Aggregate Functions in SQL: Solutions

Generated Examples

May 8, 2025

Contents

Dataset for PostgreSQL

The following SQL code creates and populates the necessary tables for the exercises. (Same as in exercises document).

```sql
-- Dataset for Aggregate Functions Exercises (PostgreSQL)

-- Drop tables if they exist to ensure a clean setup
DROP TABLE IF EXISTS employee_tasks CASCADE;
DROP TABLE IF EXISTS projects CASCADE;
DROP TABLE IF EXISTS employees CASCADE;
DROP TABLE IF EXISTS departments CASCADE;

-- departments table
CREATE TABLE departments (
    department_id SERIAL PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL UNIQUE,
    location VARCHAR(100)
);

-- employees table
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    salary DECIMAL(10, 2) CHECK (salary > 0),
    hire_date DATE,
    department_id INT REFERENCES departments(department_id),
    manager_id INT REFERENCES employees(employee_id),
    bonus_percentage NUMERIC(4, 2) CHECK (bonus_percentage >= 0 AND bonus_percentage <= 100.00),
    performance_rating INT CHECK (performance_rating >= 1 AND performance_rating <= 5)
);

-- projects table
CREATE TABLE projects (
    project_id SERIAL PRIMARY KEY,
    project_name VARCHAR(150) NOT NULL UNIQUE,
    start_date DATE,
    end_date DATE,
    budget DECIMAL(12, 2) CHECK (budget > 0),
    lead_employee_id INT REFERENCES employees(employee_id)
);

-- employee_tasks table
CREATE TABLE employee_tasks (
    task_id SERIAL PRIMARY KEY,
    employee_id INT NOT NULL REFERENCES employees(employee_id),
    project_id INT NOT NULL REFERENCES projects(project_id),
    task_description TEXT,
    hours_spent DECIMAL(5, 2) CHECK (hours_spent >= 0),
    task_date DATE,
    status VARCHAR(20) DEFAULT 'Pending' CHECK (status IN ('Pending', 'In Progress', 'Completed', 'Cancelled'))
);

-- Populate departments
INSERT INTO departments (department_name, location) VALUES
('Human Resources', 'New York'),
('Engineering', 'San Francisco'),
('Marketing', 'Chicago'),
('Sales', 'Boston'),
('Research', 'Austin');

-- Populate employees
INSERT INTO employees (first_name, last_name, email, salary, hire_date, department_id, manager_id, bonus_percentage, performance_rating) VALUES
('Alice', 'Smith', 'alice.smith@example.com', 70000.00, '2020-01-15', 2, NULL, 10.00, 4), -- Manager for Eng
('Bob', 'Johnson', 'bob.johnson@example.com', 120000.00, '2019-03-01', 2, 1, 15.00, 5),
('Charlie', 'Brown', 'charlie.brown@example.com', 95000.00, '2021-07-20', 2, 1, 12.50, 4),
('Diana', 'Lee', 'diana.lee@example.com', 80000.00, '2022-06-10', 2, 1, NULL, 3),
('Eve', 'Davis', 'eve.davis@example.com', 60000.00, '2020-05-01', 1, NULL, 5.00, 4), -- Manager for HR
('Frank', 'Miller', 'frank.miller@example.com', 55000.00, '2021-08-25', 1, 5, 7.50, 3),
('Grace', 'Wilson', 'grace.wilson@example.com', 58000.00, '2022-01-10', 1, 5, NULL, 4),
('Henry', 'Moore', 'henry.moore@example.com', 90000.00, '2019-11-05', 3, NULL, 10.00, 5), -- Manager for Marketing
('Ivy', 'Taylor', 'ivy.taylor@example.com', 75000.00, '2020-02-17', 3, 8, 8.00, 4),
('Jack', 'Anderson', 'jack.anderson@example.com', 72000.00, '2021-09-30', 3, 8, 9.50, 3),
('Karen', 'Thomas', 'karen.thomas@example.com', 65000.00, '2023-01-20', 3, 8, NULL, 4),
('Leo', 'Jackson', 'leo.jackson@example.com', 110000.00, '2018-07-14', 4, NULL, 20.00, 5), -- Manager for Sales
('Mia', 'White', 'mia.white@example.com', 85000.00, '2019-04-01', 4, 12, 18.00, 4),
('Noah', 'Harris', 'noah.harris@example.com', 82000.00, '2020-10-15', 4, 12, 17.50, 4),
('Olivia', 'Martin', 'olivia.martin@example.com', 78000.00, '2021-12-01', 4, 12, NULL, 3),
('Paul', 'Garcia', 'paul.garcia@example.com', 130000.00, '2017-05-22', 5, NULL, 22.00, 5), -- Manager for Research
('Quinn', 'Martinez', 'quinn.martinez@example.com', 100000.00, '2018-09-10', 5, 16, 15.00, 5),
('Ruby', 'Robinson', 'ruby.robinson@example.com', 92000.00, '2019-06-05', 5, 16, 14.00, 4),
('Sam', 'Clark', 'sam.clark@example.com', 88000.00, '2020-11-11', 5, 16, NULL, 3),
('Tina', 'Rodriguez', 'tina.rodriguez@example.com', 105000.00, '2022-03-15', 2, 1, 10.00, 5),
('Uma', 'Lewis', 'uma.lewis@example.com', 62000.00, '2021-05-10', 1, 5, 6.00, 4),
('Victor', 'Walker', 'victor.walker@example.com', 77000.00, '2022-08-01', 3, 8, 7.00, 3),
('Wendy', 'Hall', 'wendy.hall@example.com', 90000.00, '2020-01-20', 4, 12, 19.00, 5),
('Xavier', 'Allen', 'xavier.allen@example.com', 115000.00, '2021-02-18', 5, 16, 16.00, 4),
('Yara', 'Young', 'yara.young@example.com', 71000.00, '2023-04-01', 2, 1, 5.00, 3);

-- Update manager_id for Alice, Eve, Henry, Leo, Paul (they were NULL, now they are their own managers for simplicity or a designated top manager if exists)
-- For this dataset, let's assume they are top-level or report to someone outside this scope.
-- Or, let's make Alice the overall CEO reporting to no one.
UPDATE employees SET manager_id = 1 WHERE employee_id IN (5, 8, 12, 16); -- Other managers report to Alice
UPDATE employees SET manager_id = NULL WHERE employee_id = 1;


-- Populate projects
INSERT INTO projects (project_name, start_date, end_date, budget, lead_employee_id) VALUES
('Project Alpha', '2023-01-15', '2023-06-30', 150000.00, 2), -- Bob (Eng)
('Project Beta', '2023-03-01', '2023-09-30', 250000.00, 17), -- Quinn (Research)
('Project Gamma', '2023-05-10', '2023-12-31', 100000.00, 9), -- Ivy (Marketing)
('Project Delta', '2023-07-01', '2024-01-31', 300000.00, 13), -- Mia (Sales)
('Project Epsilon', '2023-09-01', '2024-03-31', 75000.00, 3),  -- Charlie (Eng)
('Project Zeta', '2024-01-01', '2024-06-30', 220000.00, 18);  -- Ruby (Research)

-- Populate employee_tasks
INSERT INTO employee_tasks (employee_id, project_id, task_description, hours_spent, task_date, status) VALUES
(2, 1, 'Initial design phase', 40.5, '2023-01-20', 'Completed'),
(3, 1, 'Frontend development', 120.0, '2023-03-15', 'Completed'),
(4, 1, 'Backend development', 150.75, '2023-04-10', 'In Progress'),
(20, 1, 'API integration', 80.0, '2023-05-01', 'In Progress'),
(17, 2, 'Literature review', 60.0, '2023-03-10', 'Completed'),
(18, 2, 'Experiment setup', 90.5, '2023-05-01', 'In Progress'),
(19, 2, 'Data collection', 70.0, '2023-06-15', 'Pending'),
(24, 2, 'Preliminary analysis', 30.25, '2023-07-01', 'Pending'),
(9, 3, 'Market research', 50.0, '2023-05-15', 'Completed'),
(10, 3, 'Campaign strategy', 75.25, '2023-06-20', 'In Progress'),
(11, 3, 'Content creation', 100.0, '2023-08-01', 'In Progress'),
(22, 3, 'Social media outreach', 40.0, '2023-07-10', 'Pending'),
(13, 4, 'Lead generation plan', 45.0, '2023-07-05', 'Completed'),
(14, 4, 'Client outreach', 110.0, '2023-08-15', 'In Progress'),
(15, 4, ’Sales calls’, 95.5, ’2023-09-01’, ’In Progress’),
(23, 4, ’Contract negotiation’, 60.75, ’2023-09-20’, ’Pending’),
(3, 5, ’Requirements gathering’, 30.0, ’2023-09-05’, ’Completed’),
(4, 5, ’Prototyping’, 80.0, ’2023-10-10’, ’In Progress’),
(20, 5, ’User testing setup’, 40.5, ’2023-11-01’, ’Pending’),
(25, 5, ’Documentation’, 25.0, ’2023-11-15’, ’Pending’),
(18, 6, ’Advanced algorithm design’, 120.0, ’2024-01-10’, ’In Progress’),
(19, 6, ’Simulation runs’, 100.0, ’2024-02-15’, ’Pending’),
(24, 6, ’Results validation’, 70.25, ’2024-03-01’, ’Pending’),
(6, 1, ’HR support for Alpha team’, 10.0, ’2023-02-01’, ’Completed’),
(7, 3, ’HR support for Gamma team’, 12.0, ’2023-06-01’, ’Completed’),
(2, 1, ’Additional design review’, 15.0, ’2023-02-20’, ’Completed’),
(3, 1, ’Bug fixing phase 1’, 25.0, ’2023-04-01’, ’Completed’),
(17, 2, ’Grant proposal writing’, 35.0, ’2023-04-05’, ’Completed’),
(9, 3, ’Ad copy review’, 18.0, ’2023-07-01’, ’Completed’),
(13, 4, ’Sales deck preparation’, 22.0, ’2023-07-20’, ’Completed’),
(2, 5, ’Technical specification’, 33.0, ’2023-09-20’, ’In Progress’),
(4, 1, ’Final testing’, 50.0, ’2023-05-15’, ’Cancelled’), -- Cancelled task
(10, 3, ’Competitor analysis’, 30.0, ’2023-06-01’, ’Completed’),
(14, 4, ’Follow-up emails’, 20.0, ’2023-09-05’, ’In Progress’),
(18, 2, ’Refine experiment design’, 25.0, ’2023-05-20’, ’In Progress’),
(20, 1, ’Security audit’, 40.0, ’2023-05-20’, ’Pending’),
(25, 5, ’Deployment planning’, 15.0, ’2023-12-01’, ’Pending’),
(2, 1, ’Documentation for design’, 20.0, ’2023-02-25’, ’Completed’),
(3, 5, ’Frontend module for Epsilon’, 60.0, ’2023-10-25’, ’In Progress’),
(9, 3, ’Press release draft’, 25.0, ’2023-08-10’, ’In Progress’),
(13, 4, ’CRM data update’, 10.0, ’2023-09-10’, ’Completed’),
(17, 6, ’Research paper outline’, 40.0, ’2024-01-20’, ’Pending’),
(4, 1, ’Performance optimization’, 0.0, ’2023-05-01’, ’Pending’), -- Task with 0 hours
(6, 2, ’Recruitment for Project Beta’, 15.0, ’2023-03-15’, ’Completed’),
(7, 4, ’Onboarding new sales members’, 18.0, ’2023-07-10’, ’In Progress’),
(11, 3, ’Video ad script’, 30.0, ’2023-08-15’, ’Pending’),
(15, 4, ’Quarterly sales report’, 12.0, ’2023-09-25’, ’Pending’),
(19, 6, ’Lab maintenance’, 8.0, ’2024-02-20’, ’Pending’),
(22, 3, ’Influencer outreach’, 22.0, ’2023-07-25’, ’In Progress’),
(23, 4, ’Legal review of contracts’, 16.0, ’2023-09-28’, ’Pending’);
```

Listing 1: Dataset (identical to exercises document)

1 Practice Meanings, Values, Relations, and Advantages - Solutions

Exercise i.1: Overall Company Metrics

Problem: Calculate the total number of employees, the total salary expenditure, the average salary, the minimum salary, and the maximum salary across all employees. What is the main advantage of using aggregate functions for this?

```sql
SELECT
    COUNT(*) AS total_employees,
    SUM(salary) AS total_salary_expenditure,
    AVG(salary) AS average_salary,
    MIN(salary) AS minimum_salary,
    MAX(salary) AS maximum_salary
FROM employees;
```

Exercise i.2: Department Employee Listing

Problem: For each department, list the department name, the number of employees in that department, and a comma-separated list of all employee first names in that department, ordered alphabetically by first name. How does STRING_AGG help here?

```sql
SELECT
    d.department_name,
    COUNT(e.employee_id) AS number_of_employees,
    STRING_AGG(e.first_name, ', ' ORDER BY e.first_name) AS
   employee_first_names
FROM departments d
JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
ORDER BY d.department_name;
```

Exercise i.3: Understanding Different COUNTs

Problem: Find the total number of employees, the number of employees with a bonus_percentage recorded, and the number of distinct performance_rating values. Explain the difference in meaning for each COUNT.

```sql
SELECT
    COUNT(*) AS total_employees_asterisk,
    COUNT(bonus_percentage) AS employees_with_bonus_data,
    COUNT(DISTINCT performance_rating) AS distinct_performance_ratings
FROM employees;
```

Exercise i.4: Median Salary and Mode Performance Rating

Problem: Calculate the median salary for employees in the ’Engineering’ department and the most common (mode) performance rating for the entire company. What is the value of PERCENTILE_CONT and MODE?

```sql
SELECT
    (SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary)
     FROM employees e
```

```
      JOIN departments d ON e.department_id = d.department_id
      WHERE d.department_name = 'Engineering') AS
    median_engineering_salary,
     MODE() WITHIN GROUP (ORDER BY performance_rating) AS
    mode_performance_rating_overall
FROM employees
LIMIT 1;
```

Exercise i.5: Project Task Hours Distribution

Problem: For each project, display its name, total hours spent, and the variance and standard deviation of hours spent on its tasks. How do VARIANCE and STDDEV help understand data distribution?

```
SELECT
    p.project_name,
    SUM(et.hours_spent) AS total_hours_on_project,
    VARIANCE(et.hours_spent) AS variance_hours_spent,
    STDDEV(et.hours_spent) AS stddev_hours_spent
FROM projects p
JOIN employee_tasks et ON p.project_id = et.project_id
GROUP BY p.project_id, p.project_name
HAVING COUNT(et.task_id) > 1
ORDER BY p.project_name;
```

Exercise i.6: Departmental and Cumulative Salaries

Problem: Show the total salary for each department. Also, show the cumulative salary within each department as employees are ordered by their hire date (earliest first). What is the advantage of the window aggregate here?

```
SELECT
    e.first_name,
    e.last_name,
    d.department_name,
    e.salary,
    e.hire_date,
    SUM(e.salary) OVER (PARTITION BY e.department_id) AS
   department_total_salary,
    SUM(e.salary) OVER (PARTITION BY e.department_id ORDER BY e.
   hire_date ASC, e.employee_id ASC) AS cumulative_salary_in_department
FROM employees e
JOIN departments d ON e.department_id = d.department_id
ORDER BY d.department_name, e.hire_date, e.employee_id;
```

2 Practice Disadvantages and Potential Pitfalls - Solutions

Exercise ii.1: Loss of Detail with Average

Problem: Calculate the average salary for the 'Engineering' department. What specific salary information is lost when you only look at this average?

```sql
SELECT AVG(salary) AS average_engineering_salary
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE d.department_name = 'Engineering';
```

Exercise ii.2: Misleading Aggregate without GROUP BY

Problem: Consider the query `SELECT department_id, MAX(salary) FROM employees;`. Why might this query be misleading or incorrect if the user intends to find the maximum salary *for each department*? What is the potential pitfall?

```sql
-- The problematic query would cause an error in strict SQL:
-- SELECT department_id, MAX(salary) FROM employees;
-- In systems that allow it (like older MySQL), the department_id would
    be arbitrary.
-- The intended, correct query to get max salary PER department is:
SELECT d.department_name, MAX(e.salary)
FROM employees e
JOIN departments d ON e.department_id = d.department_id
GROUP BY d.department_name;
```

Exercise ii.3: NULL Handling in AVG()

Problem: Calculate the average `bonus_percentage` for all employees. How does `AVG()` handle `NULL` values in `bonus_percentage`, and how could this be misleading if not understood?

```sql
SELECT AVG(bonus_percentage) AS average_bonus_percentage_ignoring_nulls
FROM employees;

-- To illustrate the difference if NULLs were treated as 0:
SELECT SUM(COALESCE(bonus_percentage, 0)) / COUNT(*) AS
   average_bonus_treating_nulls_as_zero
FROM employees;
```

Exercise ii.4: Aggregate in WHERE Clause

Problem: A manager wants to find departments where the average employee performance rating is below 3.5. They try to write: `SELECT department_id, AVG(performance_rating) FROM employees WHERE AVG(performance_rating) < 3.5 GROUP BY department_id;`. Why will this query fail, and what is the disadvantage or common mistake illustrated here regarding aggregate function placement?

```sql
-- The query fails because aggregates like AVG cannot be used directly
   in the WHERE clause.
-- The correct approach uses HAVING:
SELECT d.department_name, AVG(e.performance_rating) AS avg_rating
FROM employees e
JOIN departments d ON e.department_id = d.department_id
GROUP BY d.department_id, d.department_name
HAVING AVG(e.performance_rating) < 3.5;
```

3 Inefficient vs. Efficient Aggregate Usage - Solutions

Exercise iii.1: Counting Tasks Inefficiently

Problem: A junior analyst needs to find the total number of tasks for 'Project Alpha'. They write a script that fetches all task IDs for 'Project Alpha' into an application and then counts them using application code. Provide the efficient SQL aggregate function solution.

```sql
SELECT COUNT(et.task_id) AS total_tasks_for_project_alpha
FROM employee_tasks et
JOIN projects p ON et.project_id = p.project_id
WHERE p.project_name = 'Project Alpha';
```

Exercise iii.2: Calculating Average Salary Inefficiently

Problem: To find the average salary of employees hired in 2020, a developer first queries for all salaries of employees hired in 2020, then sums them up and divides by the count in their programming language. How can this be done efficiently in a single SQL query using aggregate functions?

```sql
SELECT AVG(salary) AS average_salary_2020_hires
FROM employees
WHERE EXTRACT(YEAR FROM hire_date) = 2020;
```

Exercise iii.3: Finding Max Salary Per Department Inefficiently

Problem: A data scientist wants to get a list of departments and, for each, the maximum salary. They write separate queries for each department: `SELECT MAX(salary) FROM employees WHERE department_id = 1;`, then `SELECT MAX(salary) FROM employees WHERE department_id = 2;`, etc. for all departments. Provide a single, efficient SQL query.

```sql
SELECT
    d.department_name,
    MAX(e.salary) AS max_salary_in_department
FROM employees e
JOIN departments d ON e.department_id = d.department_id
GROUP BY d.department_id, d.department_name
ORDER BY d.department_name;
```

Exercise iii.4: Filtering by Total Hours Inefficiently

Problem: An HR assistant needs to find all employees who have logged more than 150 total hours on tasks. They fetch all tasks for every employee, sum the hours in a spreadsheet, and then filter. How can this be done with an efficient SQL query using aggregates and `HAVING`?

```sql
SELECT
    e.first_name,
    e.last_name,
    SUM(et.hours_spent) AS total_hours_logged
FROM employees e
```

```sql
JOIN employee_tasks et ON e.employee_id = et.employee_id
GROUP BY e.employee_id, e.first_name, e.last_name
HAVING SUM(et.hours_spent) > 150
ORDER BY total_hours_logged DESC;
```

4 Hardcore Problem Combining Concepts - Solutions

Exercise iv.1: Top Employees by Salary per Department with Aggregates and Ranking

Problem: For each department, identify the top 2 employees by salary. For these employees, show their full name, department name, salary, their salary rank within the department (dense rank), the total salary expenditure for their department, and their salary as a percentage of their department's total salary. Only include departments with at least 3 employees. Order results by department name and then by rank.

```sql
WITH DepartmentSalaries AS (
    SELECT
        d.department_id,
        d.department_name,
        SUM(e.salary) AS department_total_salary,
        COUNT(e.employee_id) AS num_employees_in_department
    FROM departments d
    JOIN employees e ON d.department_id = e.department_id
    GROUP BY d.department_id, d.department_name
),
RankedSalaries AS (
    SELECT
        e.employee_id,
        e.first_name,
        e.last_name,
        e.salary,
        e.department_id,
        DENSE_RANK() OVER (PARTITION BY e.department_id ORDER BY e.salary DESC) as salary_rank_in_department
    FROM employees e
)
SELECT
    rs.first_name || ' ' || rs.last_name AS employee_full_name,
    ds.department_name,
    rs.salary,
    rs.salary_rank_in_department,
    ds.department_total_salary,
    ROUND((rs.salary / ds.department_total_salary) * 100, 2) AS pct_of_department_total_salary
FROM RankedSalaries rs
JOIN DepartmentSalaries ds ON rs.department_id = ds.department_id
WHERE rs.salary_rank_in_department <= 2 AND ds.num_employees_in_department >= 3
ORDER BY ds.department_name, rs.salary_rank_in_department;
```

Exercise iv.2: Project Metrics, Budget Ranking, and Cumulative Budget

Problem: List all projects. For each project, show its name, budget, total hours spent by all employees on that project, and the average hours spent per task on that project. Additionally, rank projects by their budget (highest first). For projects that started in 2023, also show the running total of budgets for projects started in 2023, ordered by their start date.

```sql
WITH ProjectTaskAggregates AS (
    SELECT
        p.project_id,
        p.project_name,
        p.budget,
        p.start_date,
        COALESCE(SUM(et.hours_spent), 0) AS total_project_hours,
        COALESCE(AVG(et.hours_spent), 0) AS avg_hours_per_task,
        COUNT(et.task_id) AS num_tasks
    FROM projects p
    LEFT JOIN employee_tasks et ON p.project_id = et.project_id
    GROUP BY p.project_id, p.project_name, p.budget, p.start_date
),
ProjectBudgetRanking AS (
    SELECT
        *,
        RANK() OVER (ORDER BY budget DESC) AS budget_rank,
        CASE
            WHEN EXTRACT(YEAR FROM start_date) = 2023
            THEN SUM(budget) OVER (PARTITION BY EXTRACT(YEAR FROM start_date) ORDER BY start_date ASC, project_id ASC)
            ELSE NULL
        END AS running_total_budget_2023_projects
    FROM ProjectTaskAggregates
)
SELECT
    project_name,
    budget,
    budget_rank,
    total_project_hours,
    CASE WHEN num_tasks > 0 THEN avg_hours_per_task ELSE 0 END AS avg_hours_per_task, -- Avoid division by zero if no tasks
    running_total_budget_2023_projects
FROM ProjectBudgetRanking
ORDER BY budget_rank, start_date;
```

Exercise iv.3: Employees Above Department Average Salary with Ranking

Problem: For every employee, display their full name, department name, salary, the average salary of their department, and their salary's rank (using ROW_NUMBER for unique ranks) within their department. Then, filter this list to show only employees who earn more than their department's average salary and whose hire date is after '2020-01-01'. Order the final result by department name and then by salary in descending order.

```sql
WITH EmployeeDepartmentAvgSalary AS (
    SELECT
        e.employee_id,
        e.first_name,
        e.last_name,
        d.department_name,
        e.department_id,
        e.salary,
        e.hire_date,
        AVG(e.salary) OVER (PARTITION BY e.department_id) AS department_avg_salary,
        ROW_NUMBER() OVER (PARTITION BY e.department_id ORDER BY e.salary DESC, e.employee_id ASC) AS salary_rank_in_department
    FROM employees e
    JOIN departments d ON e.department_id = d.department_id
)
SELECT
    first_name || ' ' || last_name AS employee_full_name,
    department_name,
    salary,
    ROUND(department_avg_salary, 2) AS department_avg_salary,
    salary_rank_in_department,
    hire_date
FROM EmployeeDepartmentAvgSalary
WHERE salary > department_avg_salary
  AND hire_date > '2020-01-01'
ORDER BY department_name, salary DESC;
```