Project 3

Hotel Management Database

Group 35
Erin Addison, Sabrina Filion, Tina Ding, Daria Vorsina

Relational Schema:

- Guests(GID, Name, Email, Phone)
- CasualGuest(GID)
  - FOREIGN KEY (GID) REFERENCES Guests
- VIPGuest(GID, TierDiscount)
  - FOREIGN KEY (GID) REFERENCES Guests
- Rooms(RoomNum, RoomType, Capacity)
- Reservations(ResID, CheckIn, CheckOut, GuestCount, RoomNum NOT NULL, GID NOT NULL, BookDate, CardNum, Cost)
  - FOREIGN KEY (RoomNum) REFERENCES Rooms
  - FOREIGN KEY (GID) REFERENCES Guests
- Staff(SID, Name, Birthdate)
- Assistants(SID, Tier) FOREIGN KEY (SID) REFERENCES Staff
- Cleaners(SID)
  - FOREIGN KEY (SID) REFERENCES Staff
- Hosts(SID, ResID)
  - FOREIGN KEY (SID) REFERENCES Assistants
  - FOREIGN KEY (ResID) REFERENCES Reservations
- Assigned(SID, GID)
  - FOREIGN KEY (SID) REFERENCES Assistants
  - FOREIGN KEY (GID) REFERENCES VIPGuest
- Cleans(SID, ResID, Tips)
  - FOREIGN KEY (SID) REFERENCES Cleaners
  - FOREIGN KEY (ResID) REFERENCES Reservations
- Amenities(Name, OpenTime, CloseTime)

- PaidAmenities(<u>Name</u>, Cost)
    - FOREIGN KEY (Name) REFERENCES Amenities
- Charges(<u>Name,</u> <u>ResID</u>, CardNum)
    - FOREIGN KEY (Name) REFERENCES PaidAmenities
    - FOREIGN KEY (ResID) REFERENCES Reservations

Note that while the relational schema above is identical to the one presented in P2, we have refactored attributes representing time and dates in our database to use the TIME and DATE data types instead of VARCHAR. Below is the list of the tables and attributes that were refactored:

- Reservations table: CheckIn, CheckOut, and BookDate were refactored from VARCHAR(8) to DATE
    - Before: 'YY/MM/DD'
    - After: 'YYYY-MM-DD'
- Amenities table: OpenTime and CloseTime were refactored from VARCHAR(8) to TIME
    - Before: 'HH:mm'
    - After: 'HH.mm.ss'
- Staff table: Birthdate was refactored from VARCHAR(8) to DATE
    - Before: 'YY/MM/DD'
    - After: 'YYYY-MM-DD'

Stored Procedure

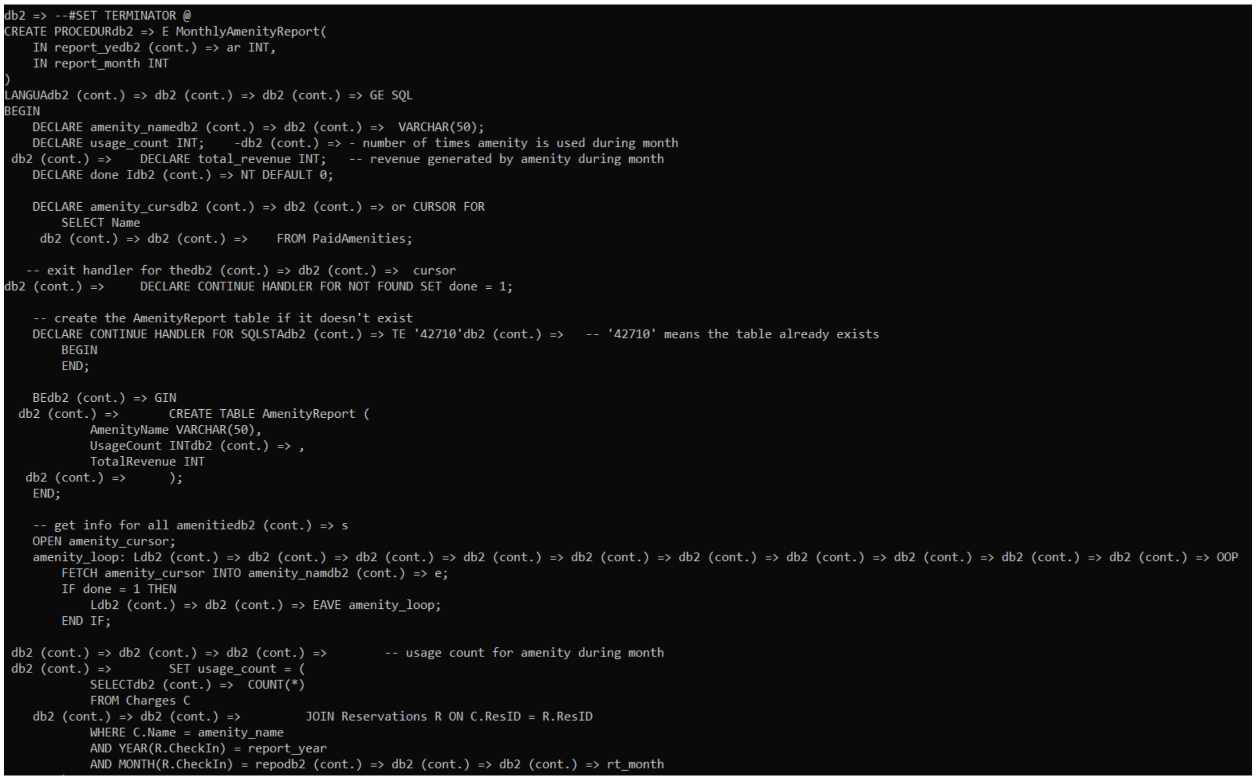
(a) The *MonthlyAmenityReport* procedure generates a monthly report on the usage (number of times used) and revenue of the hotel's paid amenities for a given month and year. The usage and revenue for the given month are calculated for each paid amenity using the *PaidAmenities*, *Charges* and *Reservations* tables and stored in a new relation called *AmenityReport*. Note that we are creating a new relation *AmenityReport*, so it should not exist before the procedure is run. This procedure allows the hotel to analyze amenity popularity and profitability, which can help with decision-making.

(b)
```
--#SET TERMINATOR @
CREATE PROCEDURE MonthlyAmenityReport(
  IN report_year INT,
  IN report_month INT
)
LANGUAGE SQL
```

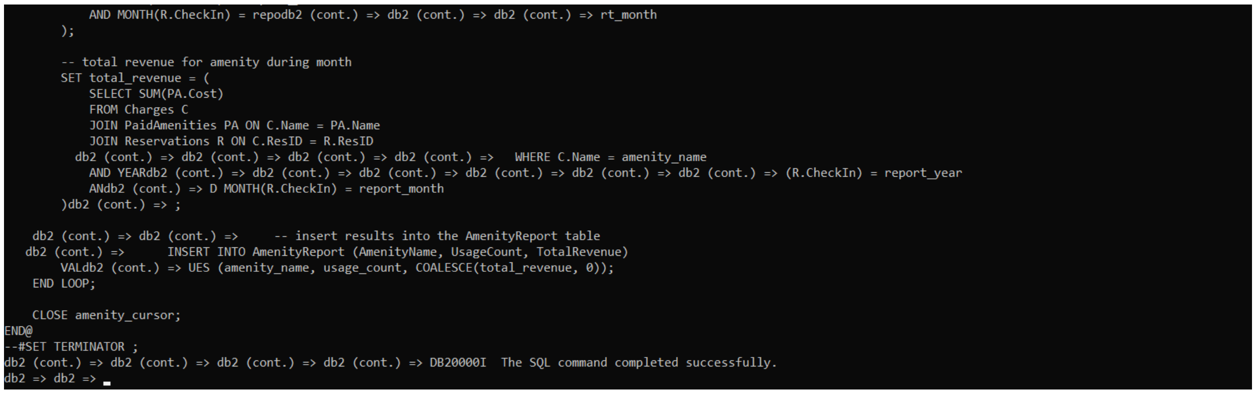
```
BEGIN
   DECLARE amenity_name VARCHAR(50);
   DECLARE usage_count INT;        -- number of times amenity is used during month
   DECLARE total_revenue INT;      -- revenue generated by amenity during month
   DECLARE done INT DEFAULT 0;

   DECLARE amenity_cursor CURSOR FOR
      SELECT Name
      FROM PaidAmenities;

  -- exit handler for the cursor
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   -- create the AmenityReport table if it doesn't exist
   DECLARE CONTINUE HANDLER FOR SQLSTATE '42710'  -- '42710' means the
table already exists
      BEGIN
      END;

   BEGIN
      CREATE TABLE AmenityReport (
         AmenityName VARCHAR(50),
         UsageCount INT,
         TotalRevenue INT
      );
   END;

   -- get info for all amenities
   OPEN amenity_cursor;
   amenity_loop: LOOP
      FETCH amenity_cursor INTO amenity_name;
      IF done = 1 THEN
         LEAVE amenity_loop;
      END IF;

      -- usage count for amenity during month
      SET usage_count = (
         SELECT COUNT(*)
         FROM Charges C
         JOIN Reservations R ON C.ResID = R.ResID
```

```
        WHERE C.Name = amenity_name
        AND YEAR(R.CheckIn) = report_year
        AND MONTH(R.CheckIn) = report_month
      );

      -- total revenue for amenity during month
      SET total_revenue = (
        SELECT SUM(PA.Cost)
        FROM Charges C
        JOIN PaidAmenities PA ON C.Name = PA.Name
        JOIN Reservations R ON C.ResID = R.ResID
        WHERE C.Name = amenity_name
        AND YEAR(R.CheckIn) = report_year
        AND MONTH(R.CheckIn) = report_month
      );

      -- insert results into the AmenityReport table
      INSERT INTO AmenityReport (AmenityName, UsageCount, TotalRevenue)
      VALUES (amenity_name, usage_count, COALESCE(total_revenue, 0));
    END LOOP;

    CLOSE amenity_cursor;
END@
--#SET TERMINATOR ;
```

(c) Creating the stored procedure (copy-pasting code from (b)):

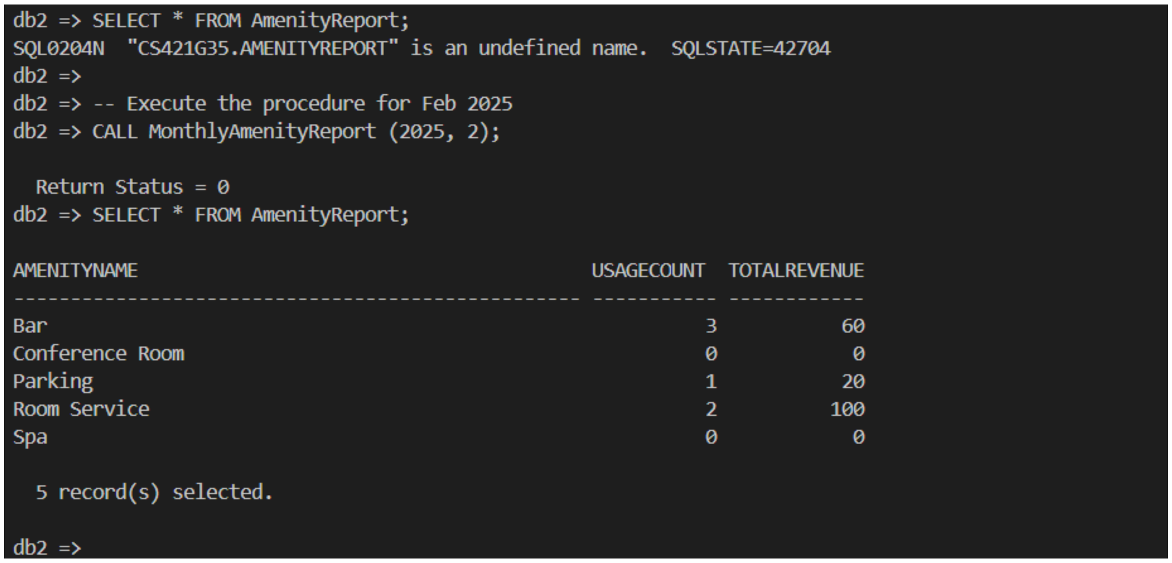


```
db2 => --#SET TERMINATOR @
CREATE PROCEDURdb2 => E MonthlyAmenityReport(
    IN report_yedb2 (cont.) => ar INT,
    IN report_month INT
)
LANGUAdb2 (cont.) => db2 (cont.) => db2 (cont.) => GE SQL
BEGIN
    DECLARE amenity_namedb2 (cont.) => db2 (cont.) =>  VARCHAR(50);
    DECLARE usage_count INT;    -db2 (cont.) => - number of times amenity is used during month
 db2 (cont.) =>    DECLARE total_revenue INT;   -- revenue generated by amenity during month
    DECLARE done Idb2 (cont.) => NT DEFAULT 0;

    DECLARE amenity_cursdb2 (cont.) => db2 (cont.) => or CURSOR FOR
        SELECT Name
     db2 (cont.) => db2 (cont.) =>    FROM PaidAmenities;

   -- exit handler for thedb2 (cont.) => db2 (cont.) =>  cursor
db2 (cont.) =>     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- create the AmenityReport table if it doesn't exist
    DECLARE CONTINUE HANDLER FOR SQLSTAdb2 (cont.) => TE '42710'db2 (cont.) =>   -- '42710' means the table already exists
        BEGIN
        END;

    BEdb2 (cont.) => GIN
  db2 (cont.) =>       CREATE TABLE AmenityReport (
            AmenityName VARCHAR(50),
            UsageCount INTdb2 (cont.) => ,
            TotalRevenue INT
   db2 (cont.) =>      );
    END;

    -- get info for all amenitiedb2 (cont.) => s
    OPEN amenity_cursor;
    amenity_loop: Ldb2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => OOP
        FETCH amenity_cursor INTO amenity_namdb2 (cont.) => e;
        IF done = 1 THEN
            Ldb2 (cont.) => db2 (cont.) => EAVE amenity_loop;
        END IF;

 db2 (cont.) => db2 (cont.) => db2 (cont.) =>        -- usage count for amenity during month
 db2 (cont.) =>        SET usage_count = (
            SELECTdb2 (cont.) =>  COUNT(*)
            FROM Charges C
    db2 (cont.) => db2 (cont.) =>         JOIN Reservations R ON C.ResID = R.ResID
            WHERE C.Name = amenity_name
            AND YEAR(R.CheckIn) = report_year
            AND MONTH(R.CheckIn) = repodb2 (cont.) => db2 (cont.) => db2 (cont.) => rt_month
```



```
            AND MONTH(R.CheckIn) = repodb2 (cont.) => db2 (cont.) => db2 (cont.) => rt_month
        );

        -- total revenue for amenity during month
        SET total_revenue = (
            SELECT SUM(PA.Cost)
            FROM Charges C
            JOIN PaidAmenities PA ON C.Name = PA.Name
            JOIN Reservations R ON C.ResID = R.ResID
          db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>   WHERE C.Name = amenity_name
            AND YEARdb2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => (R.CheckIn) = report_year
            ANdb2 (cont.) => D MONTH(R.CheckIn) = report_month
        )db2 (cont.) => ;

    db2 (cont.) => db2 (cont.) =>     -- insert results into the AmenityReport table
   db2 (cont.) =>      INSERT INTO AmenityReport (AmenityName, UsageCount, TotalRevenue)
        VALdb2 (cont.) => UES (amenity_name, usage_count, COALESCE(total_revenue, 0));
    END LOOP;

    CLOSE amenity_cursor;
END@
--#SET TERMINATOR ;
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => DB20000I  The SQL command completed successfully.
db2 => db2 => 
```

Executing the stored procedure for February 2025 and showing the created *AmenityReport* table before and after running the procedure:


```
db2 => SELECT * FROM AmenityReport;
SQL0204N  "CS421G35.AMENITYREPORT" is an undefined name.  SQLSTATE=42704
db2 =>
db2 => -- Execute the procedure for Feb 2025
db2 => CALL MonthlyAmenityReport (2025, 2);

  Return Status = 0
db2 => SELECT * FROM AmenityReport;

AMENITYNAME                                        USAGECOUNT  TOTALREVENUE
-------------------------------------------------- ----------- ------------
Bar                                                          3           60
Conference Room                                              0            0
Parking                                                      1           20
Room Service                                                 2          100
Spa                                                          0            0

  5 record(s) selected.

db2 =>
```

(d) To check if the stored procedure has the intended effect, we check the *AmenityReport* table before and after calling the procedure. See the last screenshot in (c). Note that before the procedure was called, the *AmenityReport* table did not exist. After running the procedure, the table contains the usage count and revenue generated by each paid amenity in the given month.

Application Program

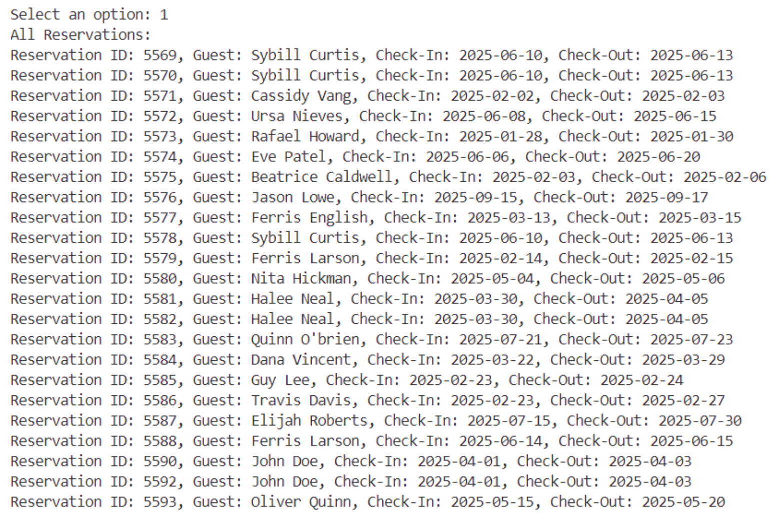
The application program for our hotel management database has the following six options: (1) look up a guest's reservations, (2) make a new reservation, (3) cancel a reservation, (4) add a new guest, (5) upgrade a guest's status, and (6) quit the application. The .java file for our application is called *databaseApp.java*.

Option 1: Look up guest reservations

This option allows the user to enter a submenu with various possibilities for looking up reservations:

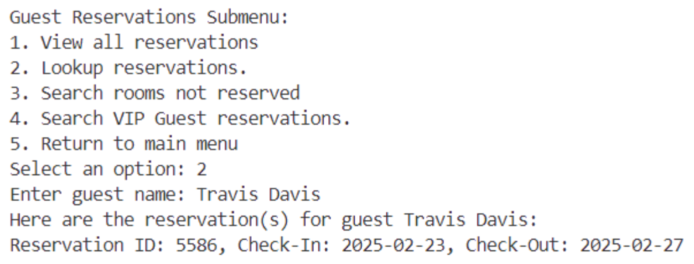
```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 1

Guest Reservations Submenu:
1. View all reservations
2. Lookup reservations.
3. Search rooms not reserved
4. Search VIP Guest reservations.
5. Return to main menu
Select an option: 3
```

Option 1.1: View all reservations



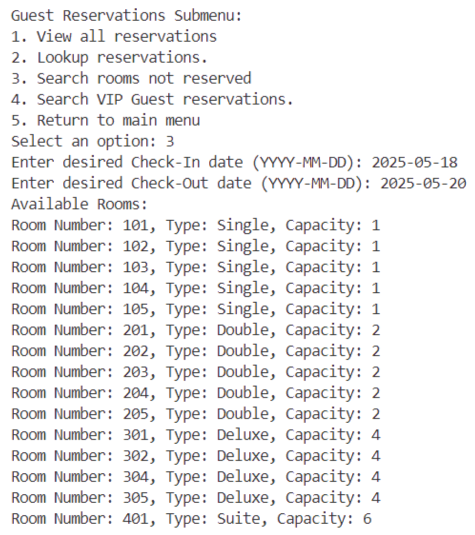
```
Select an option: 1
All Reservations:
Reservation ID: 5569, Guest: Sybill Curtis, Check-In: 2025-06-10, Check-Out: 2025-06-13
Reservation ID: 5570, Guest: Sybill Curtis, Check-In: 2025-06-10, Check-Out: 2025-06-13
Reservation ID: 5571, Guest: Cassidy Vang, Check-In: 2025-02-02, Check-Out: 2025-02-03
Reservation ID: 5572, Guest: Ursa Nieves, Check-In: 2025-06-08, Check-Out: 2025-06-15
Reservation ID: 5573, Guest: Rafael Howard, Check-In: 2025-01-28, Check-Out: 2025-01-30
Reservation ID: 5574, Guest: Eve Patel, Check-In: 2025-06-06, Check-Out: 2025-06-20
Reservation ID: 5575, Guest: Beatrice Caldwell, Check-In: 2025-02-03, Check-Out: 2025-02-06
Reservation ID: 5576, Guest: Jason Lowe, Check-In: 2025-09-15, Check-Out: 2025-09-17
Reservation ID: 5577, Guest: Ferris English, Check-In: 2025-03-13, Check-Out: 2025-03-15
Reservation ID: 5578, Guest: Sybill Curtis, Check-In: 2025-06-10, Check-Out: 2025-06-13
Reservation ID: 5579, Guest: Ferris Larson, Check-In: 2025-02-14, Check-Out: 2025-02-15
Reservation ID: 5580, Guest: Nita Hickman, Check-In: 2025-05-04, Check-Out: 2025-05-06
Reservation ID: 5581, Guest: Halee Neal, Check-In: 2025-03-30, Check-Out: 2025-04-05
Reservation ID: 5582, Guest: Halee Neal, Check-In: 2025-03-30, Check-Out: 2025-04-05
Reservation ID: 5583, Guest: Quinn O'brien, Check-In: 2025-07-21, Check-Out: 2025-07-23
Reservation ID: 5584, Guest: Dana Vincent, Check-In: 2025-03-22, Check-Out: 2025-03-29
Reservation ID: 5585, Guest: Guy Lee, Check-In: 2025-02-23, Check-Out: 2025-02-24
Reservation ID: 5586, Guest: Travis Davis, Check-In: 2025-02-23, Check-Out: 2025-02-27
Reservation ID: 5587, Guest: Elijah Roberts, Check-In: 2025-07-15, Check-Out: 2025-07-30
Reservation ID: 5588, Guest: Ferris Larson, Check-In: 2025-06-14, Check-Out: 2025-06-15
Reservation ID: 5590, Guest: John Doe, Check-In: 2025-04-01, Check-Out: 2025-04-03
Reservation ID: 5592, Guest: John Doe, Check-In: 2025-04-01, Check-Out: 2025-04-03
Reservation ID: 5593, Guest: Oliver Quinn, Check-In: 2025-05-15, Check-Out: 2025-05-20
```

Option 1.2: Lookup reservations for a certain guest



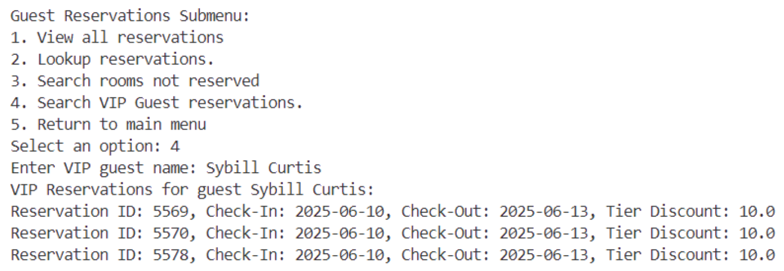
```
Guest Reservations Submenu:
1. View all reservations
2. Lookup reservations.
3. Search rooms not reserved
4. Search VIP Guest reservations.
5. Return to main menu
Select an option: 2
Enter guest name: Travis Davis
Here are the reservation(s) for guest Travis Davis:
Reservation ID: 5586, Check-In: 2025-02-23, Check-Out: 2025-02-27
```

Option 1.3: Search rooms not reserved for certain dates



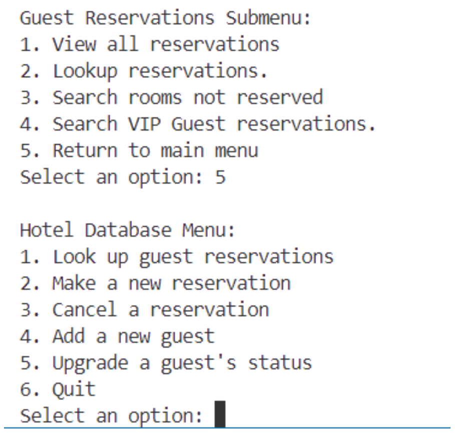
```
Guest Reservations Submenu:
1. View all reservations
2. Lookup reservations.
3. Search rooms not reserved
4. Search VIP Guest reservations.
5. Return to main menu
Select an option: 3
Enter desired Check-In date (YYYY-MM-DD): 2025-05-18
Enter desired Check-Out date (YYYY-MM-DD): 2025-05-20
Available Rooms:
Room Number: 101, Type: Single, Capacity: 1
Room Number: 102, Type: Single, Capacity: 1
Room Number: 103, Type: Single, Capacity: 1
Room Number: 104, Type: Single, Capacity: 1
Room Number: 105, Type: Single, Capacity: 1
Room Number: 201, Type: Double, Capacity: 2
Room Number: 202, Type: Double, Capacity: 2
Room Number: 203, Type: Double, Capacity: 2
Room Number: 204, Type: Double, Capacity: 2
Room Number: 205, Type: Double, Capacity: 2
Room Number: 301, Type: Deluxe, Capacity: 4
Room Number: 302, Type: Deluxe, Capacity: 4
Room Number: 304, Type: Deluxe, Capacity: 4
Room Number: 305, Type: Deluxe, Capacity: 4
Room Number: 401, Type: Suite, Capacity: 6
```

Option 1.4: Search VIP guest reservation



```
Guest Reservations Submenu:
1. View all reservations
2. Lookup reservations.
3. Search rooms not reserved
4. Search VIP Guest reservations.
5. Return to main menu
Select an option: 4
Enter VIP guest name: Sybill Curtis
VIP Reservations for guest Sybill Curtis:
Reservation ID: 5569, Check-In: 2025-06-10, Check-Out: 2025-06-13, Tier Discount: 10.0
Reservation ID: 5570, Check-In: 2025-06-10, Check-Out: 2025-06-13, Tier Discount: 10.0
Reservation ID: 5578, Check-In: 2025-06-10, Check-Out: 2025-06-13, Tier Discount: 10.0
```

Option 1.5 Return to the main menu

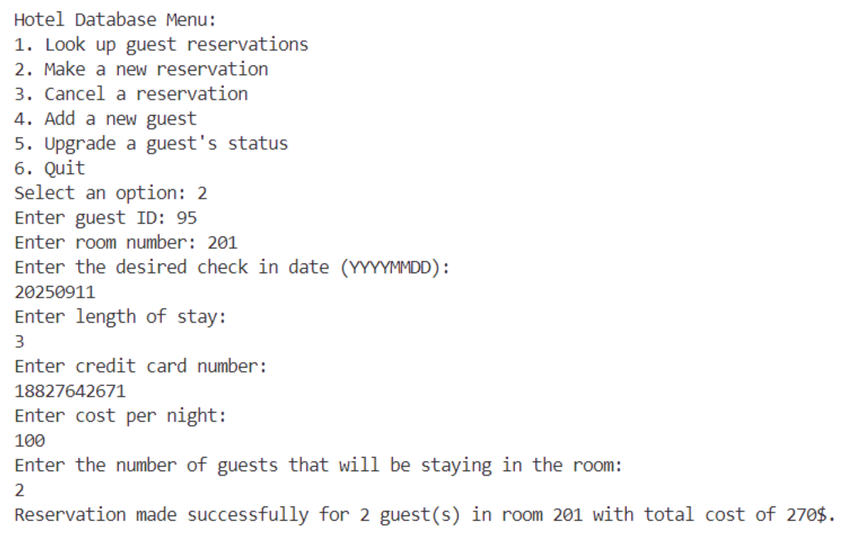


```
Guest Reservations Submenu:
1. View all reservations
2. Lookup reservations.
3. Search rooms not reserved
4. Search VIP Guest reservations.
5. Return to main menu
Select an option: 5

Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option:
```

Option 2: Make a new reservation

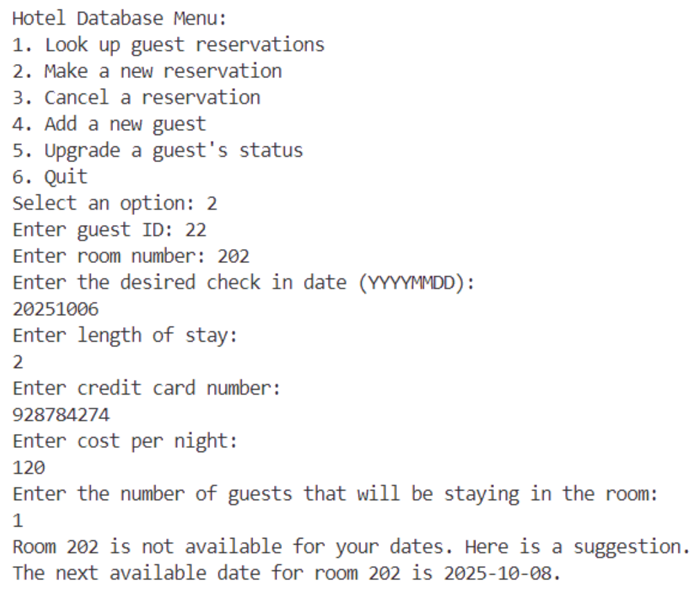
This option allows the user to create a new reservation at the hotel. This functionality ensures that the number of guests entered by the user respects the room's capacity. It also applies a discount to the total cost if the guest is a VIP. The cost per night is input as it may vary by season. If the room is already booked for the dates provided, it suggests an alternative by printing the next available date for that room. If successful, the resulting reservation has a unique resID created for it.

Example 1 – A successful new reservation, with applied discount:



```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 2
Enter guest ID: 95
Enter room number: 201
Enter the desired check in date (YYYYMMDD):
20250911
Enter length of stay:
3
Enter credit card number:
18827642671
Enter cost per night:
100
Enter the number of guests that will be staying in the room:
2
Reservation made successfully for 2 guest(s) in room 201 with total cost of 270$.
```

Example 2 – A reservation where the room is unavailable for the desired dates:

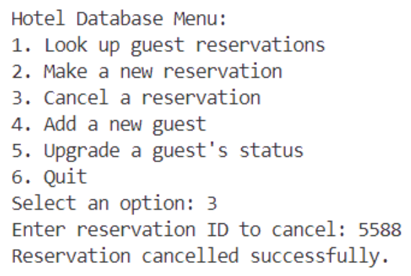


```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 2
Enter guest ID: 22
Enter room number: 202
Enter the desired check in date (YYYYMMDD):
20251006
Enter length of stay:
2
Enter credit card number:
928784274
Enter cost per night:
120
Enter the number of guests that will be staying in the room:
1
Room 202 is not available for your dates. Here is a suggestion.
The next available date for room 202 is 2025-10-08.
```

Option 3: Cancel a reservation

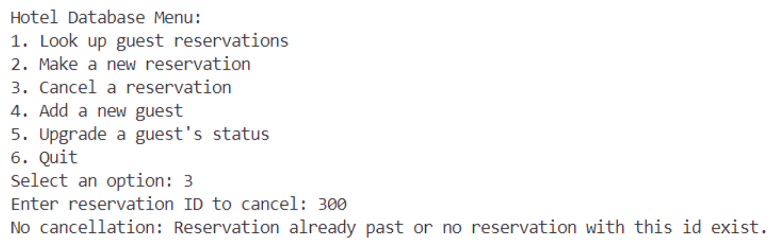
This option cancels a reservation with a given reservation ID. It only cancels reservations that are in the future, as cancelling past reservations does not logically make sense and would only erase useful data from the database. It also removes any dependencies.

Example 1 – A successful reservation cancellation:



```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 3
Enter reservation ID to cancel: 5588
Reservation cancelled successfully.
```

Example 2 – Cancelling an invalid reservation:

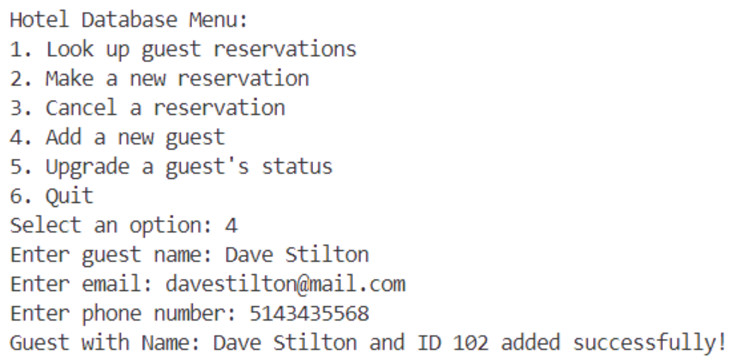


```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 3
Enter reservation ID to cancel: 300
No cancellation: Reservation already past or no reservation with this id exist.
```

Option 4: Adding a new guest

This option allows the user to add a new guest to the database. This also generates a guest ID that is so far unused.

Example:

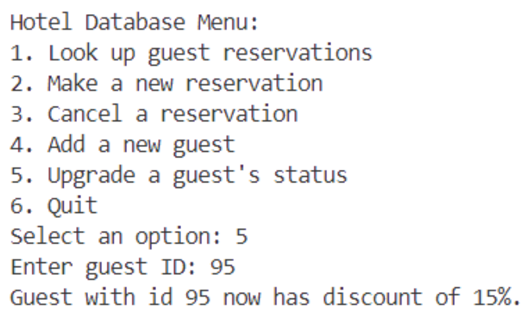


```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 4
Enter guest name: Dave Stilton
Enter email: davestilton@mail.com
Enter phone number: 5143435568
Guest with Name: Dave Stilton and ID 102 added successfully!
```

Option 5: Upgrade a guest's status

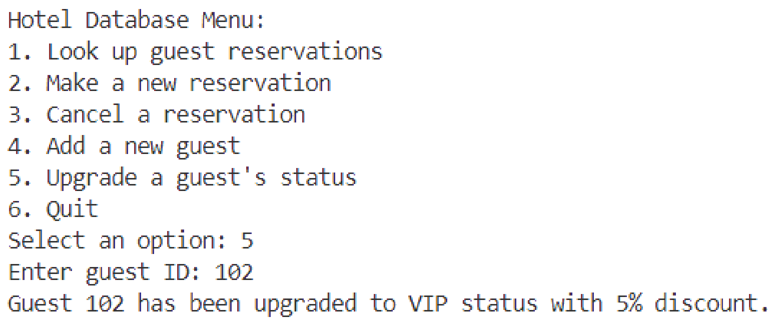
This option upgrades the status of a guest given their guest ID. If a guest is already VIP, their discount is increased to one tier higher. If a guest is casual, it upgrades them to the lowest VIP tier.

Example 1 – Upgrading guest 95 who was already VIP with a discount of 10%:



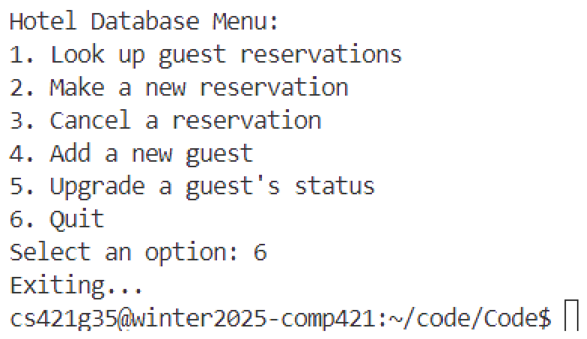
```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 5
Enter guest ID: 95
Guest with id 95 now has discount of 15%.
```

Example 2 – Upgrading guest 102 who was a casual guest to VIP:



```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 5
Enter guest ID: 102
Guest 102 has been upgraded to VIP status with 5% discount.
```

Option 6: Quit the application

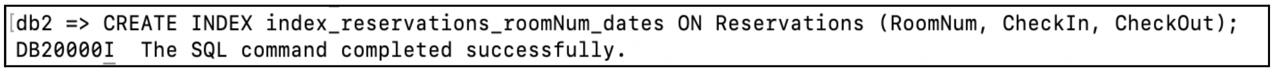


```
Hotel Database Menu:
1. Look up guest reservations
2. Make a new reservation
3. Cancel a reservation
4. Add a new guest
5. Upgrade a guest's status
6. Quit
Select an option: 6
Exiting...
cs421g35@winter2025-comp421:~/code/Code$
```

Indexing

Index 1

(a) **CREATE INDEX index_reservations_roomNum_dates ON Reservations (RoomNum, CheckIn, CheckOut);**



```
db2 => CREATE INDEX index_reservations_roomNum_dates ON Reservations (RoomNum, CheckIn, CheckOut);
DB20000I  The SQL command completed successfully.
```

(b) This Index would help speed up searches for specific room numbers and their check in/out times, and therefore help speed up guest check in and reservation creations and cancellations.

For example, suppose our hotel is very large and popular with many rooms and reservations planned in advance.

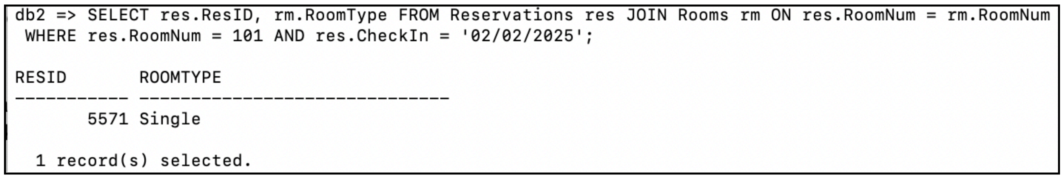
1) This Index allows for quick checking of room availability. Without it, the database would have to scan all reservations and check each ResID's row's RoomNum and date, a slow process for a large table.

2) Suppose the front desk receives a noise complaint about a room. With this Index, they can quickly search up the room number, without checking each and every row, while checking that the room's date **range** matches with the day when the complaint was made. Additionally, if the complaint does not know the specific RoomNum, this index also allows for the front desk to quickly find a range of possible room numbers to contact.

I expect it to speed up search queries on RoomNum and CheckIn/CheckOut, such as direct **indexing**, **range** scans or just wanting to see Reservations **ordered** by RoomNum or CheckIn/Out date.

Additionally because RoomNum is a foreign key, adding our index can help with **JOINs** on RumNumber.

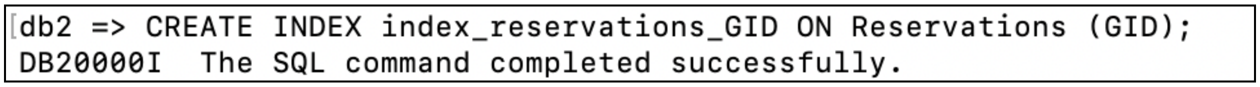
<u>Example:</u>
Index 1 speeds up checking if a reservation has a large enough room capacity.



```
db2 => SELECT res.ResID, rm.RoomType FROM Reservations res JOIN Rooms rm ON res.RoomNum = rm.RoomNum
 WHERE res.RoomNum = 101 AND res.CheckIn = '02/02/2025';

RESID       ROOMTYPE
----------- ------------------------------
       5571 Single

  1 record(s) selected.
```

Index 2

(a) **CREATE INDEX index_reservations_GID ON Reservations (GID) CLUSTER ;**



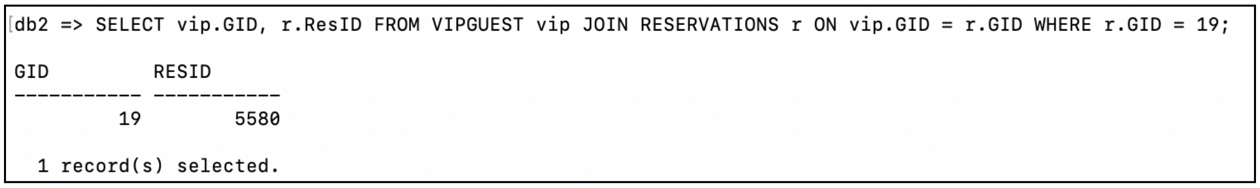
```
db2 => CREATE INDEX index_reservations_GID ON Reservations (GID);
DB20000I  The SQL command completed successfully.
```

(b) A simple but commonly used query used in our application is joining the Reservations table with other tables using the shared attribute GID. By creating a clustered index for

Reservations GID we can speed up these kinds of JOIN queries, such as when we look up and cancel reservations made by certain guests. Additionally, it will improve the speed of searches on Reservation GIDs, since Reservations are physically stored in order of GID.

*Making an index of ResID would also be useful as it is used to link tables as often as the GID attribute in our application. However, GID is more useful in that we can list all guest reservations made in order of Guests.

Example:
Suppose we join the table VIPGuests with Reservations on GID where we are looking for all the reservations made by a VIP with GID 19,
SELECT vip.GID, r.ResID FROM VIPGUEST vip JOIN RESERVATIONS r ON vip.GID = r.GID WHERE r.GID = 19;



```
db2 => SELECT vip.GID, r.ResID FROM VIPGUEST vip JOIN RESERVATIONS r ON vip.GID = r.GID WHERE r.GID = 19;

GID         RESID
----------- -----------
         19        5580

  1 record(s) selected.
```

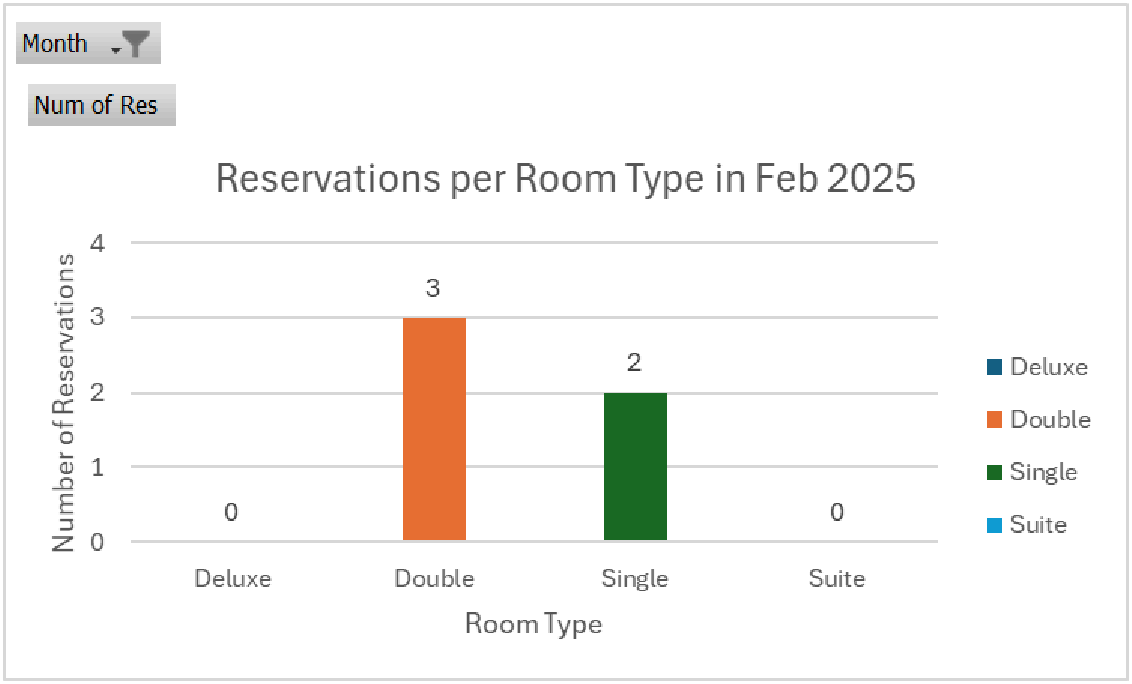
If Reservations GID is indexed, the database can quickly find the matching row to join VIPGuest with Reservations. Otherwise, it would have to compare Reservation's GID to VIPGuest's row by row.

Visualization

Vis 1

(a)
```
EXPORT TO q6-vis1.csv OF DEL MODIFIED BY NOCHARDEL
WITH MonthList (Month) AS (
        VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (11), (12)
),
RoomTypes AS ( SELECT DISTINCT RoomType FROM Rooms )
SELECT  m.Month, rt.RoomType, COUNT(DISTINCT r.ResID) AS NumRes
FROM MonthList m
CROSS JOIN RoomTypes rt  -- keep all month/roomtype combinations
LEFT JOIN Rooms ro ON ro.RoomType = rt.RoomType
LEFT JOIN Reservations r
ON MONTH(r.CheckIn) = m.Month
AND YEAR(r.CheckIn) = 2025
AND r.RoomNum = ro.RoomNum
GROUP BY m.Month, rt.RoomType
ORDER BY m.Month, rt.RoomType;
```

(b) Image of the chart:



(c) The graph displays the number of reservations made for each room type (Single, Double, Deluxe, Suite) in the hotel for a given month of 2025, based on the check-in date. The chart dynamically updates based on the selected month, allowing the user to compare the reservations across different months. In the image above, we have selected the month of February. This graph can help the hotel understand booking patterns and provide insights for decision-making, such as room pricing and resource allocation.

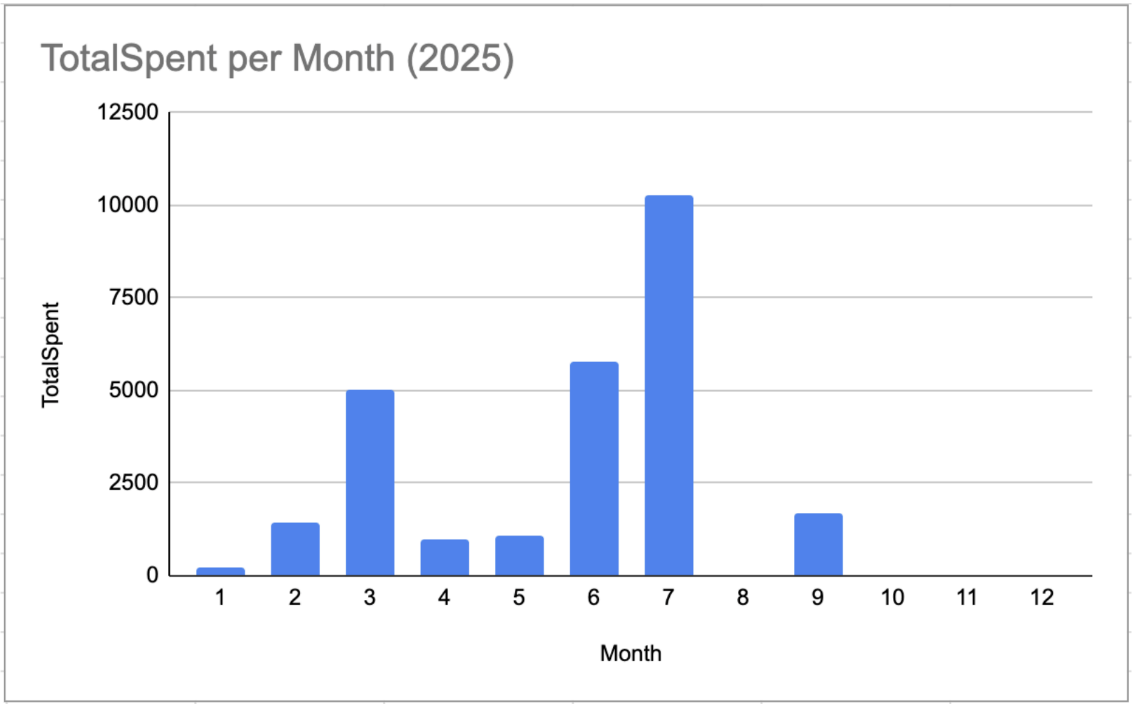
Note: the Excel spreadsheet for Vis1 is *q6-vis1.xlsx* (included in the submission). We first created a *csv* file, but to include the chart in the file, we had to save it as a *xlsx* file. The Excel spreadsheet contains two sheets: *q6-vis1*, which contains the original data from the SQL query and manually added headers, and *Chart*, which contains the created pivot table for the visualization and the final chart.

Vis 2

(a) :

```
EXPORT TO q6-vis2.csv OF DEL MODIFIED BY NOCHARDEL
WITH MonthList (Month) AS (
        VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (11), (12)
),
MonthlySpending AS (
SELECT MONTH(r.CheckIn) AS Month, SUM(r.Cost) AS TotalSpent
FROM Reservations r
WHERE YEAR(r.CheckIn) = 2025 -- all reservations are in 2025 but here to show usage
GROUP BY MONTH(r.CheckIn)
)
SELECT m.Month, COALESCE(ms.TotalSpent, 0) AS TotalSpent
FROM MonthList m
LEFT JOIN MonthlySpending ms ON m.Month = ms.Month
ORDER BY m.Month;
```

(b)



(c) The graph displays the total amount of money spent on reservations for each month. Each month is represented on the X-axis and is based on each reservation's CheckIn date, the total amount spent is represented on the y-axis. The graph allows us to see larger trends in the data, such as which months are most popular and which are the least. This info could be used to decide when increased or decreased discounts might be useful so that the hotel can encourage visitation during the low season or profit off of the high season. This graph only includes costs for reservations, not extra amenity costs guests could charge.

Note: the Excel spreadsheet for Vis2 is ***q6-vis2.xlsx***
the csv file is ***q6-vis2.csv***
and the png file is ***q6-vis2-chart.png***

Creativity - Trigger

Two triggers were created: APPLYVIPDISCOUNT and ASSIGN_TO_VIP.

***Note** that triggers are initialized once with:

```
db2 -td@ -vf vip_assistant_assignment.sql
db2 -td@ -vf  vip_discount_trigger.sql
```

They are initialized already, since the testing has been conducted as shown (c).
To test follow steps as in (d).

1. APPLYVIPDISCOUNT Trigger

a) **Description of Trigger Method 1 Action:** Trigger created which automatically updates a guest from Casual to VIP status instead of doing this with manual modifications. The trigger executes after an insertion is done to Reservations for a new guest. Additionally, the trigger creates a separate table which logs the VIPGuest ID (GID), the original cost (OriginalCost) and the cost after the discount has been applied (DiscountCost). Finally, the trigger automatically deletes the casual guest entry from CasualGuest and adds the guest to VIPGuest. If the guest is already VIP, it applies the next level of TierDiscount to the cost (additional 5%).

b) **create_vip_log.sql**

- Script that creates a log (table) that stores the VIP guest GID, the OriginalCost (normal reservation price) and the DiscountCost (cost after application of VIPDiscount)

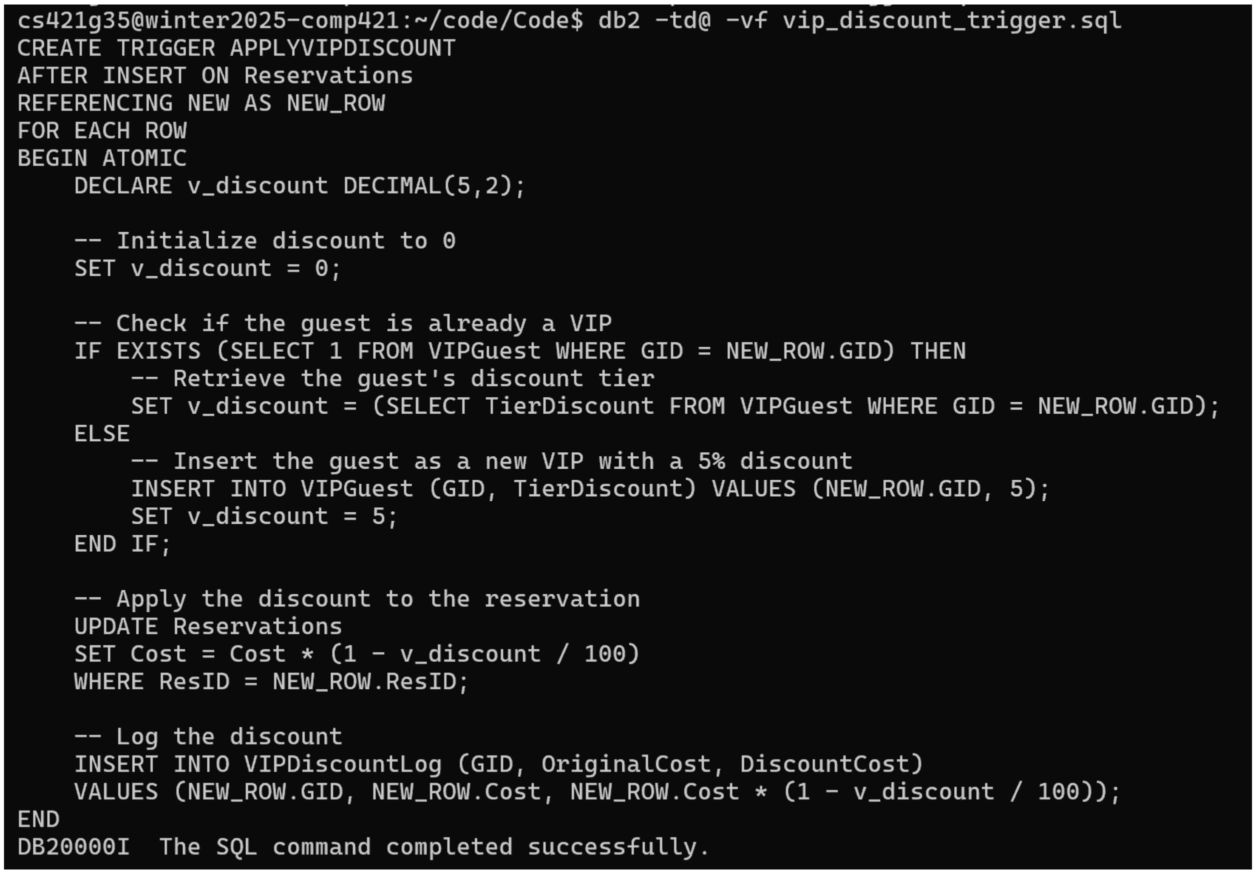
```
CREATE OR REPLACE PROCEDURE createVIPDiscountLog()
BEGIN
   DECLARE v_count INT DEFAULT 0;

   -- Check if table exists
   SELECT COUNT(*) INTO v_count FROM SYSCAT.TABLES WHERE TABNAME
= 'VIPDISCOUNTLOG';

   -- If table doesn't exist, create it
   IF v_count = 0 THEN
      EXECUTE IMMEDIATE '
         CREATE TABLE VIPDiscountLog (
            LogID INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
            GID INT NOT NULL,
            OriginalCost DECIMAL(10,2) NOT NULL,
            DiscountCost DECIMAL(10,2) NOT NULL,
            FOREIGN KEY (GID) REFERENCES VIPGuest(GID)
        )';
    END IF;
END@
```

vip_discount_trigger.sql

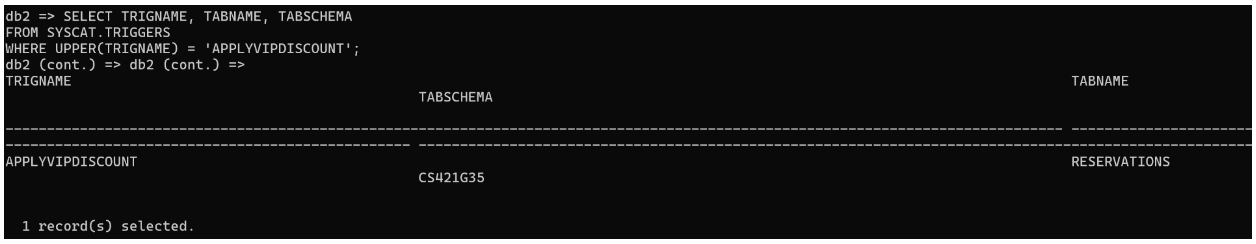
- Script that creates the APPLYVIPDISCOUNT trigger which upon a new entry in Reservations, will apply a discount. If it's an existing VIPGuest then it will apply the respective discount for that VIPGuest. If it's a new guest then a 5% discount and move the guest up to VIP status with a TierDiscount of 5. The costs will be updated in Reservations. The new Guest will be added to the VIPGuest table. Finally, the Guest's GID, the OriginalCost and the DiscounCost will be logged in the VIPDiscountLog.

```
CREATE TRIGGER APPLYVIPDISCOUNT
AFTER INSERT ON Reservations
REFERENCING NEW AS NEW_ROW
FOR EACH ROW
BEGIN ATOMIC
    DECLARE v_discount DECIMAL(5,2);

    -- Initialize discount to 0
    SET v_discount = 0;

    -- Check if the guest is already a VIP
    IF EXISTS (SELECT 1 FROM VIPGuest WHERE GID = NEW_ROW.GID) THEN
        -- Retrieve the guest's discount tier
        SET v_discount = (SELECT TierDiscount FROM VIPGuest WHERE GID =
NEW_ROW.GID);
    ELSE
        -- Insert the guest as a new VIP with a 5% discount
        INSERT INTO VIPGuest (GID, TierDiscount) VALUES (NEW_ROW.GID, 5);
        SET v_discount = 5;
    END IF;

    -- Apply the discount to the reservation
    UPDATE Reservations
    SET Cost = Cost * (1 - v_discount / 100)
    WHERE ResID = NEW_ROW.ResID;

    -- Log the discount
    INSERT INTO VIPDiscountLog (GID, OriginalCost, DiscountCost)
    VALUES (NEW_ROW.GID, NEW_ROW.Cost, NEW_ROW.Cost * (1 - v_discount /
100));
END@
```

c) Trigger Execution and Validation:



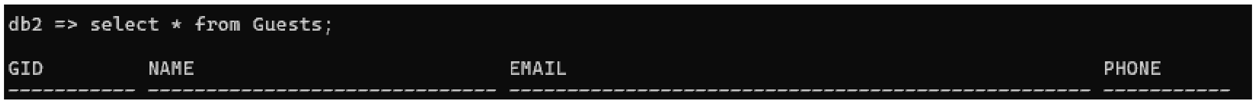
```
cs421g35@winter2025-comp421:~/code/Code$ db2 -td@ -vf vip_discount_trigger.sql
CREATE TRIGGER APPLYVIPDISCOUNT
AFTER INSERT ON Reservations
REFERENCING NEW AS NEW_ROW
FOR EACH ROW
BEGIN ATOMIC
    DECLARE v_discount DECIMAL(5,2);

    -- Initialize discount to 0
    SET v_discount = 0;

    -- Check if the guest is already a VIP
    IF EXISTS (SELECT 1 FROM VIPGuest WHERE GID = NEW_ROW.GID) THEN
        -- Retrieve the guest's discount tier
        SET v_discount = (SELECT TierDiscount FROM VIPGuest WHERE GID = NEW_ROW.GID);
    ELSE
        -- Insert the guest as a new VIP with a 5% discount
        INSERT INTO VIPGuest (GID, TierDiscount) VALUES (NEW_ROW.GID, 5);
        SET v_discount = 5;
    END IF;

    -- Apply the discount to the reservation
    UPDATE Reservations
    SET Cost = Cost * (1 - v_discount / 100)
    WHERE ResID = NEW_ROW.ResID;

    -- Log the discount
    INSERT INTO VIPDiscountLog (GID, OriginalCost, DiscountCost)
    VALUES (NEW_ROW.GID, NEW_ROW.Cost, NEW_ROW.Cost * (1 - v_discount / 100));
END
DB20000I  The SQL command completed successfully.
```



```
db2 => SELECT TRIGNAME, TABNAME, TABSCHEMA
FROM SYSCAT.TRIGGERS
WHERE UPPER(TRIGNAME) = 'APPLYVIPDISCOUNT';
db2 (cont.) => db2 (cont.) =>
TRIGNAME                                                                                                                         TABNAME
                                                  TABSCHEMA

-------------------------------------------------------------------------------------------------------------------------------- --------------------
------------------------------------------------- ------------------------------------------------------------------------------------------------------
APPLYVIPDISCOUNT                                                                                                                 RESERVATIONS
                                                  CS421G35

  1 record(s) selected.
```

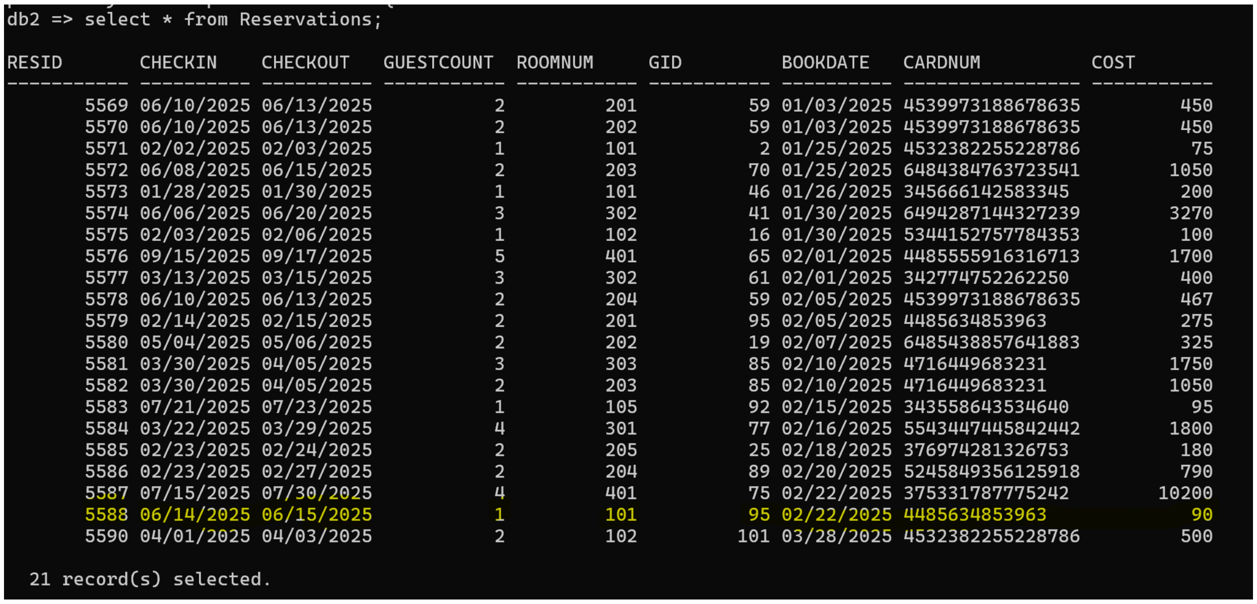
d) Demonstration of Trigger Effect:

Verify Action with an Existing Guest With a New Reservation:

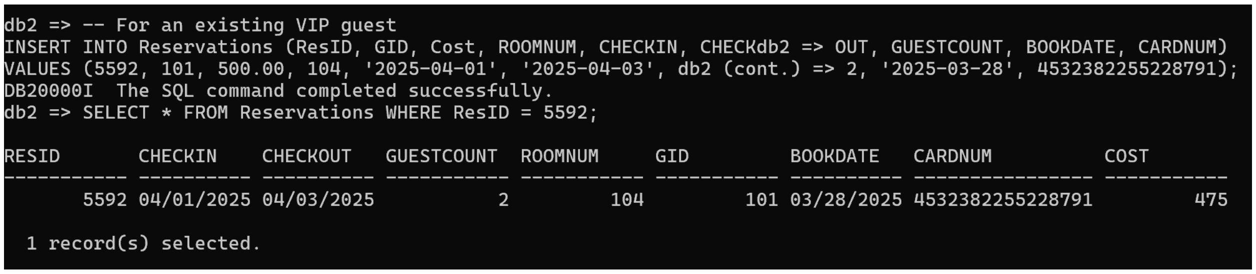


```
db2 => select * from Guests;

GID         NAME                           EMAIL                                              PHONE
----------- ------------------------------ -------------------------------------------------- -----------
```

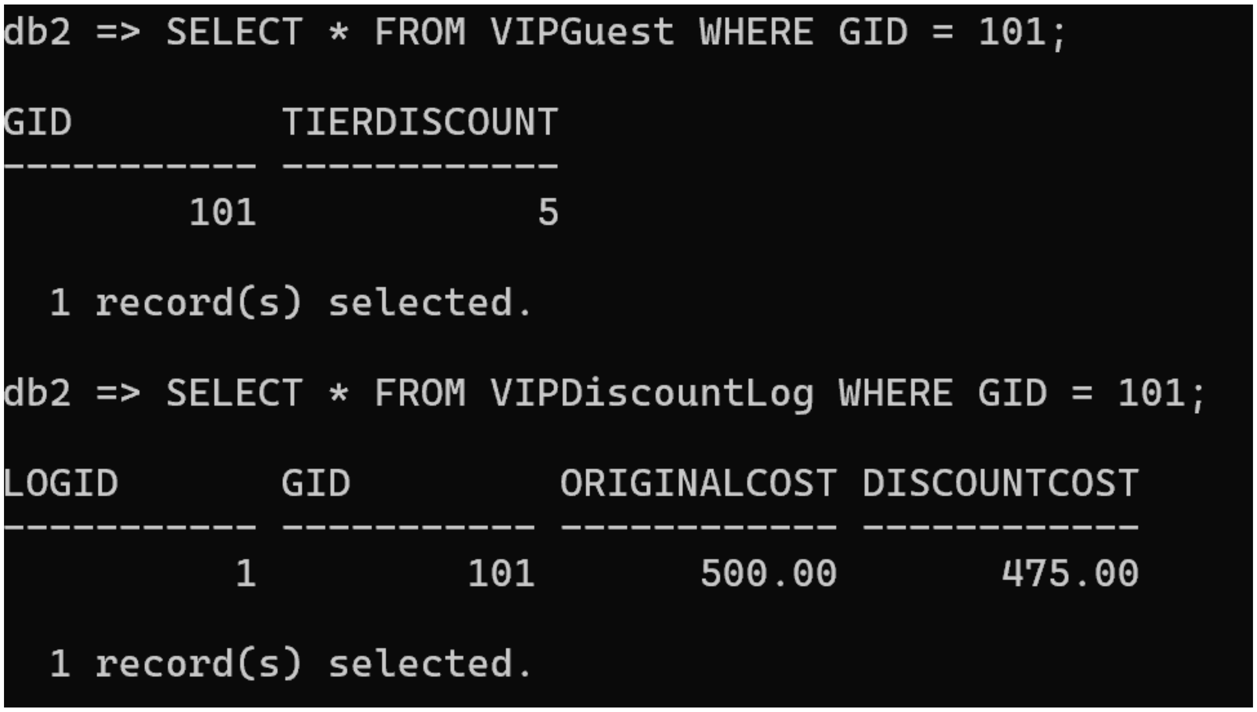
```
        101 John Doe                                     john.doe@example.com                                                   1234567890
```



```
db2 => select * from Reservations;

RESID       CHECKIN    CHECKOUT   GUESTCOUNT  ROOMNUM     GID         BOOKDATE   CARDNUM          COST
----------- ---------- ---------- ----------- ----------- ----------- ---------- ---------------- -----------
       5569 06/10/2025 06/13/2025           2         201          59 01/03/2025 4539973188678635         450
       5570 06/10/2025 06/13/2025           2         202          59 01/03/2025 4539973188678635         450
       5571 02/02/2025 02/03/2025           1         101           2 01/25/2025 4532382255228786          75
       5572 06/08/2025 06/15/2025           2         203          70 01/25/2025 6484384763723541        1050
       5573 01/28/2025 01/30/2025           1         101          46 01/26/2025 345666142583345          200
       5574 06/06/2025 06/20/2025           3         302          41 01/30/2025 6494287144327239        3270
       5575 02/03/2025 02/06/2025           1         102          16 01/30/2025 5344152757784353         100
       5576 09/15/2025 09/17/2025           5         401          65 02/01/2025 4485555916316713        1700
       5577 03/13/2025 03/15/2025           3         302          61 02/01/2025 342774752262250          400
       5578 06/10/2025 06/13/2025           2         204          59 02/05/2025 4539973188678635         467
       5579 02/14/2025 02/15/2025           2         201          95 02/05/2025 4485634853963            275
       5580 05/04/2025 05/06/2025           2         202          19 02/07/2025 6485438857641883         325
       5581 03/30/2025 04/05/2025           3         303          85 02/10/2025 4716449683231           1750
       5582 03/30/2025 04/05/2025           2         203          85 02/10/2025 4716449683231           1050
       5583 07/21/2025 07/23/2025           1         105          92 02/15/2025 343558643534640           95
       5584 03/22/2025 03/29/2025           4         301          77 02/16/2025 5543447445842442        1800
       5585 02/23/2025 02/24/2025           2         205          25 02/18/2025 376974281326753          180
       5586 02/23/2025 02/27/2025           2         204          89 02/20/2025 5245849356125918         790
       5587 07/15/2025 07/30/2025           4         401          75 02/22/2025 375331787775242        10200
       5588 06/14/2025 06/15/2025           1         101          95 02/22/2025 4485634853963             90
       5590 04/01/2025 04/03/2025           2         102         101 03/28/2025 4532382255228786         500

  21 record(s) selected.
```

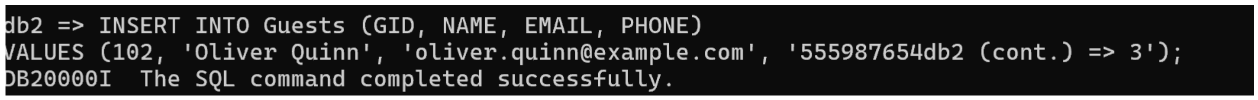


```
db2 => -- For an existing VIP guest
INSERT INTO Reservations (ResID, GID, Cost, ROOMNUM, CHECKIN, CHECKdb2 => OUT, GUESTCOUNT, BOOKDATE, CARDNUM)
VALUES (5592, 101, 500.00, 104, '2025-04-01', '2025-04-03', db2 (cont.) => 2, '2025-03-28', 4532382255228791);
DB20000I  The SQL command completed successfully.
db2 => SELECT * FROM Reservations WHERE ResID = 5592;

RESID       CHECKIN    CHECKOUT   GUESTCOUNT  ROOMNUM     GID         BOOKDATE   CARDNUM          COST
----------- ---------- ---------- ----------- ----------- ----------- ---------- ---------------- -----------
       5592 04/01/2025 04/03/2025           2         104         101 03/28/2025 4532382255228791         475

  1 record(s) selected.
```

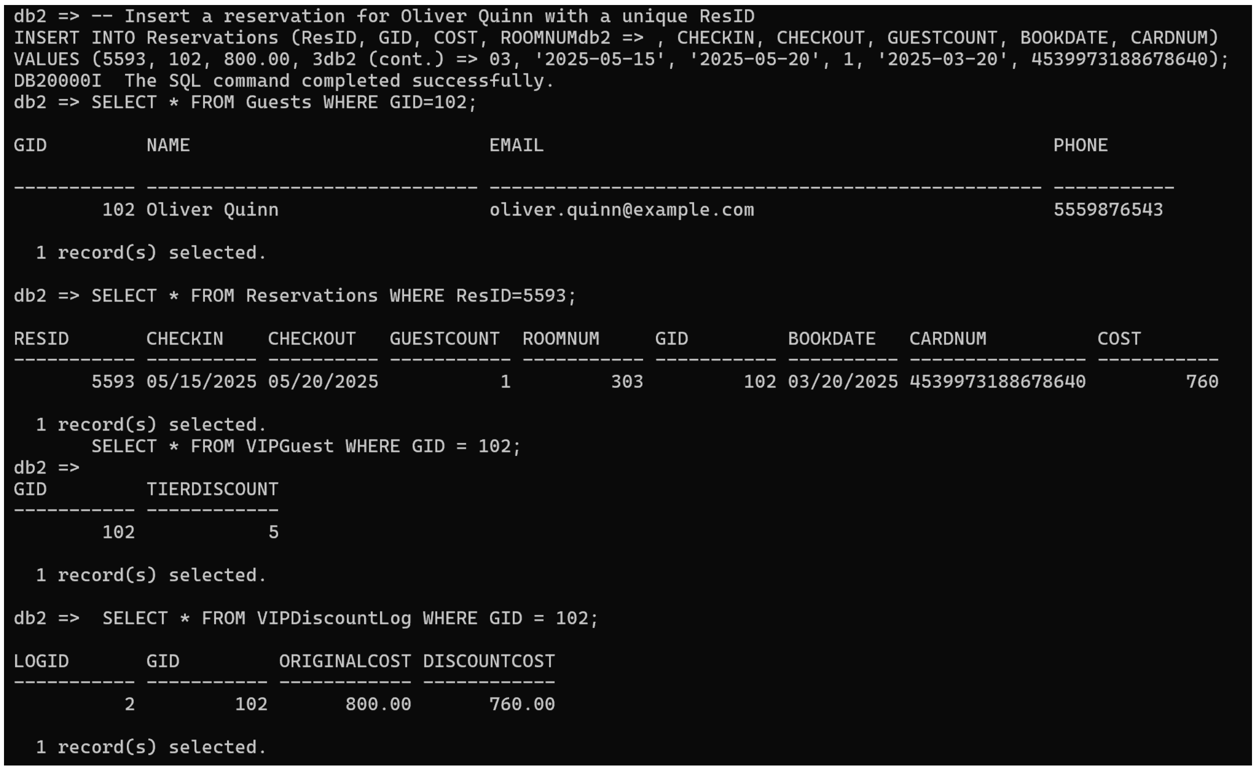


```
db2 => SELECT * FROM VIPGuest WHERE GID = 101;

GID         TIERDISCOUNT
----------- ------------
        101            5

  1 record(s) selected.

db2 => SELECT * FROM VIPDiscountLog WHERE GID = 101;

LOGID       GID         ORIGINALCOST DISCOUNTCOST
----------- ----------- ------------ ------------
          1         101       500.00       475.00

  1 record(s) selected.
```

Verify Action with a New Guest:



```
db2 => INSERT INTO Guests (GID, NAME, EMAIL, PHONE)
VALUES (102, 'Oliver Quinn', 'oliver.quinn@example.com', '555987654db2 (cont.) => 3');
DB20000I  The SQL command completed successfully.
```



```
db2 => -- Insert a reservation for Oliver Quinn with a unique ResID
INSERT INTO Reservations (ResID, GID, COST, ROOMNUMdb2 => , CHECKIN, CHECKOUT, GUESTCOUNT, BOOKDATE, CARDNUM)
VALUES (5593, 102, 800.00, 3db2 (cont.) => 03, '2025-05-15', '2025-05-20', 1, '2025-03-20', 4539973188678640);
DB20000I  The SQL command completed successfully.
db2 => SELECT * FROM Guests WHERE GID=102;

GID         NAME                           EMAIL                                              PHONE

----------- ------------------------------ -------------------------------------------------- -----------
        102 Oliver Quinn                   oliver.quinn@example.com                           5559876543

  1 record(s) selected.

db2 => SELECT * FROM Reservations WHERE ResID=5593;

RESID       CHECKIN    CHECKOUT   GUESTCOUNT  ROOMNUM     GID         BOOKDATE   CARDNUM          COST
----------- ---------- ---------- ----------- ----------- ----------- ---------- ---------------- -----------
       5593 05/15/2025 05/20/2025           1         303         102 03/20/2025 4539973188678640         760

  1 record(s) selected.
       SELECT * FROM VIPGuest WHERE GID = 102;
db2 =>
GID         TIERDISCOUNT
----------- ------------
        102            5

  1 record(s) selected.

db2 =>  SELECT * FROM VIPDiscountLog WHERE GID = 102;

LOGID       GID         ORIGINALCOST DISCOUNTCOST
----------- ----------- ------------ ------------
          2         102       800.00       760.00

  1 record(s) selected.
```

2. ASSIGN_TO_VIP Trigger

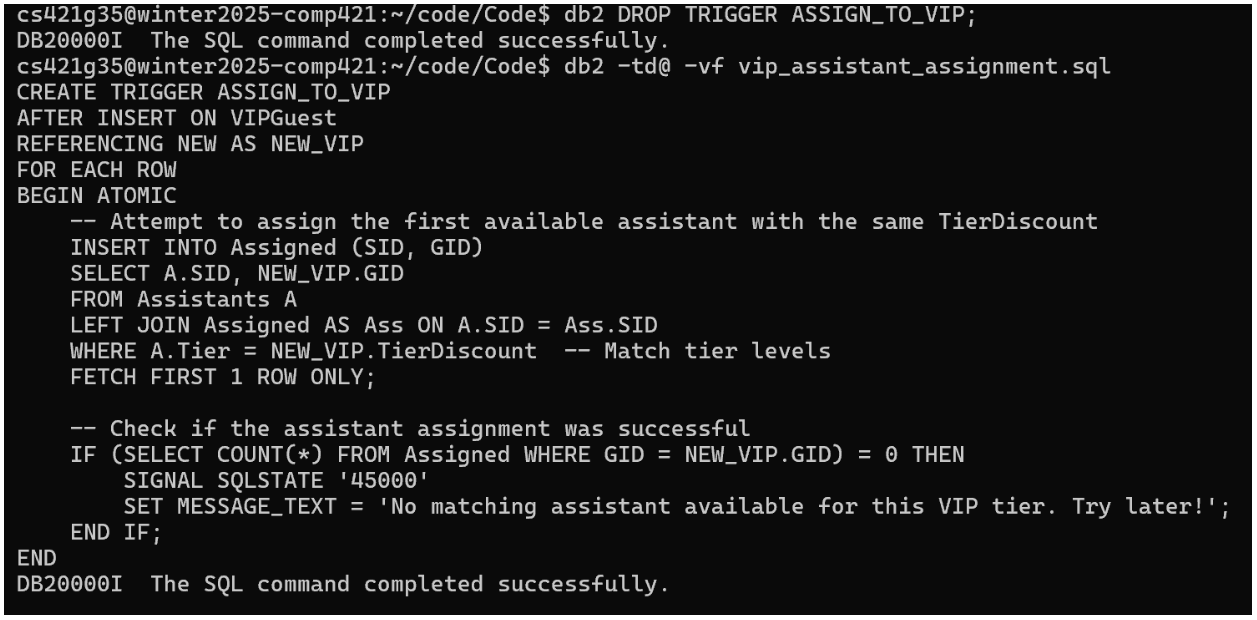
a) **Description of Trigger Method 2 Action:** Trigger created which automatically assigns an Assistant to a VIP Guest after a VIP Guest has been added to the VIPGuest table. The trigger checks if there is an available assistant for the VIP guest. If there isn't it notifies the user, otherwise it automatically pairs the VIPGuest to an Assistant based on the tier.

b) **vip_assistant_assignment.sql**

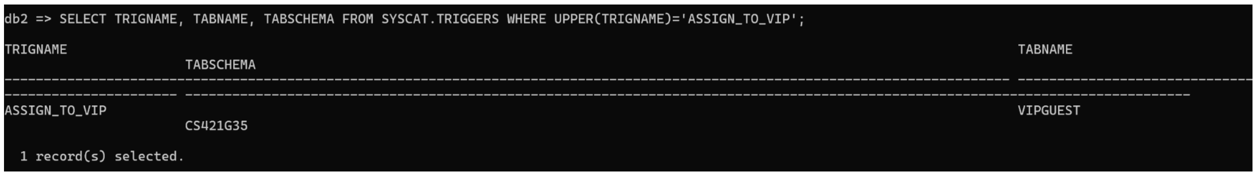
- Script to create the trigger ASSIGN_TO_VIP which acts as described in (a).

```
CREATE TRIGGER ASSIGN_TO_VIP
AFTER INSERT ON VIPGuest
REFERENCING NEW AS NEW_VIP
FOR EACH ROW
BEGIN ATOMIC
    -- Attempt to assign the first available assistant with the same TierDiscount
    INSERT INTO Assigned (SID, GID)
    SELECT A.SID, NEW_VIP.GID
    FROM Assistants A
    LEFT JOIN Assigned AS Ass ON A.SID = Ass.SID
    WHERE A.Tier = NEW_VIP.TierDiscount  -- Match tier levels
    FETCH FIRST 1 ROW ONLY;

    -- Check if the assistant assignment was successful
    IF (SELECT COUNT(*) FROM Assigned WHERE GID = NEW_VIP.GID) = 0
THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No matching assistant available for this VIP tier. Try
later!';
    END IF;
END@
```

c) Trigger Execution and Validation:



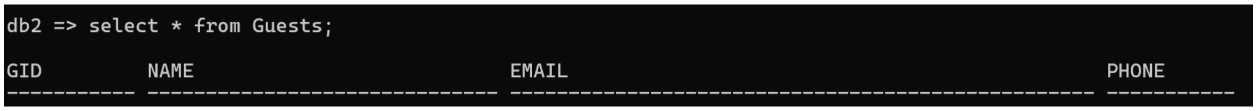
```
cs421g35@winter2025-comp421:~/code/Code$ db2 DROP TRIGGER ASSIGN_TO_VIP;
DB20000I  The SQL command completed successfully.
cs421g35@winter2025-comp421:~/code/Code$ db2 -td@ -vf vip_assistant_assignment.sql
CREATE TRIGGER ASSIGN_TO_VIP
AFTER INSERT ON VIPGuest
REFERENCING NEW AS NEW_VIP
FOR EACH ROW
BEGIN ATOMIC
    -- Attempt to assign the first available assistant with the same TierDiscount
    INSERT INTO Assigned (SID, GID)
    SELECT A.SID, NEW_VIP.GID
    FROM Assistants A
    LEFT JOIN Assigned AS Ass ON A.SID = Ass.SID
    WHERE A.Tier = NEW_VIP.TierDiscount  -- Match tier levels
    FETCH FIRST 1 ROW ONLY;

    -- Check if the assistant assignment was successful
    IF (SELECT COUNT(*) FROM Assigned WHERE GID = NEW_VIP.GID) = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No matching assistant available for this VIP tier. Try later!';
    END IF;
END
DB20000I  The SQL command completed successfully.
```



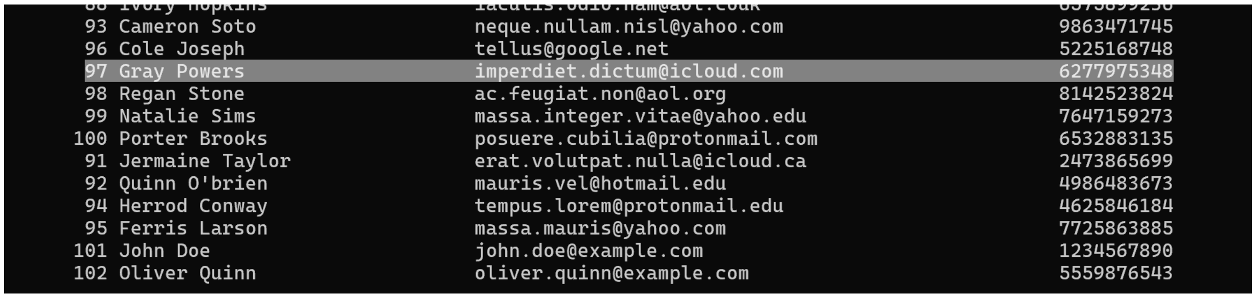
```
db2 => SELECT TRIGNAME, TABNAME, TABSCHEMA FROM SYSCAT.TRIGGERS WHERE UPPER(TRIGNAME)='ASSIGN_TO_VIP';

TRIGNAME                                                                                                         TABNAME
                     TABSCHEMA
---------------------------------------------------------------------------------------------------------------- -------------------------------
-------------------- --------------------------------------------------------------------------------------------------------------------
ASSIGN_TO_VIP                                                                                                    VIPGUEST
                     CS421G35

  1 record(s) selected.
```

d) Demonstration of Trigger Effect:

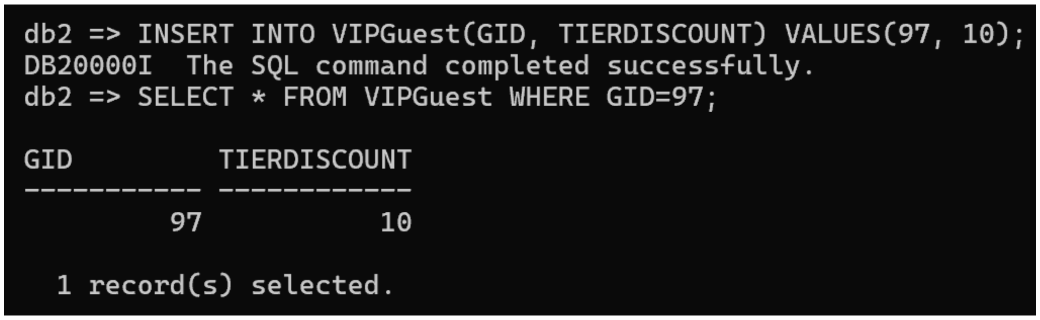
Upon Addition of a New VIP Guest:



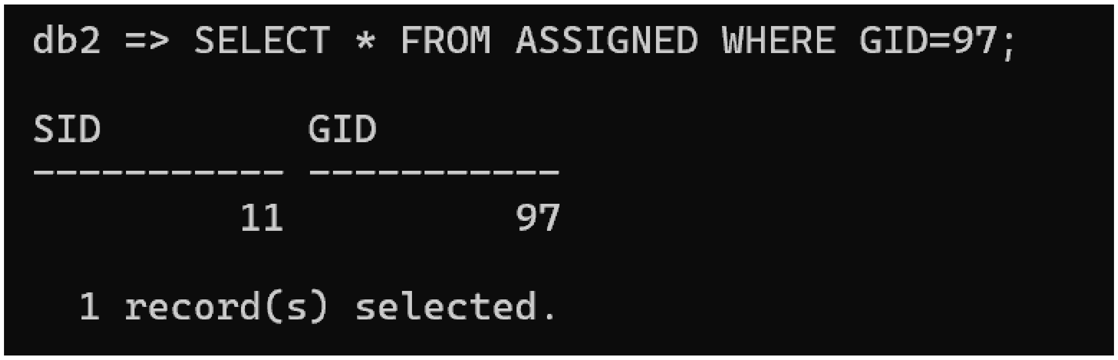
```
db2 => select * from Guests;

GID         NAME                           EMAIL                                              PHONE
----------- ------------------------------ -------------------------------------------------- ----------
```



```
         93 Cameron Soto                   neque.nullam.nisl@yahoo.com                        9863471745
         96 Cole Joseph                    tellus@google.net                                  5225168748
         97 Gray Powers                    imperdiet.dictum@icloud.com                        6277975348
         98 Regan Stone                    ac.feugiat.non@aol.org                             8142523824
         99 Natalie Sims                   massa.integer.vitae@yahoo.edu                      7647159273
        100 Porter Brooks                  posuere.cubilia@protonmail.com                     6532883135
         91 Jermaine Taylor                erat.volutpat.nulla@icloud.ca                      2473865699
         92 Quinn O'brien                  mauris.vel@hotmail.edu                             4986483673
         94 Herrod Conway                  tempus.lorem@protonmail.edu                        4625846184
         95 Ferris Larson                  massa.mauris@yahoo.com                             7725863885
        101 John Doe                       john.doe@example.com                               1234567890
        102 Oliver Quinn                   oliver.quinn@example.com                           5559876543
```



```
db2 => INSERT INTO VIPGuest(GID, TIERDISCOUNT) VALUES(97, 10);
DB20000I  The SQL command completed successfully.
db2 => SELECT * FROM VIPGuest WHERE GID=97;

GID         TIERDISCOUNT
----------- ------------
         97           10

  1 record(s) selected.
```



```
db2 => SELECT * FROM ASSIGNED WHERE GID=97;

SID         GID
----------- -----------
         11          97

  1 record(s) selected.
```

How we worked together

We had a Zoom meeting at the beginning to establish how we wanted to split the work. We each worked on around 1-2 tasks and cross-verified our answers.