CUSTOMER

Customer Registration Query:

```
SELECT * FROM Customer WHERE email = %s
```

Explanation: Checks if the email is already registered.

```
INSERT INTO Customer (
   email, thepassword, first_name, last_name, building_num, street_name,
   apt_num, city, state_name, zip_code, passport_number, passport_expiration,
   passport_country, date_of_birth
) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);
```

Explanation: Inserts customer details/information after validating that the email is unique.

Customer Login Query:

```
SELECT * FROM Customer WHERE email = %s AND thepassword = %s;
```

Explanation: Matches the entered email and hashed password with database records.

Search Flights Query:

```
SELECT
   f.flight_number,
   f.airline_name,
   f.departure_datetime,
   f.arrival_datetime,
   f.base_price,
   f.flight_status,
   a1.airport_name AS departure_airport,
   a1.city AS departure_city,
   a2.airport_name AS destination_airport,
   a2.city AS destination_city,
   ap.number_of_seats
FROM
   Flight AS f
JOIN Airport AS a1 ON f.departure_airport_code = a1.airport_code
JOIN Airport AS a2 ON f.arrival_airport_code = a2.airport_code
JOIN Airplane AS ap ON f.airplane_id = ap.airplane_id
WHERE
   a1.airport_name = %s
   AND a2.airport_name = %s
   AND DATE(f.departure_datetime) = %s
```
Explanation: After getting the intended flight information from users, I write this query to join multiple tables (Flight, Airport, Airplane) to fetch comprehensive flight details.

Ticket Count Query:

```
SELECT COUNT(*) AS tickets_sold
FROM Ticket
WHERE flight_number = %s AND departure_datetime = %s AND airline_name = %s AND
is_canceled = 0
```
Explanation: This query is to calculate the amount of tickets sold.

Backend Code:

```
capacity = flight['number_of_seats']
        if tickets_sold >= 0.8 * capacity:
            flight['calculated_price'] = int(round(flight['base_price'] * 1.25, 2))
        else:
            flight['calculated_price'] = flight['base_price']
```
Explanation: After getting the number of tickets_sold from the above query, adjust the calculated_price if the tickets_sold is greater than 80% of the total number of tickets for a flight.

Flight Check Query:

```
SELECT f.*, a.number_of_seats
FROM Flight f
JOIN Airplane a ON f.airplane_id = a.airplane_id
WHERE f.flight_number = %s AND f.departure_datetime = %s AND
LOWER(f.airline_name) = LOWER(%s)
```
Explanation: This query ensures the provided flight details are valid.

Ticket Insert Query:

```
INSERT INTO Ticket (
   flight_number, departure_datetime, airline_name, calculated_ticket_price
) VALUES (%s, %s, %s, %s)
```
Explanation: This query inserts ticket information into the ticket table after a purchase is successful.

Purchase Insert Query:

```
INSERT INTO Purchases (
    ticket_id, email, name_on_card, card_number, card_type,
    purchase_datetime, card_expiration_date, passenger_first_name,
    passenger_last_name, passenger_birthofdate
) VALUES (%s, %s, %s, %s, %s, NOW(), %s, %s, %s, %s)
```

Explanation: This query inserts all the data provided by customers into the purchases table. And for the purchase_datetime, I use the NOW() function that returns the date and time the purchase is made.

View Upcoming Flights Query:

```
SELECT
    t.flight_number,
    t.departure_datetime,
    t.airline_name,
    t.calculated_ticket_price,
    p.passenger_first_name,
    p.passenger_last_name,
    p.card_type,
    t.ticket_id
FROM
    Ticket AS t
JOIN Purchases AS p ON t.ticket_id = p.ticket_id
WHERE
    p.email = %s AND t.departure_datetime > NOW()
ORDER BY
    t.departure_datetime;
```

Explanation: This query joins the ticket and purchases table to fetch flight information so that customers can see the upcoming flight. The departure_datetime is set to be greater than NOW().

Ticket Verification Query:

```
SELECT t.ticket_id, t.departure_datetime
FROM Ticket t
JOIN Purchases p ON t.ticket_id = p.ticket_id
WHERE p.email = %s AND t.ticket_id = %s
```

Explanation: Prevents unauthorized cancellations.

Purchase Deletion Query:

```
DELETE FROM Purchases WHERE ticket_id = %s AND email = %s
```

Explanation: Makes the seat available again.

Completed Flights Query:

```
SELECT
    t.ticket_id,
    t.flight_number,
    t.airline_name,
    f.departure_datetime,
    f.arrival_datetime,
    t.calculated_ticket_price,
    f.flight_status
FROM
    Ticket AS t
JOIN Purchases AS p ON t.ticket_id = p.ticket_id
JOIN Flight AS f ON t.flight_number = f.flight_number AND t.departure_datetime =
f.departure_datetime AND t.airline_name = f.airline_name
WHERE
    p.email = %s AND f.departure_datetime < NOW()
```

Explanation: This query shows all the past flights. Departure_datetime is set to < NOW(). This flight is displayed so customers can rate those previous flights.

Insert/Update Rating Query:

```
INSERT INTO Takes (flight_number, departure_datetime, airline_name, email, comment,
rating)
VALUES (%s, %s, %s, %s, %s, %s)
ON DUPLICATE KEY UPDATE comment = %s, rating = %s;
```

Explanation: This query first gets a rating and a comment from customers. And those data are inserted into this takes table.

Past Year Total Spending Query:

```
SELECT SUM(calculated_ticket_price) AS total_spent
FROM Ticket
JOIN Purchases ON Ticket.ticket_id = Purchases.ticket_id
WHERE Purchases.email = %s AND Purchases.purchase_datetime >= DATE_SUB(NOW(),
INTERVAL 1 YEAR);
```

Explanation: This query calculates the total spending in the past year. DATE_SUB(NOW(), INTERVAL 1 YEAR) subtracts the specified interval from the current date and time.

Month-Wise Spending Query:

```
SELECT DATE_FORMAT(Purchases.purchase_datetime, '%%Y-%%m') AS month,
    SUM(calculated_ticket_price) AS total_spent
FROM Ticket
JOIN Purchases ON Ticket.ticket_id = Purchases.ticket_id
WHERE Purchases.email = %s AND Purchases.purchase_datetime >= DATE_SUB(NOW(),
INTERVAL 6 MONTH)
GROUP BY month
ORDER BY month DESC;
```

Explanation: This query retrieves monthly spending for the last 6 months. DATE_SUB(NOW(), INTERVAL 6 MONTH) sets the range to be the past 6 months.

Custom Date Range Query:

```
SELECT SUM(calculated_ticket_price) AS total_spent
FROM Ticket
JOIN Purchases ON Ticket.ticket_id = Purchases.ticket_id
WHERE Purchases.email = %s AND Purchases.purchase_datetime BETWEEN %s AND %s;
```

Explanation: This query calculates total spending in a user-specified date range.

Custom Range Month-Wise Spending Query:

```
SELECT DATE_FORMAT(Purchases.purchase_datetime, '%%Y-%%m') AS month,
    SUM(calculated_ticket_price) AS total_spent
FROM Ticket
JOIN Purchases ON Ticket.ticket_id = Purchases.ticket_id
WHERE Purchases.email = %s AND Purchases.purchase_datetime BETWEEN %s AND %s
GROUP BY month
ORDER BY month DESC;
```

Explanation: This query retrieves month-wise spending in the custom date range.

AIRLINE STAFF

Airline Staff Registration:

```
SELECT * FROM Airline_Staff WHERE username = %s
```

Checks if the inputted username already exists within the Airline_staff table. If it already exists, raise an error.

```
INSERT INTO Airline_Staff (
    username, thepassword, airline_name, first_name, last_name, date_of_birth
  ) VALUES (%s, %s, %s, %s, %s, %s)
```

Inserts a new Airline_Staff entry into the Airline_Staff table, with all the data inputted through the form.

Airline Staff View Flights:

```
SELECT f.*, a1.airport_name AS departure_airport, a2.airport_name AS arrival_airport
    FROM Flight AS f
    JOIN Airport AS a1 ON f.departure_airport_code = a1.airport_code
    JOIN Airport AS a2 ON f.arrival_airport_code = a2.airport_code
    WHERE f.airline_name = %s
    AND f.departure_datetime BETWEEN %s AND %s
```

This query is executed twice in this use case:

Default view: The query will execute using the airline_name retrieved from the session and departure datetime is automatically set to between the current time and current time + 30 days, showing all the future flights for this airline for the next 30 days.

View with inputted form. The query will execute using the airline_name retrieved from the session and the Airline Staff is able to view flights using their own parametrics.

Airline Staff Create New Flights:

```
SELECT * FROM Airplane WHERE airplane_id = %s AND airline_name = %s
```

Before creating a new flight, the application checks whether the airplane being added belongs to the airline associated with the airline staff currently logged in.

```
SELECT * FROM Maintenance
    WHERE airplane_id = %s AND airline_name = %s
    AND (start_datetime <= %s AND end_datetime >= %s)
```

This query is used to check if the airplane the staff wants to add is currently under Maintenance. If it is, deny the create flight request.

```
INSERT INTO Flight (flight_number, departure_datetime, airline_name, airplane_id,
    base_price, flight_status, arrival_airport_code, departure_airport_code, arrival_datetime)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
```

This query adds a new entry to the flight table with all the given information from the form and the session.

Airline Staff Change Status of Flight:

```
UPDATE Flight
        SET flight_status = %s, departure_datetime = DATE_ADD(departure_datetime,
INTERVAL %s MINUTE)
        WHERE flight_number = %s AND airline_name = %s
```

If you want to change a status of a flight from on-time to delayed, the application will update the flight status within the Flight table as well as update the departure datetime to be +(how long the delay it) is. The staff can specify how long.

Airline Staff Add Airplane:

```
INSERT INTO Airplane (airplane_id, airline_name, manufacturing_company, model_number,
manufacturing_date, number_of_seats)
    VALUES (%s, %s, %s, %s, %s, %s)
```

Adds a new airplane into the Airplane table using information from the inputted Form by the user staff.

```
SELECT * FROM Airplane WHERE airline_name = %s
```

This query is used on the confirmation page. After adding a new airplane, it displays all airplanes owned by the airline that the logged-in staff member works for.

Airline Staff Add Airport:

```
INSERT INTO Airport (airport_code, airport_name, city, country, num_of_terminals,
airport_type)
    VALUES (%s, %s, %s, %s, %s, %s)
```

Adds a new airport into the Airport table using information from the inputted Form by the user staff.

Airline Staff View Flight Ratings:

```
SELECT f.flight_number, AVG(t.rating) AS average_rating, GROUP_CONCAT(t.comment SEPARATOR '; ') AS
comments
  FROM Flight AS f
  LEFT JOIN Takes AS t ON f.flight_number = t.flight_number AND f.airline_name = t.airline_name
  WHERE f.airline_name = %s
  GROUP BY f.flight_number
```

This query retrieves the flight number, average rating, and comments from all the flights that the airline_name (retrieved from session) has operated.

Airline Staff Schedule Maintenance:

```
INSERT INTO Maintenance (airplane_id, airline_name, start_datetime, end_datetime)
    VALUES (%s, %s, %s, %s)
```

Inserts a new entry into the Maintenance table with all given information from the forms and session data.

Airline Staff View Frequent Customers:

```
SELECT t.email, COUNT(t.flight_number) AS flight_count
  FROM Takes AS t
  JOIN Flight AS f ON t.flight_number = f.flight_number AND t.airline_name = f.airline_name
  WHERE f.airline_name = %s AND t.departure_datetime > NOW() - INTERVAL 1 YEAR
  GROUP BY t.email
  ORDER BY flight_count DESC
  LIMIT 1
```

This query identifies the most frequent flyer for a specific airline within the past year using aggregate functions.

```
SELECT f.flight_number, f.departure_datetime, f.arrival_datetime,
          a1.airport_name AS departure_airport, a2.airport_name AS arrival_airport
     FROM Takes AS t
     JOIN Flight AS f ON t.flight_number = f.flight_number AND t.airline_name = f.airline_name
     JOIN Airport AS a1 ON f.departure_airport_code = a1.airport_code
     JOIN Airport AS a2 ON f.arrival_airport_code = a2.airport_code
     WHERE t.email = %s AND f.airline_name = %s
     ORDER BY f.departure_datetime DESC
```

This query takes an email (provided via a form) and an airline name (retrieved from the session) and returns all flights and their details that the specified customer has taken for that airline.

Airline Staff View Earned Revenue:

```
SELECT SUM(t.calculated_ticket_price) AS revenue_last_month
  FROM Ticket AS t
  JOIN Purchases AS p ON t.ticket_id = p.ticket_id
  WHERE t.airline_name = %s
  AND p.purchase_datetime BETWEEN DATE_SUB(NOW(), INTERVAL 1 MONTH) AND NOW()
```

This query returns the sum of all the sales from the past month for the given airline (retrieved from session data)

```
SELECT SUM(t.calculated_ticket_price) AS revenue_last_year
  FROM Ticket AS t
  JOIN Purchases AS p ON t.ticket_id = p.ticket_id
  WHERE t.airline_name = %s
  AND p.purchase_datetime BETWEEN DATE_SUB(NOW(), INTERVAL 1 YEAR) AND NOW()
```

This query returns the sum of all the sales from the past year for the given airline (retrieved from session data)