Database Lab

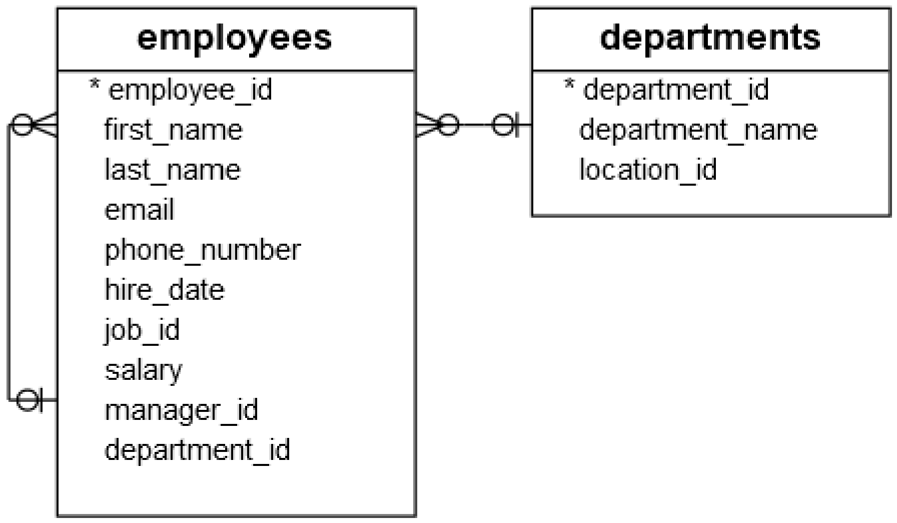
CSE 3104

Session 07

Summary: in this tutorial, you will learn about the SQL subquery and how to use the subqueries to form flexible SQL statements.

SQL subquery basic

Consider the following **employees** and **departments** tables. Run the script **creation.sql** to create tables and **insertion.sql** to insert some data into the tables:

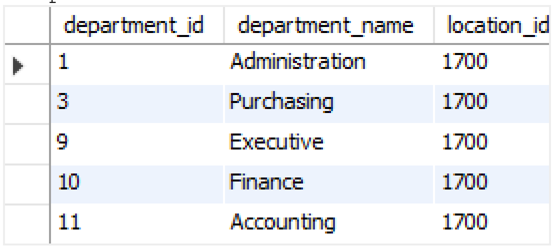


Suppose you have to find all employees who locate in the location with the id 1700. You might come up with the following solution.

First, find all departments located at the location whose id is 1700:

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

Output:

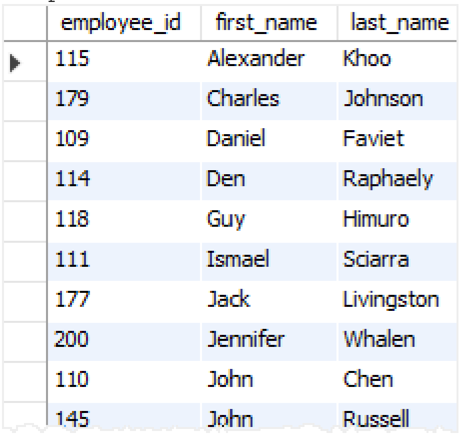


department_id	department_name	location_id
1	Administration	1700
3	Purchasing	1700
9	Executive	1700
10	Finance	1700
11	Accounting	1700

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;
```

Output:



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

This solution has two problems. To start with, you have looked at the departments table to check which department belongs to the location 1700. However, the original question was not referring to any specific departments; it referred to the location 1700.

Because of the small data volume, you can get a list of department easily. However, in the real system with high volume data, it might be problematic.

Another problem was that you have to revise the queries whenever you want to find employees who locate in a different location.

A much better solution to this problem is to use a subquery. By definition, a subquery is a query nested inside another query such as SELECT, INSERT, UPDATE, or DELETE statement. In this session, we are focusing on the subquery used with the SELECT statement.

In this example, you can rewrite combine the two queries above as follows:

```
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;
```

The query placed within the parentheses is called a subquery. It is also known as an inner query or inner select. The query that contains the subquery is called an outer query or an outer select.

To execute the query, first, the database system has to execute the subquery and substitute the subquery between the parentheses with its result – a number of department id located at the location 1700 – and then executes the outer query.

You can use a subquery in many places such as:

- With the IN or NOT IN operator
- With comparison operators
- With the EXISTS or NOT EXISTS operator
- With the ANY or ALL operator
- In the FROM clause
- In the SELECT clause

SQL subquery examples

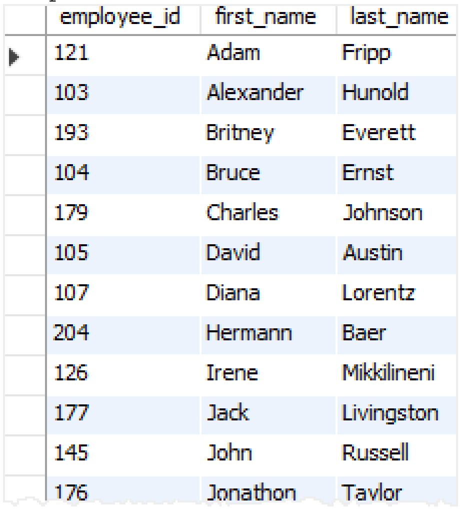
Let's take some examples of using the subqueries to understand how they work.

SQL subquery with the IN or NOT IN operator

In the previous example, you have seen how the subquery was used with the IN operator. The following example uses a subquery with the NOT IN operator to 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;
```

Output:



	employee_id	first_name	last_name
▶	121	Adam	Fripp
	103	Alexander	Hunold
	193	Britney	Everett
	104	Bruce	Ernst
	179	Charles	Johnson
	105	David	Austin
	107	Diana	Lorentz
	204	Hermann	Baer
	126	Irene	Mikkilineni
	177	Jack	Livingston
	145	John	Russell
	176	Jonathon	Taylor

SQL subquery with the comparison operator

The following syntax illustrates how a subquery is used with a comparison operator:

```
comparison_operator (subquery)
```

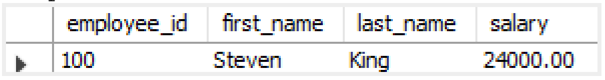
where the comparison operator is one of these operators:

- Equal (=)
- Greater than (>)
- Less than (<)
- Greater than or equal (>=)
- Less than or equal (<=)
- Not equal (!=) or (<>)

The following example finds 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;
```

Output:



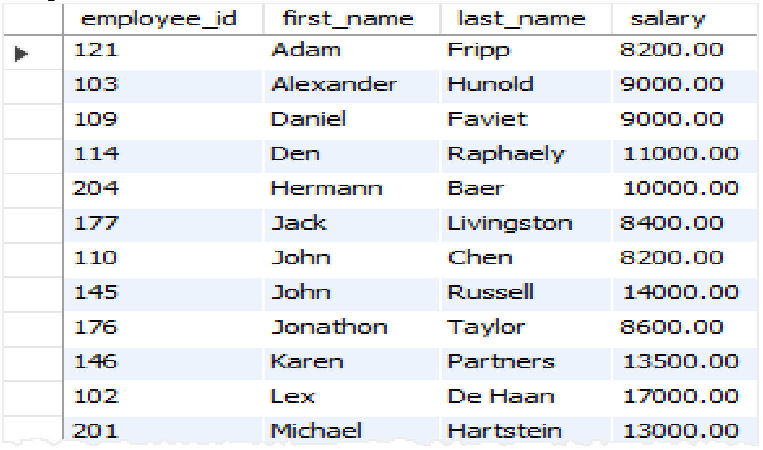
	employee_id	first_name	last_name	salary
▶	100	Steven	King	24000.00

In this example, the subquery returns the highest salary of all employees and the outer query finds the employees whose salary is equal to the highest one.

The following statement finds all employees who salaries are greater than the average salary of all employees:

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

Output:



	employee_id	first_name	last_name	salary
▶	121	Adam	Fripp	8200.00
	103	Alexander	Hunold	9000.00
	109	Daniel	Faviet	9000.00
	114	Den	Raphaely	11000.00
	204	Hermann	Baer	10000.00
	177	Jack	Livingston	8400.00
	110	John	Chen	8200.00
	145	John	Russell	14000.00
	176	Jonathon	Taylor	8600.00
	146	Karen	Partners	13500.00
	102	Lex	De Haan	17000.00
	201	Michael	Hartstein	13000.00

In this example, first, the subquery returns the average salary of all employees. Then, the outer query uses the greater than operator to find all employees whose salaries are greater than the average.

SQL subquery with the EXISTS or NOT EXISTS operator

The EXISTS operator checks for the existence of rows returned from the subquery. It returns true if the subquery contains any rows. Otherwise, it returns false.

The syntax of the EXISTS operator is as follows:

```
EXISTS (subquery )
```

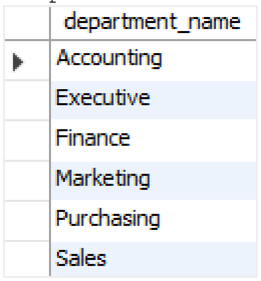
The NOT EXISTS operator is opposite to the EXISTS operator.

```
NOT EXISTS (subquery)
```

The following example finds 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;
```

Output:

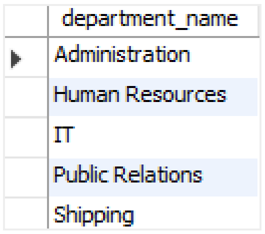


```
department_name
Accounting
Executive
Finance
Marketing
Purchasing
Sales
```

Similarly, the following statement finds all departments that do not have any employee with the salary 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;
```

Output:



```
department_name
Administration
Human Resources
IT
Public Relations
Shipping
```

SQL subquery with the ALL operator

The syntax of the subquery when it is used with the ALL operator is as follows:

```
comparison_operator ALL (subquery)
```

The following condition evaluates to true if x is greater than every value returned by the subquery.

```
x > ALL (subquery)
```

For example, suppose the subquery returns three value one, two, and three. The following condition evaluates to true if x is greater than 3.

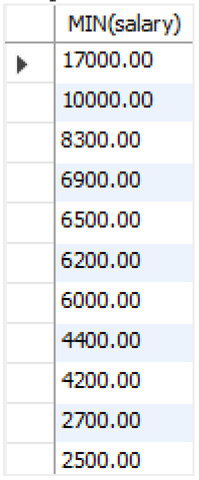
```
x > ALL (1,2,3)
```

The following query uses the GROUP BY clause and MIN() function to find the lowest salary by department:

```
SELECT
  MIN(salary)
FROM
  employees
```

```
GROUP BY department_id
ORDER BY MIN(salary) DESC;
```

Output:

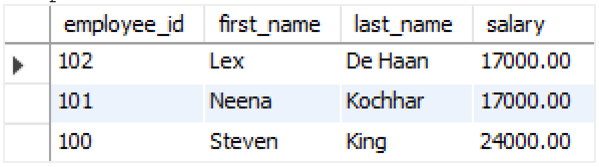


MIN(salary)
17000.00
10000.00
8300.00
6900.00
6500.00
6200.00
6000.00
4400.00
4200.00
2700.00
2500.00

The following example finds 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;
```

Output:



employee_id	first_name	last_name	salary
102	Lex	De Haan	17000.00
101	Neena	Kochhar	17000.00
100	Steven	King	24000.00

SQL subquery with the ANY operator

The following shows the syntax of a subquery with the ANY operator:

```
comparison_operator ANY (subquery)
```

For example, the following condition evaluates to true if x is greater than any value returned by the subquery. So the condition x > SOME (1,2,3) evaluates to true if x is greater than 1.

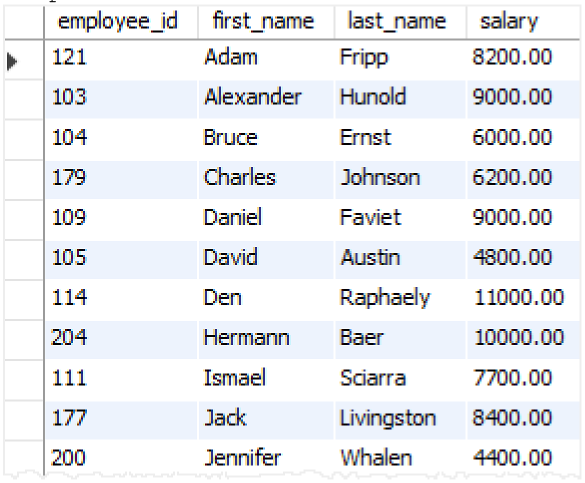
```
x > ANY (subquery)
```

Note that the SOME operator is a synonym for the ANY operator so you can use them interchangeably.

The following query finds all employees whose salaries are greater than or equal to the highest salary of every department.

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

Output:



employee_id	first_name	last_name	salary
121	Adam	Fripp	8200.00
103	Alexander	Hunold	9000.00
104	Bruce	Ernst	6000.00
179	Charles	Johnson	6200.00
109	Daniel	Faviet	9000.00
105	David	Austin	4800.00
114	Den	Raphaely	11000.00
204	Hermann	Baer	10000.00
111	Ismael	Sciarra	7700.00
177	Jack	Livingston	8400.00
200	Jennifer	Whalen	4400.00

In this example, the subquery finds the highest salary of employees in each department. The outer query looks at these values and determines which employee's salaries are greater than or equal to any highest salary by department.

SQL subquery in the FROM clause

You can use a subquery in the FROM clause of the SELECT statement as follows:

```
SELECT
    *
FROM
```

```
  (subquery) AS table_name:
```

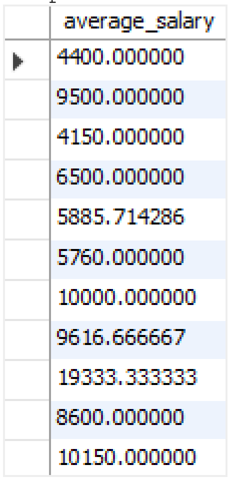
In this syntax, the table alias is mandatory because all tables in the FROM clause must have a name.

Note that the subquery specified in the FROM clause is called a derived table.

The following statement returns the average salary of every department:

```
SELECT
  AVG(salary) average_salary
FROM
  employees
GROUP BY department_id;
```

Output:

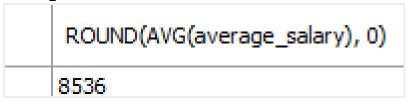


```
average_salary
4400.000000
9500.000000
4150.000000
6500.000000
5885.714286
5760.000000
10000.000000
9616.666667
19333.333333
8600.000000
10150.000000
```

You can use this query as a subquery in the FROM clause to calculate the average of average salary of departments as follows:

```
SELECT
  ROUND(AVG(average_salary), 0)
FROM
  (SELECT
    AVG(salary) average_salary
  FROM
    employees
  GROUP BY department_id) department_salary;
```

Output:



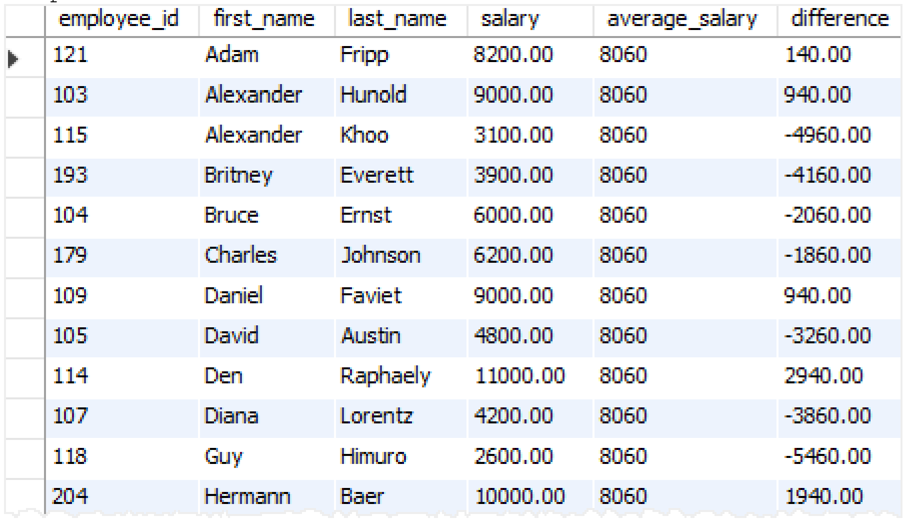
```
ROUND(AVG(average_salary), 0)
8536
```

SQL Subquery in the SELECT clause

A subquery can be used anywhere an expression can be used in the SELECT clause. The following example finds the salaries of all employees, their average salary, and the difference between the salary of each employee and the average salary.

```
SELECT
   employee_id,
   first_name,
   last_name,
   salary,
   (SELECT
         ROUND(AVG(salary), 0)
      FROM
         employees) average_salary,
   salary - (SELECT
         ROUND(AVG(salary), 0)
      FROM
         employees) difference
FROM
   employees
ORDER BY first_name , last_name;
```

Output:



employee_id	first_name	last_name	salary	average_salary	difference
121	Adam	Fripp	8200.00	8060	140.00
103	Alexander	Hunold	9000.00	8060	940.00
115	Alexander	Khoo	3100.00	8060	-4960.00
193	Britney	Everett	3900.00	8060	-4160.00
104	Bruce	Ernst	6000.00	8060	-2060.00
179	Charles	Johnson	6200.00	8060	-1860.00
109	Daniel	Faviet	9000.00	8060	940.00
105	David	Austin	4800.00	8060	-3260.00
114	Den	Raphaely	11000.00	8060	2940.00
107	Diana	Lorentz	4200.00	8060	-3860.00
118	Guy	Himuro	2600.00	8060	-5460.00
204	Hermann	Baer	10000.00	8060	1940.00

Now you should understand what an SQL subquery is and how to use subqueries to form flexible SQL statements.