SQL for Data Science

Contents

1 PostgreSQL For Data Science

1.1 SQL Query Basics (Filtering)

The SQL language is case insensitive, but the data within the cells is case sensitive.
However, the industry standard is to capitalize the SQL keywords

```
select * from employees;
SELECT * FROM employees; -- Industry Standard
-- These two statements are the same and will produce same output
```

SQL Keywords:

· SELECT - this will specify which column/attribute you want to see in a query
· FROM - specifies which table you want to select the column/attribute from
· WHERE - a condition that lets you set parameters on a given column
· LIKE - used in addition with WHERE and '% %', lets you search for parts of a word

```
SELECT * FROM employees
WHERE department LIKE 'F%nitu%' -- such as Furniture (case sensitive)
-- word must start with 'F' and contains 'nitu' somwhere in middle
```

We can filter on multiple conditions by using the AND operator or the OR operator
We can also filter with these operators together by using ( )

```
SELECT * FROM employees
WHERE department = 'Clothing' AND salary > 90000 AND region_id = 2
-- all 3 conditions must be met in order to be added to query

SELECT * FROM employees
WHERE department = 'Clothing' OR salary > 150000
-- will add any row that meets either of these conditions to the query

SELECT * FROM employees
WHERE salary < 40000 AND (department = 'Clothing' OR department = 'Pharmacy')
-- will add any row where salary is less than 40k and in either department to query
```

We can use filtering operators to further process our data. Some key operators are:

· NOT - gives us value that are not in the given search ('NOT =' is same as '!=').
· IS NULL - will return all values that are null (use IS NOT NULL to get all non-null values).
· IN - will return all values that are found within the given parameters.
· BETWEEN - will search for values within a given range (inclusive).

```
SELECT * FROM employees
WHERE NOT department = 'Clothing'; -- gives us all departments besides clothing

SELECT * FROM employees
WHERE email IS NOT NULL; -- gives all emails that don't have null value

SELECT * FROM employees
WHERE department IN ('Sports', 'Firs Aid', 'Garden');
-- return values in any of the given departments

SELECT * FROM employees
WHERE salary BETWEEN 80000 AND 100000; -- all salaries in this range
```

We can change the way the information is displayed in the output of our queries with certain keywords:

· ORDER BY - sorts the data by given column, either ascending (ASC) or descending (DESC)

· DISTINCT - will return all unique values for a given column

· LIMIT - will set how many records to show from our query (same as FETCH FIRST _ ROWS ONLY)

· AS - allows us to rename a column to a given parameter name (good for exporting queries)

```
SELECT * FROM employees
ORDER BY employee_id DESC; -- orders from employee_id 1000 down to 1

SELECT DISTINCT department FROM employees -- selects unique departments in table
ORDER BY 1 -- orders by first parameter for select statement
LIMIT 10; -- only show first 10 records

SELECT first_name AS "First Name", salary AS yearly_salary -- use " " when spaces
FROM employees
FETCH FIRST 10 ROWS ONLY; -- only show first 10 records (same as LIMIT 10)
```

1.2 Using Functions

We can use functions with the SELECT statement to alter our data in output queries (not in the table).

· UPPER( ) - converts output to all uppercase

· LOWER( ) - converts output to all lowercase

· LENGTH( ) - gives you the length for a each string in all rows for a given column

· TRIM( ) - will take off any extra space in beginning or end (good for cleaning data)

```
-- all uppercase first name, all lowercase department
SELECT LENGTH(first_name), LOWER(department)
FROM employees;

-- we can test a function without selecting from a table
SELECT LENGTH(TRIM('   HELLO   ')) -- trim extra space, return length = 5
```

We can combine multiple columns together by using concatenation || or a Boolean expression

```
-- combine first name and last name into new column called full_name
SELECT first_name || ' ' || last_name AS full_name
FROM employees;

-- use a boolean to create a new column of True or False
SELECT (salary > 140000) AS highly_paid
FROM employees
ORDER BY salary DESC; -- show all true values first

SELECT department, ('Clothing' IN department) -- creates a new column of T/F
FROM employees;

SELECT department, (department LIKE '%oth%') -- creates a new column of T/F
FROM employees; -- T if department has 'oth' in its name, otherwise F
```

1.2.1 String Functions

We can perform functions on strings in a given table and output them in our query (not in the table).

· SUBSTRING( ) - lets us extract part of a string (FROM start FOR length)
· REPLACE( ) - lets us change the name of strings to a new string
· POSITION( ) - lets us find a position of a given character
· COALESCE( ) - lets us add a value into any null cells for the output

```
-- test the substring function with a given string
SELECT SUBSTRING('This is test data' FROM 1 FOR 4) AS test_data; -- returns 'This'
-- excluding the FOR will go from position 1 to end of string

-- replace every occurance of Clothing with Attire in a new column
SELECT department, REPLACE(department, 'Clothing', 'Attire') AS modified_depart
FROM employees;

-- return a column of integers that contain the position of @ in the email column
SELECT POSITION('@' IN email)
FROM employees;

-- use position and substring to extract email domain names into a new column
SELECT email, SUBSTRING(email FROM POSITION('@' IN email)+1) AS email_domain
FROM employees; -- for above statement, +1 ignores @ sign (increment start position)

-- create new column from email with null values filled in as NONE
SELECT COALESCE(email, 'NONE') AS email_filled
FROM employees;
```

1.2.2 Grouping Functions

We can perform calculations on data and get statistical insight from these queries (for numeric data). Note that grouping functions take in multiple rows, but only output one row (the calculation).

· MAX( ) - returns the highest numeric value in a given column
· MIN( ) - returns the lowest numeric value in a given column
· AVG( ) - returns the average numeric value in a given column
· ROUND(,) - rounds our data to a given decimal place value
· COUNT( ) - gives the total number of records in a column (excludes null values)
· SUM( ) - sums the numeric values in a given column

```
-- find the highest, lowest, and average paid salary in our table
SELECT MAX(salary) FROM employees;
SELECT MIN(salary) FROM employees;
SELECT ROUND(AVG(salary),3) FROM employees; -- average rounded to 3 decimal places

-- find total number of records in our table
SELECT COUNT(*) FROM employees;
-- note: we often use count(*) to make sure all records are counted incase of null's

-- find the yearly amount paid to employees
SELECT SUM(salary) FROM employees;
```

1.3 Grouping Data / Computing Aggregates

We can use the GROUP BY command to group records that are the same in a given column.

· Note that 'group by' comes after *where* clause but before *order by*.

· We can call the parameter for 'group/order by' with the column position in the select statement.

· Any non-aggregate columns in select list must also by mentioned in group by clause.

```sql
-- group salaries by department for region's 4, 5, 6, and 7
SELECT department, SUM(salary)
FROM employees
WHERE region_id IN (4,5,6,7)
GROUP BY department;

-- get number of employees, average/min/max salary for each department and
-- order from highest employee count to lowest
SELECT department, COUNT(*) AS num_of_employees,
  ROUND(AVG(salary), 3) AS average_salary,
  MIN(salary) AS lowest_salary,
  MAX(salary) AS highest_salary
FROM employees
GROUP BY 1 -- department
ORDER BY 2 DESC; -- num_of_employees

-- get the number of employees by gender for each department
SELECT department, gender, COUNT(*) AS num_of_employees
FROM employees
GROUP BY 1, 2 -- gender is non-aggregate function (must be included)
ORDER BY 1; -- order by department (easier to read)

-- how many people have the same first name (name and count)
SELECT first_name, COUNT(*) as occurances
FROM employees
GROUP BY 1
HAVING COUNT(*) > 1;

-- get unique domain names for email and the number of employees having that domain
SELECT SUBSTRING(email FROM POSITION('@' IN email)+1) AS email_domain,
COUNT(*) AS num_of_employees
FROM employees
WHERE email IS NOT NULL
GROUP BY 1
ORDER BY 2 DESC;

-- get the min/max/avg salary for each gender in every region
SELECT gender, region_id, MIN(salary) AS min_salary,
MAX(salary) AS max_salary,
ROUND(AVG(salary)) AS avg_salary -- round nearest whole num
FROM employees
GROUP BY 1,2
ORDER BY 1,2 ASC;
```

We can use the HAVING command to filter data that has been grouped (similar to where clause).

· This command is used to filter *aggregated* data.

· Note that the having command comes after the *group by* and before the *order by* clauses

```sql
-- get all department names that have more than 35 employees
SELECT department, COUNT(*)
FROM eployees
GROUP BY 1
HAVING COUNT(*) > 35
ORDER BY 1;
```

1.4 Using Subqueries

We can refer to columns of specific sources by specifying the table name before the column in the SELECT statement. We can make this even more simple by giving the sources aliases as their reference name.

· Note for future - we can have select statements nested in from statements (new source of data).

```
-- aliase employees table as 'e' and departments as 'd'
SELECT d.department -- get department column from departments table
FROM employees e, departments d;
```

We can use subqueries in the WHERE/FROM clause (acts as a source from which data can be pulled from). In the from clause, we need to give the subquery an alias.

· The inner query (subquery) returns a list of data that we can pull from.

· Renaming a subqueries column names means we must reference these names in the outer query.

· We can have multiple sources of data (subqueries) in our FROM clause.

```
-- select all employees who work in a department not listed in departments table
SELECT * FROM employees
WHERE department NOT IN (SELECT department from departments);

-- select first name and salary from subquery 'a' of employees with salary > 150k
SELECT a.first_name, a.salary
FROM (SELECT * FROM employees WHERE salary > 150000) a

-- renaming inner query names (same as above problem)
SELECT a.employee_name, a.yearly_salary
FROM (SELECT first_name AS employee_name, salary AS yearly_salary
      FROM employees WHERE salary > 150000) a

-- select all employees who work in the electronics division
SELECT * FROM employees
WHERE department IN (SELECT department
FROM departments WHERE division = 'Electronics');

-- select all employees that work in Asia or Canada and make over 130k
SELECT * FROM employees
WHERE region_id IN (SELECT region_id FROM regions WHERE country IN ('Asia','Canada'))
AND salary > 130000;
```

We can use subqueries in the SELECT statement, but we must make sure that the subquery only returns one record. This can be used to compare all records of outer query to one value from inner query (see example below).

```
-- INVALID SYNTAX EXAMPLE
SELECT first_name, salary, (SELECT first_name FROM employees)
FROM employees;
/* this will not run, the inner query will try to return all 1000 first_names
  for each record in outer query. */

/* select first_name and department of employee and how much less they make than
   the highet paid employee (in Asia and Canada) */
SELECT first_name, department,
       ((SELECT MAX(salary) from employees) - salary) AS less_income
FROM employees
WHERE region_id IN (SELECT region_id FROM regions
                    WHERE country IN ('Asia','Canada'));
```

We can us the ALL/ANY clause in the where/having clause, and they have the following function:

· ANY - will return true if any of the subquery values meet the condition.

· ALL - will return true if all subquery values meet the condition.

You can use all operator $>, <, >=, <=$ when using these clauses.

Note that ANY will still return values inside the subquery (tricky to use).

```sql
-- select all employees who's region_id is greater than US region id's (1,2,3)
SELECT * FROM employees
WHERE region_id > ALL (SELECT region_id FROM regions WHERE country='United States');

/* select all employees that work in the kids devision and the dates
at which they were hired is greater than all hire dates of employees
in the maintenance department */
SELECT * FROM employees
WHERE department IN (SELECT department FROM departments WHERE division = 'Kids')
AND hire_date > ALL (SELECT hire_date FROM employees
                     WHERE department = 'Maintenance');

-- select the salary that occures the most often (and is the highest value)
SELECT salary, COUNT(*) FROM employees
GROUP BY 1
ORDER BY 2 DESC, 1 DESC
LIMIT 1;
-- same as...
SELECT salary FROM employees
GROUP BY salary
HAVING COUNT(*) >= ALL(SELECT COUNT(*) FROM employees GROUP BY salary)
ORDER BY 1 DESC
LIMIT 1;

-- find the average salary excluding the lowest and highest paid employee
SELECT ROUND(AVG(salary)) FROM employees
WHERE salary < ALL (SELECT MAX(salary) FROM employees)
AND salary > ALL (SELECT MIN(salary) FROM employees);
```

1.5 The CASE Clause

1.6 Practice Problems with Solutions

(1.1 SQL Query Basics)

```sql
/* First name and email of females that work in the tools department having a
   salary greater than 110,000 */
SELECT first_name, email FROM employees
WHERE gender = 'F' AND department = 'Tools' AND salary > 110000;

/* First name and hire date of employees who earn more than 165,000 as well as
   employees that work in the sports department and are men */
SELECT first_name, hire_date FROM employees
WHERE salary > 165000 OR (department = 'Sports' and gender = 'M');

/* First name and hire date of employees hired during Jan 1, 2002 and Jan 1, 2004 */
SELECT first_name, hire_date FROM employees
WHERE hire_date BETWEEN '2002-01-01' AND '2004-01-01';

/* All columns from male employees who work in the automotive department and earn
   more than 40,000 and less than 100,000 as well as females that work in the
   toy department */
SELECT * FROM employees
WHERE gender = 'M' AND department = 'Automotive'
AND (salary BETWEEN 40000 and 100000)
OR (gender = 'F' AND department = 'Toys');
```

(1.2 Using Functions)

```sql
/* Write a query against the professors table that can output the following in the
   result: "Chong works in the Science department" */
SELECT last_name || ' works in the ' || department || ' department'
FROM professors
WHERE last_name = 'Chong';

/* Write a query that says if a professor mis highly paid (above 95000)
   in the format "It is false that professor Chong is highly paid" */
SELECT 'It is ' || (salary > 95000) || ' that professor ' || last_name ||
       ' is highly paid'
FROM professors;

/* Write a query that returns all of the records and columns from the professors
   table but shortens the department names to only the first three characters in
   upper case. */
SELECT last_name, UPPER(SUBSTRING(department FROM 1 FOR 3)) AS department_abrv,
       salary, hire_date
FROM professors;

/* Write a query that returns the highest and lowest salary from the professors
   table excluding the professor named 'Wilson'. */
SELECT MAX(salary) AS max_salary, MIN(salary) AS min_salary
FROM professors
WHERE last_name != 'Wilson';

/* Write a query that will display the hire date of the professor that has been
   teaching the longest. */
SELECT MIN(hire_date) FROM professors;
```

(1.3 Grouping Data / Computing Aggregates)

```sql
-- Write a query that displays only the state with the largest amount of fruit supply
SELECT state
FROM fruit_imports
GROUP BY 1
ORDER BY SUM(supply) DESC
LIMIT 1;

-- Write a query that returns the most expensive cost_per_unit of every season.
SELECT season, MAX(cost_per_unit)
FROM fruit_imports
GROUP BY 1;

-- Write a query that returns the state that has more than 1 import of the same fruit
SELECT state
FROM fruit_imports
GROUP BY 1, name
HAVING COUNT(name) > 1;

-- Write a query that returns the seasons that produce either 3 fruits or 4 fruits.
SELECT season
FROM fruit_imports
GROUP BY 1
HAVING ((COUNT(name) = 3) OR (COUNT(name) = 4));

/* Write a query that takes into consideration the supply and cost_per_unit columns
   for determining the total cost and returns the most expensive state with the total
   cost. */
SELECT state, SUM(supply * cost_per_unit) AS total_cost
FROM fruit_imports
GROUP BY 1
ORDER BY 2 DESC
LIMIT 1;

-- Execute the below SQL script and write a query that returns the count of 4.
CREATE table fruits (fruit_name varchar(10));
INSERT INTO fruits VALUES ('Orange');
INSERT INTO fruits VALUES ('Apple');
INSERT INTO fruits VALUES (NULL);
INSERT INTO fruits VALUES (NULL);

SELECT COUNT(COALESCE(fruit_name, 'SOMEVALUE'))
FROM fruits;
```

(1.4 Using Subqueries)