```
-- Viveka Agrawal
-- 1. SQL DDL statements --
CREATE SCHEMA IF NOT EXISTS zot_pets;
CREATE SCHEMA
SET search_path TO zot_pets;
SET
-- Use CASCADE to automatically drop dependent views and objects
DROP TABLE IF EXISTS "appointment_service" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "purchase" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "appointment" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "pet" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "service" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "product" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "groomer" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "pet_owner" CASCADE;
DROP TABLE
DROP TABLE IF EXISTS "user" CASCADE;
DROP TABLE
DROP TYPE IF EXISTS service_type_enum;
DROP TYPE
CREATE TYPE service_type_enum AS ENUM ('Bath', 'Haircut', 'Nail
Trim');
CREATE TYPE
CREATE TABLE "user" (
"user_id" SERIAL PRIMARY KEY,
"name" VARCHAR(255) NOT NULL,
"email" VARCHAR(255) NOT NULL UNIQUE,
"street" VARCHAR(255),
"city" VARCHAR(100),
"state" VARCHAR(100),
"join_date" DATE NOT NULL DEFAULT CURRENT_DATE
);
CREATE TABLE
CREATE TABLE "pet_owner" (
"user_id" INT PRIMARY KEY,
"payment_info" VARCHAR(255),
CONSTRAINT "fk_pet_owner_user"
FOREIGN KEY("user_id")
REFERENCES "user"("user_id")
ON DELETE CASCADE
);
CREATE TABLE
CREATE TABLE "groomer" (
```

```
"user_id" INT PRIMARY KEY,
"bio" TEXT,
"certification" VARCHAR(255),
"rating" NUMERIC(3, 2) CHECK ("rating" >= 0 AND "rating" <= 5),
CONSTRAINT "fk_groomer_user"
FOREIGN KEY("user_id")
REFERENCES "user"("user_id")
ON DELETE CASCADE
);
CREATE TABLE
CREATE TABLE "pet" (
"owner_id" INT NOT NULL,
"pet_id" SERIAL NOT NULL,
"name" VARCHAR(255) NOT NULL,
"species" VARCHAR(100),
"breed" VARCHAR(100),
"dob" DATE,
"notes" TEXT,
"photo_urls" VARCHAR(2048),
PRIMARY KEY ("owner_id", "pet_id"),
CONSTRAINT "fk_pet_owner"
FOREIGN KEY("owner_id")
REFERENCES "pet_owner"("user_id")
ON DELETE CASCADE
);
CREATE TABLE
CREATE TABLE "service" (
"groomer_id" INT NOT NULL,
"service_id" SERIAL NOT NULL,
"service_type" service_type_enum NOT NULL,
"description" TEXT,
"price" NUMERIC(10, 2) NOT NULL CHECK ("price" >= 0),
PRIMARY KEY ("groomer_id", "service_id"),
CONSTRAINT "fk_service_groomer"
FOREIGN KEY("groomer_id")
REFERENCES "groomer"("user_id")
ON DELETE CASCADE
);
CREATE TABLE
CREATE TABLE "product" (
"product_id" SERIAL PRIMARY KEY,
"name" VARCHAR(255) NOT NULL,
"price" NUMERIC(10, 2) NOT NULL CHECK ("price" >= 0)
);
CREATE TABLE
CREATE TABLE "appointment" (
"appointment_id" SERIAL PRIMARY KEY,
"pet_owner_id" INT NOT NULL,
"pet_id" INT NOT NULL,
"appointment_datetime" TIMESTAMP WITH TIME ZONE NOT NULL,
```

```
CONSTRAINT "fk_appointment_pet"
FOREIGN KEY("pet_owner_id", "pet_id")
REFERENCES "pet"("owner_id", "pet_id")
ON DELETE CASCADE
);
CREATE TABLE
CREATE TABLE "appointment_service" (
"appointment_id" INT NOT NULL,
"service_groomer_id" INT NOT NULL,
"service_id" INT NOT NULL,
PRIMARY KEY ("appointment_id", "service_groomer_id", "service_id"),
CONSTRAINT "fk_appointment"
FOREIGN KEY("appointment_id")
REFERENCES "appointment"("appointment_id")
ON DELETE CASCADE,
CONSTRAINT "fk_service"
FOREIGN KEY("service_groomer_id", "service_id")
REFERENCES "service"("groomer_id", "service_id")
ON DELETE CASCADE
);
CREATE TABLE
CREATE TABLE "purchase" (
"purchase_id" SERIAL PRIMARY KEY,
"pet_owner_id" INT NOT NULL,
"product_id" INT NOT NULL,
"groomer_id" INT NOT NULL,
"purchase_date" DATE NOT NULL DEFAULT CURRENT_DATE,
CONSTRAINT "fk_purchase_owner"
FOREIGN KEY("pet_owner_id")
REFERENCES "pet_owner"("user_id")
ON DELETE CASCADE,
CONSTRAINT "fk_purchase_product"
FOREIGN KEY("product_id")
REFERENCES "product"("product_id")
ON DELETE RESTRICT,
CONSTRAINT "fk_purchase_groomer"
FOREIGN KEY("groomer_id")
REFERENCES "groomer"("user_id")
ON DELETE SET NULL
);
CREATE TABLE
-- Pets table field design observations
/*

owner_id: This value is INT since it is a number assigned to the owner
of a pet. It must be included as it
can't be NULL.

pet_id: This value is SERIAL, which is an auto-number which makes each
pet unique. This works with owner_id
```

to make sure there are no duplicate pets. It must be included as it can't be NULL.

name: This value is VARCHAR(255). Its a text field for a pet's name. It can store up to 255 characters
for long names. It must be included as it can't be NULL.

species: This value is VARCHAR(100). It specifies an animal type like a dog or a cat. It can be an optional
field since the species of a pet may be unknown.

breed: This value is VARCHAR(100). It specifies the breed information of a pet. It can be an optional
field since the breed of a pet may be unknown.

notes: This value is TEXT to provide extra information about the pet in words. It can be an optional
field used to write special facts about a pet.

dob: This value is DATE to store the birth date of a pet. It can be an optional field since some pets
have unknown birthdays.

photo_urls: This value is VARCHAR(2048) as it stores multiple photo URLs. It has space for several URLs
but can run out of space for pets that have many photos.

*/
-- 2. Data Loading (COPY commands) --
\copy zot_pets.user(user_id, name, email, street, city, state, join_date) FROM '/Users/viv/Downloads/ZotPetsData/user.csv' WITH (FORMAT csv, HEADER true);
COPY 150
\copy zot_pets.pet_owner(user_id, payment_info) FROM '/Users/viv/Downloads/ZotPetsData/pet_owner.csv' WITH (FORMAT csv, HEADER true);
COPY 143
\copy zot_pets.groomer(user_id, bio, certification, rating) FROM '/Users/viv/Downloads/ZotPetsData/groomer.csv' WITH (FORMAT csv, HEADER true);
COPY 45
\copy zot_pets.pet(owner_id, pet_id, name, species, breed, dob, notes, photo_urls) FROM '/Users/viv/Downloads/ZotPetsData/pet.csv' WITH (FORMAT csv, HEADER true);
COPY 200
\copy zot_pets.service(groomer_id, service_id, service_type, description, price) FROM '/Users/viv/Downloads/ZotPetsData/service.csv' WITH (FORMAT csv, HEADER true);
COPY 250
\copy zot_pets.product(product_id, name, price) FROM '/Users/viv/Downloads/ZotPetsData/product.csv' WITH (FORMAT csv, HEADER true);

```
COPY 8
\copy zot_pets.appointment(appointment_id, pet_owner_id, pet_id,
appointment_datetime) FROM '/Users/viv/Downloads/ZotPetsData/
appointment.csv' WITH (FORMAT csv, HEADER true);
COPY 400
\copy zot_pets.appointment_service(appointment_id, service_groomer_id,
service_id) FROM '/Users/viv/Downloads/ZotPetsData/
appointment_service.csv' WITH (FORMAT csv, HEADER true);
COPY 602
\copy zot_pets.purchase(purchase_id, pet_owner_id, product_id,
groomer_id, purchase_date) FROM '/Users/viv/Downloads/ZotPetsData/
purchase.csv' WITH (FORMAT csv, HEADER true);
COPY 500
-- 3. Query Answers --
-- Problem A --
-- I used subqueries to make sure to only use 1 SELECT statement
SELECT (SELECT COUNT(*) FROM zot_pets.user) AS number_of_users,
       (SELECT COUNT(*) FROM zot_pets.pet) AS number_of_pets,
       (SELECT COUNT(*) FROM zot_pets.appointment) AS
number_of_appointments;
 number_of_users | number_of_pets | number_of_appointments
-----------------+----------------+------------------------
             150 |            200 |                    400
(1 row)

-- Problem B --
-- Since 2 tables are being used, the user's id and groomer's id must
be joined
SELECT u.name, g.rating
FROM zot_pets.user u, zot_pets.groomer g
WHERE u.user_id = g.user_id
        AND g.rating >= 4.5
ORDER BY g.rating DESC;
        name        | rating
--------------------+--------
 Connie Wiggins     |   4.98
 Tyler Perez        |   4.93
 Stephen Cox        |   4.91
 Michelle Martinez  |   4.87
 Jeffrey Martinez   |   4.86
 Sabrina Bryan      |   4.78
 Dr. Angela Skinner |   4.77
 Jeffrey Spencer    |   4.59
 Mark Collins       |   4.52
 Katherine Jacobson |   4.52
(10 rows)

-- Problem C --
-- In order to get the address, I combined street, city, and state as
one entity
```

```
SELECT u.user_id, u.email, (u.street, u.city, u.state) AS address
FROM zot_pets.user u
JOIN zot_pets.groomer g ON u.user_id = g.user_id
JOIN zot_pets.pet_owner po ON u.user_id = po.user_id
WHERE u.email LIKE '%@example.com';
 user_id |             email             |
address
---------+-------------------------------
+-----------------------------------------------
      26 | ycarter@example.com           | ("96307 Stacy Plaza Apt.
273","Port Barbara",OR)
      29 | shelbyhardy@example.com       | ("9152 Moore Crest","Lake
Joshuabury",TX)
      56 | tammy60@example.com           | ("594 Vargas Fall Apt.
393","East Bradleyside",NM)
      90 | wardkatie@example.com         | ("3737 Lara Courts Suite
318",Montgomeryfurt,CT)
      93 | phillipsfaith@example.com     | ("3825 Ellis Estate Apt.
839",Jerryville,CO)
     104 | bmorrison@example.com         | ("993 Reynolds
Loaf",Kelleyfurt,ME)
     111 | andrew06@example.com          | ("3362 Matthew
Meadow","East Jacqueline",IN)
     112 | kentdarren@example.com        | ("0216 Hall Loop Suite
944",Josephberg,AS)
     124 | hernandezjennifer@example.com | ("50709 Kelly Forge","Port
Carolyn",TX)
     133 | carsonbenjamin@example.com    | ("092 Lance Tunnel Apt.
858",Linside,NH)
     135 | wreyes@example.com            | ("3277 Tara Stream Apt.
597",Martinezton,PW)
(11 rows)

-- Problem D --
-- I separated the date and time from the timestamp in order to get
the individual values
SELECT a.appointment_id, a.appointment_datetime::date AS date,
a.appointment_datetime::time AS time, s.service_type
FROM zot_pets.appointment a, zot_pets.appointment_service aps,
zot_pets.service s
WHERE a.appointment_id = aps.appointment_id
  AND aps.service_id = s.service_id
  AND aps.service_groomer_id = s.groomer_id
  AND a.pet_id = 5
ORDER BY a.appointment_datetime ASC;
 appointment_id |    date    |   time   | service_type
----------------+------------+----------+--------------
             94 | 2024-09-13 | 17:29:57 | Haircut
             94 | 2024-09-13 | 17:29:57 | Nail Trim
            179 | 2025-08-03 | 12:20:04 | Nail Trim
```

```
            179 | 2025-08-03 | 12:20:04 | Haircut
            179 | 2025-08-03 | 12:20:04 | Haircut
(5 rows)

-- Problem E --
-- I used GROUP BY to get the number of baths, haircuts, and nail
trims separately
SELECT service_type, COUNT(*) AS total_times_service_performed
FROM zot_pets.service
WHERE groomer_id = 11
GROUP BY service_type
ORDER BY service_type;
 service_type | total_times_service_performed
--------------+-------------------------------
 Bath         |                             2
 Haircut      |                             1
 Nail Trim    |                             2
(3 rows)

-- Problem F --
-- I used the JOIN statements to get from user to pet_owner to pet to
access pet information
-- I used HAVING COUNT(DISTINCT) to count the number of unique species
SELECT u.name, u.email
FROM zot_pets.user u
JOIN zot_pets.pet_owner po ON u.user_id = po.user_id
JOIN zot_pets.pet p ON po.user_id = p.owner_id
GROUP BY u.user_id, u.name, u.email
HAVING COUNT(DISTINCT p.species) >= 3;
       name       |           email
------------------+---------------------------
 Candice Gonzales | annahorn@example.com
 Heidi Ryan       | annemoore@example.org
 Matthew Chambers | elizabeth81@example.net
 Morgan Ramos     | griffithglenda@example.org
 Michael Lopez    | john43@example.net
 Brenda Good      | john70@example.net
 John Baker       | karen21@example.net
 Jeffrey Ross     | lindahernandez@example.org
 Steven Ford      | mike66@example.net
 Darren Boone     | scottmichael@example.org
 Tammy Parker     | tmoore@example.net
(11 rows)

-- Problem G --
-- I used the BOOL_OR() function to make sure baths and haircuts are
included but not nail trims
-- I used GROUP BY groomer id to make sure the services provided by
each groomer are together in the output
SELECT g.user_id, u.email
```

```
FROM zot_pets.groomer g, zot_pets.user u, zot_pets.service s,
zot_pets.appointment_service aps
WHERE g.user_id = u.user_id
  AND g.user_id = s.groomer_id
  AND s.groomer_id = aps.service_groomer_id
  AND s.service_id = aps.service_id
GROUP BY g.user_id, u.email
HAVING BOOL_OR(s.service_type = 'Bath')
   AND BOOL_OR(s.service_type = 'Haircut')
   AND NOT BOOL_OR(s.service_type = 'Nail Trim')
ORDER BY g.user_id ASC;
 user_id |           email
---------+---------------------------
      68 | torressusan@example.org
      89 | bknight@example.net
     109 | john70@example.net
     134 | callahannatalie@example.org
(4 rows)

-- Problem H --
-- I made sure to filter out NULL values and empty photo_urls so that
they do not count as part of the count
-- I used LIMIT to return only the pets with most photos
SELECT pet_id AS "PetID", name AS "Name",
array_length(string_to_array(photo_urls, ','), 1) AS num_photos
FROM zot_pets.pet
WHERE photo_urls IS NOT NULL
        AND photo_urls != ''
ORDER BY num_photos DESC
LIMIT 10;
 PetID |   Name    | num_photos
-------+-----------+------------
     1 | Erin      |          3
     1 | Catherine |          3
     1 | Jason     |          3
     1 | Shawn     |          3
     1 | Cheryl    |          3
     1 | Patricia  |          3
     1 | Amber     |          3
     1 | Gregory   |          3
     1 | Lori      |          3
     1 | Jason     |          3
(10 rows)

-- Problem I --
-- This part calculates the annual product spending per customer from
purchases
-- I used EXTRACT(YEAR FROM) to find the year from the purchase date
WITH product_spending AS (
    SELECT EXTRACT(YEAR FROM p.purchase_date) AS year, p.pet_owner_id
```

```
AS user_id, SUM(pd.price) AS total_product_spending
    FROM zot_pets.purchase p, zot_pets.product pd
    WHERE p.product_id = pd.product_id
    GROUP BY year, p.pet_owner_id
),
-- This part calculates the annual service spending per customer from
appointments
-- I used EXTRACT(YEAR FROM) to find the year from the appointment
timestamp
appointment_spending AS (
    SELECT EXTRACT(YEAR FROM a.appointment_datetime) AS year,
a.pet_owner_id, SUM(s.price) AS total_service_spending
    FROM zot_pets.appointment a, zot_pets.appointment_service appts,
zot_pets.service s
    WHERE a.appointment_id = appts.appointment_id
      AND appts.service_id = s.service_id
      AND appts.service_groomer_id = s.groomer_id
    GROUP BY year, a.pet_owner_id
),
-- This part combines product and service spending
-- I used UNION ALL to get all the spending records and then group
them together
-- I used MAX and SUM to combine the product and service spending for
each customer
combined_prod_serve_spend AS (
    SELECT
        user_id AS pet_owner_id, year,
                MAX(total_service_spending) AS total_service_spending,
        MAX(total_product_spending) AS total_product_spending,
        MAX(total_service_spending) + MAX(total_product_spending) AS
total_spending
    FROM (
                -- Service spending records
        SELECT pet_owner_id AS user_id, year, 0 AS
total_product_spending, total_service_spending
        FROM appointment_spending
                UNION ALL
        -- Product spending records
        SELECT user_id, year, total_product_spending, 0 AS
total_service_spending
        FROM product_spending
    )
    GROUP BY user_id, year
),
-- This part calculates the previous year's spending for comparison
-- I used self-joins on the combined table to find spending from the
previous year
-- I used COALESCE to handle cases where no previous year data exists
combined_with_previous AS (
    SELECT
```

```
        current_year.pet_owner_id,
        current_year.year,
                current_year.total_service_spending,
        current_year.total_product_spending,
        current_year.total_spending,
        COALESCE(previous_year.total_spending, 0) AS
previous_year_spending
    FROM combined_prod_serve_spend current_year
    LEFT JOIN combined_prod_serve_spend previous_year
        ON current_year.pet_owner_id = previous_year.pet_owner_id
        AND current_year.year = previous_year.year + 1
),
-- This part ranks customers by total spending within each year
-- I used ROW_NUMBER() to assign rank 1 to the highest spender each
year
customer_rankings AS (
    SELECT
        spending.year,
        customer.user_id,
        customer.name,
        customer.email,
        spending.total_product_spending,
        spending.total_service_spending,
        spending.total_spending,
        spending.previous_year_spending,
        ROW_NUMBER() OVER (PARTITION BY spending.year ORDER BY
spending.total_spending DESC) AS ranking
    FROM combined_with_previous spending, zot_pets.user customer
    WHERE spending.pet_owner_id = customer.user_id
)
-- This part shows the final result (finding the top customer where
rank = 1 for each year)
SELECT
    year,
    user_id,
    name,
    email,
        total_service_spending,
    total_product_spending,
    total_spending,
    previous_year_spending
FROM customer_rankings
WHERE ranking = 1
ORDER BY year;
 year | user_id |      name      |            email            |
total_service_spending | total_product_spending | total_spending |
previous_year_spending
------+---------+----------------+-----------------------------
+----------------------+----------------------+---------------
+----------------------
```

```
 2023 |     127 | Joseph Li      | gcardenas@example.org       |
546.57 |                  56.73 |         603.30 |
0
 2024 |       8 | William Flores | alicia18@example.com        |
1121.60 |                 190.67 |        1312.27 |
75.85
 2025 |     134 | Mark Collins   | callahannatalie@example.org |
1498.62 |                       0 |        1498.62 |
319.07
(3 rows)

-- Problem J --
-- I created the view to calculate lifetime value
CREATE VIEW PetOwnerLTV AS
SELECT user_id, SUM(spending) AS calculated_ltv
FROM (
    -- This part calculates the service spending, which is the sum of
all service prices from appointments
    SELECT a.pet_owner_id AS user_id, s.price AS spending
    FROM zot_pets.appointment a, zot_pets.appointment_service aps,
zot_pets.service s
    WHERE a.appointment_id = aps.appointment_id
      AND aps.service_groomer_id = s.groomer_id
      AND aps.service_id = s.service_id
    UNION ALL
    -- This part calculates the product spending, which is the sum of
all product prices from purchases
    SELECT p.pet_owner_id AS user_id, pr.price AS spending
    FROM zot_pets.purchase p, zot_pets.product pr
    WHERE p.product_id = pr.product_id
)
GROUP BY user_id;
CREATE VIEW
 -- Table alteration DDL:
-- This part adds the new column
ALTER TABLE zot_pets.pet_owner
ADD COLUMN lifetime_value NUMERIC(10, 2) NOT NULL DEFAULT 0.00;
ALTER TABLE

 -- Table update query:
-- This part updates the new column with values from the view
-- I used COALESCE to handle the cases where pet owners have no
spending history
--      and set those cases to 0.00 to satisfy the NOT NULL constraint
UPDATE zot_pets.pet_owner po
SET lifetime_value = COALESCE(
    (
        SELECT calculated_ltv
        FROM PetOwnerLTV ltv
        WHERE ltv.user_id = po.user_id
```

```
    ),
    0.00
);
UPDATE 143

 -- Change verification query:

-- This part checks whether the lifetime_value column correctly stores
PetOwnerLTV view calculations
-- I used WHERE conditions to connect pet_owner, user, and PetOwnerLTV
view
-- This orders by the highest LTV to make sure that the top spenders
are correctly materialized
SELECT po.user_id, u.name, po.lifetime_value AS materialized_ltv,
ltv.calculated_ltv AS view_ltv
FROM zot_pets.pet_owner po, zot_pets.user u, PetOwnerLTV ltv
WHERE po.user_id = u.user_id
  AND po.user_id = ltv.user_id
ORDER BY po.lifetime_value DESC
LIMIT 10;
 user_id |       name       | materialized_ltv | view_ltv
---------+------------------+------------------+----------
     100 | Steven Ford      |          1997.32 |  1997.32
     134 | Mark Collins     |          1841.75 |  1841.75
     127 | Joseph Li        |          1837.60 |  1837.60
      40 | Nicole Dean      |          1820.15 |  1820.15
     139 | James Black      |          1805.25 |  1805.25
      28 | Morgan Ramos     |          1700.62 |  1700.62
      34 | Tammy Parker     |          1681.73 |  1681.73
     109 | Brenda Good      |          1639.20 |  1639.20
     150 | Candice Gonzales |          1571.36 |  1571.36
       8 | William Flores   |          1509.28 |  1509.28
(10 rows)
```