6 Retrieving Data by Using Subqueries

Objectives

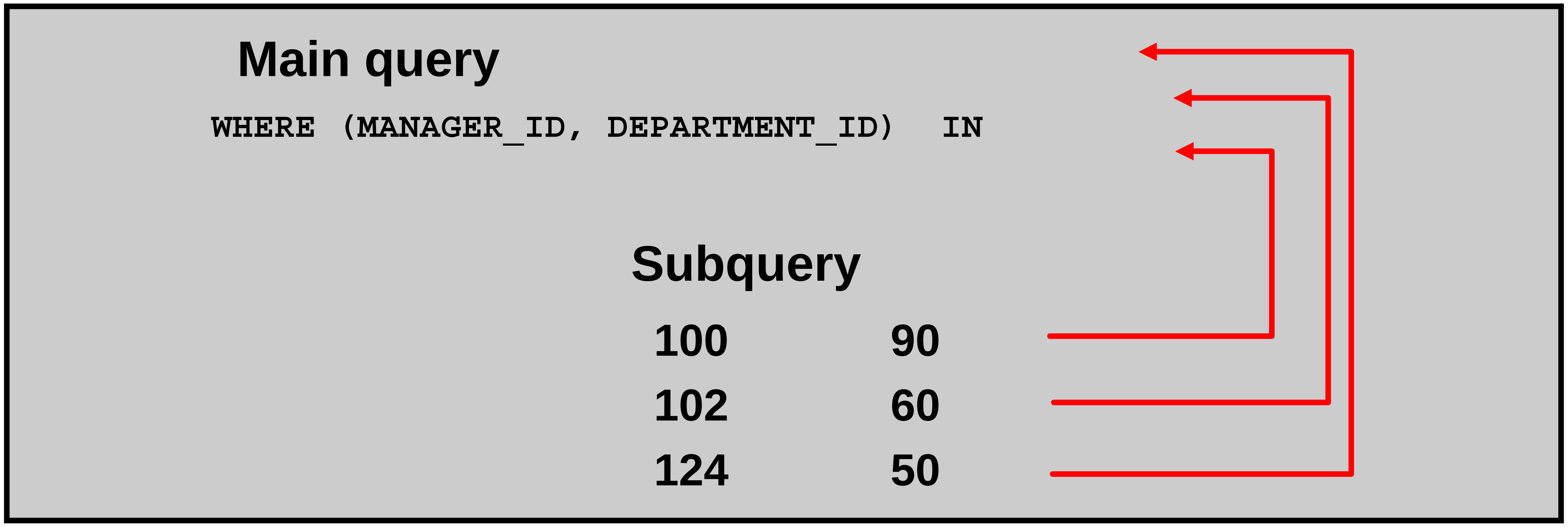
After completing this lesson, you should be able to do the following:

- Write a multiple-column subquery
- Use scalar subqueries in SQL
- Solve problems with correlated subqueries
- Update and delete rows by using correlated subqueries
- Use the `EXISTS` and `NOT EXISTS` operators
- Use the `WITH` clause

Lesson Agenda

- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

Multiple-Column Subqueries



Each row of the main query is compared to values from a multiple-row and multiple-column subquery.

Column Comparisons

Multiple-column comparisons involving subqueries can be:

- Nonpairwise comparisons
- Pairwise comparisons

Pairwise Comparison Subquery

Display the details of the employees who are managed by the same manager and work in the same department as employees with the first name of “John.”

```
SELECT  employee_id, manager_id, department_id
FROM    empl_demo
WHERE  (manager_id, department_id) IN
                          (SELECT manager_id, department_id
                           FROM empl_demo
                           WHERE first_name = 'John')
AND first_name <> 'John';
```

Nonpairwise Comparison Subquery

Display the details of the employees who are managed by the same manager as the employees with the first name of “John” and work in the same department as the employees with the first name of “John.”

```
SELECT  employee_id, manager_id, department_id
FROM    empl_demo
WHERE   manager_id IN
                  (SELECT manager_id
                   FROM empl_demo
                   WHERE first_name = 'John')
AND department_id IN
                 (SELECT department_id
                  FROM empl_demo
                  WHERE first_name = 'John')
AND first_name <> 'John';
```

Lesson Agenda

- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

Scalar Subquery Expressions

- A scalar subquery expression is a subquery that returns exactly one column value from one row.
- Scalar subqueries can be used in:
  - The condition and expression part of `DECODE` and `CASE`
  - All clauses of `SELECT` except `GROUP BY`
  - The `SET` clause and `WHERE` clause of an `UPDATE` statement

Scalar Subqueries: Examples

- Scalar subqueries in `CASE` expressions:

```
SELECT employee_id, last_name,
       (CASE
                                    20
        WHEN department_id =
                  (SELECT department_id
                   FROM departments
                      WHERE location_id = 1800)
         THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```

- Scalar subqueries in the `ORDER BY` clause:

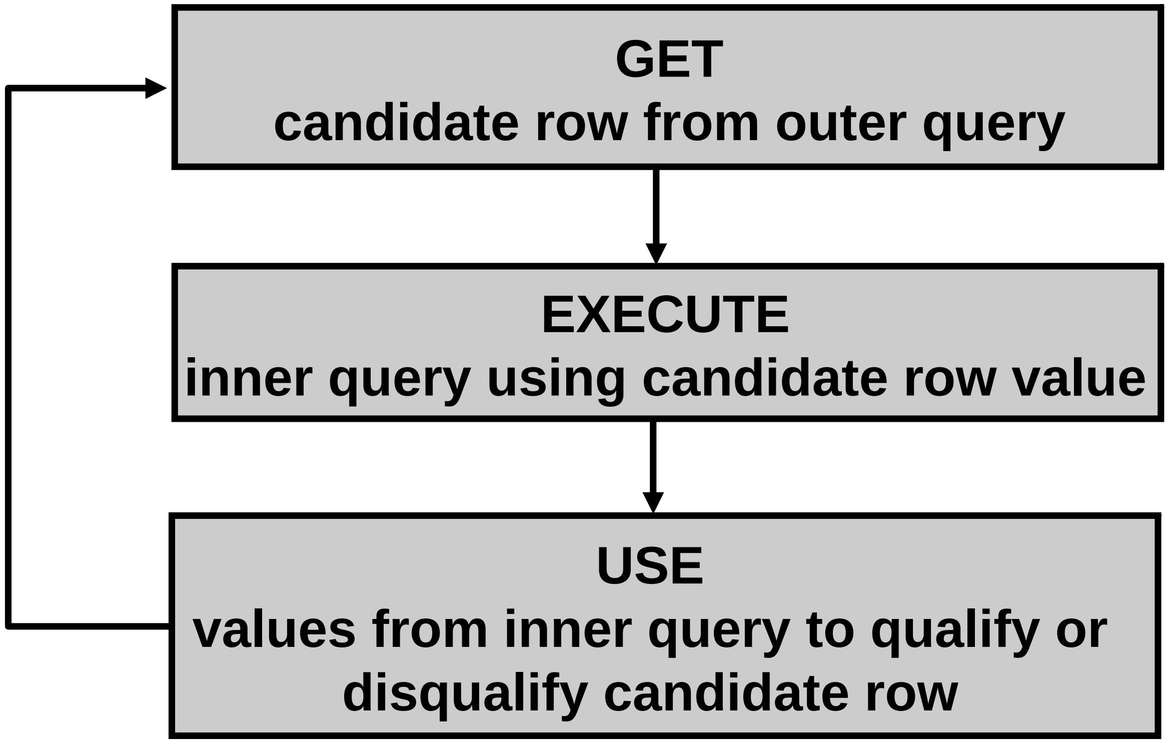
```
SELECT   employee_id, last_name
FROM     employees e
ORDER BY (SELECT department_name
           FROM departments d
            WHERE e.department_id = d.department_id);
```

Lesson Agenda

- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

Correlated Subqueries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.


GET
candidate row from outer query
EXECUTE
inner query using candidate row value
USE
values from inner query to qualify or disqualify candidate row

Correlated Subqueries

The subquery references a column from a table in the parent query.

```
SELECT column1, column2, ...
FROM   table1   Outer_table
WHERE  column1 operator
                      (SELECT   column1, column2
                       FROM     table2
                       WHERE    expr1 =
                            Outer_table.expr2);
```

Using Correlated Subqueries

Find all employees who earn more than the average salary in their department.

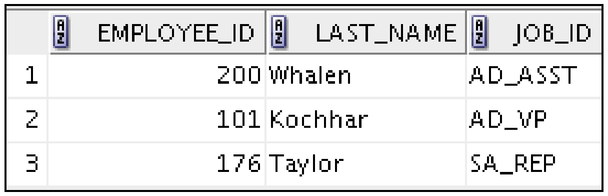
```
SELECT last_name, salary, department_id
FROM   employees outer_table
WHERE  salary >
                (SELECT AVG(salary)
                  FROM   employees inner_table
                  WHERE inner_table.department_id =
                  outer_table.department_id);
```

Each time a row from the outer query is processed, the inner query is evaluated.

Using Correlated Subqueries

Display details of those employees who have changed jobs at least twice.

```
SELECT e.employee_id, last_name,e.job_id
FROM   employees e
WHERE  2 <= (SELECT COUNT(*)
             FROM   job_history
             WHERE  employee_id = e.employee_id);
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	200	Whalen	AD_ASST
2	101	Kochhar	AD_VP
3	176	Taylor	SA_REP

Lesson Agenda

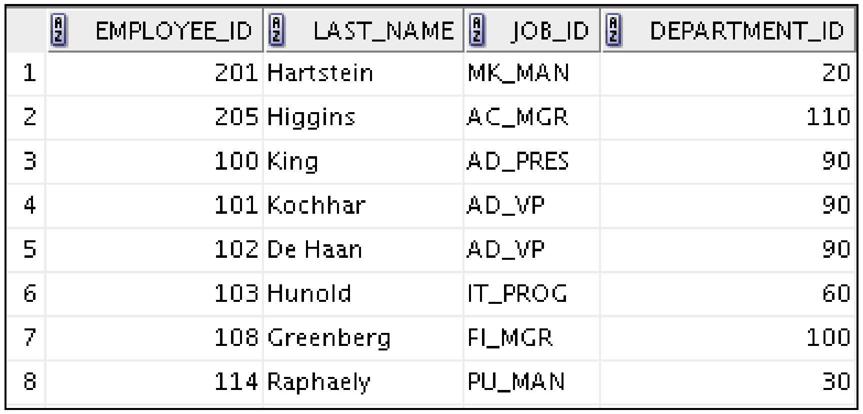
- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

Using the `EXISTS` Operator

- The `EXISTS` operator tests for existence of rows in the results set of the subquery.
- If a subquery row value is found:
  - The search does not continue in the inner query
  - The condition is flagged `TRUE`
- If a subquery row value is not found:
  - The condition is flagged `FALSE`
  - The search continues in the inner query

Using the `EXISTS` Operator

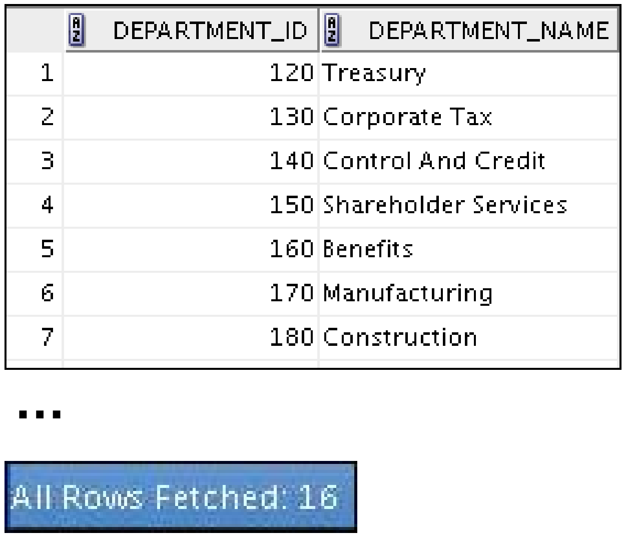
```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE   EXISTS ( SELECT 'X'
                 FROM   employees
                 WHERE  manager_id =
                        outer.employee_id);
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	201	Hartstein	MK_MAN	20
2	205	Higgins	AC_MGR	110
3	100	King	AD_PRES	90
4	101	Kochhar	AD_VP	90
5	102	De Haan	AD_VP	90
6	103	Hunold	IT_PROG	60
7	108	Greenberg	FI_MGR	100
8	114	Raphaely	PU_MAN	30

Find All Departments That Do Not Have Any Employees

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM   employees
                  WHERE  department_id = d.department_id);
```



	DEPARTMENT_ID	DEPARTMENT_NAME
1	120	Treasury
2	130	Corporate Tax
3	140	Control And Credit
4	150	Shareholder Services
5	160	Benefits
6	170	Manufacturing
7	180	Construction

…

All Rows Fetched: 16

Correlated `UPDATE`

Use a correlated subquery to update rows in one table based on rows from another table.

```
UPDATE table1 alias1
SET    column = (SELECT expression
                 FROM   table2 alias2
                 WHERE  alias1.column =
                        alias2.column);
```

Using Correlated `UPDATE`

- Denormalize the `EMPL6` table by adding a column to store the department name.
- Populate the table by using a correlated update.

```
ALTER TABLE empl6
ADD(department_name VARCHAR2(25));
```

```
UPDATE empl6 e
SET    department_name =
              (SELECT department_name
               FROM   departments d
               WHERE  e.department_id = d.department_id);
```

Correlated `DELETE`

Use a correlated subquery to delete rows in one table based on rows from another table.

```
DELETE FROM table1 alias1
WHERE  column operator
               (SELECT expression
                FROM   table2 alias2
                WHERE  alias1.column = alias2.column);
```

Using Correlated `DELETE`

Use a correlated subquery to delete only those rows from the `EMPL6` table that also exist in the `EMP_HISTORY` table.

```
DELETE FROM empl6 E
WHERE employee_id =
           (SELECT employee_id
            FROM   emp_history
            WHERE  employee_id = E.employee_id);
```

Lesson Agenda

- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

`WITH` Clause

- Using the `WITH` clause, you can use the same query block in a `SELECT` statement when it occurs more than once within a complex query.
- The `WITH` clause retrieves the results of a query block and stores it in the user's temporary tablespace.
- The `WITH` clause may improve performance.

`WITH` Clause: Example

Using the `WITH` clause, write a query to display the department name and total salaries for those departments whose total salary is greater than the average salary across departments.

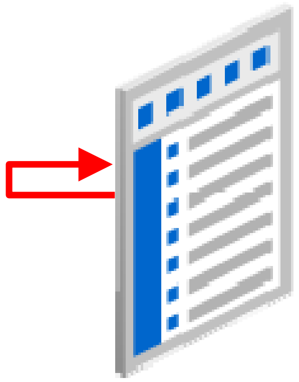
WITH Clause: Example

```
WITH
dept_costs  AS (
   SELECT d.department_name, SUM(e.salary) AS dept_total
   FROM   employees e JOIN departments d
   ON     e.department_id = d.department_id
   GROUP BY d.department_name),
avg_cost    AS (
   SELECT SUM(dept_total)/COUNT(*) AS dept_avg
   FROM   dept_costs)
SELECT *
FROM   dept_costs
WHERE  dept_total >
        (SELECT dept_avg
         FROM avg_cost)
ORDER BY department_name;
```

Recursive `WITH` Clause

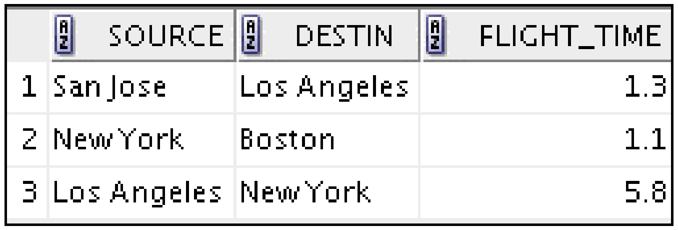
The Recursive `WITH` clause

- Enables formulation of recursive queries.
- Creates query with a name, called the Recursive `WITH` element name
- Contains two types of query blocks member: anchor and a recursive
- Is ANSI-compatible



Recursive `WITH` Clause: Example

FLIGHTS Table

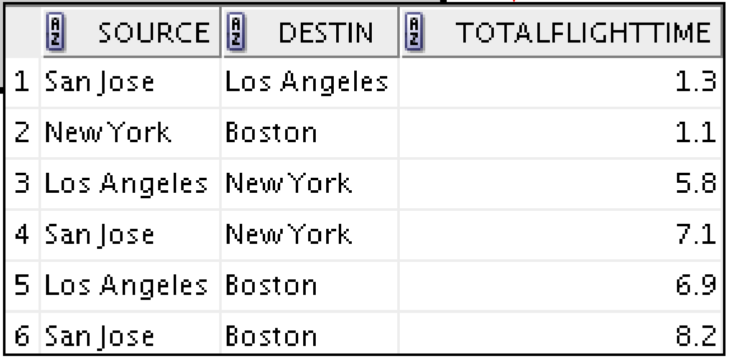


	SOURCE	DESTIN	FLIGHT_TIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8

1

```
WITH Reachable_From (Source, Destin, TotalFlightTime) AS
(
    SELECT Source, Destin, Flight_time
    FROM Flights
  UNION ALL
   SELECT incoming.Source, outgoing.Destin,
          incoming.TotalFlightTime+outgoing.Flight_time
   FROM Reachable_From incoming, Flights outgoing
   WHERE incoming.Destin = outgoing.Source
)
SELECT Source, Destin, TotalFlightTime
FROM Reachable_From;
```

2



	SOURCE	DESTIN	TOTALFLIGHTTIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8
4	San Jose	New York	7.1
5	Los Angeles	Boston	6.9
6	San Jose	Boston	8.2

3

Quiz

With a correlated subquery, the inner `SELECT` statement drives the outer `SELECT` statement.

1. True
2. False

Summary

In this lesson, you should have learned that:

- A multiple-column subquery returns more than one column
- Multiple-column comparisons can be pairwise or nonpairwise
- A multiple-column subquery can also be used in the `FROM` clause of a `SELECT` statement

Summary

- Correlated subqueries are useful whenever a subquery must return a different result for each candidate row
- The `EXISTS` operator is a Boolean operator that tests the presence of a value
- Correlated subqueries can be used with `SELECT`, `UPDATE`, and `DELETE` statements
- You can use the `WITH` clause to use the same query block in a `SELECT` statement when it occurs more than once

Practice 6: Overview

This practice covers the following topics:

- Creating multiple-column subqueries
- Writing correlated subqueries
- Using the `EXISTS` operator
- Using scalar subqueries
- Using the `WITH` clause