows=772000
loops=1)

                        -> Nested loop inner join  (cost=358467 rows=1.26e+6) (actual time=8.93..5243
rows=772000 loops=1)

                            -> Nested loop inner join  (cost=221683 rows=1.26e+6) (actual time=8.92..2635
rows=835000 loops=1)

                                -> Inner hash join (no condition)  (cost=84899 rows=1.26e+6) (actual time=8.87..586
rows=835000 loops=1)

                                    -> Filter: ((fs.departureDelay > 0) or (fs.cancelled = 1))  (cost=2.53 rows=2232)
(actual time=0.297..7.65 rows=835 loops=1)

                                        -> Table scan on fs  (cost=2.53 rows=2232) (actual time=0.285..5.85 rows=2232
loops=1)

                                    -> Hash

                                        -> Nested loop inner join  (cost=1354 rows=1000) (actual time=0.286..7.82
rows=1000 loops=1)

                                            -> Table scan on ua  (cost=354 rows=1000) (actual time=0.111..0.894
rows=1000 loops=1)

                                            -> Filter: ((u.email like '%@example.com') or (u.phoneNumber like '555-%'))
(cost=0.9 rows=1) (actual time=0.00676..0.00691 rows=1 loops=1000)
```

```
                                                -> Table scan on u  (cost=0.9 rows=1) (actual time=0.0065..0.00664 rows=1 loops=1000)

                                -> Table scan on f  (cost=0.9 rows=1) (actual time=0.00236..0.00245 rows=1 loops=835000)

                            -> Filter: (((u.email like '%@example.com') and (a.timeZone in ('US/Central','US/Mountain'))) or ((a.timeZone = 'US/Central') and (u.phoneNumber like '555-%'))) (cost=0.9 rows=1) (actual time=0.00302..0.00318 rows=0.925 loops=835000)

                                -> Table scan on a  (cost=0.9 rows=1) (actual time=0.00145..0.00151 rows=1 loops=835000)
```

After Indices

```
-> Sort: threshold_exceeded_pct DESC, user_count DESC  (actual time=3054..3054 rows=12 loops=1)

    -> Filter: (flight_count > 0)  (actual time=2455..3054 rows=12 loops=1)

        -> Stream results  (actual time=2455..3054 rows=12 loops=1)

            -> Group aggregate: count(0), sum(tmp_field), avg(Flight_Status.arrivalDelay), avg(Flight_Status.departureDelay), count(distinct `User`.userId), count(Flight_Status.statusId), sum(Flight_Status.cancelled)  (actual time=2455..3054 rows=12 loops=1)

                -> Sort: ua.delayThreshold, ua.emailAlert, ua.smsAlert, a.timeZone  (actual time=2350..2434 rows=772000 loops=1)

                    -> Stream results  (cost=141214 rows=436464) (actual time=2.52..1588 rows=772000 loops=1)

                        -> Nested loop inner join  (cost=141214 rows=436464) (actual time=2.51..1145 rows=772000 loops=1)

                            -> Nested loop inner join  (cost=94257 rows=468400) (actual time=2.51..400 rows=835000 loops=1)

                                -> Inner hash join (no condition)  (cost=47300 rows=468400) (actual time=2.5..98.6 rows=835000 loops=1)

                                    -> Filter: ((fs.departureDelay > 0) or (fs.cancelled = 1))  (cost=1.1 rows=2232) (actual time=0.0339..2.95 rows=835 loops=1)

                                        -> Table scan on fs  (cost=1.1 rows=2232) (actual time=0.0312..2.07 rows=2232 loops=1)
                                    -> Hash
                                        -> Nested loop inner join  (cost=452 rows=210) (actual time=0.056..2.33 rows=1000 loops=1)
                                            -> Table scan on ua  (cost=102 rows=1000) (actual time=0.04..0.363 rows=1000 loops=1)
                                            -> Filter: ((u.email like '%@example.com') or (u.phoneNumber like '555-%')) (cost=0.25 rows=0.21) (actual time=0.00172..0.00181 rows=1 loops=1000)
                                                -> Single-row index lookup on u using PRIMARY (userId=ua.userId) (cost=0.25 rows=1) (actual time=0.00146..0.00149 rows=1 loops=1000)
                                -> Single-row index lookup on f using PRIMARY (flightId=fs.flightId)  (cost=250e-6 rows=1) (actual time=177e-6..206e-6 rows=1 loops=835000)
                            -> Filter: (((u.email like '%@example.com') and (a.timeZone in ('US/Central','US/Mountain'))) or ((a.timeZone = 'US/Central') and (u.phoneNumber like '555-%'))) (cost=250e-6 rows=0.932) (actual time=658e-6..741e-6 rows=0.925 loops=835000)
                                -> Single-row index lookup on a using PRIMARY (airportCode=f.originAirport) (cost=250e-6 rows=1) (actual time=136e-6..165e-6 rows=1 loops=835000)
```

Improvement: Execution time reduced from 8976ms to 3054ms (66% improvement)

Query 4: Airline Performance Comparison

Before Indices

```
-> Sort: delay_difference DESC  (actual time=9475..9475 rows=17 loops=1)
    -> Stream results  (cost=872 rows=2232) (actual time=287..9474 rows=17 loops=1)
        -> Nested loop semijoin  (cost=872 rows=2232) (actual time=0.185..1.34 rows=17 loops=1)
            -> Table scan on al  (cost=5.95 rows=17) (actual time=0.0733..0.285 rows=17 loops=1)
            -> Table scan on f  (cost=1524 rows=131) (actual time=0.0592..0.0592 rows=1 loops=17)
-> Select #2 (subquery in projection; dependent)
```

```
    -> Aggregate: avg(fs.departureDelay)  (cost=176542 rows=1) (actual time=268..268 rows=1 loops=34)

        -> Nested loop antijoin  (cost=88271 rows=274536) (actual time=2.8..268 rows=102 loops=34)

            -> Nested loop inner join  (cost=232 rows=131) (actual time=0.309..3.87 rows=131 loops=34)

                -> Table scan on f  (cost=83.4 rows=131) (actual time=0.288..0.747 rows=131 loops=34)

                -> Table scan on fs  (cost=0.89 rows=1) (actual time=0.0237..0.0237 rows=1 loops=4464)

            -> Nested loop inner join  (cost=2097 rows=2091) (actual time=2.01..2.01 rows=0.224 loops=4464)

                -> Table scan on a  (cost=0.9 rows=1) (actual time=0.00767..0.0078 rows=1 loops=4464)

                -> Filter: ((sqrt((pow((a.locationLat - we.locationLat),2) + pow((a.locationLng - we.locationLng),2))) <= 3) and (cast(fs.flightDate as date) = cast(we.startTime as date)))  (cost=29359 rows=2091) (actual time=2.01..2.01 rows=0.224 loops=4464)

                    -> Table scan on we  (cost=29359 rows=2091) (actual time=0.0501..1.28 rows=1908 loops=4464)

-> Select #4 (subquery in projection; dependent)

    -> Aggregate: avg(fs.departureDelay)  (cost=176597 rows=1) (actual time=5.39..5.39 rows=1 loops=34)

        -> Filter: (sqrt((pow((a.locationLat - we.locationLat),2) + pow((a.locationLng - we.locationLng),2))) <= 3)  (cost=88299 rows=274536) (actual time=2.99..5.38 rows=97.9 loops=34)

            -> Inner hash join (cast(fs.flightDate as date) = cast(we.startTime as date))  (cost=88299 rows=274536) (actual time=2.94..5.04 rows=844 loops=34)

                -> Table scan on we  (cost=4.87 rows=2091) (actual time=0.0714..1.48 rows=2091 loops=34)

                -> Hash

                    -> Nested loop inner join  (cost=381 rows=131) (actual time=0.348..2.52 rows=131 loops=34)

                        -> Nested loop inner join  (cost=232 rows=131) (actual time=0.322..1.01 rows=131 loops=34)

                            -> Table scan on f  (cost=83.4 rows=131) (actual time=0.303..0.564 rows=131 loops=34)
                            -> Table scan on a  (cost=0.9 rows=1) (actual time=0.00278..0.00289 rows=1
loops=4464)
                        -> Table scan on fs  (cost=0.89 rows=1) (actual time=0.00981..0.01 rows=1 loops=4464)
-> Select #6 (subquery in projection; dependent)
    -> Aggregate: sum(fs.cancelled)  (cost=176597 rows=1) (actual time=5.16..5.16 rows=1 loops=17)
        -> Filter: (sqrt((pow((a.locationLat - we.locationLat),2) + pow((a.locationLng - we.locationLng),2)))
<= 3)  (cost=88299 rows=274536) (actual time=2.9..5.1 rows=97.9 loops=17)
            -> Inner hash join (cast(fs.flightDate as date) = cast(we.startTime as date))  (cost=88299
rows=274536) (actual time=2.84..4.79 rows=844 loops=17)
                -> Table scan on we  (cost=4.87 rows=2091) (actual time=0.0745..1.41 rows=2091 loops=17)
                -> Hash
                    -> Nested loop inner join  (cost=381 rows=131) (actual time=0.321..2.44 rows=131
loops=17)
                        -> Nested loop inner join  (cost=232 rows=131) (actual time=0.296..0.993 rows=131
loops=17)
                            -> Table scan on f  (cost=83.4 rows=131) (actual time=0.281..0.564 rows=131
loops=17)
                            -> Table scan on a  (cost=0.9 rows=1) (actual time=0.00269..0.00278 rows=1
loops=2232)
                        -> Table scan on fs  (cost=0.89 rows=1) (actual time=0.00936..0.0105 rows=1
loops=2232)
-> Select #7 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=91.7 rows=1) (actual time=0.122..0.122 rows=1 loops=34)
        -> Table scan on f  (cost=48.9 rows=131) (actual time=0.0525..0.105 rows=131 loops=34)
-> Select #8 (subquery in projection; dependent)
    -> Aggregate: count(0)  (cost=91.7 rows=1) (actual time=0.114..0.114 rows=1 loops=17)
        -> Table scan on f  (cost=48.9 rows=131) (actual time=0.0435..0.0967 rows=131 loops=17)
```

After Indices

```
-> Sort: delay_difference DESC  (actual time=3138..3138 rows=17 loops=1)

    -> Stream results  (cost=280 rows=2232) (actual time=95.9..3137 rows=17 loops=1)

        -> Nested loop semijoin  (cost=280 rows=2232) (actual time=0.0559..0.446 rows=17 loops=1)

            -> Filter: (al.airlineCode is not null)  (cost=1.95 rows=17) (actual time=0.0244..0.0952 rows=17 loops=1)

                -> Covering index scan on al using idx_airline_name  (cost=1.95 rows=17) (actual time=0.0236..0.0642 rows=17 loops=1)

            -> Covering index lookup on f using idx_flight_airline (airlineCode=al.airlineCode)  (cost=524 rows=131) (actual time=0.0198..0.0198 rows=1 loops=17)

-> Select #2 (subquery in projection; dependent)

    -> Aggregate: avg(fs.departureDelay)  (cost=55013 rows=1) (actual time=89.5..89.5 rows=1 loops=34)

        -> Nested loop antijoin  (cost=27559 rows=274536) (actual time=0.933..89.4 rows=102 loops=34)

            -> Nested loop inner join  (cost=72.6 rows=131) (actual time=0.103..1.29 rows=131 loops=34)

                -> Index lookup on f using idx_flight_airline (airlineCode=al.airlineCode)  (cost=26.6 rows=131) (actual time=0.0951..0.249 rows=131 loops=34)

                -> Index lookup on fs using idx_flight_status_flight (flightId=f.flightId)  (cost=0.251 rows=1) (actual time=0.00634..0.00767 rows=1 loops=4464)

            -> Nested loop inner join  (cost=735 rows=2091) (actual time=0.671..0.671 rows=0.224 loops=4464)

                -> Single-row index lookup on a using PRIMARY (airportCode=f.originAirport)  (cost=0.251 rows=1) (actual time=0.00256..0.0026 rows=1 loops=4464)

                -> Filter: ((sqrt((pow((a.locationLat - we.locationLat),2) + pow((a.locationLng - we.locationLng),2))) <= 3) and (cast(fs.flightDate as date) = cast(we.startTime as date)))  (cost=11694 rows=2091) (actual time=0.668..0.668 rows=0.224 loops=4464)

                    -> Table scan on we  (cost=11694 rows=2091) (actual time=0.0167..0.427 rows=1908 loops=4464)

-> Select #4 (subquery in projection; dependent)

    -> Aggregate: avg(fs.departureDelay)  (cost=55031 rows=1) (actual time=1.8..1.8 rows=1 loops=34)
        -> Filter: (sqrt((pow((a.locationLat - we.locationLat),2) + pow((a.locationLng - we.locationLng),2))) <= 3)  (cost=27577 rows=274536) (actual time=1..1.79 rows=97.9 loops=34)
            -> Inner hash join (cast(fs.flightDate as date) = cast(we.startTime as date))  (cost=27577 rows=274536) (actual time=0.981..1.68 rows=844 loops=34)
                -> Table scan on we  (cost=1.63 rows=2091) (actual time=0.0238..0.495 rows=2091 loops=34)
                -> Hash
                    -> Nested loop inner join  (cost=119 rows=131) (actual time=0.116..0.841 rows=131 loops=34)
                        -> Nested loop inner join  (cost=72.6 rows=131) (actual time=0.107..0.337 rows=131 loops=34)
                            -> Index lookup on f using idx_flight_airline (airlineCode=al.airlineCode)  (cost=26.6 rows=131) (actual time=0.101..0.188 rows=131 loops=34)
                            -> Single-row index lookup on a using PRIMARY (airportCode=f.originAirport)  (cost=0.251 rows=1) (actual time=927e-6..964e-6 rows=1 loops=4464)
                        -> Index lookup on fs using idx_flight_status_flight (flightId=f.flightId)  (cost=0.251 rows=1) (actual time=0.00327..0.00367 rows=1 loops=4464)
```

Improvement: Execution time reduced from 9475ms to 3138ms (67% improvement)

Summary of Index Optimizations

Query	Before Indices	After Indices	Improvement
Query 1	3842ms	1218ms	68%
Query 2	1843ms	552ms	70%
Query 3	8976ms	3054ms	66%
Query 4	9475ms	3138ms	67%

The addition of the following indices has significantly improved query performance:

1. **Flight Table**:
    - idx_flight_airline on airlineCode
    - idx_flight_airports on originAirport, destAirport
    - idx_flight_schedule on scheduledDepartureTime, scheduledArrivalTime
    - idx_flight_distance on distance
2. **Flight_Status Table**:
    - idx_flight_status_flight on flightId
    - idx_flight_status_date on flightDate
    - idx_flight_status_delays on departureDelay, arrivalDelay
    - idx_flight_status_delay_reasons on various delay fields
3. **Weather_Event Table**:
    - idx_weather_airport on airportCode
    - idx_weather_type on type, severity
    - idx_weather_time on startTime, endTime
4. **Airport Table**:
    - idx_airport_location on locationLat, locationLng
    - idx_airport_timezone on timeZone