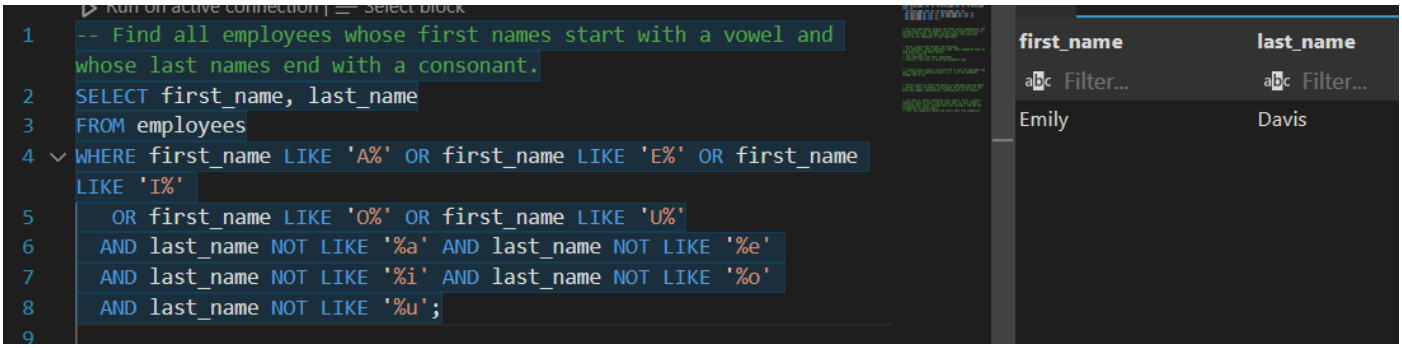
Task 1 Queries Outputs

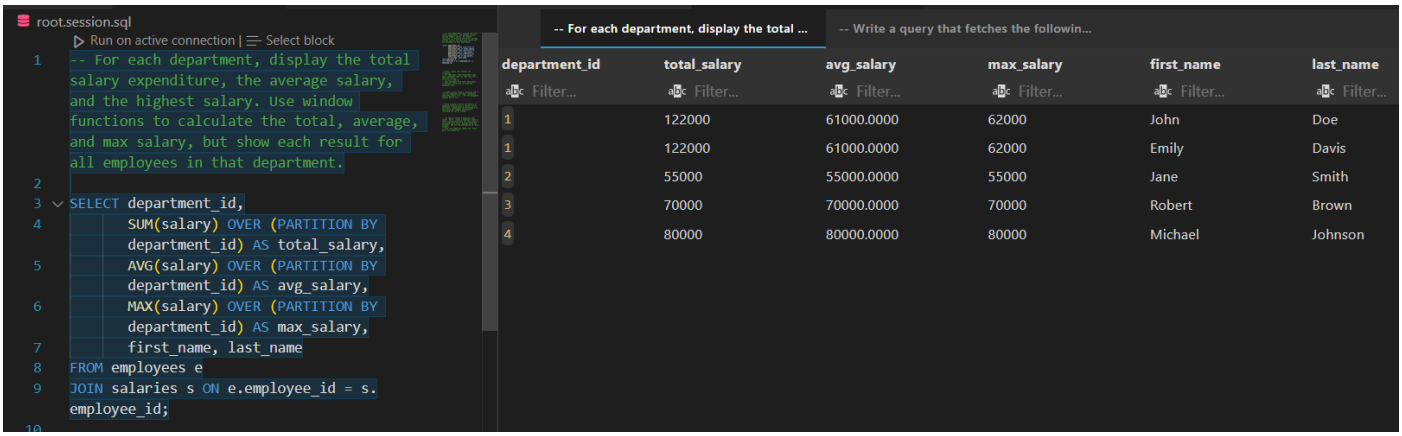
Find all employees whose first names start with a vowel and whose last names end with a consonant.



```sql
-- Find all employees whose first names start with a vowel and
whose last names end with a consonant.
SELECT first_name, last_name
FROM employees
WHERE first_name LIKE 'A%' OR first_name LIKE 'E%' OR first_name
LIKE 'I%'
   OR first_name LIKE 'O%' OR first_name LIKE 'U%'
  AND last_name NOT LIKE '%a' AND last_name NOT LIKE '%e'
  AND last_name NOT LIKE '%i' AND last_name NOT LIKE '%o'
  AND last_name NOT LIKE '%u';
```

first_name	last_name
Filter...	Filter...
Emily	Davis

For each department, display the total salary expenditure, the average salary, and the highest salary. Use window functions to calculate the total, average, and max salary, but show each result for all employees in that department.



```sql
-- For each department, display the total
salary expenditure, the average salary,
and the highest salary. Use window
functions to calculate the total, average,
and max salary, but show each result for
all employees in that department.

SELECT department_id,
       SUM(salary) OVER (PARTITION BY
       department_id) AS total_salary,
       AVG(salary) OVER (PARTITION BY
       department_id) AS avg_salary,
       MAX(salary) OVER (PARTITION BY
       department_id) AS max_salary,
       first_name, last_name
FROM employees e
JOIN salaries s ON e.employee_id = s.
employee_id;
```

department_id	total_salary	avg_salary	max_salary	first_name	last_name
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
1	122000	61000.0000	62000	John	Doe
1	122000	61000.0000	62000	Emily	Davis
2	55000	55000.0000	55000	Jane	Smith
3	70000	70000.0000	70000	Robert	Brown
4	80000	80000.0000	80000	Michael	Johnson

Write a query that fetches the following:

All employees, their department name, their manager's name (if they have one), and their salary.

You will need to:

Join employees with their department.

Perform a self-join to fetch the manager's name.

** no info about manager is given

Create a query using a recursive CTE to list all employees and their respective reporting chains (i.e., list the manager's manager and so on).

** no info about manager is given

Write a query to fetch the details of employees earning above a certain salary threshold. Investigate the performance of this query and suggest improvements, including the use of indexes

-- Taking average of all employees salary as the salary threshold

```
mysql> SET profiling = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SELECT e.employee_id, e.first_name, e.last_name, s.salary
    -> FROM employees e
    -> JOIN salaries s ON e.employee_id = s.employee_id
    -> WHERE s.salary > (SELECT AVG(salary) FROM salaries);
+-------------+------------+-----------+--------+
| employee_id | first_name | last_name | salary |
+-------------+------------+-----------+--------+
|         103 | Robert     | Brown     |  70000 |
|         105 | Michael    | Johnson   |  80000 |
+-------------+------------+-----------+--------+
2 rows in set (0.00 sec)

mysql> CREATE INDEX idx_salary ON salaries(salary);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_employee_id ON salaries(employee_id);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT e.employee_id, e.first_name, e.last_name, s.salary
    -> FROM employees e
    -> JOIN salaries s ON e.employee_id = s.employee_id
    -> WHERE s.salary > (SELECT AVG(salary) FROM salaries);
+-------------+------------+-----------+--------+
| employee_id | first_name | last_name | salary |
+-------------+------------+-----------+--------+
|         103 | Robert     | Brown     |  70000 |
|         105 | Michael    | Johnson   |  80000 |
+-------------+------------+-----------+--------+
2 rows in set (0.00 sec)

mysql> SHOW profiles \G
```

```
Query_ID: 10
Duration: 0.00095300
   Query: SELECT e.employee_id, e.first_name, e.last_name, s.salary
FROM employees e
JOIN salaries s ON e.employee_id = s.employee_id
WHERE s.salary > (SELECT AVG(salary) FROM salaries)
*************************** 11. row ***************************
Query_ID: 11
Duration: 0.07844850
   Query: CREATE INDEX idx_salary ON salaries(salary)
*************************** 12. row ***************************
Query_ID: 12
Duration: 0.03532000
   Query: CREATE INDEX idx_employee_id ON salaries(employee_id)
*************************** 13. row ***************************
Query_ID: 13
Duration: 0.00086725
   Query: SELECT e.employee_id, e.first_name, e.last_name, s.salary
FROM employees e
JOIN salaries s ON e.employee_id = s.employee_id
WHERE s.salary > (SELECT AVG(salary) FROM salaries)
13 rows in set, 1 warning (0.00 sec)
```

A difference of (0.000953 - 0.000867) = 0.000086 seconds

You need to create a detailed sales report. First, create a temporary table to store interim sales data for each product, including total sales, average sales per customer, and the top salesperson for each product.

Hint: Use temporary tables and insert data from subqueries.

-- Table Creation Queries

```
mysql> CREATE TABLE products (
    ->     product_id INT PRIMARY KEY,
    ->     product_name VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> CREATE TABLE salespersons (
    ->     salesperson_id INT PRIMARY KEY,
    ->     first_name VARCHAR(50),
    ->     last_name VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE TABLE sales (
    ->     sales_id INT PRIMARY KEY,
    ->     product_id INT,
    ->     salesperson_id INT,
    ->     sales_amount DECIMAL(10, 2),
    ->     FOREIGN KEY (product_id) REFERENCES products(product_id),
    ->     FOREIGN KEY (salesperson_id) REFERENCES salespersons(salesperson_id)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> CREATE TEMPORARY TABLE temp_sales_report (
    ->     product_id INT,
    ->     total_sales DECIMAL(10, 2),
    ->     avg_sales_per_customer DECIMAL(10, 2),
    ->     top_salesperson VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

-- Inserting Sample Data

```
mysql> -- Sample Data for Products
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO products (product_id, product_name) VALUES
    -> (1, 'Product A'),
    -> (2, 'Product B');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Sample Data for Salespersons
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO salespersons (salesperson_id, first_name, last_name) VALUES
    -> (1, 'John', 'Doe'),
    -> (2, 'Jane', 'Smith');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Sample Data for Sales
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO sales (sales_id, product_id, salesperson_id, sales_amount) VALUES
    -> (1, 1, 1, 100.00),
    -> (2, 1, 2, 150.00),
    -> (3, 2, 1, 200.00),
    -> (4, 2, 2, 300.00);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

-- Inserting require analysis data to temp table

```
mysql> -- Inserting require analysis data to temp table
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO temp_sales_report (product_id, total_sales, avg_sales_per_customer, top_salesperson)
    -> SELECT
    ->     p.product_id,
    ->     SUM(s.sales_amount) AS total_sales,
    ->     AVG(s.sales_amount) AS avg_sales_per_customer,
    ->     (SELECT CONCAT(sp.first_name, ' ', sp.last_name)
    ->      FROM salespersons sp
    ->      JOIN sales s2 ON sp.salesperson_id = s2.salesperson_id
    ->      WHERE s2.product_id = p.product_id
    ->      GROUP BY sp.salesperson_id
    ->      ORDER BY SUM(s2.sales_amount) DESC
    ->      LIMIT 1) AS top_salesperson
    -> FROM products p
    -> JOIN sales s ON p.product_id = s.product_id
    -> GROUP BY p.product_id;
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

-- Displaying analysis result table (temp table)

```
mysql>  SELECT * FROM temp_sales_report;
+------------+-------------+-----------------------+-----------------+
| product_id | total_sales | avg_sales_per_customer | top_salesperson |
+------------+-------------+-----------------------+-----------------+
|          1 |      250.00 |                125.00 | Jane Smith      |
|          2 |      500.00 |                250.00 | Jane Smith      |
+------------+-------------+-----------------------+-----------------+
2 rows in set (0.00 sec)
```

-- Cleanup (Optional)

```
mysql> DROP TEMPORARY TABLE IF EXISTS temp_sales_report;
Query OK, 0 rows affected (0.00 sec)
```