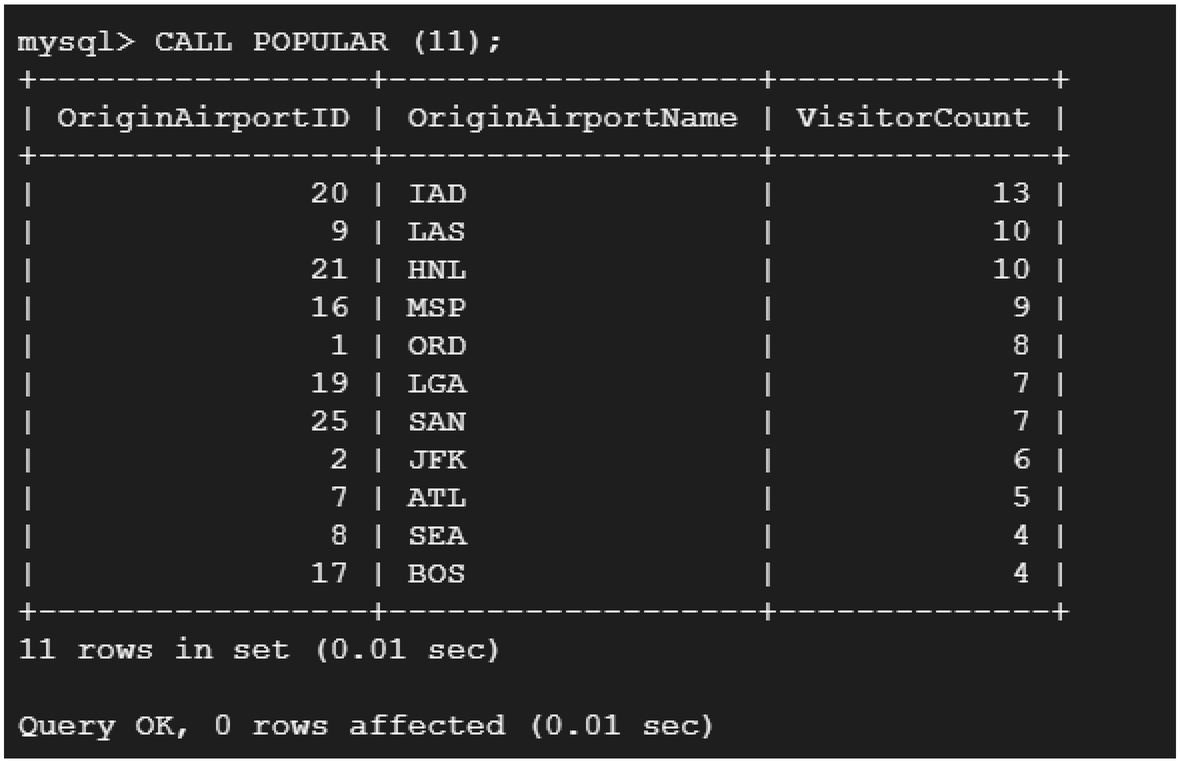
Stage 4 SQL Code

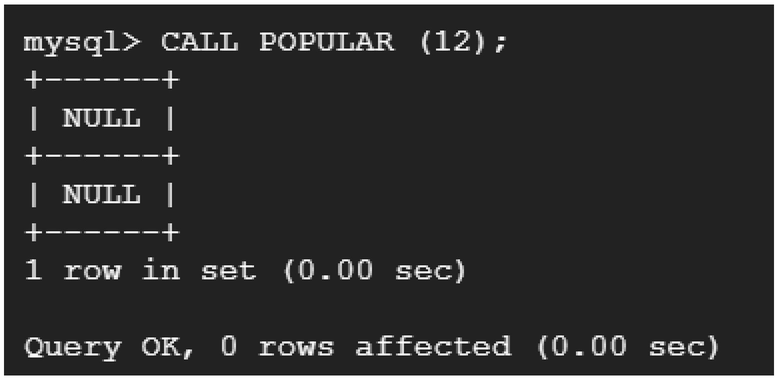
Stored Procedure:

```
DELIMITER //

CREATE PROCEDURE Popular(dest INT)
BEGIN
   DECLARE P_count INT;

SELECT COUNT(DISTINCT B.UserID)
  INTO P_count
FROM Booking B
JOIN Flight F
  ON B.FlightID = F.FlightID
GROUP BY F.Destination
HAVING F.Destination = dest;

   IF P_count > 60 THEN
      SELECT
         A2.AirportID           AS OriginAirportID,
         A2.AirportName         AS OriginAirportName,
         COUNT(DISTINCT B.UserID) AS VisitorCount
      FROM Booking B
      JOIN Flight  F  ON B.FlightID = F.FlightID
      JOIN Airport A2 ON F.Departure = A2.AirportID
      WHERE F.Destination = dest
      GROUP BY
         A2.AirportID,
         A2.AirportName
      ORDER BY
         VisitorCount DESC;

   ELSE
     SELECT NULL;
   END IF;
END
//
DELIMITER ;
```



```
mysql> CALL POPULAR (11);
+-----------------+-------------------+--------------+
| OriginAirportID | OriginAirportName | VisitorCount |
+-----------------+-------------------+--------------+
|              20 | IAD               |           13 |
|               9 | LAS               |           10 |
|              21 | HNL               |           10 |
|              16 | MSP               |            9 |
|               1 | ORD               |            8 |
|              19 | LGA               |            7 |
|              25 | SAN               |            7 |
|               2 | JFK               |            6 |
|               7 | ATL               |            5 |
|               8 | SEA               |            4 |
|              17 | BOS               |            4 |
+-----------------+-------------------+--------------+
11 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

CALL POPULAR (12);



```
mysql> CALL POPULAR (12);
+------+
| NULL |
+------+
| NULL |
+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

(how to drop the table)
DROP PROCEDURE IF EXISTS Popular;

Transaction:

```
DELIMITER //

DROP PROCEDURE IF EXISTS SaveFlightByAirportCap;
CREATE PROCEDURE SaveFlightByAirportCap(
   IN pUserID    INT,
   IN pFlightID  INT,
   IN pQuantity  INT
)
BEGIN
   DECLARE vDest        INT;
   DECLARE vPopCount    INT DEFAULT 0;
   DECLARE vFlightCount INT DEFAULT 0;

   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
   START TRANSACTION;

   SELECT Destination
     INTO vDest
   FROM Flight
   WHERE FlightID = pFlightID
   FOR UPDATE;

   IF vDest IS NULL THEN
      ROLLBACK;
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Flight not found';
   END IF;

   SELECT IFNULL(SUM(b.Quantity),0)
     INTO vPopCount
   FROM Booking AS b
   JOIN Flight   AS f ON b.FlightID = f.FlightID
   WHERE f.Destination = vDest
   GROUP BY f.Destination
   FOR UPDATE;

   SELECT IFNULL(SUM(Quantity),0)
     INTO vFlightCount
   FROM Booking
   WHERE FlightID = pFlightID
   FOR UPDATE;

   IF vFlightCount + pQuantity > 70 THEN
```

```
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot book: flight is over capacity';
    END IF;

    IF vPopCount + pQuantity > 150 THEN
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot book: airport is
overpopulated';
    ELSE
        INSERT INTO Booking (UserID, FlightID, Quantity)
        SELECT
          pUserID,
          f.FlightID,
          pQuantity
        FROM Flight AS f
          JOIN Airport AS dep  ON f.Departure   = dep.AirportID
          JOIN Airport AS dest ON f.Destination = dest.AirportID
        WHERE f.FlightID = pFlightID
          AND NOT EXISTS (
            SELECT 1
              FROM Booking AS b2
             WHERE b2.UserID   = pUserID
               AND b2.FlightID = pFlightID
          );

        IF ROW_COUNT() = 0 THEN
            ROLLBACK;
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot book: duplicate booking';
        ELSE
            COMMIT;
        END IF;
    END IF;
END
//
DELIMITER ;
```

Calling the function

```
CALL SaveFlightByAirportCap(123, 456, 2);
```

Trigger:

(Inserts into Booked_For after a new Booking is made, check that it doesn't already exist)

```
DELIMITER $$

CREATE TRIGGER after_booking_insert
AFTER INSERT ON Booking
FOR EACH ROW
BEGIN
   IF NOT EXISTS (
         SELECT 1 FROM Booked_For
         WHERE SavedFlightID = NEW.SavedFlightID
           AND FlightID = NEW.FlightID
  ) THEN
         INSERT INTO Booked_For (SavedFlightID, FlightID)
         VALUES (NEW.SavedFlightID, NEW.FlightID);
   END IF;
END $$

DELIMITER ;
```

(Deletes from Booked_For after a Booking is deleted, check that it exists first)

```
DELIMITER $$

CREATE TRIGGER after_booking_delete
AFTER DELETE ON Booking
FOR EACH ROW
BEGIN
   IF EXISTS (
         SELECT 1 FROM Booked_For
         WHERE SavedFlightID = OLD.SavedFlightID
  ) THEN
         DELETE FROM Booked_For
         WHERE SavedFlightID = OLD.SavedFlightID;
  END IF;
END $$

DELIMITER ;
```

Constraints:

The constraint requirement is covered by the primary and foreign keys within our tables. In particular:

- The Users table has UserId as the primary key and AirportID as a foreign key
- The Airport table has AirportID as the primary key
- The Company table has CompanyID as the primary key
- The Booking table has SavedFlightID as the primary key, UserID as a foreign key, and FlightID as a foreign key
- The Booked_For table has the key pair (SavedFlightID, FlightID) as the primary key, SavedFlightID as a foreign key, and FlightID as a foreign key
- The Flight table has FlightID as the primary key, Departure as a foreign key, Destination as a foreign key, and CompanyID as a foreign key