# **PostgreSQL Exam Preparation I**

|  |  |
| --- | --- |
| **Database Design** | **Creators without Board Games** |
| CREATE TABLE categories(          id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,          name VARCHAR(50) NOT NULL      );        CREATE TABLE addresses(          id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,          street\_name VARCHAR(100) NOT NULL,          street\_number INT NOT NULL,          town VARCHAR(30) NOT NULL,          country VARCHAR(50) NOT NULL,          zip\_code INT NOT NULL,            CONSTRAINT ck\_street\_number\_is\_positive          CHECK (street\_number > 0),          CONSTRAINT ck\_zip\_code\_is\_positive          CHECK (zip\_code > 0)      );    CREATE TABLE publishers(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(30) NOT NULL,      address\_id INT NOT NULL,      website VARCHAR(40),      phone VARCHAR(20),      FOREIGN KEY (address\_id)      REFERENCES addresses(id)      ON UPDATE CASCADE      ON DELETE CASCADE  );  CREATE TABLE players\_ranges(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      min\_players INT NOT NULL,      max\_players INT NOT NULL,      CONSTRAINT ck\_min\_players\_is\_positive      CHECK (min\_players > 0),      CONSTRAINT ck\_max\_players\_is\_positive      CHECK (max\_players > 0)  );  CREATE TABLE creators(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      first\_name VARCHAR(30) NOT NULL,      last\_name VARCHAR(30) NOT NULL,      email VARCHAR(30) NOT NULL  );  CREATE TABLE board\_games(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(30) NOT NULL,      release\_year INT NOT NULL,      rating NUMERIC(2) NOT NULL,      category\_id INT NOT NULL,      publisher\_id INT NOT NULL,      players\_range\_id INT NOT NULL,      CONSTRAINT ck\_release\_year\_is\_positive      CHECK (release\_year > 0),      FOREIGN KEY (category\_id)      REFERENCES categories(id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (publisher\_id)      REFERENCES publishers(id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (players\_range\_id)      REFERENCES players\_ranges(id)      ON UPDATE CASCADE      ON DELETE CASCADE  );  CREATE TABLE creators\_board\_games(      creator\_id INT NOT NULL,      board\_game\_id INT NOT NULL,      FOREIGN KEY (creator\_id)      REFERENCES creators(id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (board\_game\_id)      REFERENCES board\_games(id)      ON UPDATE CASCADE      ON DELETE CASCADE  ); | SELECT      cr.id,      CONCAT(cr.first\_name, ' ', cr.last\_name) AS "creator\_name",      cr.email  FROM      creators AS cr  LEFT JOIN      creators\_board\_games AS cbg  ON      cr.id = cbg.creator\_id  WHERE      cbg.creator\_id IS NULL  ORDER BY      "creator\_name"  ; |
| **Insert** | **First 5 Board Games** |
| INSERT INTO board\_games(name, release\_year, rating, category\_id, publisher\_id, players\_range\_id)  VALUES      ('Deep Blue', 2019, 5.67, 1, 15, 7),      ('Paris', 2016, 9.78, 7, 1, 5),      ('Catan: Starfarers', 2021, 9.87, 7, 13, 6),      ('Bleeding Kansas', 2020, 3.25, 3, 7, 4),      ('One Small Step', 2019, 5.75, 5, 9, 2);    INSERT INTO publishers(name, address\_id, website, phone)  VALUES      ('Agman Games', 5, 'www.agmangames.com', '+16546135542'),      ('Amethyst Games',  7, 'www.amethystgames.com', '+15558889992'),      ('BattleBooks', 13, 'www.battlebooks.com', '+12345678907'); | SELECT      bg.name,      bg.rating,      cat.name AS "category\_name"  FROM      board\_games AS bg  JOIN      players\_ranges AS pr  ON      pr.id = bg.players\_range\_id  JOIN      categories AS cat  ON      cat.id = bg.category\_id  WHERE      (bg.rating > 7 AND bg.name ILIKE '%a%')      OR      (bg.rating > 7.50 AND pr.min\_players >= 2 AND pr.max\_players <= 5)  ORDER BY      bg.name,      bg.rating DESC  LIMIT 5  ; |
| **Update** | **Creators with Emails** |
| UPDATE players\_ranges  SET  max\_players = max\_players + 1  WHERE  min\_players = 2  AND  max\_players = 2  ;  UPDATE board\_games  SET  "name" = "name" || ' V2'  WHERE  release\_year >= 2020  ; | SELECT      CONCAT(cr.first\_name, ' ', cr.last\_name) AS "full\_name",      cr.email,      MAX(bg.rating)  FROM      creators AS cr  JOIN      creators\_board\_games AS cbg  ON      cbg.creator\_id = cr.id  JOIN      board\_games AS bg  ON      cbg.board\_game\_id = bg.id  GROUP BY      "full\_name",      cr.email  HAVING      cr.email LIKE '%.com'  ORDER BY      "full\_name"  ; |
| **Delete** | **Creators by Rating** |
| DELETE FROM board\_games  WHERE      publisher\_id IN(          SELECT              id          FROM              publishers          WHERE              address\_id IN(                  SELECT                      id                  FROM                      addresses                  WHERE                      town LIKE 'L%'              )      );  DELETE FROM publishers  WHERE      address\_id IN(          SELECT              id          FROM              addresses          WHERE              town LIKE 'L%'      );  DELETE FROM      addresses  WHERE      town LIKE 'L%'; | SELECT      cr.last\_name,      CEIL(AVG(bg.rating)) AS "average\_rating",      p.name AS "publisher\_name"  FROM      creators AS cr  JOIN      creators\_board\_games AS cbg  ON      cbg.creator\_id = cr.id  JOIN      board\_games AS bg  ON      cbg.board\_game\_id = bg.id  JOIN      publishers AS p  ON      bg.publisher\_id = p.id  GROUP BY      cr.last\_name,      p.name  HAVING      p.name = 'Stonemaier Games'  ORDER BY      "average\_rating" DESC  ; |
| **Board Games by Release Year** | **Creator of Board Games** |
| SELECT      name,      rating  FROM      board\_games  ORDER BY      release\_year ASC,      name DESC  ; | CREATE OR REPLACE FUNCTION fn\_get\_volunteers\_count\_from\_department(searched\_volunteers\_department VARCHAR(30))  RETURNS INT AS  $$  DECLARE  **volunteers\_count INT;**  BEGIN  **volunteers\_count := (**  SELECT  COUNT(\*)  FROM  volunteers AS v  JOIN  volunteers\_departments AS vd  ON  vd.id = v.department\_id  WHERE  vd.department\_name = searched\_volunteers\_department);  **RETURN volunteers\_count;**  END  $$  LANGUAGE plpgsql; |
| **Board Games by Category** | **Search for Board Games** |
| SELECT      bg.id,      bg.name,      bg.release\_year,      cat.name  FROM      board\_games AS bg  JOIN      categories AS cat  ON      bg.category\_id= cat.id  WHERE      cat.name IN ('Strategy Games', 'Wargames')  ORDER BY      bg.release\_year DESC  ; | CREATE OR REPLACE PROCEDURE sp\_animals\_with\_owners\_or\_not(  **IN animal\_name VARCHAR(30),**  **OUT owner\_name VARCHAR(30)**  )  AS  $$  BEGIN  **owner\_name** := (  SELECT  o.name -> **OR COALESCE(o.name, 'For adoption ') + NO IF**  FROM  animals AS a  JOIN  owners AS o  ON  a.owner\_id = o.id  WHERE  a.name = animal\_name  );  IF **owner\_name** IS NULL THEN owner\_name := 'For adoption';  END IF;  END  $$  LANGUAGE plpgsql; |

# **PostgreSQL Exam Preparation II**

|  |  |
| --- | --- |
| **Database Design** | **Drivers and Cars** |
| CREATE TABLE addresses(  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  name VARCHAR(100) NOT NULL  );  CREATE TABLE categories(  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  name VARCHAR(10) NOT NULL  );  CREATE TABLE clients(  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  full\_name VARCHAR(50) NOT NULL,  phone\_number VARCHAR(20) NOT NULL  );  CREATE TABLE drivers(  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  first\_name VARCHAR(30) NOT NULL,  last\_name VARCHAR(30) NOT NULL,  **age INT NOT NULL,**  **rating NUMERIC(2, 1) DEFAULT 5.5,**  **CONSTRAINT ck\_age\_is\_positive**  **CHECK (age > 0)**  );  CREATE TABLE cars(  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  make VARCHAR(20) NOT NULL,  model VARCHAR(20),  year INT NOT NULL DEFAULT 1,  mileage INT DEFAULT 1,  condition CHAR(1) NOT NULL,  category\_id INT NOT NULL,  CONSTRAINT ck\_year\_positive\_number  CHECK (year > 0),  CONSTRAINT ck\_mileage\_positive\_number  CHECK (mileage > 0),  FOREIGN KEY (category\_id)  REFERENCES categories(id)  ON UPDATE CASCADE  ON DELETE CASCADE  );  CREATE TABLE courses(  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  from\_address\_id INT NOT NULL,  start TIMESTAMP NOT NULL,  bill NUMERIC(10, 2) DEFAULT 10,  car\_id INT NOT NULL,  client\_id INT NOT NULL,  FOREIGN KEY (from\_address\_id)  REFERENCES addresses (id)  ON UPDATE CASCADE  ON DELETE CASCADE,  FOREIGN KEY (car\_id)  REFERENCES cars (id)  ON UPDATE CASCADE  ON DELETE CASCADE,  FOREIGN KEY (client\_id)  REFERENCES clients (id)  ON UPDATE CASCADE  ON DELETE CASCADE,  CONSTRAINT ck\_bill\_positive\_number  CHECK (bill > 0)  );  CREATE TABLE cars\_drivers(  car\_id INT NOT NULL,  driver\_id INT NOT NULL,  FOREIGN KEY (car\_id)  REFERENCES cars (id)  ON UPDATE CASCADE  ON DELETE CASCADE,  FOREIGN KEY (driver\_id)  REFERENCES drivers (id)  ON UPDATE CASCADE  ON DELETE CASCADE  ); | SELECT      d.first\_name,      d.last\_name,      c.make,      c.model,      c.mileage  FROM      drivers AS d  JOIN      cars\_drivers AS cd  ON      d.id = cd.driver\_id  JOIN      cars AS c  ON      c.id = cd.car\_id  WHERE      mileage IS NOT NULL  ORDER BY      c.mileage DESC,      d.first\_name  ; |
| **Insert** | **Number of Courses for Each Car** |
| **INSERT INTO clients(full\_name, phone\_number)**  **SELECT**  **CONCAT(d.first\_name, ' ', d.last\_name),**  **CONCAT('(088) 9999', d.id \* 2)**  FROM  drivers AS d  WHERE  d.id BETWEEN 10 AND 20  ; | SELECT  ca.id AS "car\_id",  ca.make,  ca.mileage,  COUNT(co.id) AS "count\_of\_courses",  ROUND(AVG(co.bill), 2) AS "average\_bill"  FROM  cars AS ca  **LEFT JOIN**  courses AS co  ON  co.car\_id = ca.id  GROUP BY  ca.id  HAVING  COUNT(co.id) <> 2  ORDER BY  "count\_of\_courses" DESC,  "car\_id"  ; |
| **Update** | **Regular Clients** |
| UPDATE  cars  SET  condition = 'C'  WHERE  **(mileage > 800000 OR mileage IS NULL)**  **AND**  **year <= 2010**  **AND**  **make <> 'Mercedes-Benz'**  ; | SELECT      cl.full\_name,      COUNT(co.client\_id) AS "count\_of\_cars",      SUM(co.bill) AS "total\_sum"  FROM      clients AS cl  JOIN      courses AS co  ON      cl.id = co.client\_id  GROUP BY      cl.full\_name  HAVING      cl.full\_name LIKE '\_a%'      AND      COUNT(co.client\_id) > 1  ORDER BY      cl.full\_name  ; |
| **Delete** | **Full Information of Courses** |
| DELETE FROM clients  WHERE  **id NOT IN**  **(SELECT client\_id FROM courses)**  **AND**  **LENGTH(full\_name) > 3**  **;** | SELECT  ad.name,  **CASE**  **WHEN EXTRACT(HOUR FROM co.start) BETWEEN 6 AND 20 THEN 'Day'** ->**'Day'** (**between 6 and 20, inclusive**) or **'Night'**(**between 21 and 5, inclusive**)  **ELSE 'Night'**  **END AS "day\_time",**  co.bill,  cl.full\_name,  ca.make,  ca.model,  cat.name AS "category\_name"  FROM  addresses AS ad  JOIN  courses AS co  ON  ad.id = co.from\_address\_id  JOIN  clients AS cl  ON  co.client\_id = cl.id  JOIN  cars AS ca  ON  ca.id = co.car\_id  JOIN  categories AS cat  ON  ca.category\_id = cat.id  ORDER BY  co.id  ; |
| **Cars** | **Find all Courses by Client’s Phone Number** |
| SELECT      make,      model,      condition  FROM      cars  ORDER BY      id  ; | CREATE OR REPLACE FUNCTION fn\_courses\_by\_client(phone\_num VARCHAR(20))  RETURNS INT AS  $$      DECLARE number\_of\_courses INT;      BEGIN          number\_of\_courses := (              SELECT                  COUNT(co.client\_id)              FROM                  courses AS co              JOIN                  clients AS cl              ON                  co.client\_id = cl.id              WHERE                  cl.phone\_number = phone\_num          );          RETURN              number\_of\_courses;      END  $$  LANGUAGE plpgsql; |
|  | **Full Info for Address** |
|  | CREATE OR REPLACE PROCEDURE sp\_courses\_by\_address(  address\_name VARCHAR(100)  )  AS  $$  BEGIN - > Before beginning the insertion process into the table, ensure that you **TRUNCATE** **any existing results from the table** at the **start of the procedure**.  **TRUNCATE search\_results;**  **INSERT INTO search\_results(**  **address\_name,**  **full\_name,**  **level\_of\_bill,**  **make,**  **condition,**  **category\_name**  **)**  SELECT  ad.name AS "address\_name",  cl.full\_name,  CASE  WHEN co.bill BETWEEN 0 AND 20 THEN 'Low'  WHEN co.bill BETWEEN 21 AND 30 THEN 'Medium'  ELSE 'High'  END AS "level\_of\_bill",  ca.make,  ca.condition,  cat.name AS "category\_name"  FROM  addresses AS ad  JOIN  courses AS co  ON  ad.id = co.from\_address\_id  JOIN  clients AS cl  ON  co.client\_id = cl.id  JOIN  cars AS ca  ON  ca.id = co.car\_id  JOIN  categories AS cat  ON  ca.category\_id = cat.id  WHERE  ad.name = address\_name  ORDER BY  ca.make,  cl.full\_name;  END  $$  LANGUAGE plpgsql; |

# **PostgreSQL Exam Preparation III**

|  |  |
| --- | --- |
| **Database Design** | **Creators without Board Games** |
| CREATE TABLE categories(          id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,          name VARCHAR(50) NOT NULL      );        CREATE TABLE addresses(          id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,          street\_name VARCHAR(100) NOT NULL,          street\_number INT NOT NULL,          town VARCHAR(30) NOT NULL,          country VARCHAR(50) NOT NULL,          zip\_code INT NOT NULL,            CONSTRAINT ck\_street\_number\_is\_positive          CHECK (street\_number > 0),          CONSTRAINT ck\_zip\_code\_is\_positive          CHECK (zip\_code > 0)      );    CREATE TABLE publishers(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(30) NOT NULL,      address\_id INT NOT NULL,      website VARCHAR(40),      phone VARCHAR(20),      FOREIGN KEY (address\_id)      REFERENCES addresses(id)      ON UPDATE CASCADE      ON DELETE CASCADE  );  CREATE TABLE players\_ranges(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      min\_players INT NOT NULL,      max\_players INT NOT NULL,      CONSTRAINT ck\_min\_players\_is\_positive      CHECK (min\_players > 0),      CONSTRAINT ck\_max\_players\_is\_positive      CHECK (max\_players > 0)  );  CREATE TABLE creators(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      first\_name VARCHAR(30) NOT NULL,      last\_name VARCHAR(30) NOT NULL,      email VARCHAR(30) NOT NULL  );  CREATE TABLE board\_games(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(30) NOT NULL,      release\_year INT NOT NULL,      rating NUMERIC(2) NOT NULL,      category\_id INT NOT NULL,      publisher\_id INT NOT NULL,      players\_range\_id INT NOT NULL,      CONSTRAINT ck\_release\_year\_is\_positive      CHECK (release\_year > 0),      FOREIGN KEY (category\_id)      REFERENCES categories(id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (publisher\_id)      REFERENCES publishers(id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (players\_range\_id)      REFERENCES players\_ranges(id)      ON UPDATE CASCADE      ON DELETE CASCADE  );  CREATE TABLE creators\_board\_games(      creator\_id INT NOT NULL,      board\_game\_id INT NOT NULL,      FOREIGN KEY (creator\_id)      REFERENCES creators(id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (board\_game\_id)      REFERENCES board\_games(id)      ON UPDATE CASCADE      ON DELETE CASCADE  ); | SELECT      cr.id,      CONCAT(cr.first\_name, ' ', cr.last\_name) AS "creator\_name",      cr.email  FROM      creators AS cr  LEFT JOIN      creators\_board\_games AS cbg  ON      cr.id = cbg.creator\_id  WHERE      cbg.creator\_id IS NULL  ORDER BY      "creator\_name"  ; |
| **Insert** | **First 5 Board Games** |
| INSERT INTO board\_games(name, release\_year, rating, category\_id, publisher\_id, players\_range\_id)  VALUES      ('Deep Blue', 2019, 5.67, 1, 15, 7),      ('Paris', 2016, 9.78, 7, 1, 5),      ('Catan: Starfarers', 2021, 9.87, 7, 13, 6),      ('Bleeding Kansas', 2020, 3.25, 3, 7, 4),      ('One Small Step', 2019, 5.75, 5, 9, 2);    INSERT INTO publishers(name, address\_id, website, phone)  VALUES      ('Agman Games', 5, 'www.agmangames.com', '+16546135542'),      ('Amethyst Games',  7, 'www.amethystgames.com', '+15558889992'),      ('BattleBooks', 13, 'www.battlebooks.com', '+12345678907'); | Get the initial **5** board games with a **"rating"** higher than **7.00** that either contain the letter **'a'** in the board game **"name"** OR have a **"rating"** greater than **7.50**, and have a player count range between **2** and **5**.  SELECT  bg.name,  bg.rating,  cat.name AS "category\_name"  FROM  board\_games AS bg  JOIN  players\_ranges AS pr  ON  pr.id = bg.players\_range\_id  JOIN  categories AS cat  ON  cat.id = bg.category\_id  WHERE  **(bg.rating > 7 AND bg.name ILIKE '%a%')**  **OR**  **(bg.rating > 7.50 AND pr.min\_players >= 2 AND pr.max\_players <= 5)**  ORDER BY  bg.name,  bg.rating DESC  LIMIT 5  ; |
| **Update** | **Creators with Emails** |
| The next assignment is to update the **"players\_ranges"** table by increasing the **maximum player** count by **1** for board games that have a player range of **[2, 2]**. Furthermore, you need to change the **names** **of "board\_games"** that were published in **2020 or later** by adding **' V2'** to the end of their original names.  UPDATE players\_ranges  SET  max\_players = max\_players + 1  WHERE  **min\_players = 2**  **AND**  **max\_players = 2**  ;  UPDATE board\_games  SET  **"name" = "name" || ' V2' NOT WITH CONCAT**  WHERE  release\_year >= 2020  ; | SELECT      CONCAT(cr.first\_name, ' ', cr.last\_name) AS "full\_name",      cr.email,      MAX(bg.rating)  FROM      creators AS cr  JOIN      creators\_board\_games AS cbg  ON      cbg.creator\_id = cr.id  JOIN      board\_games AS bg  ON      cbg.board\_game\_id = bg.id  GROUP BY      "full\_name",      cr.email  HAVING      cr.email LIKE '%.com'  ORDER BY      "full\_name"  ; |
| **Delete** | **Creators by Rating** |
| In the **"addresses"** table, remove all countries that have a **"town"** starting with the letter **'L'**. Take into consideration that there might be **conflicts with foreign key constraints**.  **-- Delete records from referencing tables first**  DELETE FROM board\_games  WHERE  publisher\_id IN(  SELECT  id  FROM  publishers  WHERE  address\_id IN(  SELECT  id  FROM  addresses  WHERE  town LIKE 'L%'  )  );  DELETE FROM publishers  WHERE  address\_id IN(  SELECT  id  FROM  addresses  WHERE  town LIKE 'L%'  );  DELETE FROM  addresses  WHERE  town LIKE 'L%'; | SELECT      cr.last\_name,      CEIL(AVG(bg.rating)) AS "average\_rating",      p.name AS "publisher\_name"  FROM      creators AS cr  JOIN      creators\_board\_games AS cbg  ON      cbg.creator\_id = cr.id  JOIN      board\_games AS bg  ON      cbg.board\_game\_id = bg.id  JOIN      publishers AS p  ON      bg.publisher\_id = p.id  GROUP BY      cr.last\_name,      p.name  HAVING      p.name = 'Stonemaier Games'  ORDER BY      "average\_rating" DESC  ; |
| **Board Games by Release Year** | **Creator of Board Games** |
| SELECT      name,      rating  FROM      board\_games  ORDER BY      release\_year ASC,      name DESC  ; | CREATE OR REPLACE FUNCTION fn\_creator\_with\_board\_games(name VARCHAR(30))  RETURNS INT AS  $$      DECLARE total\_board\_games INT;      BEGIN          total\_board\_games := (              SELECT                  COUNT(cbg.board\_game\_id)              FROM                  creators AS cr              JOIN                  creators\_board\_games AS cbg              ON                  cr.id = cbg.creator\_id              WHERE                  cr.first\_name = name          );          RETURN total\_board\_games;      END  $$  LANGUAGE plpgsql; |
| **Board Games by Category** | **Search for Board Games** |
| SELECT      bg.id,      bg.name,      bg.release\_year,      cat.name  FROM      board\_games AS bg  JOIN      categories AS cat  ON      bg.category\_id= cat.id  WHERE      cat.name IN ('Strategy Games', 'Wargames')  ORDER BY      bg.release\_year DESC  ; | CREATE TABLE search\_results (      id SERIAL PRIMARY KEY,      name VARCHAR(50),      release\_year INT,      rating FLOAT,      category\_name VARCHAR(50),      publisher\_name VARCHAR(50),      min\_players VARCHAR(50),      max\_players VARCHAR(50)  );  CREATE OR REPLACE PROCEDURE usp\_search\_by\_category(  category VARCHAR(50)  )  AS  $$      BEGIN          INSERT INTO search\_results(              name,              release\_year,              rating,              category\_name,              publisher\_name,              min\_players,              max\_players          )          SELECT              bg.name,              bg.release\_year,              bg.rating,              cat.name AS "category\_name",              p.name AS "publisher\_name",              CONCAT(pr.min\_players, ' ', 'people') AS min\_players,              CONCAT(pr.max\_players, ' ', 'people') AS max\_players          FROM              board\_games AS bg          JOIN              categories AS cat          ON              bg.category\_id = cat.id          JOIN              publishers AS p          ON              bg.publisher\_id = p.id          JOIN              players\_ranges AS pr          ON              bg.players\_range\_id = pr.id          WHERE              cat.name = category          ORDER BY              "publisher\_name" ASC,              bg.release\_year DESC;      END  $$  LANGUAGE plpgsql; |

[**PostgreSQL Regular Exam - 15 June 2024**](https://alpha.judge.softuni.org/contests/postgresql-regular-exam-15-june-2024/5039)

|  |  |
| --- | --- |
| **Database Design** | **Lucky Accounts** |
| CREATE TABLE accounts(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      username VARCHAR(30) UNIQUE NOT NULL,      password VARCHAR(30) NOT NULL,      email VARCHAR(50) NOT NULL,      gender CHAR(1) NOT NULL,      age INT NOT NULL,      job\_title VARCHAR(40) NOT NULL,      ip VARCHAR(30) NOT NULL,      CONSTRAINT ck\_gender      CHECK (gender IN ('F', 'M'))  );  CREATE TABLE addresses(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      street VARCHAR(30) NOT NULL,      town VARCHAR(30) NOT NULL,      country VARCHAR(30) NOT NULL,      account\_id INT NOT NULL,        FOREIGN KEY (account\_id)      REFERENCES accounts(id)      ON DELETE CASCADE      ON UPDATE CASCADE  );  CREATE TABLE photos(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      description TEXT,      capture\_date TIMESTAMP NOT NULL,      views INT DEFAULT 0 NOT NULL,      CONSTRAINT ck\_views      CHECK (views >= 0)  );  CREATE TABLE comments(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      content VARCHAR(255) NOT NULL,      published\_on TIMESTAMP NOT NULL,      photo\_id INT NOT NULL,        FOREIGN KEY (photo\_id)      REFERENCES photos(id)      ON DELETE CASCADE      ON UPDATE CASCADE  );  CREATE TABLE accounts\_photos(      account\_id INT NOT NULL,      photo\_id INT NOT NULL,      PRIMARY KEY (account\_id, photo\_id),        FOREIGN KEY (account\_id)      REFERENCES accounts(id)      ON DELETE CASCADE ON UPDATE CASCADE,      FOREIGN KEY (photo\_id)      REFERENCES photos(id)      ON DELETE CASCADE ON UPDATE CASCADE  );  CREATE TABLE likes(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      photo\_id INT NOT NULL,      account\_id INT NOT NULL,        FOREIGN KEY (photo\_id)      REFERENCES photos(id)      ON DELETE CASCADE      ON UPDATE CASCADE,      FOREIGN KEY (account\_id)      REFERENCES accounts(id)      ON DELETE CASCADE      ON UPDATE CASCADE  ); | SELECT      CONCAT(a.id, ' ', a.username) AS "id\_username",      a.email  FROM      accounts AS a  JOIN      accounts\_photos AS ap  ON      a.id = ap.account\_id  JOIN      photos AS p  ON      p.id = ap.photo\_id  WHERE      ap.account\_id = ap.photo\_id  ORDER BY      ap.account\_id ASC  ; **Count Likes and Comments** |
| **Insert** | **Count Likes and Comments** |
| You need to insert data records into the '**addresses**' table, derived from the '**accounts**' table.  For **accounts** identified as **female gender**, insert corresponding data into the '**addresses**' table with the following values:   * street -set it tothe **username** of the **account**. * town - set it to the **password** of the **account**. * country - set it to the **ip** of the **account**. * account\_id - set it to the **age** of the **account**.   INSERT INTO addresses(**street, town, country, account\_id**)  SELECT  **username,**  **password,**  **ip,**  **age**  FROM  accounts  WHERE  gender = 'F'  ; | SELECT  p.id AS "photo\_id",  **COUNT(DISTINCT l.id) AS likes\_count,**  COUNT(DISTINCT c.id) AS comments\_count  FROM  photos AS p  **LEFT** JOIN  likes AS l  ON  p.id = l.photo\_id  **LEFT** JOIN  comments AS c  ON  c.photo\_id = p.id  GROUP BY  p.id  ORDER BY  likes\_count DESC,  comments\_count DESC,  p.id ASC  ; |
| **Update** | **Photos Captured on the Tenth Day** |
| UPDATE addresses  SET  **country =**  CASE  WHEN country LIKE 'B%' THEN 'Blocked'  WHEN country LIKE 'T%' THEN 'Test'  WHEN country LIKE 'P%' THEN 'In Progress'  **ELSE country**  END  ; | SELECT  CONCAT(**LEFT(description, 10),** '...') AS "summary",  TO\_CHAR(capture\_date, 'DD.MM HH24:MI') AS "date"  FROM  photos  WHERE  EXTRACT('DAY' FROM capture\_date) = 10  ORDER BY  capture\_date DESC  ; |
| **Delete** | **Get Accounts Photos Count** |
| DELETE FROM addresses  WHERE      id % 2 = 0      AND      street ILIKE '%r%'  ; | CREATE OR REPLACE FUNCTION udf\_accounts\_photos\_count(account\_username VARCHAR(30))  RETURNS INT AS  $$      DECLARE uploaded\_photos INT;      BEGIN          uploaded\_photos := (              SELECT                  COUNT(ap.account\_id)              FROM                  accounts\_photos AS ap              JOIN                  accounts AS a              ON                  a.id = ap.account\_id              WHERE                  a.username = account\_username          );          RETURN uploaded\_photos;      END  $$  LANGUAGE plpgsql; |
| **Accounts** | **Modify Accounts Job Title** |
| SELECT      username,      gender,      age  FROM      accounts  WHERE      age >= 18      AND      LENGTH(username) > 9  ORDER BY      age DESC,      username ASC  ; | CREATE OR REPLACE PROCEDURE udp\_modify\_account(  address\_street VARCHAR(30),  address\_town VARCHAR(30)  )  AS  $$  BEGIN  **UPDATE**  **accounts AS a**  SET  job\_title = CONCAT('(Remote) ', job\_title)  **FROM**  **addresses AS ad**  WHERE  address\_street = ad.street  AND  address\_town = ad.town  AND  a.id = ad.account\_id;  END  $$  LANGUAGE plpgsql; |
| **Top 3 Most Commented Photos** |  |
| SELECT      p.id AS photo\_id,      p.capture\_date,      p.description,      COUNT(c.photo\_id) AS comments\_count  FROM      photos AS p  JOIN      comments as c  ON      p.id = c.photo\_id  WHERE      p.description IS NOT NULL  GROUP BY      p.id,      p.capture\_date,      p.description  ORDER BY      comments\_count DESC,      photo\_id ASC  LIMIT 3  ; |  |

# **PostgreSQL Retake Exam - 7 August 2024**

|  |  |
| --- | --- |
| **Database Design** | **Casting** |
| CREATE TABLE countries(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(40) UNIQUE NOT NULL,      continent VARCHAR(40) NOT NULL,      currency VARCHAR(5)  );  CREATE TABLE categories(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(50) UNIQUE NOT NULL  );  CREATE TABLE actors(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      first\_name VARCHAR(50) NOT NULL,      last\_name VARCHAR(50) NOT NULL,      birthdate DATE NOT NULL,      height INT,      awards INT DEFAULT 0 NOT NULL,      country\_id INT NOT NULL,      CONSTRAINT ck\_awards      CHECK (awards >= 0),      FOREIGN KEY (country\_id)      REFERENCES countries (id)      ON UPDATE CASCADE      ON DELETE CASCADE  );  CREATE TABLE productions\_info(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      rating DECIMAL(4,2) NOT NULL,      duration INT NOT NULL,      budget DECIMAL(10,2),      release\_date DATE NOT NULL,      has\_subtitles BOOLEAN DEFAULT FALSE NOT NULL,      synopsis TEXT,        CONSTRAINT ck\_duration      CHECK (duration > 0)  );  CREATE TABLE productions(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      title VARCHAR(70) UNIQUE NOT NULL,      country\_id INT NOT NULL,      production\_info\_id INT UNIQUE NOT NULL,        FOREIGN KEY (country\_id)      REFERENCES countries (id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (production\_info\_id)      REFERENCES productions\_info (id)      ON UPDATE CASCADE      ON DELETE CASCADE  );  CREATE TABLE productions\_actors(      production\_id INT NOT NULL,      actor\_id INT NOT NULL,      PRIMARY KEY (production\_id, actor\_id),      FOREIGN KEY (production\_id)      REFERENCES productions (id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (actor\_id)      REFERENCES actors (id)      ON UPDATE CASCADE      ON DELETE CASCADE  );  CREATE TABLE categories\_productions(      category\_id INT NOT NULL,      production\_id INT NOT NULL,      PRIMARY KEY (category\_id, production\_id),      FOREIGN KEY (category\_id)      REFERENCES categories (id)      ON UPDATE CASCADE      ON DELETE CASCADE,      FOREIGN KEY (production\_id)      REFERENCES productions (id)      ON UPDATE CASCADE      ON DELETE CASCADE  ); | SELECT  CONCAT(first\_name, ' ', last\_name) AS full\_name,  **CONCAT(LOWER(LEFT(first\_name, 1)), LOWER(RIGHT(last\_name, 2)), LENGTH(last\_name), '@sm-cast.com') AS email,**  awards  FROM  actors AS a  **WHERE**  **id NOT IN(**  **SELECT actor\_id FROM productions\_actors - > actors not participating in any** productions  )  ORDER BY  awards DESC,  id ASC  ; |
| **Insert** | **Nominees** |
| **Insert data** into the **actors'** table with the **following values**:  • **first\_name** - set it to the **first name** of the actor but **reversed**.  • **last\_name** - set it to the **last name** of the actor but **reversed**.  • **birthdate** - set it to the **birthdate** of the **actor** but **2 days earlier**.  **• height - set it to the height of the actor plus 10. Note that there should not be NULL height values for the new records. See the provided examples.**  • **awards** - set them to the **country\_id**.  • **country\_id** - set it to the **id** of **Armenia** (Do **not** hard-code the id).  INSERT INTO actors(first\_name, last\_name, birthdate, height, awards, country\_id)  SELECT  REVERSE(first\_name),  REVERSE(last\_name),  birthdate - INTERVAL '2 days',  **COALESCE(height, 0) + 10,**  country\_id,  (SELECT id FROM countries WHERE name = 'Armenia')  FROM  actors  WHERE  id BETWEEN 10 AND 20  ; | SELECT  c.name AS country\_name,  COUNT(p.id) AS productions\_count,  **COALESCE(AVG(pi.budget), 0) AS avg\_budget - > Display zeros when there are no related data for average budgets.**  FROM  countries AS c  JOIN  productions AS p  ON  c.id = p.country\_id  JOIN  productions\_info AS pi  ON  pi.id = p.production\_info\_id  GROUP BY  c.name  HAVING  COUNT(p.id) >= 1  ORDER BY  productions\_count DESC,  country\_name ASC  ; |
| **Update** | **Classify by Rating** |
| **Increase** the **production's duration** for **productions** with **no synopsis and** the following **ids**  UPDATE productions\_info  SET  **duration = duration + CASE**  **WHEN id < 15 THEN 15**  **WHEN id >= 20 THEN 20**  **ELSE 0**  **END**  WHERE  synopsis IS NULL  ; | SELECT      p.title,      CASE          WHEN pi.rating <= 3.50 THEN 'poor'          WHEN pi.rating BETWEEN  3.50 AND 8.00 THEN 'good'          ELSE 'excellent'      END AS rating,      CASE          WHEN pi.has\_subtitles = 'true' THEN 'Bulgarian'          ELSE 'N/A'      END AS subtitles,      COUNT(pa.actor\_id) AS actors\_count  FROM      productions AS p  JOIN      productions\_info AS pi  ON      pi.id = p.production\_info\_id  JOIN      productions\_actors AS pa  ON      pa.production\_id = p.id  GROUP BY      p.title,      pi.rating,      pi.has\_subtitles  ORDER BY      rating ASC,      actors\_count DESC,      p.title ASC  ; |
| **Delete** | **Productions Count by Category** |
| DELETE FROM countries  WHERE id NOT IN (  **SELECT country\_id FROM actors**  **UNION**  **SELECT country\_id FROM productions**  ); | CREATE OR REPLACE FUNCTION udf\_category\_productions\_count(category\_name VARCHAR(50))  RETURNS VARCHAR AS  $$  DECLARE productions\_total\_number VARCHAR;  BEGIN  productions\_total\_number := (  SELECT  COUNT(cp.category\_id)  FROM  categories AS c  JOIN  categories\_productions AS cp  ON  c.id = cp.category\_id  WHERE  c.name = category\_name  );  **RETURN CONCAT('Found', ' ', productions\_total\_number, ' ', 'productions.');**  END  $$  LANGUAGE plpgsql; |
| **Countries** | **Awarded Production** |
| SELECT      id,      name,      continent,      currency  FROM      countries  WHERE      continent = 'South America'      AND      (currency LIKE 'P%'      OR      currency LIKE 'U%')  ORDER BY      currency DESC,      id ASC  ; | CREATE OR REPLACE PROCEDURE udp\_awarded\_production(  production\_title VARCHAR(70))  AS  $$  BEGIN  UPDATE  actors  SET  awards = awards + 1  WHERE  **id IN (**  **SELECT**  **pa.actor\_id**  **FROM**  **productions AS p**  **JOIN**  **productions\_actors AS pa**  **ON**  **p.id = pa.production\_id**  **WHERE**  **production\_title = p.title**  **);**  END  $$  LANGUAGE plpgsql; |
| **Productions by Release Year** |  |
| **Filter** **productions** that have been released in 2023 and 2024 year  SELECT  p.id,  p.title,  pi.duration,  ROUND(pi.budget, 1) AS budget,  TO\_CHAR(pi.release\_date, 'MM-YY') AS release\_date  FROM  productions AS p  RIGHT JOIN  productions\_info AS pi  ON  p.production\_info\_id = pi.id  WHERE  **pi.release\_date BETWEEN '2023-01-01' AND '2024-12-31'**  AND  pi.budget > 1500000.00  ORDER BY  budget ASC,  pi.duration DESC,  p.id ASC  LIMIT 3  ; |  |

**PostgreSQL Exam - 14 October 2023**

|  |  |
| --- | --- |
| **Database Design** | **Teams with Player Count and Fan Base** |
| CREATE TABLE towns(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(45) NOT NULL  );  CREATE TABLE stadiums(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(45) NOT NULL,      capacity INT NOT NULL CHECK (capacity > 0),      town\_id INT NOT NULL, FOREIGN KEY (town\_id) REFERENCES towns(id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE teams(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(45) NOT NULL,      established DATE NOT NULL,      fan\_base INT DEFAULT 0 NOT NULL CHECK (fan\_base >= 0),      stadium\_id INT NOT NULL, FOREIGN KEY (stadium\_id) REFERENCES stadiums(id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE coaches(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      first\_name VARCHAR(10) NOT NULL,      last\_name VARCHAR(20) NOT NULL,      salary NUMERIC(10, 2) DEFAULT 0 NOT NULL CHECK (salary >= 0),      coach\_level INT DEFAULT 0 NOT NULL CHECK (coach\_level >= 0)  );  CREATE TABLE skills\_data(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      dribbling INT DEFAULT 0 CHECK (dribbling >= 0),      pace INT DEFAULT 0 CHECK (pace >= 0),      passing INT DEFAULT 0 CHECK (passing >= 0),      shooting INT DEFAULT 0 CHECK (shooting >= 0),      speed INT DEFAULT 0 CHECK (speed >= 0),      strength INT DEFAULT 0 CHECK (strength >= 0)  );  CREATE TABLE players(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      first\_name VARCHAR(10) NOT NULL,      last\_name VARCHAR(20) NOT NULL,      age INT DEFAULT 0 NOT NULL CHECK (age >= 0),      position CHAR(1) NOT NULL,      salary NUMERIC(10, 2) DEFAULT 0 NOT NULL CHECK (salary >= 0),      hire\_date TIMESTAMP,      skills\_data\_id INT NOT NULL, FOREIGN KEY (skills\_data\_id) REFERENCES skills\_data(id) ON UPDATE CASCADE ON DELETE CASCADE,      team\_id INT, FOREIGN KEY (team\_id) REFERENCES teams(id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE players\_coaches(      player\_id INT, FOREIGN KEY (player\_id) REFERENCES players(id) ON UPDATE CASCADE ON DELETE CASCADE,      coach\_id INT, FOREIGN KEY (coach\_id) REFERENCES coaches(id) ON UPDATE CASCADE ON DELETE CASCADE  ); | SELECT      t.id,      t.name AS team\_name,      COUNT(p.team\_id) AS player\_count,      t.fan\_base  FROM      teams AS t  JOIN      players AS p  ON      t.id = p.team\_id  GROUP BY      t.id,      team\_name,      t.fan\_base  HAVING      t.fan\_base > 30000  ORDER BY       player\_count DESC,       t.fan\_base DESC  ; |
| **Insert** | **Coaches, Players Skills and Teams Overview** |
| INSERT INTO coaches(first\_name, last\_name, salary, coach\_level)  SELECT      first\_name,      last\_name,      salary \* 2,      LENGTH(first\_name) AS coach\_level  FROM      players  WHERE      hire\_date < '2013-12-13 07:18:46'  ; | SELECT  CONCAT(c.first\_name, ' ', c.last\_name) AS **coach\_full\_name**,  CONCAT(p.first\_name, ' ', p.last\_name) AS **player\_full\_name**,  t.name AS **team\_name**,  sd.passing,  sd.shooting,  sd.speed  FROM  coaches AS c  JOIN  players\_coaches AS pc  ON  c.id = pc.coach\_id  JOIN  players AS p  ON  p.id = pc.player\_id  JOIN  skills\_data AS sd  ON  p.skills\_data\_id = sd.id  JOIN  teams AS t  ON  p.team\_id = t.id  GROUP BY  **team\_name,**  sd.passing,  sd.shooting,  sd.speed,  **coach\_full\_name,**  **player\_full\_name**  ORDER BY  coach\_full\_name ASC,  player\_full\_name DESC  ; |
| **Update** | **Stadium Teams Information** |
| UPDATE coaches  SET  salary = salary \* coach\_level  **WHERE**  **id IN (**  **SELECT coach\_id FROM players\_coaches WHERE coach\_id IS NOT NULL**  **)**  AND  first\_name LIKE 'C%'  ; | CREATE OR REPLACE FUNCTION fn\_stadium\_team\_name(stadium\_name VARCHAR(30))  RETURNS TABLE (team\_name VARCHAR(50))  AS  $$      BEGIN          RETURN QUERY              SELECT                  t.name              FROM                  teams AS t              JOIN                  stadiums AS s              ON                  s.id = t.stadium\_id              WHERE                  s.name = stadium\_name              ORDER BY                  t.name;      END  $$  LANGUAGE plpgsql; |
| **Delete** | **Player Team Finder** |
| To ensure the accurate update of our database, your assignment is to **remove all instances of these promoted players** from the "**players**" table. More specifically, eliminate players **hired before** '2013-12-13 07:18:46'. Additionally, delete all associated records for these players from the "**players\_coaches**" table.  DELETE FROM players\_coaches  WHERE  player\_id IN (  SELECT  id  FROM players  WHERE  hire\_date < '2013-12-13 07:18:46'  )  ;  DELETE FROM players  WHERE  hire\_date < '2013-12-13 07:18:46'  ; | CREATE OR REPLACE PROCEDURE sp\_players\_team\_name(      IN player\_name VARCHAR(50),      OUT team\_name VARCHAR(45)  )  AS  $$      BEGIN      team\_name := (          SELECT              name          FROM              teams AS t          JOIN              players AS p          ON              t.id = p.team\_id          WHERE              p.first\_name || ' ' || p.last\_name = player\_name);          IF team\_name IS NULL THEN team\_name := 'The player currently has no team';          END IF;      END  $$  LANGUAGE plpgsql; |
| **Players** |  |
| SELECT      CONCAT(first\_name, ' ', last\_name) AS full\_name,      age,      hire\_date  FROM      players  WHERE      first\_name LIKE 'M%'  ORDER BY      age DESC,      full\_name ASC  ; |  |
| **Offensive Players without Team** |  |
| SELECT      p.id,      CONCAT(p.first\_name, ' ', p.last\_name) AS full\_name,      p.age,      p.position,      p.salary,      sd.pace,      sd.shooting  FROM      players AS p  JOIN      skills\_data AS sd  ON      p.skills\_data\_id = sd.id  WHERE      p.position = 'A'      AND      sd.pace + sd.shooting > 130      AND      p.team\_id IS NULL  ; |  |

[**PostgreSQL Retake Exam - 04 December 2023**](https://alpha.judge.softuni.org/contests/postgresql-retake-exam-04-december-2023/4439)

|  |  |
| --- | --- |
| **Database Design** | **Products** |
| CREATE TABLE countries(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(50) UNIQUE NOT NULL  );  CREATE TABLE customers(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      first\_name VARCHAR(25) NOT NULL,      last\_name VARCHAR(50) NOT NULL,      gender CHAR(1) CHECK (gender IN ('M', 'F')),      age INT NOT NULL CHECK (age > 0),      phone\_number char(10) NOT NULL,      country\_id INT NOT NULL, FOREIGN KEY (country\_id) REFERENCES countries (id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE products(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(25) NOT NULL,      description VARCHAR(250),      recipe TEXT,      price NUMERIC(10, 2) NOT NULL CHECK (price > 0)  );  CREATE TABLE feedbacks(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      description VARCHAR(255),      rate NUMERIC(4, 2) CHECK (rate BETWEEN 0 AND 10),      product\_id INT NOT NULL, FOREIGN KEY (product\_id) REFERENCES products (id) ON UPDATE CASCADE ON DELETE CASCADE,      customer\_id INT NOT NULL, FOREIGN KEY (customer\_id) REFERENCES customers (id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE distributors(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(25) UNIQUE NOT NULL,      address VARCHAR(30) NOT NULL,      summary VARCHAR(200) NOT NULL,      country\_id INT NOT NULL, FOREIGN KEY (country\_id) REFERENCES countries (id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE ingredients(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(30) NOT NULL,      description VARCHAR(200),      country\_id INT NOT NULL, FOREIGN KEY (country\_id) REFERENCES countries (id) ON UPDATE CASCADE ON DELETE CASCADE,      distributor\_id INT NOT NULL, FOREIGN KEY (distributor\_id) REFERENCES distributors (id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE products\_ingredients(      product\_id INT, FOREIGN KEY (product\_id) REFERENCES products (id) ON UPDATE CASCADE ON DELETE CASCADE,      ingredient\_id INT, FOREIGN KEY (ingredient\_id) REFERENCES ingredients (id) ON UPDATE CASCADE ON DELETE CASCADE  ); | SELECT      name,      recipe,      price  FROM      products  WHERE      price BETWEEN 10 and 20  ORDER BY      price DESC  ; |
| **Insert** | **Negative Feedback** |
| the requests **the creation of a new table named "gift\_recipients"** with the following columns:   * **"id"** - an automatically generated primary key; * **"name"**- concatenation of customers’ **"first\_name"** and **"last\_name"**; * **"country\_id"** - the country **"id"** of the individuals eligible to receive a gift; * **"gift\_sent"** - a Boolean flag with a default value of **"False"**   As part of the data insertion process, you should update the **"gift\_sent"** column to **"True"** for customers whose country **"id"** matches one of the following values: **7, 8, 14, 17, 26**.  **CREATE TABLE gift\_recipients**(  id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  name VARCHAR(50) NOT NULL,  country\_id INT NOT NULL,  gift\_sent BOOLEAN DEFAULT FALSE  );  INSERT INTO gift\_recipients(name, country\_id, gift\_sent)  SELECT  CONCAT(first\_name, ' ', last\_name) AS name,  country\_id,  CASE  WHEN country\_id IN (7, 8, 14, 17, 26) THEN TRUE  ELSE FALSE  END AS gift\_sent  FROM  customers  ; | SELECT      f.product\_id,      f.rate,      f.description,      f.customer\_id,      c.age,      c.gender  FROM      customers AS c  JOIN      feedbacks AS f  ON      c.id = f.customer\_id  WHERE      f.rate < 5.0      AND      c.gender = 'F'      AND      c.age > 30  ORDER BY      f.product\_id DESC  ; |
| **Update** | **High Average Price and Multiple Feedbacks** |
| UPDATE      products AS p  SET      price = price \* 1.10  FROM      feedbacks AS f  WHERE      f.rate > 8  ; | SELECT      p.name AS product\_name,      ROUND(AVG(p.price), 2) AS average\_price,      COUNT(f.id) AS total\_feedbacks  FROM      products AS p  JOIN      feedbacks AS f  ON      f.product\_id = p.id  GROUP BY      product\_name,      p.price  HAVING      p.price > 15      AND      COUNT(f.id) > 1  ORDER BY      total\_feedbacks ASC,      average\_price DESC  ; |
| **Delete** | **Specific Ingredients** |
| DELETE FROM  distributors  WHERE  name LIKE 'L%'  ;  DELETE FROM  ingredients  WHERE  distributor\_id IN(  SELECT  id  FROM  distributors  WHERE  name LIKE 'L%'  )  ;  DELETE FROM  **products\_ingredients**  WHERE  ingredient\_id IN(  SELECT  id  **FROM**  **distributors**  **WHERE**  **name LIKE 'L%'**  )  ; | SELECT      i.name AS ingredient\_name,      p.name AS product\_name,      d.name AS distributor\_name  FROM      distributors AS d  JOIN      ingredients AS i  ON      d.id = i.distributor\_id  JOIN      products\_ingredients AS pi  ON      i.id = pi.ingredient\_id  JOIN      products AS p  ON      p.id = pi.product\_id  WHERE      i.name ILIKE '%mustard%'      AND      d.country\_id = 16  ORDER BY      p.name ASC  ; |
| **Middle Range Distributors** | **Customer’s Country** |
| SELECT      d.name AS distributor\_name,      i.name AS ingredient\_name,      p.name AS product\_name,      AVG(f.rate) AS average\_rate  FROM      distributors AS d  JOIN      ingredients AS i  ON      d.id = i.distributor\_id  JOIN      products\_ingredients AS pi  ON      i.id = pi.ingredient\_id  JOIN      products AS p  ON      p.id = pi.product\_id  JOIN      feedbacks AS f  ON      p.id = f.product\_id  GROUP BY      distributor\_name,      ingredient\_name,      product\_name  HAVING      AVG(f.rate) BETWEEN 5 and 8  ORDER BY      d.name,      i.name,      p.name  ; | CREATE OR REPLACE PROCEDURE sp\_customer\_country\_name(  IN customer\_full\_name VARCHAR(50),  OUT country\_name VARCHAR(50))  AS  $$      BEGIN          country\_name := (              SELECT                  c.name              FROM                  countries AS c              JOIN                  customers AS cust              ON                  cust.country\_id = c.id              WHERE                  customer\_full\_name = CONCAT(cust.first\_name, ' ', cust.last\_name)          );      END  $$  LANGUAGE plpgsql; |
| **Customer Feedback** |  |
| CREATE OR REPLACE FUNCTION fn\_feedbacks\_for\_product(product\_name VARCHAR(25))  RETURNS TABLE(      customer\_id INT,      customer\_name VARCHAR(75),      feedback\_description VARCHAR(255),      feedback\_rate NUMERIC(4, 2)  )  AS  $$      BEGIN          RETURN QUERY (              SELECT                  c.id AS customer\_id,                  c.first\_name AS customer\_name,                  f.description AS feedback\_description,                  f.rate AS feedback\_rate              FROM                  customers AS c              JOIN                  feedbacks AS f              ON                  c.id = f.customer\_id              JOIN                  products AS p              ON                  p.id = f.product\_id              WHERE                  p.name = product\_name              ORDER BY                  c.id ASC          );      END  $$  LANGUAGE plpgsql; |  |

**PostgreSQL Regular Exam - 11 October 2025**

|  |  |
| --- | --- |
| **Database Design** | **Items On Sale** |
| CREATE TABLE brands(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(50) UNIQUE NOT NULL  );  CREATE TABLE classifications(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(30) UNIQUE NOT NULL  );  CREATE TABLE customers(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      first\_name VARCHAR(30) NOT NULL,      last\_name VARCHAR(30) NOT NULL,      address VARCHAR(150) NOT NULL,      phone VARCHAR(30) UNIQUE NOT NULL,      loyalty\_card BOOLEAN DEFAULT FALSE NOT NULL  );  CREATE TABLE items(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      name VARCHAR(50) NOT NULL,      quantity INT NOT NULL CHECK (quantity >= 0),      price DECIMAL(12, 2) NOT NULL CHECK (price > 0.00),      description text,      brand\_id INT NOT NULL, FOREIGN KEY (brand\_id) REFERENCES brands(id) ON UPDATE CASCADE ON DELETE CASCADE,      classification\_id INT NOT NULL, FOREIGN KEY (classification\_id) REFERENCES classifications(id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE orders(      id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,      created\_at TIMESTAMP DEFAULT NOW() NOT NULL,      customer\_id INT NOT NULL, FOREIGN KEY (customer\_id) REFERENCES customers(id) ON UPDATE CASCADE ON DELETE CASCADE  );  CREATE TABLE reviews(      customer\_id INT NOT NULL, FOREIGN KEY (customer\_id) REFERENCES customers(id) ON UPDATE CASCADE ON DELETE CASCADE,      item\_id INT NOT NULL, FOREIGN KEY (item\_id) REFERENCES items(id) ON UPDATE CASCADE ON DELETE CASCADE,      PRIMARY KEY (customer\_id, item\_id),      created\_at TIMESTAMP DEFAULT NOW() NOT NULL,      rating DECIMAL(3, 1) DEFAULT 0.0 NOT NULL CHECK (rating <= 10.0)  );  CREATE TABLE orders\_items(      order\_id INT NOT NULL, FOREIGN KEY (order\_id) REFERENCES orders(id) ON UPDATE CASCADE ON DELETE CASCADE,      item\_id INT NOT NULL, FOREIGN KEY (item\_id) REFERENCES items(id) ON UPDATE CASCADE ON DELETE CASCADE,      PRIMARY KEY (order\_id, item\_id),      quantity INT NOT NULL CHECK (quantity >= 0)  ); | SELECT      i.name,      CONCAT(UPPER(b.name), '/', LOWER(c.name)) AS promotion,      CONCAT('On sale: ', i.description) AS description,      i.quantity  FROM      items AS i  LEFT JOIN      orders\_items AS oi  ON      i.id = oi.item\_id  JOIN      brands AS b  ON      i.brand\_id = b.id  JOIN      classifications AS c  ON      i.classification\_id = c.id  WHERE      oi.item\_id IS NULL  ORDER BY      i.quantity DESC,      i.name ASC  ; |
| **Insert** | **Customers without Reviews** |
| INSERT INTO items(name, quantity, price, description, brand\_id, classification\_id)  SELECT      'Item' || r.created\_at AS name,      r.customer\_id AS quantity,      r.rating \* 5 AS price,      NULL AS description,      r.item\_id AS brand\_id,      (          SELECT              MIN(item\_id)          FROM reviews      ) AS classification\_id  FROM      reviews AS r  ORDER BY      r.item\_id ASC  LIMIT 10  ; | SELECT      c.id AS customer\_id,      CONCAT(c.first\_name, ' ', c.last\_name) AS full\_name,      COUNT(o.customer\_id) AS total\_orders,      CASE          WHEN c.loyalty\_card = TRUE THEN 'Loyal Customer'          ELSE 'Regular Customer'      END AS loyalty\_status  FROM      customers AS c  JOIN      orders AS o  ON      c.id = o.customer\_id  WHERE      c.id NOT IN(          SELECT              customer\_id          FROM              reviews      )  GROUP BY      c.id  HAVING      COUNT(o.customer\_id) >= 1  ORDER BY      total\_orders DESC,      customer\_id ASC  ; |
| **Update** | **Top-rated Items** |
| UPDATE      reviews  SET      rating =      CASE          WHEN item\_id = customer\_id THEN 10.0          WHEN customer\_id > item\_id THEN 5.5          ELSE rating      END  ; | SELECT      i.name AS item\_name,      ROUND(AVG(r.rating), 2) AS average\_rating,      COUNT(r.item\_id) AS total\_reviews,      b.name AS brand\_name,      c.name AS classification\_name  FROM      items AS i  JOIN      reviews AS r  ON      i.id = r.item\_id  JOIN      brands AS b  ON      i.brand\_id = b.id  JOIN      classifications AS c  ON      i.classification\_id = c.id  GROUP BY      item\_name,      brand\_name,      classification\_name  HAVING      COUNT(r.item\_id) >= 3  ORDER BY      average\_rating DESC,      item\_name ASC  LIMIT 3  ; |
| **Delete** | **Items by Classification** |
| DELETE FROM customers  WHERE      id NOT IN(          SELECT              customer\_id          FROM              orders      )  ; | CREATE OR REPLACE FUNCTION udf\_classification\_items\_count(classification\_name VARCHAR(30))  RETURNS VARCHAR AS  $$      DECLARE          items\_total\_number INT;          result VARCHAR(50);      BEGIN          SELECT              COUNT(i.id) INTO items\_total\_number          FROM              items AS i          JOIN              classifications AS c          ON              i.classification\_id = c.id          WHERE              c.name = classification\_name;          IF items\_total\_number > 0 THEN result := 'Found ' || items\_total\_number || ' items.';          ELSE result := 'No items found.';          END IF;          RETURN result;      END  $$  LANGUAGE plpgsql; |
| **Customers** | **Update Loyalty Status** |
| SELECT      id,      last\_name,      loyalty\_card  FROM      customers  WHERE      last\_name ILIKE '%m%'      AND      loyalty\_card = TRUE  ORDER BY      last\_name DESC,      id ASC  ; | CREATE OR REPLACE PROCEDURE udp\_update\_loyalty\_status(min\_orders INT)  AS  $$      BEGIN          UPDATE customers AS c          SET              loyalty\_card = TRUE          WHERE              c.id IN(                  SELECT                      o.customer\_id                  FROM                      orders AS o                  GROUP BY                      o.customer\_id                  HAVING                      COUNT(o.id) >= min\_orders              );      END  $$  LANGUAGE plpgsql; |
| **Orders By Date** |  |
| SELECT      id,      TO\_CHAR(created\_at, 'DD-MM-YYYY') AS created\_at,      customer\_id  FROM      orders  WHERE      created\_at > '01-01-2025'      AND      customer\_id BETWEEN 15 AND 30  ORDER BY      created\_at ASC,      customer\_id DESC,      id ASC  LIMIT 5  ; |  |