Set Returning Functions & JSON and Array Functions

Data Transformation and Aggregation: Exercises

May 19, 2025

Contents

Global Dataset for PostgreSQL

The following SQL code creates and populates the necessary tables for the exercises. Execute this script in your PostgreSQL environment before attempting the exercises.

```sql
-- Dataset for Exercises
-- Drop tables if they exist to ensure a clean setup
DROP TABLE IF EXISTS EventCalendar CASCADE;
DROP TABLE IF EXISTS ProjectAssignments CASCADE;
DROP TABLE IF EXISTS Employees CASCADE;
DROP TABLE IF EXISTS SystemLogs CASCADE;
DROP TABLE IF EXISTS ServiceSubscriptions CASCADE;

-- Create Tables
CREATE TABLE Employees (
    employeeId INT PRIMARY KEY,
    employeeName VARCHAR(100) NOT NULL,
    hireDate DATE NOT NULL,
    department VARCHAR(50),
    skills TEXT[], -- For unnest, array_append, array_length
    performanceReviews JSONB -- For JSON functions, e.g., '[{"year": 2022, "rating": 4, "notes": "Good progress"}, ...]'
);

CREATE TABLE ProjectAssignments (
    assignmentId SERIAL PRIMARY KEY,
    projectId INT NOT NULL,
    projectName VARCHAR(100),
    employeeId INT REFERENCES Employees(employeeId),
    role VARCHAR(50),
    assignmentHours INT,
    assignmentData JSONB -- e.g., '{ "milestones": [{"name": "Phase 1", "status": "completed"}, {"name": "Phase 2", "status": "pending"}], "budget": 5000.00 }'
);

CREATE TABLE SystemLogs (
    logId SERIAL PRIMARY KEY,
    logTimestamp TIMESTAMP NOT NULL,
    serviceName VARCHAR(50),
    logLevel VARCHAR(10), -- e.g., INFO, ERROR, WARN
    logDetails JSONB -- e.g., '{ "clientIp": "192.168.1.10", "requestPath": "/api/data", "statusCode": 200, "userContext": {"userId": 101, "sessionId": "xyz"} }'
);

CREATE TABLE ServiceSubscriptions (
    subscriptionId SERIAL PRIMARY KEY,
    userId INT,
    customerName VARCHAR(100),
    serviceType VARCHAR(50),
    startDate DATE NOT NULL,
    endDate DATE, -- Can be NULL for ongoing subscriptions
    monthlyFee DECIMAL(10,2),
    features JSONB -- e.g., '{ "storageLimitGB": 50, "prioritySupport": true, "addons": ["backup", "monitoring"] }'
);

CREATE TABLE EventCalendar (
    eventId SERIAL PRIMARY KEY,
    eventName VARCHAR(100),
    eventCategory VARCHAR(50),
    eventStartDate DATE,
    eventEndDate DATE,
    expectedAttendees INT,
    bookedResources TEXT[] -- e.g., '{"Room A", "Projector X"}'
);


-- Populate Tables
INSERT INTO Employees (employeeId, employeeName, hireDate, department, skills, performanceReviews) VALUES
(1, 'Alice Wonderland', '2020-01-15', 'Engineering', ARRAY['SQL', 'Python', 'Data Analysis'], '[{"year": 2022, "rating": 4, "notes": "Exceeded expectations in Q3"},
```

```
    {"year": 2023, "rating": 5, "notes": "Top performer"}]'),
(2, 'Bob The Builder', '2019-03-01', 'Engineering', ARRAY['Java', 'Spring', '
    Microservices'], '[{"year": 2022, "rating": 3, "notes": "Met expectations"}, {"year
    ": 2023, "rating": 4, "notes": "Improved significantly"}]'),
(3, 'Charlie Brown', '2021-07-30', 'Sales', ARRAY['Communication', 'Negotiation', 'CRM'
    ], '[{"year": 2023, "rating": 4, "notes": "Good sales figures"}]'),
(4, 'Diana Prince', '2018-05-10', 'HR', ARRAY['Recruitment', 'Employee Relations', '
    Legal Knowledge'], NULL),
(5, 'Edward Scissorhands', '2022-11-01', 'Engineering', ARRAY['Python', 'Machine
    Learning'], '[{"year": 2023, "rating": 5, "notes": "Innovative solutions"}]'),
(6, 'Fiona Gallagher', '2023-02-15', 'Sales', ARRAY['CRM', 'Presentations'], '[]'), --
    Empty JSON array
(7, 'George Jetson', '2017-09-01', 'Management', ARRAY['Leadership', 'Strategy'], '[{"
    year": 2022, "rating": 5, "notes": "Excellent leadership"}, {"year": 2023, "rating":
     4, "notes": "Managed team well through transition"}]');

INSERT INTO ProjectAssignments (projectId, projectName, employeeId, role,
    assignmentHours, assignmentData) VALUES
(101, 'Data Warehouse Migration', 1, 'Lead Data Engineer', 120, '{ "milestones": [{"name
    ": "Schema Design", "status": "completed"}, {"name": "ETL Development", "status": "
    in-progress"}], "budget": 20000.00, "critical": true }'),
(101, 'Data Warehouse Migration', 2, 'Backend Developer', 80, '{ "milestones": [{"name":
     "API Integration", "status": "pending"}], "budget": 15000.00, "critical": true }'),
(102, 'Mobile App Development', 2, 'Lead Mobile Developer', 150, '{ "milestones": [{"
    name": "UI/UX Design", "status": "completed"}, {"name": "Frontend Dev", "status": "
    completed"}, {"name": "Backend Dev", "status": "in-progress"}], "budget": 50000.00,
    "critical": false }'),
(103, 'Sales Platform Upgrade', 3, 'Sales Lead', 100, '{ "milestones": [{"name": "
    Requirement Gathering", "status": "completed"}], "budget": 10000.00, "critical":
    false }'),
(101, 'Data Warehouse Migration', 5, 'ML Engineer', 60, '{ "milestones": [{"name": "
    Model Training", "status": "in-progress"}], "budget": 12000.00, "critical": true }')
    ,
(104, 'HR System Implementation', 4, 'HR Specialist', 90, NULL); -- No JSON data

INSERT INTO SystemLogs (logTimestamp, serviceName, logLevel, logDetails) VALUES
('2023-10-01 10:00:00', 'AuthService', 'INFO', '{ "message": "User login successful", "
    userId": 1, "clientIp": "192.168.0.10" }'),
('2023-10-01 10:05:00', 'OrderService', 'ERROR', '{ "message": "Payment processing
    failed", "orderId": 123, "errorCode": "P5001", "details": {"reason": "Insufficient
    funds"} }'),
('2023-10-01 10:10:00', 'ProductService', 'WARN', '{ "message": "Low stock warning", "
    productId": "XYZ123", "currentStock": 5 }'),
('2023-10-02 11:00:00', 'AuthService', 'INFO', '{ "message": "User login successful", "
    userId": 2, "clientIp": "192.168.0.15" }'),
('2023-10-02 11:15:00', 'OrderService', 'INFO', '{ "message": "Order placed", "orderId":
     124, "items": ["itemA", "itemB"], "totalAmount": 75.50 }'),
(NOW() - INTERVAL '1 day', 'ReportingService', 'DEBUG', '{ "queryId": "q123", "
    executionTimeMs": 1500, "parameters": {"startDate": "2023-01-01", "endDate":
    "2023-01-31"} }');

INSERT INTO ServiceSubscriptions (userId, customerName, serviceType, startDate, endDate,
     monthlyFee, features) VALUES
(101, 'Customer Alpha', 'Premium Cloud Storage', '2023-01-01', '2023-12-31', 20.00, '{ "
    storageLimitGB": 100, "prioritySupport": true, "addons": ["versioning", "encryption
    "] }'),
(102, 'Customer Beta', 'Basic VPN Service', '2023-03-15', NULL, 5.00, '{ "dataCapMB":
    5000, "prioritySupport": false, "serverLocations": ["US", "EU"] }'),
(103, 'Customer Gamma', 'Standard Streaming', '2023-05-01', '2024-04-30', 10.00, '{ "
    resolution": "1080p", "profiles": 4, "offlineDownload": true }'),
(101, 'Customer Alpha', 'Analytics Suite', '2023-06-01', NULL, 50.00, '{ "users": 5, "
    dashboards": 10, "dataSources": ["db1", "s3"] }'),
(104, 'Customer Delta', 'Premium Cloud Storage', '2022-11-01', '2023-11-01', 18.00, '{ "
    storageLimitGB": 100, "prioritySupport": true, "addons": ["backup"] }');

INSERT INTO EventCalendar (eventName, eventCategory, eventStartDate, eventEndDate,
    expectedAttendees, bookedResources) VALUES
('Tech Conference 2024', 'Conference', '2024-03-10', '2024-03-12', 500, ARRAY['Main Hall
    ', 'Audio System', 'Projectors']),
('Product Launch Q1', 'Marketing', '2024-02-15', '2024-02-15', 100, ARRAY['Meeting Room
    Alpha', 'Catering Service']),
('Team Building Workshop', 'HR', '2024-04-05', '2024-04-05', 30, ARRAY['Outdoor Space',
    'Activity Kits']),
```

```
('Quarterly Review Meeting', 'Management', '2024-01-20', '2024-01-20', 15, ARRAY['Board Room']),
('Holiday Party 2023', 'Social', '2023-12-15', '2023-12-15', 150, NULL); -- No resources booked
```

Listing 1: Global Dataset for Exercises

1 Set Returning Functions (generate_series, unnest)

1.1 Meaning, values, relations (with previous concepts), advantages

SRF.1.1 Monthly Active Subscription Report

Problem: For each `serviceType` in `ServiceSubscriptions`, generate a list of all months (as the first day of the month) between January 2023 and December 2023. For each of these generated months, count how many subscriptions of that `serviceType` were active during that month. A subscription is active if the generated month falls within its `startDate` and `endDate` (or if `endDate` is NULL, it's considered active indefinitely past its `startDate`). Display the `serviceType`, the generated month, and the count of active subscriptions.

Concept Focus: Using `generate_series` to create a date series for reporting. Advantage: easily create time dimensions for temporal analysis. Relation: `JOIN` with existing tables, `GROUP BY` for aggregation, date functions for comparison.

SRF.1.2 Employee Skills Breakdown

Problem: List each employee and each of their individual skills on a separate row. Also, show the employee's department. Exclude employees who have no listed skills.

Concept Focus: Using `unnest` to normalize array data into rows. Advantage: enables relational operations (joins, filters) on individual array elements. Relation: `JOIN` (implicitly with the same table via unnesting), `SELECT` specific columns.

1.2 Disadvantages of all its technical concepts

SRF.2.1 Potential Performance Issue with `generate_series`

Problem: Imagine you need to generate a record for every second in a full year for sensor data simulation. Write a query that *would* generate this series (but limit it to 10 for safety in this exercise). Explain why running this for a full year without careful planning (e.g., for direct insertion or large-scale processing) could be a disadvantage.

Concept Focus: Disadvantage of `generate_series` - potential for creating excessively large datasets.

SRF.2.2 Row Explosion with `unnest`

Problem: Consider the `Employees` table. If an employee had 100 skills, and you `unnest` their skills and then `JOIN` this with a `ProjectAssignments` table where they are on 10 projects, how many rows could this potentially generate for this single employee in the combined result before any aggregation? Explain the disadvantage.

Concept Focus: Disadvantage of `unnest` - row explosion leading to performance degradation if not handled correctly.

1.3 Cases where people lose advantages due to inefficient solutions

SRF.3.1 Inefficiently Generating Date Sequences

Problem: A common task is to get a list of dates for a specific month. Show an inefficient way to generate all days in January 2024 using a recursive CTE (a more complex approach than `generate_series` for this simple task). Then, provide the efficient solution using `generate_series`. Highlight why the `generate_series` approach is advantageous.

Concept Focus: Contrasting `generate_series` with a more verbose/complex recursive CTE for simple series generation.

SRF.3.2 Inefficiently Handling Array Elements

Problem: An employee `Alice Wonderland` (employeeId 1) has skills stored in an array. You want to find if she has the skill 'Python'. Show an inefficient way to check this (e.g., by converting the array to a string and using `LIKE`). Then, show the efficient way using array operators or `unnest` with a `WHERE` clause. Highlight the advantages of the efficient SQL-native array approach.

Concept Focus: Contrasting inefficient string manipulation for array searching with SQL-native array operations or `unnest`.

1.4 Hardcore problem combining previous concepts

SRF.4.1 Comprehensive Project Health and Skill Utilization Report

Problem: Generate a report for the first 6 months of 2023 (January to June). The report should show:

1. `reportMonth` (first day of each month).
2. `projectName`.
3. `totalAssignedHours` for that project in that month (sum of `assignmentHours` for employees whose assignment *could* be active in that month – assume `assignmentHours` are per month if the project is active). For simplicity, consider a project assignment active if its `employeeId` is on a project.
4. `criticalProjectFlag` (boolean, true if `assignmentData ->> 'critical'` is true for *any* assignment on that project).
5. `listOfDistinctSkillsUtilized`: A comma-separated string of all distinct skills possessed by employees assigned to that project. Only include skills of employees who were hired *before or during* the `reportMonth`.
6. `averageYearsOfService`: Average years of service (from `hireDate` to `reportMonth`) of employees assigned to that project and hired before or during the `reportMonth`. Round to 2 decimal places.

Filter the report to include only projects that had at least one employee assigned who possesses the skill 'Python'. Order the results by `reportMonth` and then by `projectName`.

Previous concepts to use: `generate_series`, `unnest`, CTEs (basic or nested), Joins (INNER, LEFT), Aggregations (SUM, AVG, STRING_AGG), Date functions (DATE_TRUNC, EXTRACT or age calculation), String functions (CONCAT), Subqueries (possibly in WHERE or SELECT), CASE statements, `COALESCE`.

2 JSON and Array Functions (jsonb_extract_path_text, jsonb_array_elements, jsonb_build_object, array_append, array_length)

Note: The dataset uses JSONB for JSON columns, so jsonb_ prefixed functions are generally applicable.

2.1 Meaning, values, relations (with previous concepts), advantages

JAF.1.1 Extracting Specific Log Information

Problem: From the SystemLogs table, extract the clientIp and userId for all 'INFO' level logs from 'AuthService'. The userId is nested within userContext in logDetails. If userId is not present, display NULL.

Concept Focus: jsonb_extract_path_text (or ->> operator) for pulling specific values from JSON. Advantage: Direct access to nested JSON data without complex parsing. Relation: WHERE clause for filtering.

JAF.1.2 Expanding Performance Review Details

Problem: For each employee who has performance reviews, list each review on a separate row, showing the employee's name, the review year, and rating.

Concept Focus: jsonb_array_elements to transform a JSON array within a field into multiple rows. Advantage: Normalizes JSON array data for relational processing. Relation: JOIN (implicit with jsonb_array_elements).

JAF.1.3 Constructing a Simplified Project Overview JSON

Problem: For each project in ProjectAssignments, create a new JSONB object containing the projectName and a list of employeeIds assigned to it.

Concept Focus: jsonb_build_object to create JSON objects dynamically, ARRAY_AGG or jsonb_agg (previous concept) to gather employee IDs. Advantage: Creating structured JSON output from relational data. Relation: GROUP BY for aggregation.

JAF.1.4 Updating Event Resources and Checking Count

Problem: For the 'Tech Conference 2024' event, add 'WiFi Access Point' to its bookedResources. Then, display the event name and the new total number of booked resources for this event. (Simulate the update in a SELECT statement or describe the UPDATE and then SELECT).

Concept Focus: array_append to add an element to an array, array_length to get the size of an array. Advantage: Simple and efficient array manipulation.

2.2 Disadvantages of all its technical concepts

JAF.2.1 Performance of Complex JSON Queries vs. Normalized Data

Problem: Suppose the logDetails in SystemLogs often contains deeply nested structures (e.g., 5-10 levels deep). Explain the disadvantage of frequently querying very

specific, deeply nested values from such JSONB columns compared to having those specific values in their own indexed relational columns.

Concept Focus: Disadvantage of JSON functions - potential performance overhead for deeply nested or complex queries if not properly indexed (GIN/GiST) or if compared to highly optimized relational access.

JAF.2.2 Array Overuse and Normalization

Problem: The `Employees` table has a `skills` array. If these skills also had attributes (e.g., 'skillLevel', 'yearsOfExperienceWithSkill'), explain the disadvantage of trying to store this richer skill information within the single `TEXT[]` array (e.g., by encoding it like `'Python:Expert:5yrs'`) versus creating a separate `EmployeeSkills` table.

Concept Focus: Disadvantage of arrays - can lead to denormalization and make querying/updating complex attributes of array elements difficult.

2.3 Cases where people lose advantages due to inefficient solutions

JAF.3.1 Inefficiently Querying JSON Data with String Matching

Problem: From `SystemLogs`, find all logs where the `logDetails` JSONB contains a key `orderId` with a value of `123`. Show an inefficient way to do this by casting `logDetails` to text and using `LIKE`. Then, provide the efficient solution using JSONB operators. Highlight the advantages of the JSONB-specific approach.

Concept Focus: Contrasting inefficient string matching on stringified JSON with efficient JSONB operators.

JAF.3.2 Storing Multiple Flags as a Comma-Separated String Instead of JSON/Array

Problem: A common inefficient practice is storing multiple boolean flags or categorical tags as a single comma-separated string in a `VARCHAR` column (e.g., `flags VARCHAR(255)` with value `'active,premium,verified'`). Suppose we have such a column named `userTags` in a hypothetical `Users` table. Show how one might inefficiently query for users who have the 'premium' tag using `LIKE`. Then, describe how using a `TEXT[]` (array) or a `JSONB` (e.g., {`"active": true, "premium": true, "verified": true`}) would be more advantageous for querying and management.

Concept Focus: Contrasting comma-separated strings with arrays or JSONB for storing multiple discrete values/flags.

2.4 Hardcore problem combining previous concepts

JAF.4.1 Advanced Customer Subscription Feature Analysis and Aggregation

Problem: Generate a JSONB report for each `customerName` from `ServiceSubscriptions`. The report should:

1. Be a single JSONB object per customer.
2. Top-level keys: `customerName`, `totalMonthlyFee`, `activeServicesCount`, `serviceDetails` (a JSON array).

3. `totalMonthlyFee`: Sum of `monthlyFee` for all *currently active* subscriptions (`endDate` IS NULL or `endDate > CURRENT_DATE`).
4. `activeServicesCount`: Count of *currently active* subscriptions.
5. `serviceDetails`: A JSON array, where each element is a JSONB object representing one of their subscriptions (both active and past). Each object should contain:
   - `serviceType`
   - `status`: 'Active' or 'Expired' (based on `endDate` vs `CURRENT_DATE`).
   - `durationMonths`: Number of full months the subscription lasted or has been active. If `endDate` is NULL, calculate up to `CURRENT_DATE`.
   - `featureList`: A JSON array of strings derived from the `addons` array within the `features` JSONB column. If `addons` doesn't exist or is not an array, this should be an empty JSON array `[]`.
   - `hasPrioritySupport`: Boolean, extracted from `features ->> 'prioritySupport'`. Default to `false` if not present.

Filter results to include only customers who have at least one subscription with 'prioritySupport' set to true in their `features`. Order customers by their `totalMonthlyFee` (for active subscriptions) in descending order.

Previous concepts to use: `jsonb_build_object`, `jsonb_array_elements_text` (for processing `features -> 'addons'`), `jsonb_extract_path_text` (or `->`, `->>`), `jsonb_agg`, `COALESCE`, `CASE` statements, `SUM`, `COUNT`, Date functions (`AGE`, `EXTRACT`, `CURRENT_DATE`), CTEs, Joins (if needed, though most can be done with aggregations and subqueries on `ServiceSubscriptions`), `FILTER` clause in aggregation.