Project 2

Hotel Management Database

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

---

2. 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(Name, Cost)
    - FOREIGN KEY (Name) REFERENCES Amenities
- Charges(Name, ResID, CardNum)
    - FOREIGN KEY (Name) REFERENCES PaidAmenities
    - FOREIGN KEY (ResID) REFERENCES Reservations

3. Pending Constraints

- The Guests ISA Casual/VIP hierarchy is covering and non-overlapping. Guests are either considered Casual guests or VIP guests; therefore, all GIDs in the Guests table should appear as GIDs in either the CasualGuest table or the VIPGuest table.

- If a guest wants to reserve multiple rooms, they have to make multiple separate reservations. This means that someone could reserve the same room multiple times for the same date. However, we cannot make room numbers unique in the *Reservations* table as others may need to reserve the same room at a later date. The application must make sure that guests cannot book the same room for any overlapping period of time.

- Assistant and VIP tiers in the *Assistants* and *VIPGuest* tables could be non-matching. The application must make sure they match when assigning assistants in the *Assigned* table.

4. Load Data

Please see the *loaddata.sql* and *loaddata.log* files included in the submission.

5. SQL Queries

Query 1

(a) This query returns the IDs of the rooms that are free at a certain time interval (June 10th 2025 to June 13th 2025). This is obviously a very useful query for the hotel reservation system.

(b)
```
SELECT r.roomnum
FROM Rooms r
WHERE r.RoomNum NOT IN (
  SELECT res.RoomNum
  FROM Reservations res
  WHERE NOT  ( res.CheckOut <= '25/06/10' OR res.CheckIn >= '25/06/13' )
) ORDER BY r.roomnum;
```

(c) Executing the query:

```
db2 => SELECT r.roomnum FROM Rooms r WHERE r.RoomNum NOT IN
 ( SELECT res.RoomNum FROM Reservations res WHEdb2 (cont.) => RE NOT
( res.CheckOut <= '25/06/10' OR res.CheckIdb2 (cont.) => n >= '25/06/13' )
) ORDER BY r.roomnum;
db2 (cont.) =>
ROOMNUM
-----------
        101
        102
        103
        104
        105
        205
        301
        303
        304
        305
        401

  11 record(s) selected.
```

Query 2

(a) This query returns the name and phone number of guests who had reserved a double room in February 2025 based on the check-out date, along with the name of the assistant that hosted their stay.

(b)
```
SELECT g.name AS GuestName, g.phone AS GuestPhone, s.name AS
AssistantName
FROM Staff s, Guests g, Reservations r, Hosts h
WHERE s.sid = h.sid AND h.resid = r.resid AND r.gid = g.gid
AND r.roomnum IN (
  SELECT ro.roomnum
  FROM Rooms ro
  WHERE ro.roomtype = 'Double')
AND r.CheckOut >= '25/02/01'
AND r.CheckOut < '25/03/01' ;
```

(c) Executing the query:

```
db2 => SELECT g.name AS GuestName, g.phone AS GuestPhone, s.name AS AssistantName
FROM Staff s, Guests g, Rdb2 (cont.) => eservations r, Hosts h
WHERE s.sid = h.sid AND h.rdb2 (cont.) => esid = r.resid AND r.gid = g.gid
AND r.roomnum INdb2 (cont.) =>  (
  SELECT ro.roomnum
  FROM Rooms ro
  WHERE ro.db2 (cont.) => db2 (cont.) => db2 (cont.) => roomtype = 'Double'
)
AND r.CheckOut >= '25/02/01'db2 (cont.) => db2 (cont.) =>
AND r.CheckOut < '25/03/01'
;
db2 (cont.) => db2 (cont.) =>
GUESTNAME                      GUESTPHONE  ASSISTANTNAME
------------------------------ ----------- ------------------------------
Ferris Larson                  7725863885  Michael Maddox
Guy Lee                        6193376542  Ingrid Velez
Travis Davis                   4664935746  Ingrid Velez

  3 record(s) selected.
```

Query 3

(a) This query calculates the total amount of money collected from a certain paid amenity, in the last month. Here, the total amount collected from *room service* in the last month is calculated.

(b) SELECT SUM(a.cost) AS Total
FROM Charges c, PaidAmenities a, Reservations r
WHERE a.name=c.name AND c.resid=r.resid AND a.name='Room Service' AND r.checkin >='25/02/04' AND r.checkin <='25/03/04' ;

(c) Executing the query:

```
db2 => SELECT SUM(a.cost) AS Total FROM Charges c, PaidAmenities a, Reservations r WHERE a.name=c.name A
ND c.resid=r.resid AND a.name='Room Service' AND r.checkin >='25/02/04' AND  r.checkin <='25/03/04' ;

TOTAL
-----------
         50

  1 record(s) selected.
```

Query 4

(a) For a VIP guest with a certain ID, display their name, their assigned assistant's name, and the number of times they came to the hotel. Here, we choose VIP with ID 59.

(b) SELECT g.name AS GuestName, s.Name AS HostName, COUNT(r.resID) AS VisitCount
FROM Guests g, Staff s, Assigned assign, reservations r
WHERE g.gid =assign.gid AND s.SID=assign.SID AND g.GID=59 AND r.Gid=g.gid
GROUP By g.name,s.name;

(c) Executing the query:

```
db2 => SELECT g.name AS GuestName, s.Name AS HostName, COUNT(r.resID) AS VisitCount
FROM Guests g, Staff sdb2 (cont.) => , Assigned assign, reservations r WHERE g.gid =assign.gid A
ND s.SID=assign.SID AND g.GID=59 AND r.Gid =g.gid GROUP By g.name,s.name;

GUESTNAME                    HOSTNAME                     VISITCOUNT
---------------------------- ---------------------------- -----------
Sybill Curtis                Venus Bishop                           3

  1 record(s) selected.
```

Query 5

(a) View all charges associated with reservation ID 5575, including the cost of the room and the cost of additional paid amenities.

(b) SELECT r.ResID, 'Room Cost' AS ChargeType, NULL AS AmenityName, r.Cost AS Cost, r.CardNum
FROM Reservations r
WHERE r.ResID = 5575

UNION ALL
SELECT c.ResID, 'Paid Amenity' AS ChargeType, c.Name AS AmenityName, a.Cost AS Cost, c.CardNum
FROM Charges c,PaidAmenities a
WHERE c.Name = a.Name AND c.ResID = 5575;

(c) Executing the query:

```
db2 => SELECT r.ResID, 'Room Cost' AS ChargeType, NULL AS AmenityName, r.Cost AS Cost, r.CardNum FROM Re
servations r WHERE r.ResID = 5575
UNION ALL
SELECT cdb2 (cont.) => db2 (cont.) => .ResID, 'Paid Amenity' AS ChargeType, c.Name AS AmenityName, a.Cos
t AS Cost, c.CardNum FROM Charges c,PaidAmenities a WHERE c.Name = a.Name AND c.ResID = 5575;

RESID       CHARGETYPE   AMENITYNAME                                        COST        CARDNUM
----------- ------------ -------------------------------------------------- ----------- ----------------
       5575 Paid Amenity Bar                                                         20 5344152757784353
       5575 Paid Amenity Room Service                                                50 5344152757784353
       5575 Room Cost    -                                                          100 5344152757784353

  3 record(s) selected.
```

6. SQL Modifications

Mod 1

(a) This statement updates the *TierDiscount* for all VIP guests whose average reservation cost exceeds $1000 to the highest available discount for VIPs. If a VIP guest already has the highest discount, their current discount remains unchanged.

(b) UPDATE VIPGuest
SET TierDiscount = (SELECT MAX(TierDiscount) FROM VIPGuest)
WHERE gid IN (
  SELECT gid
  FROM Reservations
  WHERE gid IN (SELECT gid FROM VIPGuest)
  GROUP BY gid
  HAVING AVG(Cost) > 1000
);

(c) To check that the update worked, we first show the average reservation cost and tier discount for each VIP guest, and compare their tier discount before and after the update. We observe that initially, VIP guests with *gid* 75, 41, and 65 have average reservation costs of over $1000, with tier discounts of 15, 5, and 10, respectively. After the update, their tier discounts are all 15. Also note that the tier discounts for all other VIP guests remain unchanged.

```
db2 => -- First check the avg reservation cost and tier discount for all VIPs before update;
db2 => SELECT r.gid, AVG(r.Cost) AS avgcost, v.TierDiscount
db2 (cont.) => FROM Reservations r, VIPGuest v WHERE r.gid = v.gid
db2 (cont.) => GROUP BY r.gid, v.TierDiscount  ORDER BY avgcost DESC;

GID         AVGCOST     TIERDISCOUNT
----------- ----------- ------------
         75       10200           15
         41        3270            5
         65        1700           10
         59         455           10
         19         325            5
         95         182           10
         92          95            5
          2          75            5

  8 record(s) selected.

db2 => -- Now we perform the update (modification)
db2 => UPDATE VIPGuest
db2 (cont.) => SET TierDiscount = (SELECT MAX(TierDiscount) FROM VIPGuest)
db2 (cont.) => WHERE gid IN ( SELECT gid FROM Reservations
db2 (cont.) => WHERE gid IN (SELECT gid FROM VIPGuest)
db2 (cont.) => GROUP BY gid HAVING AVG(Cost) > 1000
db2 (cont.) => );
DB20000I  The SQL command completed successfully.
db2 => -- Check that the update worked by checking the tier discount for all VIPs after update
db2 => SELECT r.gid, AVG(r.Cost) AS avgcost, v.TierDiscount
db2 (cont.) => FROM Reservations r, VIPGuest v WHERE r.gid = v.gid
db2 (cont.) => GROUP BY r.gid, v.TierDiscount  ORDER BY avgcost DESC;

GID         AVGCOST     TIERDISCOUNT
----------- ----------- ------------
         75       10200           15
         41        3270           15
         65        1700           15
         59         455           10
         19         325            5
         95         182           10
         92          95            5
          2          75            5

  8 record(s) selected.
```

Mod 2

(a) The hotel plans to gradually upgrade its Deluxe rooms, with a first phase taking place during the first seven days of April 2025. Only available Deluxe rooms, meaning those that are not reserved during this time period, will undergo partial maintenance, during which half of the area of each room will be closed off temporarily. The other half will still be operational as a Double room type, with a maximum capacity of two guests. To ensure that data about the rooms are correctly updated in the database, the *RoomType* and *Capacity* of the available Deluxe rooms are updated to *Double* and *2*, respectively.

(b) UPDATE Rooms r
SET Capacity = 2, RoomType = 'Double'

```
WHERE RoomNum IN (
  SELECT RoomNum
  FROM Rooms
  WHERE RoomType='Deluxe'
) AND NOT EXISTS (
  SELECT 1
  FROM Reservations res
  WHERE res.RoomNum = r.RoomNum
  AND ((res.CheckIn <= '25/04/07' AND res.CheckOut >= '25/04/01') OR
        (res.CheckIn >= '25/04/01' AND res.CheckOut <= '25/04/07'))
);
```

(c) Executing the statement in DB2:

```
db2 => -- Perform update
db2 => UPDATE Rooms r
db2 (cont.) => SET Capacity = 2, RoomType = 'Double'
db2 (cont.) => WHERE RoomNum IN (
db2 (cont.) => SELECT RoomNum FROM Rooms WHERE RoomType='Deluxe'
db2 (cont.) => ) AND NOT EXISTS (
db2 (cont.) =>   SELECT 1
db2 (cont.) =>   FROM Reservations res
db2 (cont.) =>   WHERE res.RoomNum = r.RoomNum
db2 (cont.) =>   AND ((res.CheckIn <= '25/04/07' AND res.CheckOut >= '25/04/01') OR
db2 (cont.) =>        (res.CheckIn >= '25/04/01' AND res.CheckOut <= '25/04/07'))
db2 (cont.) => );
DB20000I  The SQL command completed successfully.
db2 =>
```

7. Views

View 1 (single table)

(a) Find rooms where the guest count indicated in the reservation is greater than 4. Hotels want to check if there is enough space and beds for guests in a room or if overcrowding is occurring. They might offer an extra room if one becomes available. For some rooms, full capacity may be possible but not the most comfortable. For example, six people could fit into a certain room, but the hotel might have a policy to offer another room if it becomes available. A hotel might also want to make sure the number of people is spread out and avoid certain portions of the hotel being too busy, as this could lead to slower elevator traffic, parking issues, etc – depending on the layout of the hotel.

(b) CREATE VIEW BusyReservationInfo (ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount)
AS
SELECT ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount

FROM reservations
WHERE GuestCount >= 4;

(c) Executing the statement:

```
db2 => CREATE VIEW BusyReservationInfo (ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount)
db2 (cont.) => AS
db2 (cont.) => SELECT ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount
db2 (cont.) => FROM reservations
db2 (cont.) => WHERE GuestCount >= 4;
DB20000I  The SQL command completed successfully.
```

(d) Selecting everything from the view truncated to 5 records:

```
db2 => SELECT ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount
db2 (cont.) => FROM BusyReservationInfo;

RESID       ROOMNUM     GID         CHECKIN  CHECKOUT GUESTCOUNT
----------- ----------- ----------- -------- -------- -----------
       5576         401          65 25/09/15 25/09/17           5
       5584         301          77 25/03/22 25/03/29           4
       5587         401          75 25/07/15 25/07/30           4

  3 record(s) selected.
```

(e) Inserting a new record (reservation) into the view:

INSERT INTO BusyReservationInfo (ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount)
VALUES (9998, 401, 65, '25/09/19', '25/09/20', 5);

```
db2 => INSERT INTO BusyReservationInfo (ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount)
db2 (cont.) => VALUES (9998, 401, 65, '25/09/19', '25/09/20', 5);
DB20000I  The SQL command completed successfully.
```

(f) Above, we inserted another busy Reservation into BusyReservationInfo for Guest 65 in a room they stayed in previously that does not include a Book Date, CardNumber or Cost. The Insert into the view was successful and updated the base table Reservations through the view. Since the view definition contains all of the NOT NULL attributes of the Reservations table, we can properly insert a new Reservation.

Check New Full Reservation shows up in View →

```
db2 => SELECT ResID, RoomNum, gid, CheckIn, CheckOut, GuestCount FROM BusyReservationInfo;

RESID       ROOMNUM     GID         CHECKIN  CHECKOUT GUESTCOUNT
----------- ----------- ----------- -------- -------- -----------
       5576         401          65 25/09/15 25/09/17           5
       5584         301          77 25/03/22 25/03/29           4
       5587         401          75 25/07/15 25/07/30           4
       9998         401          65 25/09/19 25/09/20           5

  4 record(s) selected.
```

And the Insert is added to the base table, as shown below

```
db2 => select * from reservations where GuestCount >= 4;

RESID       CHECKIN  CHECKOUT GUESTCOUNT  ROOMNUM     GID         BOOKDATE CARDNUM          COST
----------- -------- -------- ----------- ----------- ----------- -------- ---------------- -----------
       5576 25/09/15 25/09/17           5         401          65 25/02/01 4485555916316713        1700
       5584 25/03/22 25/03/29           4         301          77 25/02/16 5543447445842442        1800
       5587 25/07/15 25/07/30           4         401          75 25/02/22 375331787775242        10200
       9998 25/09/19 25/09/20           5         401          65 -        -                          -

  4 record(s) selected.
```

View 2

(a) Shows information about the VIP guests (ID, name, and Tier Discount) and the ID and name of the assistant that is assigned to them. If a VIP is not assigned any assistant, the table should show no assistant associated with that VIP.

(b) CREATE VIEW VIPAssistInfo (VIPgid, VIPName, TierDiscount, AssistantID, AssistantName)
AS
SELECT V.gid as VIPgid, G.name as VIPName, V.TierDiscount as VIPTierDiscount, A.sid as AssistantID, S.name as AssistantName
FROM **Guests G**
JOIN **VIPGuest V** ON V.gid = G.gid
LEFT OUTER JOIN **Assigned Assign** ON V.gid = Assign.gid
LEFT OUTER JOIN **Assistants A** ON Assign.sid = A.sid
LEFT OUTER JOIN **Staff S** ON A.sid = S.sid ;

(c) Executing the statement:

```
db2 => CREATE VIEW VIPAssistInfo (VIPgid, VIPName, TierDiscount,  AssistantID, AssistantName)
AS
SELECT V.gid as VIPgid, G.name as VIPName, V.TierDiscount  as VIPTierDiscount, A.sid as AssistantID, S.name as AssistantName
FROM Guests G
JOIN VIPGuest V ON db2 (cont.) => V.gid = G.gid
LEFT OUTER JOIN Assigned Assign ON V.gid = Assign.gid
LEFT OUTER JOIN Assistants A ON Assign.sid = A.sid
LEFT OUTER JOIN Staff S ON A.sid = S.sid ;
DB20000I  The SQL command completed successfully.
```

(d) Selecting everything from the view truncated to 5 records:

```
db2 => SELECT VIPgid, VIPName, TierDiscount,  AssistantID, AssistantName FROM VIPAssistInfo LIMIT 5;

VIPGID      VIPNAME                         TIERDISCOUNT ASSISTANTID ASSISTANTNAME
----------- ------------------------------- ------------ ----------- ------------------------------
         19 Nita Hickman                               5           4 Ingrid Velez
          2 Cassidy Vang                               5          11 Venus Bishop
          4 Daniel Sloan                               5          11 Venus Bishop
          7 Matthew Mcfarland                          5          13 Dara William
         11 Quynn Skinner                              5          13 Dara William

  5 record(s) selected.
```

→ quick insert into **base table** to show padding

```
INSERT INTO Guests (gid,name,email,phone)
VALUES (999,'Mr.NoAssist','Mr.noAssist@gmail.ca','7729772839') ;
INSERT INTO VIPGuest (gid,TierDiscount)
VALUES  (999,15);
```

```
db2 => INSERT INTO Guests (gid,name,email,phone)
VALUES
        (999,'Mr.NoAssist','Mr.noAssist@gmail.ca','7729772839');db2 (cont.) =>
DB20000I  The SQL command completed successfully.
db2 => INSERT INTO VIPGuest (gid,TierDiscount)
VALUES
(999,15);
db2 (cont.) => db2 (cont.) => DB20000I  The SQL command completed successfully.
```

SELECT VIPgid, VIPName, TierDiscount, AssistantID,
AssistantName From VIPAssistInfo Where VIPgid = 999;

```
db2 => SELECT VIPgid, VIPName, TierDiscount, AssistantID, AssistantName From VIPAssistInfo Where VIPgid = 999;

VIPGID      VIPNAME                         TIERDISCOUNT ASSISTANTID ASSISTANTNAME
----------- ------------------------------- ------------ ----------- ------------------------------
        999 Mr.NoAssist                                15           - -

  1 record(s) selected.
```

(e) Inserting a new record into the view:

Suppose we have some new VIP Mr.Albertt like we did in View 1.

INSERT INTO Guests (gid,name,email,phone)
VALUES (998,'Mr.Albertt','Mr.Albertt@gmail.ca','7739772839') ;

INSERT INTO VIPGuest (gid,TierDiscount)
VALUES (998, 20);

And Lev Hartman is in the Assistant table with SID 5

We try to add Mr.Albertt with Lev Hartman as his assistant into VIPAssistInfo like we did in View 1 (note AssistantName can be NULL)…

INSERT INTO VIPAssistInfo (VIPgid, VIPName, TierDiscount, AssistantID)
VALUES (998, 'Mr.Albertt', 20, 5);

```
db2 => INSERT INTO VIPAssistInfo (VIPgid, VIPName, TierDiscount,  AssistantID) VALUES (998, 'Mr.Albertt', 20, 5);
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL0150N  The target fullselect, view, typed table, materialized query table,
range-clustered table, or staging table in the INSERT, DELETE, UPDATE, MERGE,
or TRUNCATE statement is a target for which the requested operation is not
permitted.  SQLSTATE=42807
```

SQLSTATE=42807

(f) This View includes the joining of multiple different tables (three LEFT OUTER JOINs) in its definition. According to the DB2 manual, inserts into a view are prohibited if the view contains complex constructs such as a **join**, a GROUP BY, or a HAVING clause. Therefore, this view cannot be updated through insertion– unlike in view 1.

To deal with this issue you can use an INSTEAD OF trigger to handle insertion.

8. Check Constraints

Check 1

(a) We added a constraint called *discountval* on the *VIPGuest* table to restrict the values in the *TierDiscount* column to only 5, 10, or 15. These values represent the discount percentages for the loyalty program tiers of the hotel's VIP guests. By applying this constraint, we ensure that only valid discount percentages are stored, allowing us to correctly identify the VIP guests in each tier and ensure they receive the appropriate benefits. Furthermore, using a named constraint allows us to easily modify the constraint later if needed, e.g., if the hotel revisits its VIP tier system and decides to adjust the tier discounts.

(b) ALTER TABLE VipGuest
ADD CONSTRAINT discountval
CHECK (TierDiscount IN (5,10,15));

(c) Running the command:

```
db2 => ALTER TABLE VIPGuest
db2 (cont.) => ADD CONSTRAINT discountval
db2 (cont.) => CHECK (TierDiscount IN (5,10,15));
DB20000I  The SQL command completed successfully.
```

(d) Inserting a record that violates the constraint (TierDiscount of 1 instead of 5, 10, or 15):

```
db2 => -- Create dummy guest for example
db2 => INSERT INTO Guests (gid, name) VALUES (1111, 'Bad VIP');
DB20000I  The SQL command completed successfully.
db2 =>
db2 => -- Insert record that violates constraint
db2 => INSERT INTO VIPGuest (gid, TierDiscount) VALUES (1111, 1);
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL0545N  The requested operation is not allowed because a row does not
satisfy the check constraint "CS421G35.VIPGUEST.DISCOUNTVAL".  SQLSTATE=23513
db2 =>
db2 => -- Remove dummy guest from Guests table
db2 => DELETE FROM Guests WHERE gid=1111;
DB20000I  The SQL command completed successfully.
db2 =>
```

Note that we first created a dummy value in the parent table since *gid* in *VIPGuest* is a foreign key to *gid* in *Guests*, then deleted it at the end.
The error returned by the database is:

> SQL0545N The requested operation is not allowed because a row does not satisfy the check constraint "CS421G35.VIPGUEST.DISCOUNTVAL". SQLSTATE=23513

Check 2

(a) We added a constraint called *checkinout* on the *Reservations* table to enforce that the *CheckOut* date selected by the guest is always later than the *CheckIn* date for every reservation. This ensures there are no illogical or incorrect bookings in terms of the dates.

(b) ALTER TABLE Reservations
ADD CONSTRAINT checkinout
CHECK (CheckOut > CheckIn);

(c) Running the command:

```
db2 => ALTER TABLE Reservations
db2 (cont.) => ADD CONSTRAINT checkinout
db2 (cont.) => CHECK (CheckOut > CheckIn);
DB20000I  The SQL command completed successfully.
```

(d) Inserting a record that violates the constraint (CheckIn on March 1, 2025 and CheckOut on February 1, 2025):

```
db2 => INSERT INTO Reservations (ResID,CheckIn,CheckOut,GuestCount,RoomNum,gid)
db2 (cont.) => VALUES (1, '25/03/01', '25/02/01', 1, 101, 1)
db2 (cont.) => ;
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL0545N  The requested operation is not allowed because a row does not
satisfy the check constraint "CS421G35.RESERVATIONS.CHECKINOUT".
SQLSTATE=23513
db2 =>
```

The error returned by the database is:

> SQL0545N The requested operation is not allowed because a row does not satisfy the check constraint "CS421G35.RESERVATIONS.CHECKINOUT".
> SQLSTATE=23513

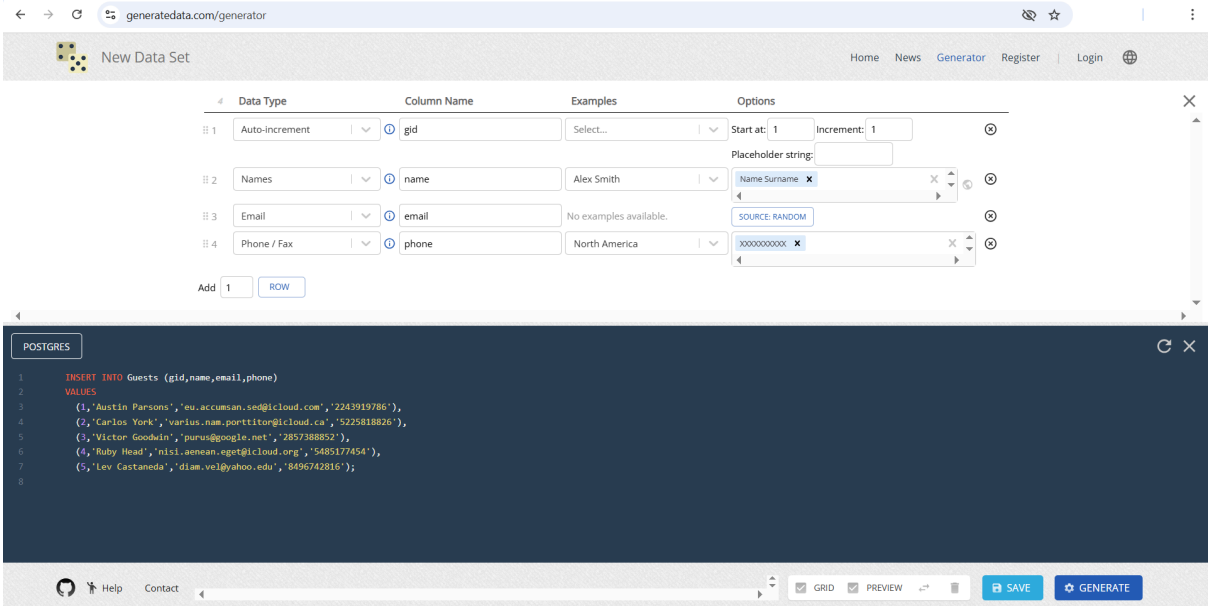
9. Creativity

Real data sets

We generated meaningful data for the *Guests* table, which stores information regarding the guests at the hotel, such as their guest id (GID), name, email, and phone number. The *CasualGuests*, *VIPGuest*, and *Reservations* tables all have a foreign key to the *Guests* table (GID). To generate the data, we used *generatedata.com*, an open source data generation tool that allows users to create realistic test data. We generated fields for the guest id, guest name, email address, and phone number using the following data types from the tool: Auto-increment, Names, Email, and Phone/Fax. See the screenshot below for details. We used the tool's "GENERATE" button to generate an insert statement in PostgreSQL format with 100 records. This insert statement was also compatible with the DB2 format so we did not need to modify it. Then, to separate the guests into the *CasualGuest*s and *VIPGuest*

tables, we manually randomly divided the guest ids (foreign keys) such that around 70% of guests are casual guests and 30% are VIP guests.

Screenshot of the options we selected in *generatedata.com* to generate the realistic data:



Advanced SQL Features

We created a trigger to enforce a rule that depends on data across two different tables, as this is not possible with a CHECK constraint:

(a) This trigger, called *check_guest_count*, creates a constraint in the *Reservations* table that ensures that the number of guests (*GuestCount*) for any given reservation does not exceed the capacity of the room being booked, which is indicated in the *Rooms* table. Whenever we try to insert a faulty reservation with respect to the guest count into the *Reservations* table, the trigger will raise an error to prevent the insertion.

(b)
```
CREATE TRIGGER check_guest_count
  BEFORE INSERT ON Reservations
  REFERENCING NEW AS new_res
  FOR EACH ROW
BEGIN ATOMIC
  DECLARE room_capacity INT;

  -- get the capacity of the room being booked
  SET room_capacity = (
    SELECT Capacity
    FROM Rooms
    WHERE RoomNum = new_res.RoomNum
  );

  -- check if the guest count exceeds the capacity
```

```
  IF new_res.GuestCount > room_capacity THEN
    SIGNAL SQLSTATE '75001' ('Guest count exceeds room capacity.');
  END IF;
END;
```

(c) To test the query, we try to insert a reservation into the *Reservations* table with a guest count that exceeds the capacity of the chosen room. Here, we choose room number 101, which has a capacity of 1, and a guest count of 2. Note that to create the trigger, we first changed the statement terminator to "@" since ";" is used when defining the trigger.

```
db2 => -- Create the trigger
db2 => --#SET TERMINATOR @
db2 => CREATE TRIGGER check_guest_count
db2 (cont.) =>    BEFORE INSERT ON Reservations
db2 (cont.) =>    REFERENCING NEW AS new_res
db2 (cont.) =>    FOR EACH ROW
db2 (cont.) => BEGIN ATOMIC
db2 (cont.) =>    DECLARE room_capacity INT;
db2 (cont.) =>    -- get the capacity of the room being booked
db2 (cont.) =>     SET room_capacity = (
db2 (cont.) =>        SELECT Capacity
db2 (cont.) =>        FROM Rooms
db2 (cont.) =>        WHERE RoomNum = new_res.RoomNum
db2 (cont.) =>      );
db2 (cont.) =>    -- check if the guest count exceeds the capacity
db2 (cont.) =>    IF new_res.GuestCount > room_capacity THEN
db2 (cont.) =>      SIGNAL SQLSTATE '75001' ('Guest count exceeds room capacity.');
db2 (cont.) =>    END IF;
db2 (cont.) => END@
DB20000I  The SQL command completed successfully.
db2 => --#SET TERMINATOR ;
db2 => -- We now test the trigger and try to insert a faulty reservation
db2 => -- First check the capacity of room with RoomNum=101
db2 => SELECT Capacity FROM Rooms WHERE RoomNum=101;

CAPACITY
-----------
          1

  1 record(s) selected.

db2 => -- Now, we try to insert a reservation with guest count of 2 > 1 for room 101
db2 => INSERT INTO Reservations (ResID, GuestCount, RoomNum, gid)
db2 (cont.) => VALUES (1111, 2, 101, 1);
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL0438N  Application raised error or warning with diagnostic text: "Guest
count exceeds room capacity.".  SQLSTATE=75001
db2 =>
```

10. How we worked together

We had a Zoom meeting at the beginning to plan the project and communicated through our group chat afterwards. We each worked on around 1-2 separate tasks while collaborating and making sure everything is cohesive.