DATA SCI 7030: Database and Analytics

Tozammel Hossain
Nov 10, 2020

Data Science & Analytics
University of Missouri

Subquery

- A query inside another query
- Three types
  - Type I
    - Uncorrelated
  - Type II
    - Correlated
  - Nested table
    - Subquery is termed as a table using aliases

Join vs Subquery

- Subquery
  - Gives structure (more modular)
  - Easier to read
  - The basis for the name Structured Query Language (SQL)
  - Computationally slow

Join vs Subquery

- Join
  - Need to select columns for multiple tables
  - Computationally faster
- Positioning (next slide)
- **Most subqueries can be rewritten as joins, and most joins can be rewritten as subqueries**

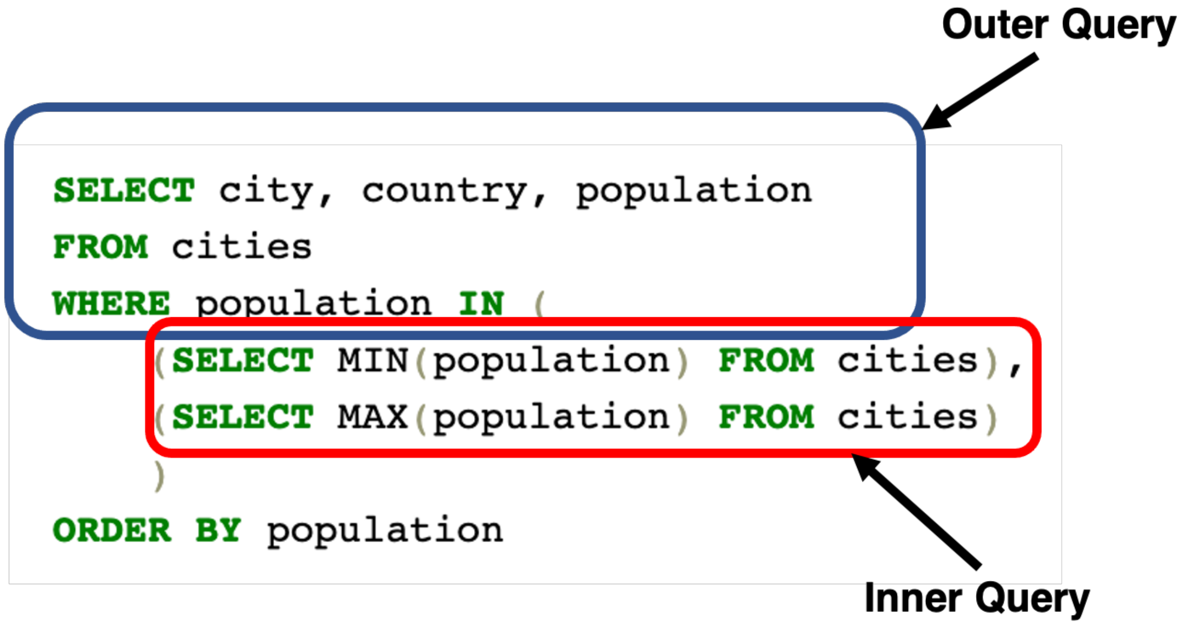
Positioning a Subquery

- SELECT list
- FROM clause
- WHERE clause
  - With the IN or NOT IN operator
  - With comparison operators
  - With the EXISTS or NOT EXISTS operator
  - With the ANY or ALL operator
- HAVING clause
- INSERT

Positioning a Subquery

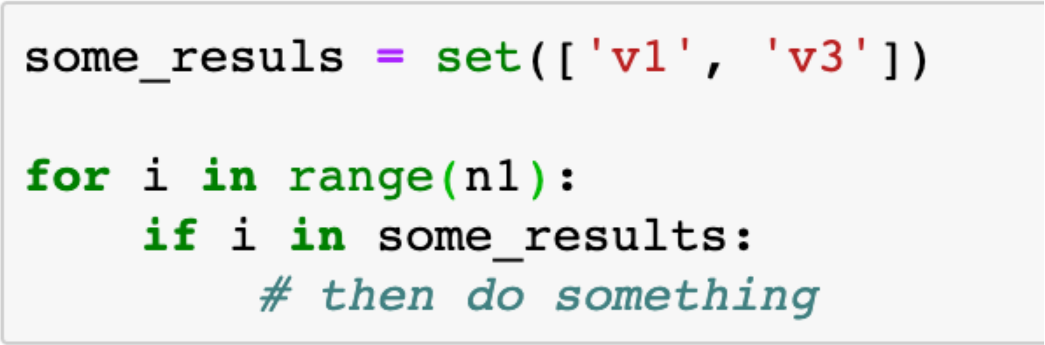
- UPDATE
- DELETE
- **Q. Where does JOIN query go?**

Type I Subquery



```sql
SELECT city, country, population
FROM cities
WHERE population IN (
      (SELECT MIN(population) FROM cities),
      (SELECT MAX(population) FROM cities)
      )
ORDER BY population
```

Similar to Python single for loop



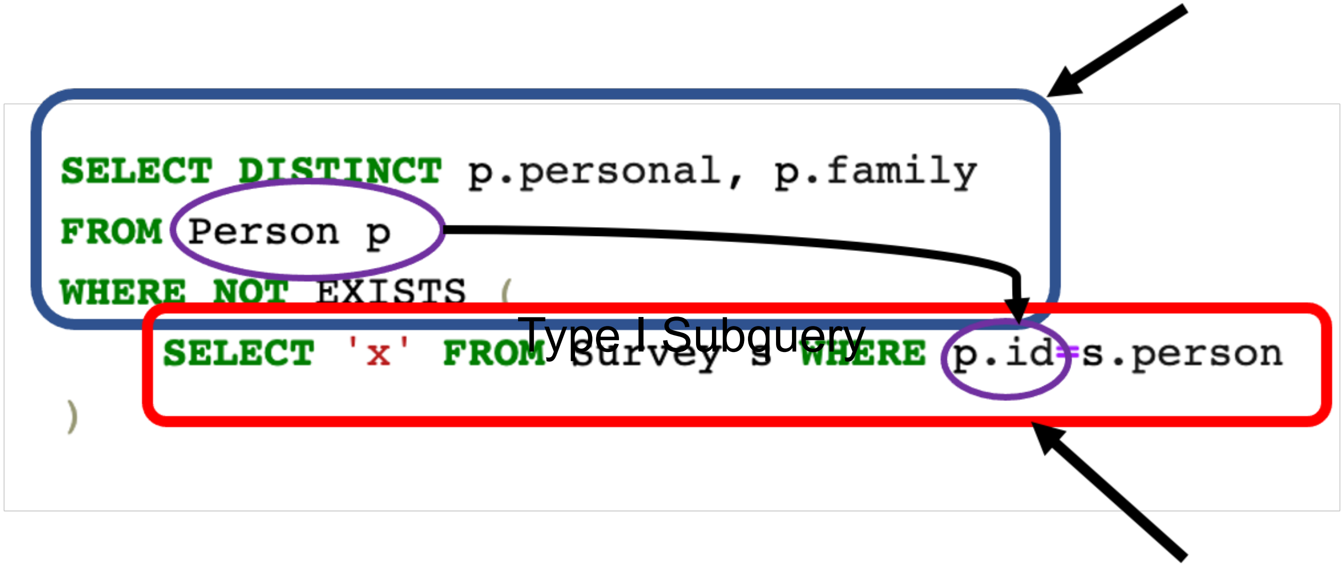
```python
some_resuls = set(['v1', 'v3'])

for i in range(n1):
    if i in some_results:
        # then do something
```

Type I Subquery

- Divide the problem into subtasks
  - Each subtask is a module
  - Develop each task as module
  - Combine the modules

Type II Subquery

Outer Query

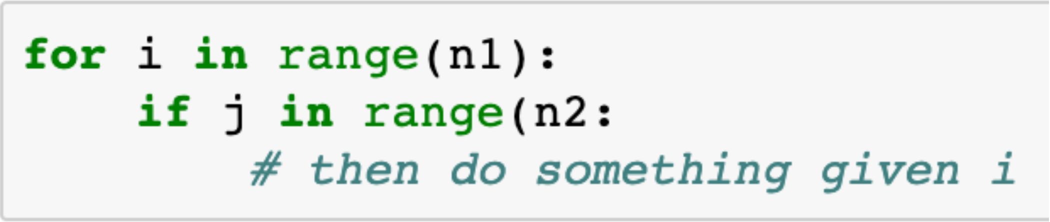


```sql
SELECT DISTINCT p.personal, p.family
FROM Person p
WHERE NOT EXISTS (
    SELECT 'x' FROM Survey s WHERE p.id=s.person
)
```

Type I Subquery

Inner Query

Similar to Python double for loop

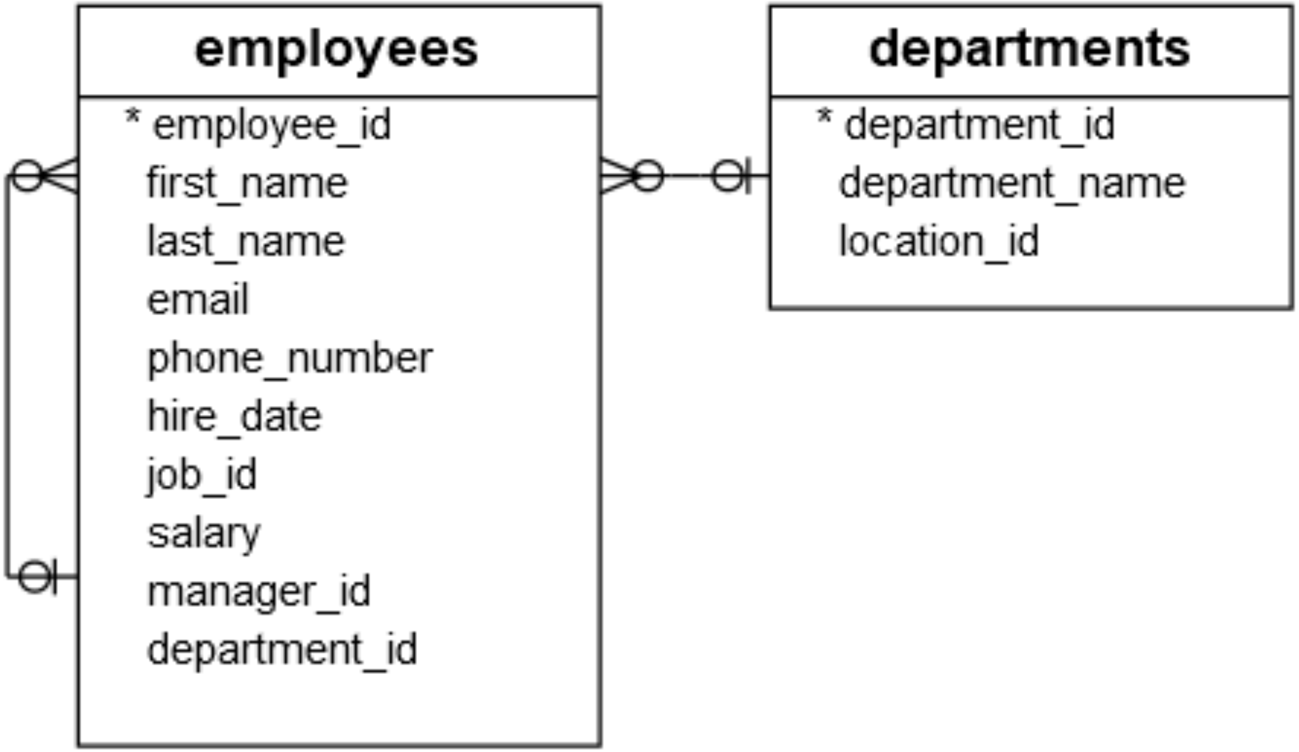


```python
for i in range(n1):
    if j in range(n2:
        # then do something given i
```

Type II Subquery

- Computationally expensive
  - Why?
- Should be avoided
  - Replace with join

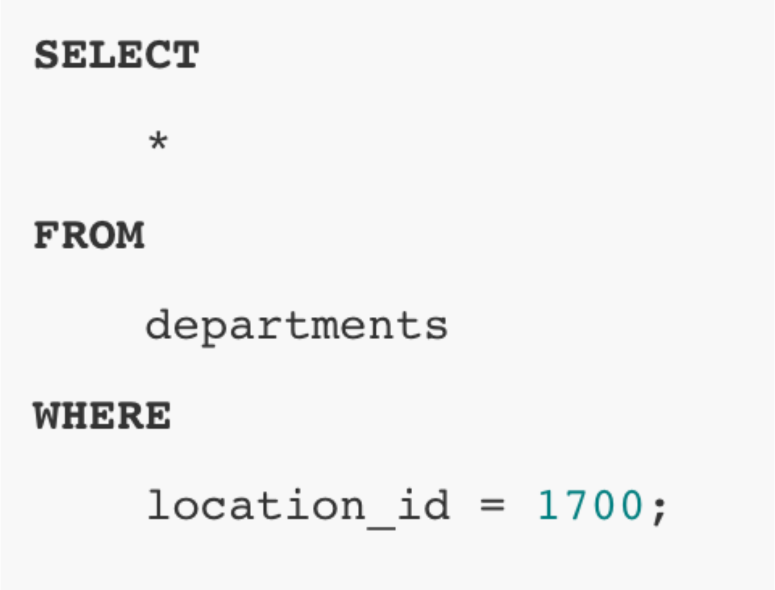
Examples



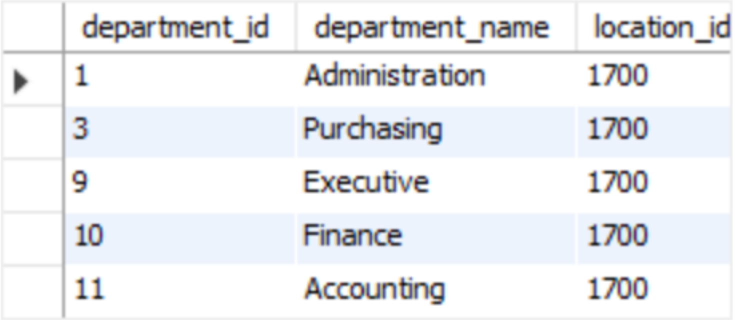
- Two one-to-many relationships
  - Why department_id and manager_id is inside the Employees table?
  - What type of keys are they?

Examples

- **Find all employees who do not locate at the location 1700**
  - First, find all departments located at the location whose id is 1700



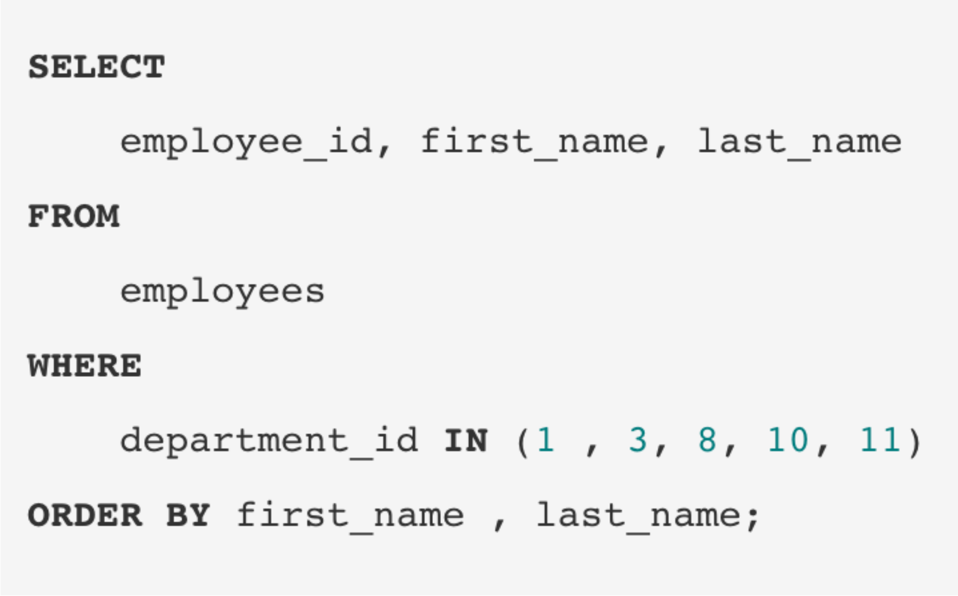
```
SELECT
    *
FROM
    departments
WHERE
    location_id = 1700;
```



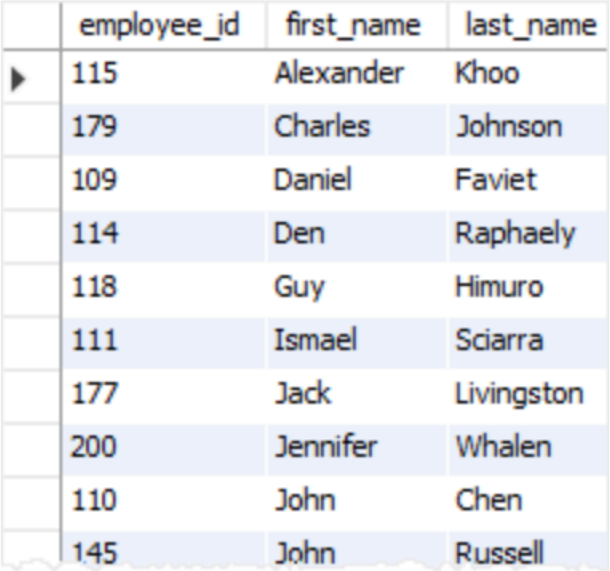
	department_id	department_name	location_id
▶	1	Administration	1700
	3	Purchasing	1700
	9	Executive	1700
	10	Finance	1700
	11	Accounting	1700

Examples

- Second, find all employees that belong to the location 1700 by using the department id list of the previous query



```
SELECT
    employee_id, first_name, last_name
FROM
    employees
WHERE
    department_id IN (1 , 3, 8, 10, 11)
ORDER BY first_name , last_name;
```

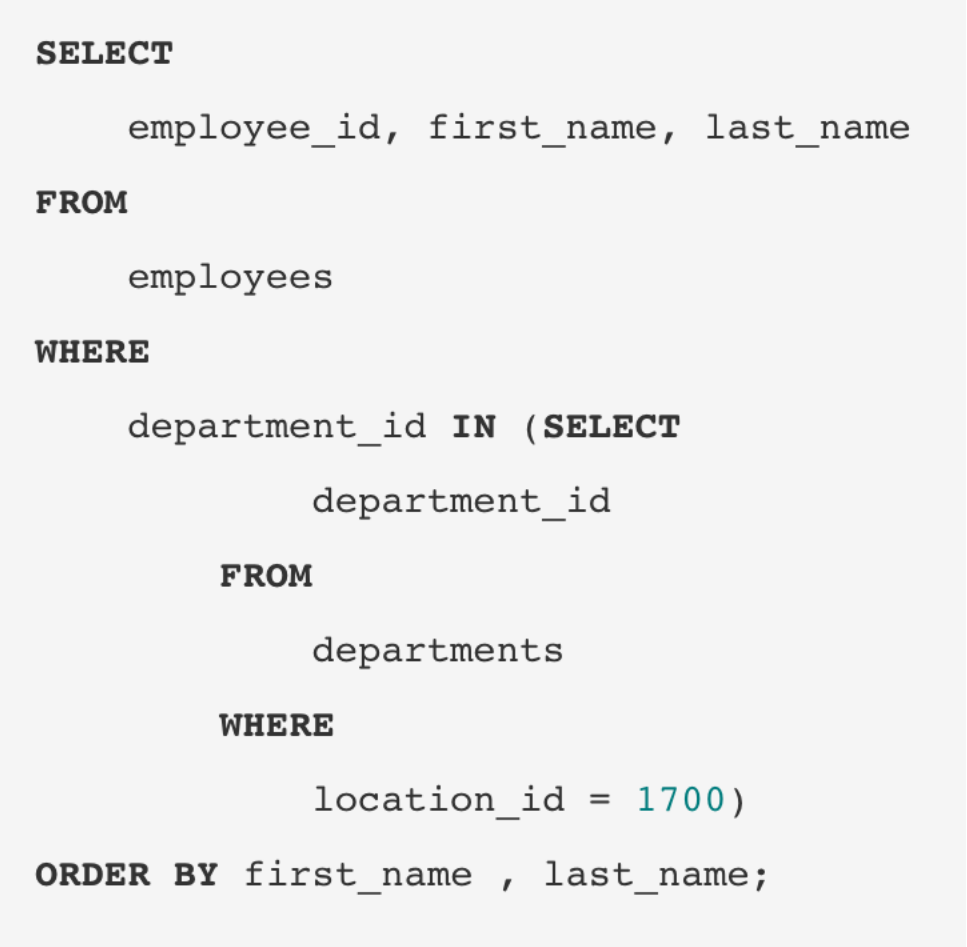


employee_id	first_name	last_name
115	Alexander	Khoo
179	Charles	Johnson
109	Daniel	Faviet
114	Den	Raphaely
118	Guy	Himuro
111	Ismael	Sciarra
177	Jack	Livingston
200	Jennifer	Whalen
110	John	Chen
145	John	Russell

Problems with the naïve solution

- The original question was not referring to any specific departments; it referred to the location 1700
  - Need to transfer the information manually to the second query
    - Think about a huge list of department ids
- Revise the queries whenever you want to find employees who locate in a different location

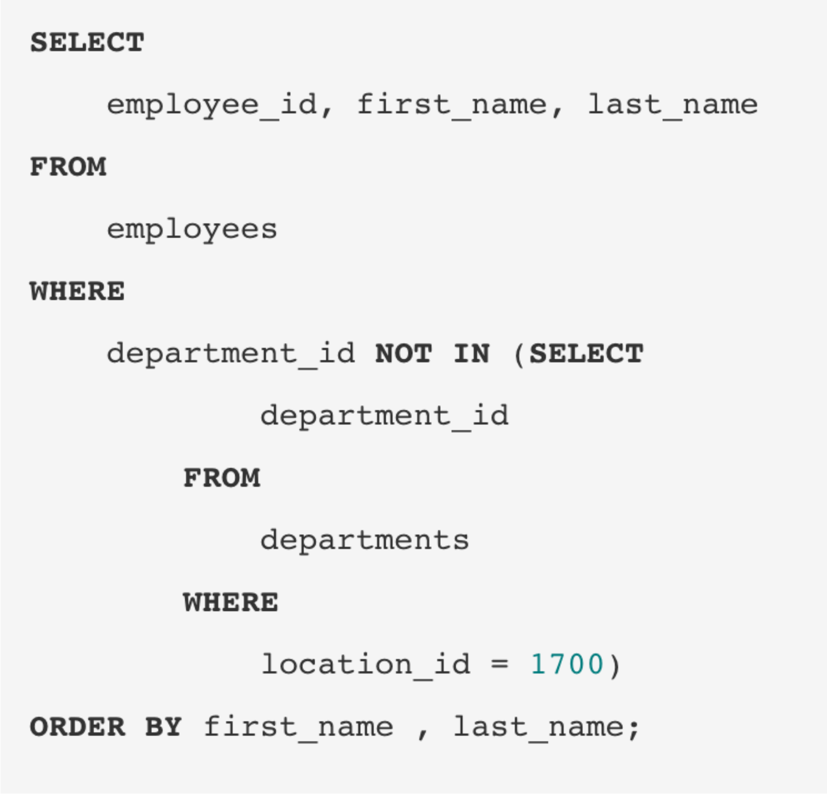
A better solution



```
SELECT
    employee_id, first_name, last_name
FROM
    employees
WHERE
    department_id IN (SELECT
            department_id
        FROM
            departments
        WHERE
            location_id = 1700)
ORDER BY first_name , last_name;
```

Example Type-I

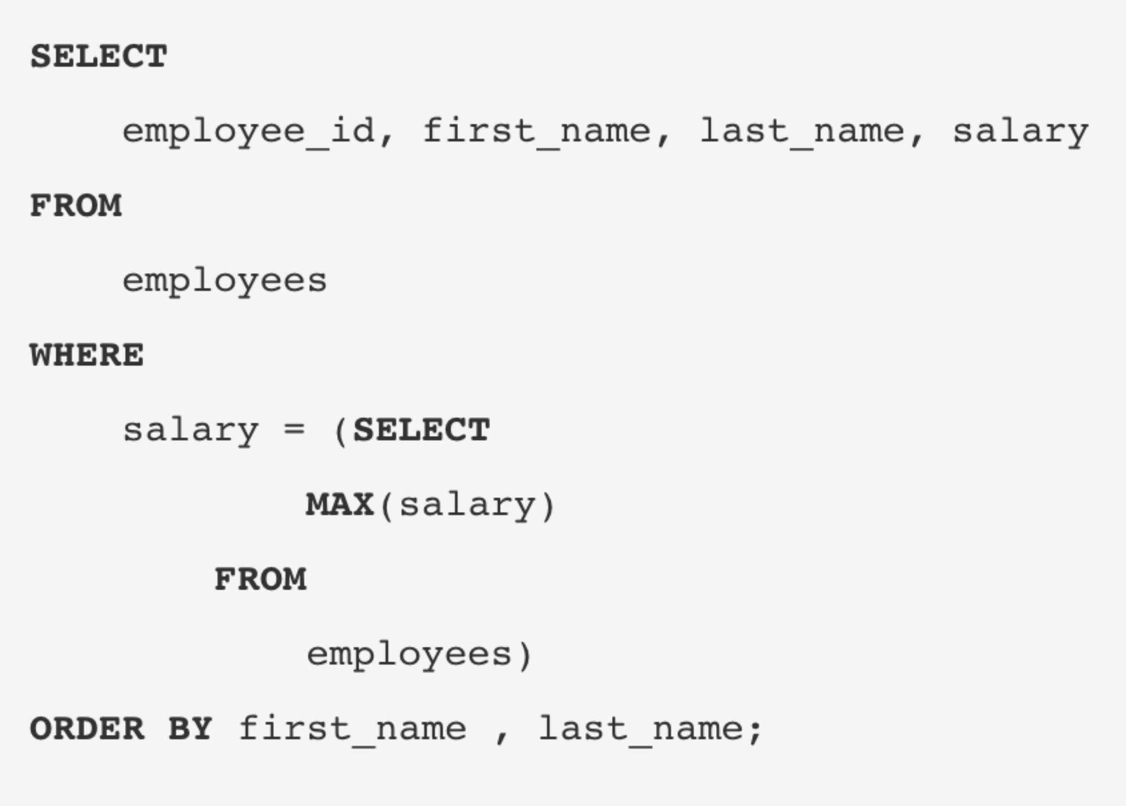
- Find all employees who do not locate at the location 1700



```
SELECT
    employee_id, first_name, last_name
FROM
    employees
WHERE
    department_id NOT IN (SELECT
            department_id
        FROM
            departments
        WHERE
            location_id = 1700)
ORDER BY first_name , last_name;
```

Example Type-I

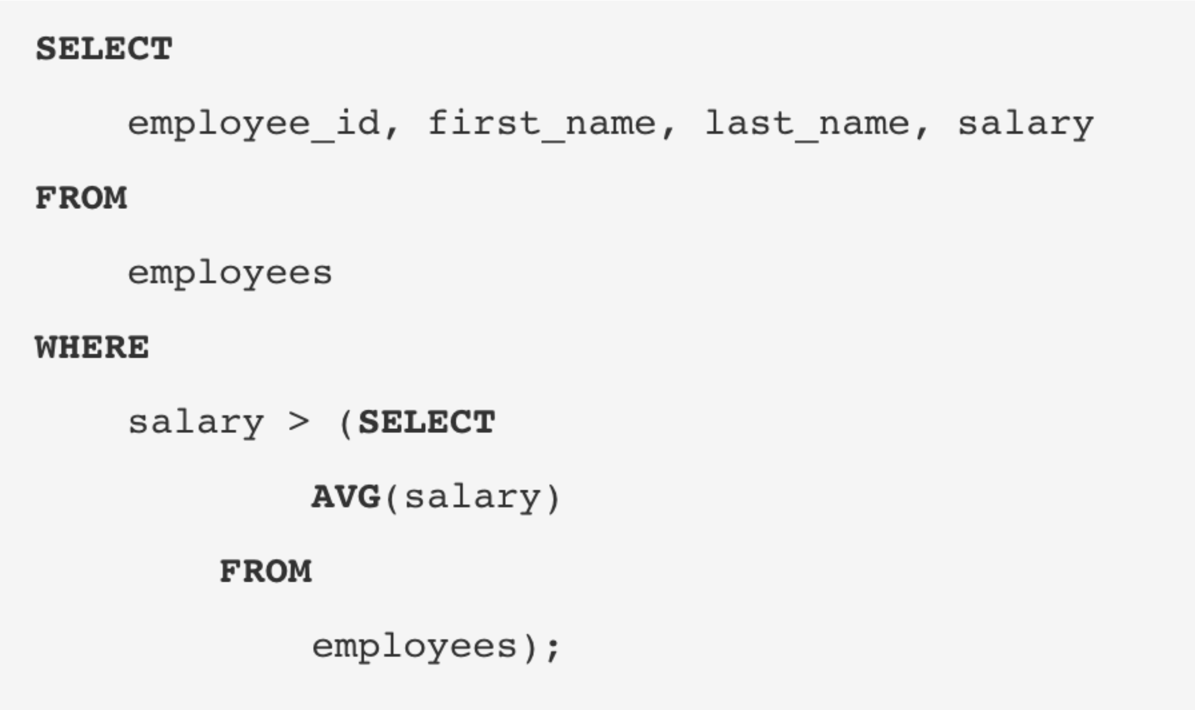
- Find the employees who have the highest salary



```
SELECT
    employee_id, first_name, last_name, salary
FROM
    employees
WHERE
    salary = (SELECT
            MAX(salary)
        FROM
            employees)
ORDER BY first_name , last_name;
```

Example Type-I

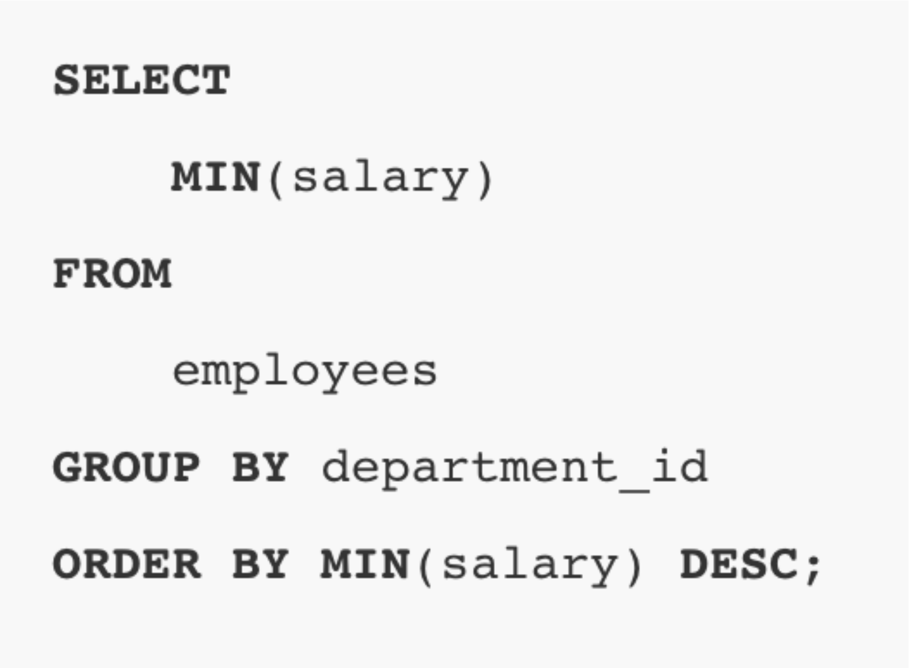
- Find all employees who salaries are greater than the average salary of all employees



```sql
SELECT
    employee_id, first_name, last_name, salary
FROM
    employees
WHERE
    salary > (SELECT
            AVG(salary)
        FROM
            employees);
```

Example-Type I

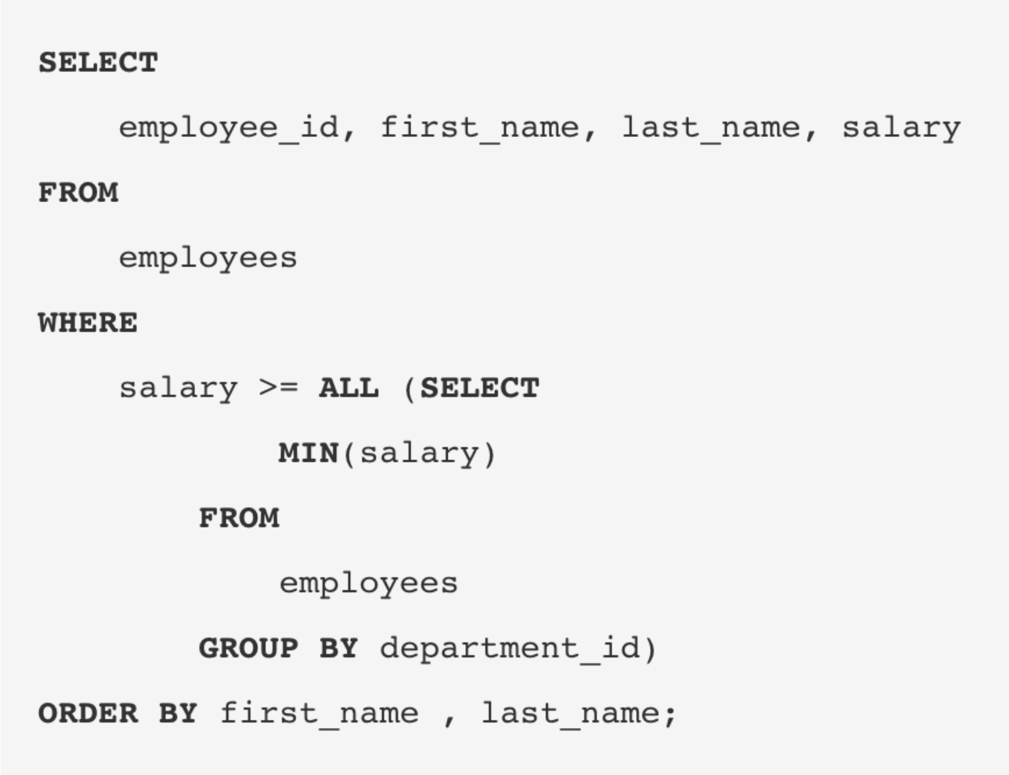
- Find the lowest salary by department



```
SELECT

    MIN(salary)

FROM

    employees

GROUP BY department_id

ORDER BY MIN(salary) DESC;
```

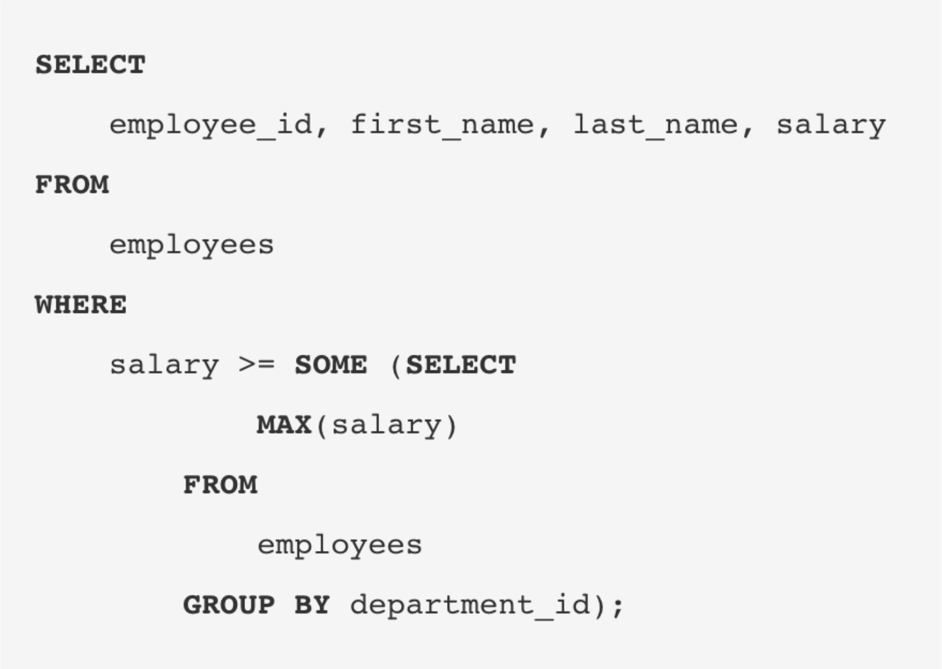
Example-Type I

- Find all employees whose salaries are greater than the lowest salary of every department



```
SELECT
    employee_id, first_name, last_name, salary
FROM
    employees
WHERE
    salary >= ALL (SELECT
            MIN(salary)
        FROM
            employees
        GROUP BY department_id)
ORDER BY first_name , last_name;
```

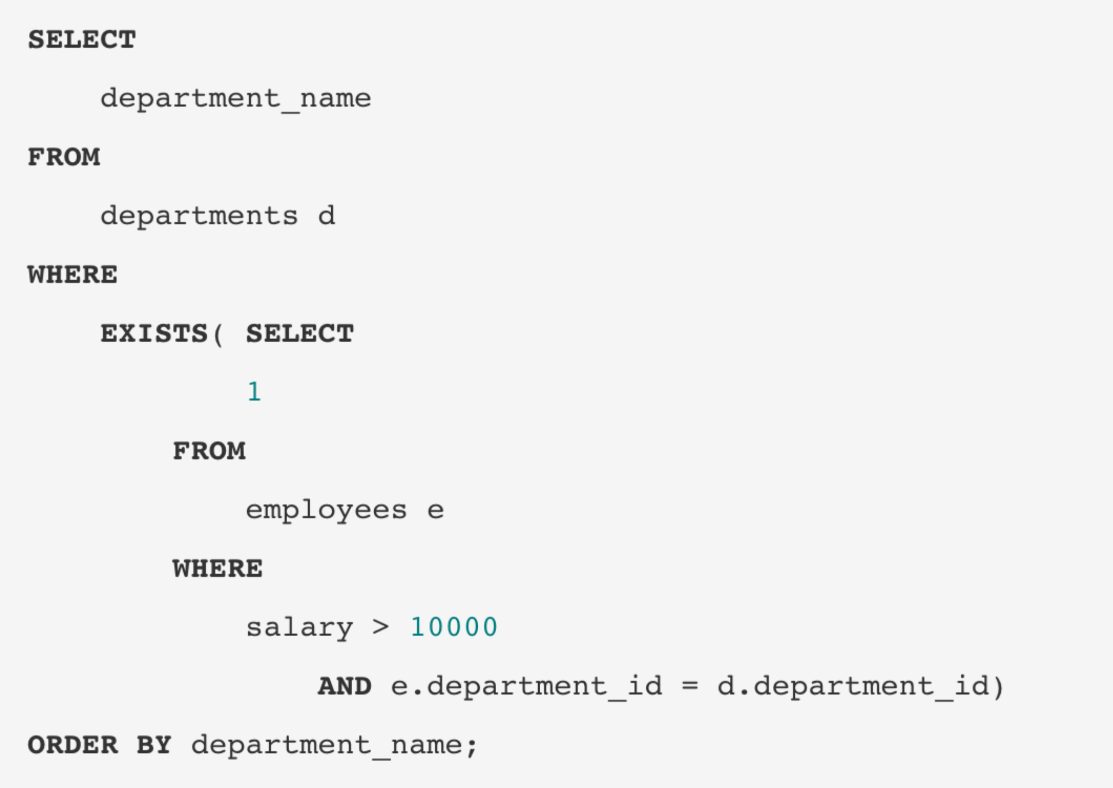
- Find all employees whose salaries are greater than or equal to the highest salary of any department



```
SELECT
    employee_id, first_name, last_name, salary
FROM
    employees
WHERE
    salary >= SOME (SELECT
            MAX(salary)
        FROM
            employees
        GROUP BY department_id);
```

Example-Type II

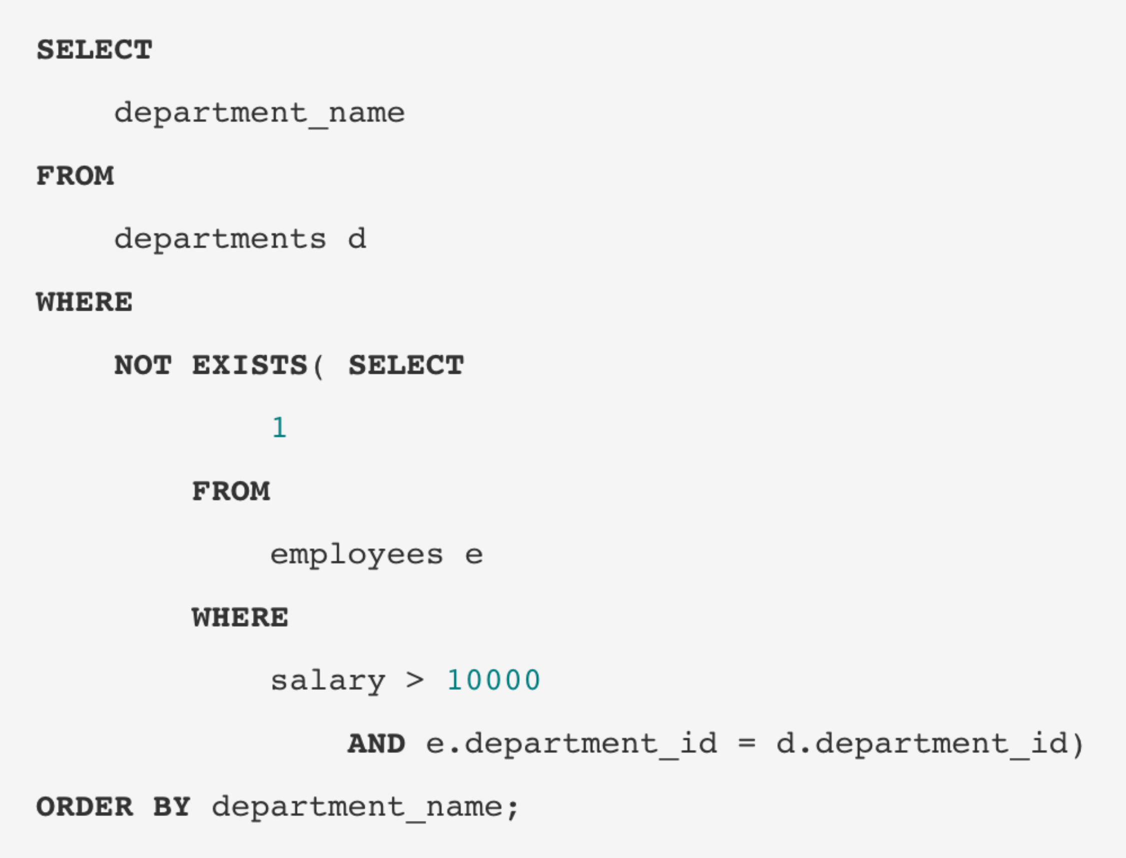
- Find all departments which have at least one employee with the salary is greater than 10,000



```
SELECT
    department_name
FROM
    departments d
WHERE
    EXISTS( SELECT
            1
        FROM
            employees e
        WHERE
            salary > 10000
                AND e.department_id = d.department_id)
ORDER BY department_name;
```

Example-Type II

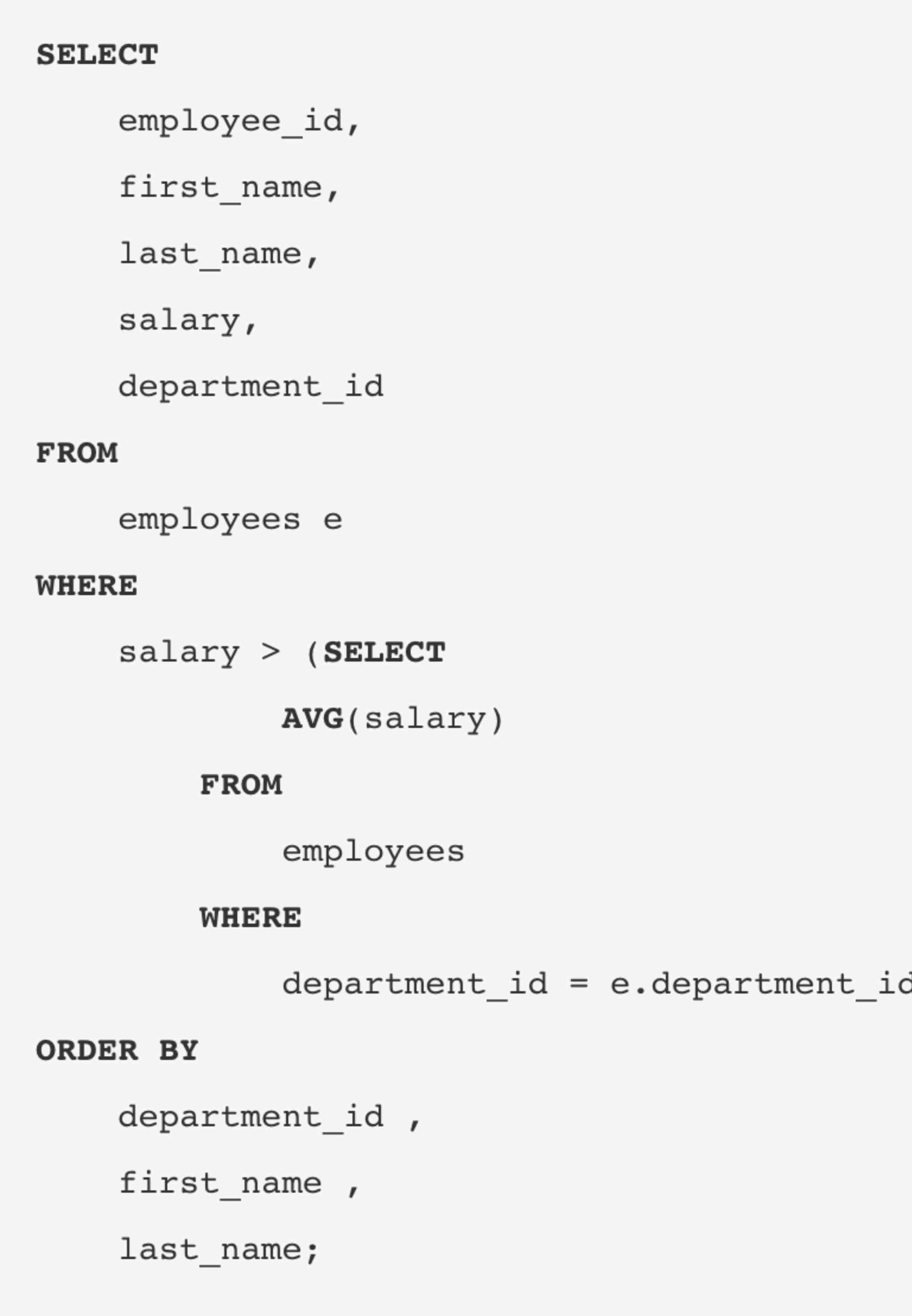
- Find all departments which do not have any employee with the salary is greater than 10,000



```
SELECT
    department_name
FROM
    departments d
WHERE
    NOT EXISTS( SELECT
            1
        FROM
            employees e
        WHERE
            salary > 10000
                AND e.department_id = d.department_id)
ORDER BY department_name;
```

Exampl

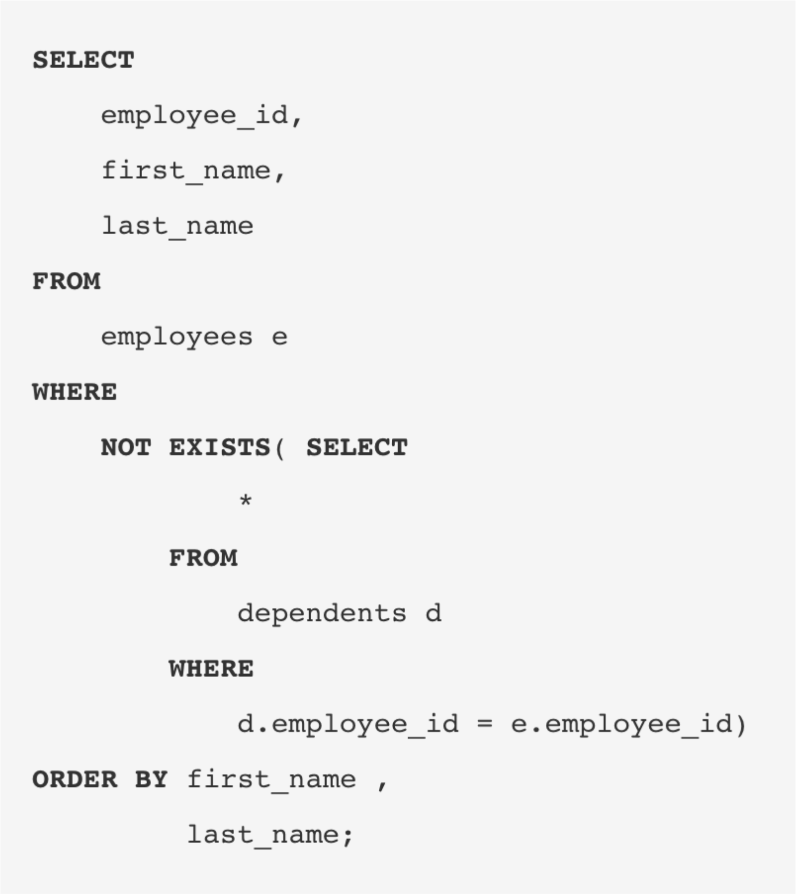
- Find all employees whose salary is higher than the average salary of the employees in their departments



```
SELECT
    employee_id,
    first_name,
    last_name,
    salary,
    department_id
FROM
    employees e
WHERE
    salary > (SELECT
            AVG(salary)
        FROM
            employees
        WHERE
            department_id = e.department_id
ORDER BY
    department_id ,
    first_name ,
    last_name;
```

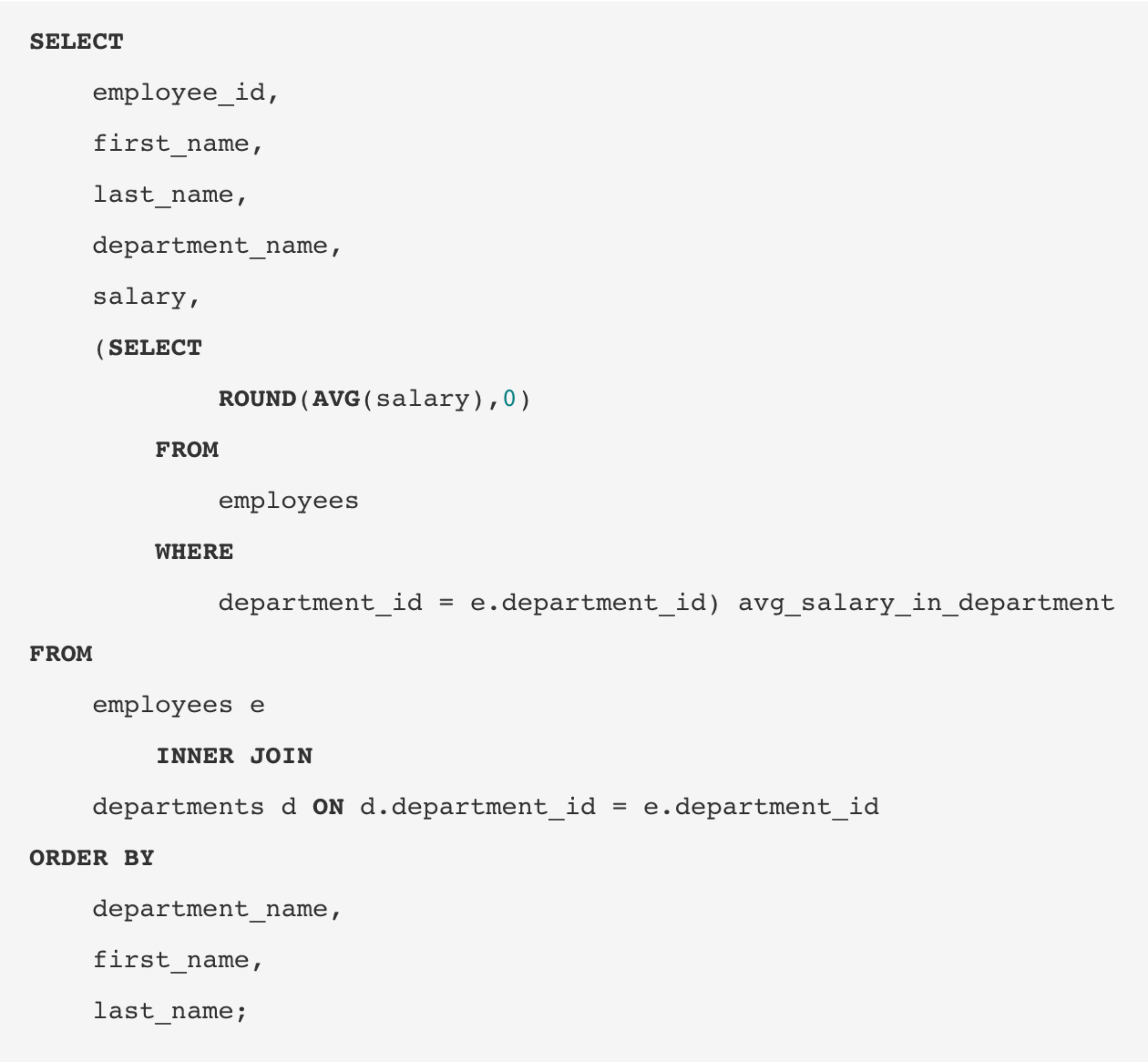
Example-Type II

- Find all employees who have no dependents



```
SELECT
    employee_id,
    first_name,
    last_name
FROM
    employees e
WHERE
    NOT EXISTS( SELECT
            *
        FROM
            dependents d
        WHERE
            d.employee_id = e.employee_id)
ORDER BY first_name ,
         last_name;
```

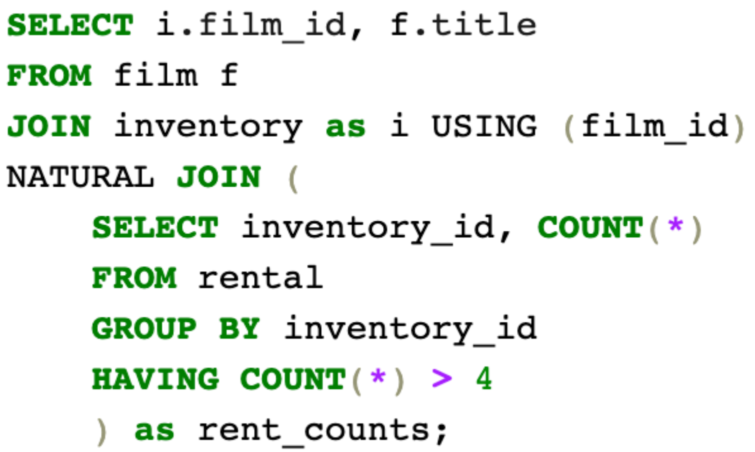
Subquery in select statement



```
SELECT
    employee_id,
    first_name,
    last_name,
    department_name,
    salary,
    (SELECT
            ROUND(AVG(salary),0)
        FROM
            employees
        WHERE
            department_id = e.department_id) avg_salary_in_department
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
ORDER BY
    department_name,
    first_name,
    last_name;
```

Nested Table Expression

- Whenever we are using the <table_A> JOIN <table_B> syntax, we are creating a table expression
- NTE: a table is an output of subquery
- Use in the FROM clause



```
SELECT i.film_id, f.title
FROM film f
JOIN inventory as i USING (film_id)
NATURAL JOIN (
    SELECT inventory_id, COUNT(*)
    FROM rental
    GROUP BY inventory_id
    HAVING COUNT(*) > 4
    ) as rent_counts;
```

Questions