

18

Advanced Subqueries

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Schedule:	Timing	Topic
	60 minutes	Lecture
	50 minutes	Practice
	110 minutes	Total

Objectives

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

- **Write a multiple-column subquery**
- **Describe and explain the behavior of subqueries when null values are retrieved**
- **Write a subquery in a `FROM` clause**
- **Use scalar subqueries in SQL**
- **Describe the types of problems that can be solved with correlated subqueries**
- **Write correlated subqueries**
- **Update and delete rows using correlated subqueries**
- **Use the `EXISTS` and `NOT EXISTS` operators**
- **Use the `WITH` clause**

ORACLE

18-2

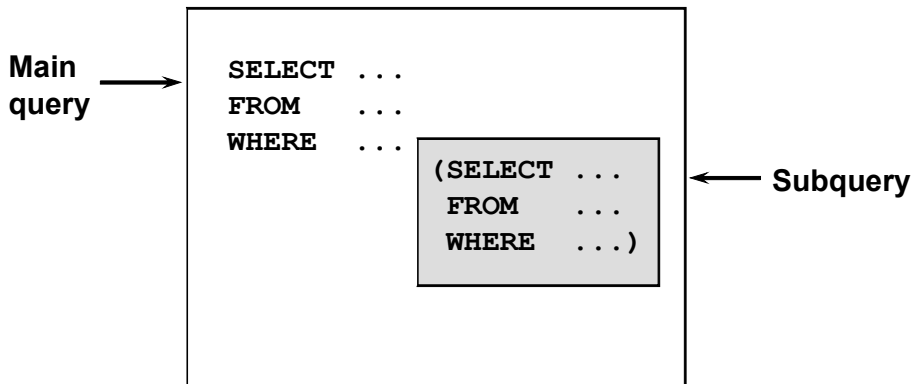
Copyright © Oracle Corporation, 2001. All rights reserved.

Lesson Aim

In this lesson, you learn how to write multiple-column subqueries and subqueries in the `FROM` clause of a `SELECT` statement. You also learn how to solve problems by using scalar, correlated subqueries and the `WITH` clause.

What Is a Subquery?

A subquery is a `SELECT` statement embedded in a clause of another SQL statement.



ORACLE®

18-3

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is a Subquery?

A *subquery* is a `SELECT` statement that is embedded in a clause of another SQL statement, called the parent statement.

The subquery (inner query) returns a value that is used by the parent statement. Using a nested subquery is equivalent to performing two sequential queries and using the result of the inner query as the search value in the outer query (main query).

Subqueries can be used for the following purposes:

- To provide values for conditions in `WHERE`, `HAVING`, and `START WITH` clauses of `SELECT` statements
- To define the set of rows to be inserted into the target table of an `INSERT` or `CREATE TABLE` statement
- To define the set of rows to be included in a view or snapshot in a `CREATE VIEW` or `CREATE SNAPSHOT` statement
- To define one or more values to be assigned to existing rows in an `UPDATE` statement
- To define a table to be operated on by a containing query. (You do this by placing the subquery in the `FROM` clause. This can be done in `INSERT`, `UPDATE`, and `DELETE` statements as well.)

Note: A subquery is evaluated once for the entire parent statement.

Instructor Note

You can skip this slide if the students are already familiar with these concepts.

Subqueries

```
SELECT select_list
FROM   table
WHERE  expr operator (SELECT select_list
                        FROM   table);
```

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outer query).

ORACLE

18-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Subqueries

You can build powerful statements out of simple ones by using subqueries. Subqueries can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself or some other table. Subqueries are very useful for writing SQL statements that need values based on one or more unknown conditional values.

In the syntax:

operator includes a comparison operator such as >, =, or IN

Note: Comparison operators fall into two classes: single-row operators (>, =, >=, <, <=, <>) and multiple-row operators (IN, ANY, ALL).

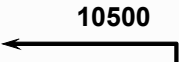
The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement. The inner and outer queries can retrieve data from either the same table or different tables.

Instructor Note

You can skip this slide if the students are already familiar with these concepts.

Using a Subquery

```
SELECT last_name
FROM employees
WHERE salary >
    (SELECT salary
     FROM employees
     WHERE employee_id = 149) ;
```



LAST_NAME
King
Kochhar
De Haan
Abel
Hartstein
Higgins

6 rows selected.

ORACLE

18-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Using a Subquery

In the example in the slide, the inner query returns the salary of the employee with employee number 149. The outer query uses the result of the inner query to display the names of all the employees who earn more than this amount.

Example

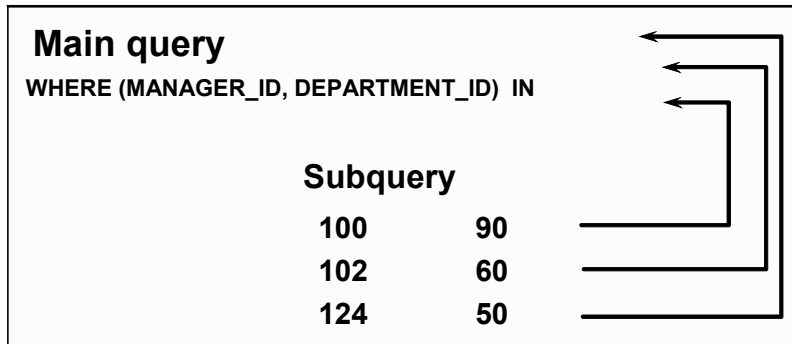
Display the names of all employees who earn less than the average salary in the company.

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary < (SELECT AVG(salary)
                FROM employees);
```

Instructor Note

You can skip this slide if the students are already familiar with these concepts.

Multiple-Column Subqueries



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

ORACLE

18-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Multiple-Column Subqueries

So far you have written single-row subqueries and multiple-row subqueries where only one column is returned by the inner `SELECT` statement and this is used to evaluate the expression in the parent select statement. If you want to compare two or more columns, you must write a compound `WHERE` clause using logical operators. Using multiple-column subqueries, you can combine duplicate `WHERE` conditions into a single `WHERE` clause.

Syntax

```
SELECT    column, column, ...
FROM      table
WHERE     (column, column, ...) IN
          (SELECT column, column, ...
           FROM    table
           WHERE   condition);
```

The graphic in the slide illustrates that the values of the `MANAGER_ID` and `DEPARTMENT_ID` from the main query are being compared with the `MANAGER_ID` and `DEPARTMENT_ID` values retrieved by the subquery. Since the number of columns that are being compared are more than one, the example qualifies as a multiple-column subquery.

Column Comparisons

Column comparisons in a multiple-column subquery can be:

- **Pairwise comparisons**
- **Nonpairwise comparisons**

ORACLE

18-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Pairwise versus Nonpairwise Comparisons

Column comparisons in a multiple-column subquery can be pairwise comparisons or nonpairwise comparisons.

In the example on the next slide, a pairwise comparison was executed in the `WHERE` clause. Each candidate row in the `SELECT` statement must have *both* the same `MANAGER_ID` column and the `DEPARTMENT_ID` as the employee with the `EMPLOYEE_ID` 178 or 174.

A multiple-column subquery can also be a nonpairwise comparison. In a nonpairwise comparison, each of the columns from the `WHERE` clause of the parent `SELECT` statement are individually compared to multiple values retrieved by the inner select statement. The individual columns can match any of the values retrieved by the inner select statement. But collectively, all the multiple conditions of the main `SELECT` statement must be satisfied for the row to be displayed. The example on the next page illustrates a nonpairwise comparison.

Pairwise Comparison Subquery

Display the details of the employees who are managed by the same manager *and* work in the same department as the employees with `EMPLOYEE_ID` 178 or 174.

```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  (manager_id, department_id) IN
      (SELECT manager_id, department_id
       FROM   employees
       WHERE  employee_id IN (178,174))
AND    employee_id NOT IN (178,174);
```

ORACLE

18-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Pairwise Comparison Subquery

The example in the slide is that of a multiple-column subquery because the subquery returns more than one column. It compares the values in the `MANAGER_ID` column and the `DEPARTMENT_ID` column of each row in the `EMPLOYEES` table with the values in the `MANAGER_ID` column and the `DEPARTMENT_ID` column for the employees with the `EMPLOYEE_ID` 178 or 174.

First, the subquery to retrieve the `MANAGER_ID` and `DEPARTMENT_ID` values for the employees with the `EMPLOYEE_ID` 178 or 174 is executed. These values are compared with the `MANAGER_ID` column and the `DEPARTMENT_ID` column of each row in the `EMPLOYEES` table. If the values match, the row is displayed. In the output, the records of the employees with the `EMPLOYEE_ID` 178 or 174 will not be displayed. The output of the query in the slide follows.

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
176	149	80

Nonpairwise Comparison Subquery

Display the details of the employees who are managed by the same manager as the employees with **EMPLOYEE_ID** 174 or 141 *and* work in the same department as the employees with **EMPLOYEE_ID** 174 or 141.

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   manager_id IN
        (SELECT  manager_id
         FROM    employees
         WHERE   employee_id IN (174,141))
AND     department_id IN
        (SELECT  department_id
         FROM    employees
         WHERE   employee_id IN (174,141))
AND     employee_id NOT IN(174,141);
```

ORACLE

18-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Nonpairwise Comparison Subquery

The example shows a nonpairwise comparison of the columns. It displays the **EMPLOYEE_ID**, **MANAGER_ID**, and **DEPARTMENT_ID** of any employee whose manager ID matches any of the manager IDs of employees whose employee IDs are either 174 or 141 and **DEPARTMENT_ID** match any of the department IDs of employees whose employee IDs are either 174 or 141.

First, the subquery to retrieve the **MANAGER_ID** values for the employees with the **EMPLOYEE_ID** 174 or 141 is executed. Similarly, the second subquery to retrieve the **DEPARTMENT_ID** values for the employees with the **EMPLOYEE_ID** 174 or 141 is executed. The retrieved values of the **MANAGER_ID** and **DEPARTMENT_ID** columns are compared with the **MANAGER_ID** and **DEPARTMENT_ID** column for each row in the **EMPLOYEES** table. If the **MANAGER_ID** column of the row in the **EMPLOYEES** table matches with any of the values of the **MANAGER_ID** retrieved by the inner subquery and if the **DEPARTMENT_ID** column of the row in the **EMPLOYEES** table matches with any of the values of the **DEPARTMENT_ID** retrieved by the second subquery, the record is displayed. The output of the query in the slide follows.

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
142	124	50
143	124	50
144	124	50
176	149	80

Using a Subquery in the FROM Clause

```
SELECT  a.last_name, a.salary,  
        a.department_id, b.salavg  
FROM    employees a, (SELECT  department_id,  
                        AVG(salary) salavg  
                        FROM    employees  
                        GROUP BY department_id) b  
WHERE   a.department_id = b.department_id  
AND     a.salary > b.salavg;
```

LAST_NAME	SALARY	DEPARTMENT_ID	SALAVG
Hartstein	13000	20	9500
Mourgos	5800	50	3500
Hunold	9000	60	6400
Zlotkey	10500	80	10033.3333
Abel	11000	80	10033.3333
King	24000	90	19333.3333
Higgins	12000	110	10150

7 rows selected.

ORACLE

18-10

Copyright © Oracle Corporation, 2001. All rights reserved.

Using a Subquery in the FROM Clause

You can use a subquery in the FROM clause of a SELECT statement, which is very similar to how views are used. A subquery in the FROM clause of a SELECT statement is also called an *inline* view. A subquery in the FROM clause of a SELECT statement defines a data source for that particular SELECT statement, and only that SELECT statement. The example on the slide displays employee last names, salaries, department numbers, and average salaries for all the employees who earn more than the average salary in their department. The subquery in the FROM clause is named b, and the outer query references the SALAVG column using this alias.

Instructor Note

You may wish to point out that the example demonstrates a useful technique to combine detail row values and aggregate data in the same output.

Scalar Subquery Expressions

- A scalar subquery expression is a subquery that returns exactly one column value from one row.
- Scalar subqueries were supported in Oracle8i only in a limited set of cases, For example:
 - `SELECT` statement (`FROM` and `WHERE` clauses)
 - `VALUES` list of an `INSERT` statement
- In Oracle9i, scalar subqueries can be used in:
 - Condition and expression part of `DECODE` and `CASE`
 - All clauses of `SELECT` except `GROUP BY`

ORACLE

18-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Scalar Subqueries in SQL

A subquery that returns exactly one column value from one row is also referred to as a scalar subquery. Multiple-column subqueries written to compare two or more columns, using a compound `WHERE` clause and logical operators, do not qualify as scalar subqueries.

The value of the scalar subquery expression is the value of the select list item of the subquery. If the subquery returns 0 rows, the value of the scalar subquery expression is `NULL`. If the subquery returns more than one row, the Oracle Server returns an error. The Oracle Server has always supported the usage of a scalar subquery in a `SELECT` statement. The usage of scalar subqueries has been enhanced in Oracle9i. You can now use scalar subqueries in:

- Condition and expression part of `DECODE` and `CASE`
- All clauses of `SELECT` except `GROUP BY`
- In the left-hand side of the operator in the `SET` clause and `WHERE` clause of `UPDATE` statement

However, scalar subqueries are not valid expressions in the following places:

- As default values for columns and hash expressions for clusters
- In the `RETURNING` clause of `DML` statements
- As the basis of a function-based index
- In `GROUP BY` clauses, `CHECK` constraints, `WHEN` conditions
- `HAVING` clauses
- In `START WITH` and `CONNECT BY` clauses
- In statements that are unrelated to queries, such as `CREATE PROFILE`

Scalar Subqueries: Examples

Scalar Subqueries in CASE Expressions

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

Scalar Subqueries in 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);
```

ORACLE

18-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Scalar Subqueries: Examples

The first example in the slide demonstrates that scalar subqueries can be used in CASE expressions. The inner query returns the value 20, which is the department ID of the department whose location ID is 1800. The CASE expression in the outer query uses the result of the inner query to display the employee ID, last names, and a value of Canada or USA, depending on whether the department ID of the record retrieved by the outer query is 20 or not.

The result of the preceding example follows:

EMPLOYEE_ID	LAST_NAME	LOCATI
100	King	USA
101	Kochhar	USA
102	De Haan	USA
...		
201	Hartstein	Canada
202	Fay	Canada
205	Higgins	USA
206	Gietz	USA

20 rows selected.

Scalar Subqueries: Examples (continued)

The second example in the slide demonstrates that scalar subqueries can be used in the `ORDER BY` clause. The example orders the output based on the `DEPARTMENT_NAME` by matching the `DEPARTMENT_ID` from the `EMPLOYEES` table with the `DEPARTMENT_ID` from the `DEPARTMENTS` table. This comparison is done in a scalar subquery in the `ORDER BY` clause. The result of the the second example follows:

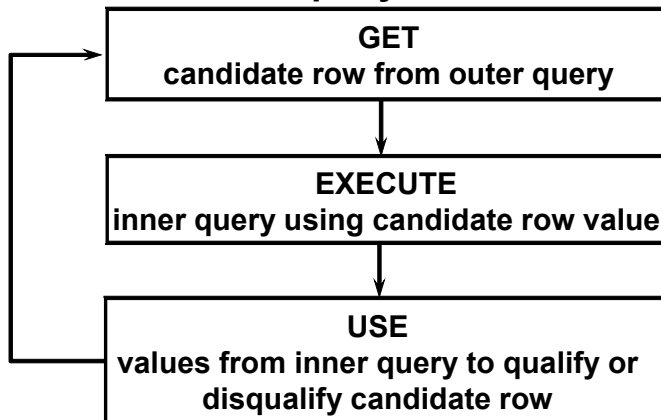
EMPLOYEE_ID	LAST_NAME
205	Higgins
206	Gietz
200	Whalen
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
201	Hartstein
202	Fay
149	Zlotkey
176	Taylor
174	Abel
EMPLOYEE_ID	LAST_NAME
124	Mourgos
141	Rajs
142	Davies
143	Matos
144	Vargas
178	Grant

20 rows selected.

The second example uses a correlated subquery. In a correlated subquery, the subquery references a column from a table referred to in the parent statement. Correlated subqueries are explained later in this lesson.

Correlated Subqueries

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



ORACLE

18-14

Copyright © Oracle Corporation, 2001. All rights reserved.

Correlated Subqueries

The Oracle Server performs a correlated subquery when the subquery references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a `SELECT`, `UPDATE`, or `DELETE` statement.

Nested Subqueries Versus Correlated Subqueries

With a normal nested subquery, the inner `SELECT` query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

Nested Subquery Execution

- The inner query executes first and finds a value.
- The outer query executes once, using the value from the inner query.

Correlated Subquery Execution

- Get a candidate row (fetched by the outer query).
- Execute the inner query using the value of the candidate row.
- Use the values resulting from the inner query to qualify or disqualify the candidate.
- Repeat until no candidate row remains.

Correlated Subqueries

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

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

ORACLE

18-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Correlated Subqueries (continued)

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result or set of results for each candidate row considered by the main query. In other words, you use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.

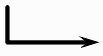
The Oracle Server performs a correlated subquery when the subquery references a column from a table in the parent query.

Note: You can use the ANY and ALL operators in a correlated subquery.

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
WHERE  salary > (SELECT AVG(salary)
                FROM   employees
                WHERE  department_id =
                      outer.department_id) ;
```



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

ORACLE

18-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Correlated Subqueries

The example in the slide determines which employees earn more than the average salary of their department. In this case, the correlated subquery specifically computes the average salary for each department.

Because both the outer query and inner query use the EMPLOYEES table in the FROM clause, an alias is given to EMPLOYEES in the outer SELECT statement, for clarity. Not only does the alias make the entire SELECT statement more readable, but without the alias the query would not work properly, because the inner statement would not be able to distinguish the inner table column from the outer table column.

Instructor Note

You may wish to indicate that the aliases used are a syntactical requirement. The alias OUTER used here is mandatory, unlike other cases where an alias is used to add clarity and readability to the SQL statement.

Using Correlated Subqueries

Display details of those employees who have switched 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
101	Kochhar	AD_VP
176	Taylor	SA_REP
200	Whalen	AD_ASST

ORACLE

18-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Using Correlated Subqueries

The example in the slide displays the details of those employees who have switched jobs at least twice. The Oracle Server evaluates a correlated subquery as follows:

1. Select a row from the table specified in the outer query. This will be the current candidate row.
2. Store the value of the column referenced in the subquery from this candidate row. (In the example in the slide, the column referenced in the subquery is `E.EMPLOYEE_ID`.)
3. Perform the subquery with its condition referencing the value from the outer query's candidate row. (In the example in the slide, group function `COUNT (*)` is evaluated based on the value of the `E.EMPLOYEE_ID` column obtained in step 2.)
4. Evaluate the `WHERE` clause of the outer query on the basis of results of the subquery performed in step 3. This determines if the candidate row is selected for output. (In the example, the number of times an employee has switched jobs, evaluated by the subquery, is compared with 2 in the `WHERE` clause of the outer query. If the condition is satisfied, that employee record is displayed.)
5. Repeat the procedure for the next candidate row of the table, and so on until all the rows in the table have been processed.

The correlation is established by using an element from the outer query in the subquery. In this example, the correlation is established by the statement `EMPLOYEE_ID = E.EMPLOYEE_ID` in which you compare `EMPLOYEE_ID` from the table in the subquery with the `EMPLOYEE_ID` from the table in the outer query.

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

ORACLE

18-18

Copyright © Oracle Corporation, 2001. All rights reserved.

The EXISTS Operator

With nesting **SELECT** statements, all logical operators are valid. In addition, you can use the **EXISTS** operator. This operator is frequently used with correlated subqueries to test whether a value retrieved by the outer query exists in the results set of the values retrieved by the inner query. If the subquery returns at least one row, the operator returns **TRUE**. If the value does not exist, it returns **FALSE**. Accordingly, **NOT EXISTS** tests whether a value retrieved by the outer query is not a part of the results set of the values retrieved by the inner query.

Using the EXISTS Operator

Find employees who have at least one person reporting to them.

```
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
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
124	Mourgos	ST_MAN	50
149	Zlotkey	SA_MAN	80
201	Hartstein	MK_MAN	20
205	Higgins	AC_MGR	110

8 rows selected.

ORACLE

18-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the EXISTS Operator

The EXISTS operator ensures that the search in the inner query does not continue when at least one match is found for the manager and employee number by the condition:

```
WHERE manager_id = outer.employee_id.
```

Note that the inner SELECT query does not need to return a specific value, so a constant can be selected. From a performance standpoint, it is faster to select a constant than a column.

Note: Having EMPLOYEE_ID in the SELECT clause of the inner query causes a table scan for that column. Replacing it with the literal X, or any constant, improves performance. This is more efficient than using the IN operator.

A IN construct can be used as an alternative for a EXISTS operator, as shown in the following example:

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  employee_id IN (SELECT manager_id
                      FROM   employees
                      WHERE  manager_id IS NOT NULL);
```

Using the NOT EXISTS Operator

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
190	Contracting

ORACLE

18-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Using the NOT EXISTS Operator

Alternative Solution

A NOT IN construct can be used as an alternative for a NOT EXISTS operator, as shown in the following example.

```
SELECT department_id, department_name
FROM departments
WHERE department_id NOT IN (SELECT department_id
                           FROM employees);
```

no rows selected

However, NOT IN evaluates to FALSE if any member of the set is a NULL value. Therefore, your query will not return any rows even if there are rows in the departments table that satisfy the WHERE condition.

Correlated UPDATE

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

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

ORACLE®

18-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Correlated UPDATE

In the case of the UPDATE statement, you can use a correlated subquery to update rows in one table based on rows from another table.

Correlated UPDATE

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

```
ALTER TABLE employees
ADD (department_name VARCHAR2 (14)) ;
```

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

ORACLE

18-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Correlated UPDATE (continued)

The example in the slide denormalizes the **EMPLOYEES** table by adding a column to store the department name and then populates the table by using a correlated update.

Here is another example for a correlated update.

Problem Statement

Use a correlated subquery to update rows in the **EMPLOYEES** table based on rows from the **REWARDS** table:

```
UPDATE employees
SET    salary = (SELECT employees.salary + rewards.pay_raise
                 FROM   rewards
                 WHERE  employee_id = employees.employee_id
                 AND    payraise_date =
                        (SELECT MAX (payraise_date)
                         FROM   rewards
                         WHERE  employee_id = employees.employee_id))
WHERE  employees.employee_id
IN     (SELECT employee_id
       FROM   rewards) ;
```

Instructor Note

In order to demonstrate the code example in the notes, you must first run the script file `\labs\cre_reward.sql`, which creates the **REWARDS** table and inserts records into the table. Remember to **rollback** the transaction if you demo the script in the slide or notes page. This is very important as if this is not done, the outputs shown in the practices will not match.

Correlated UPDATE (continued)

This example uses the REWARDS table. The REWARDS table has the columns EMPLOYEE_ID, PAY_RAISE, and PAYRAISE_DATE. Every time an employee gets a pay raise, a record with the details of the employee ID, the amount of the pay raise, and the date of receipt of the pay raise is inserted into the REWARDS table. The REWARDS table can contain more than one record for an employee. The PAYRAISE_DATE column is used to identify the most recent pay raise received by an employee.

In the example, the SALARY column in the EMPLOYEES table is updated to reflect the latest pay raise received by the employee. This is done by adding the current salary of the employee with the corresponding pay raise from the REWARDS table.

Correlated DELETE

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

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

ORACLE

18-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Correlated DELETE

In the case of a DELETE statement, you can use a correlated subquery to delete only those rows that also exist in another table. If you decide that you will maintain only the last four job history records in the JOB_HISTORY table, then when an employee transfers to a fifth job, you delete the oldest JOB_HISTORY row by looking up the JOB_HISTORY table for the MIN(START_DATE) for the employee. The following code illustrates how the preceding operation can be performed using a correlated DELETE:

```
DELETE FROM job_history JH
WHERE  employee_id =
      (SELECT employee_id
       FROM   employees E
       WHERE  JH.employee_id = E.employee_id
       AND    start_date =
             (SELECT MIN(start_date)
              FROM    job_history JH
              WHERE   JH.employee_id = E.employee_id)
       AND 5 > (SELECT COUNT(*)
                FROM    job_history JH
                WHERE   JH.employee_id = E.employee_id
                GROUP BY employee_id
                HAVING COUNT(*) >= 4));
```


Correlated DELETE

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

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

ORACLE

18-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Correlated DELETE (continued)

Example

Two tables are used in this example. They are:

- The **EMPLOYEES** table, which gives details of all the current employees
- The **EMP_HISTORY** table, which gives details of previous employees

EMP_HISTORY contains data regarding previous employees, so it would be erroneous if the same employee's record existed in both the **EMPLOYEES** and **EMP_HISTORY** tables. You can delete such erroneous records by using the correlated subquery shown in the slide.

Instructor Note

In order to demonstrate the code example in the slide, you must first run the script file `\labs\cre_emphistory.sql`, which creates the **EMP_HISTORY** table and inserts records into the table.

The 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 improves performance

ORACLE

18-26

Copyright © Oracle Corporation, 2001. All rights reserved.

The WITH clause

Using the **WITH** clause, you can define a query block before using it in a query. The **WITH** clause (formally known as `subquery_factoring_clause`) enables you to reuse the same query block in a **SELECT** statement when it occurs more than once within a complex query. This is particularly useful when a query has many references to the same query block and there are joins and aggregations.

Using the **WITH** clause, you can reuse the same query when it is high cost to evaluate the query block and it occurs more than once within a complex query. Using the **WITH** clause, the Oracle Server retrieves the results of a query block and stores it in the user's temporary tablespace. This can improve performance.

WITH Clause Benefits

- Makes the query easy to read
- Evaluates a clause only once, even if it appears multiple times in the query, thereby enhancing 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.

ORACLE

18-27

Copyright © Oracle Corporation, 2001. All rights reserved.

WITH Clause: Example

The problem in the slide would require the following intermediate calculations:

1. Calculate the total salary for every department, and store the result using a `WITH` clause.
2. Calculate the average salary across departments, and store the result using a `WITH` clause.
3. Compare the total salary calculated in the first step with the average salary calculated in the second step. If the total salary for a particular department is greater than the average salary across departments, display the department name and the total salary for that department.

The solution for the preceding problem is given in the next page.

WITH Clause: Example

WITH

```
dept_costs AS (  
  SELECT d.department_name, SUM(e.salary) AS dept_total  
  FROM   employees e, departments d  
  WHERE  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;
```

ORACLE

18-28

Copyright © Oracle Corporation, 2001. All rights reserved.

WITH Clause: Example (continued)

The SQL code in the slide is an example of a situation in which you can improve performance and write SQL more simply by using the `WITH` clause. The query creates the query names `DEPT_COSTS` and `AVG_COST` and then uses them in the body of the main query. Internally, the `WITH` clause is resolved either as an in-line view or a temporary table. The optimizer chooses the appropriate resolution depending on the cost or benefit of temporarily storing the results of the `WITH` clause.

Note: A subquery in the `FROM` clause of a `SELECT` statement is also called an in-line view.

The output generated by the SQL code on the slide will be as follows:

DEPARTMENT_NAME	DEPT_TOTAL
Executive	58000
Sales	30100

The WITH Clause Usage Notes

- It is used only with `SELECT` statements.
- A query name is visible to all `WITH` element query blocks (including their subquery blocks) defined after it and the main query block itself (including its subquery blocks).
- When the query name is the same as an existing table name, the parser searches from the inside out, the query block name takes precedence over the table name.
- The `WITH` clause can hold more than one query. Each query is then separated by a comma.

Summary

In this lesson, you should have learned the following:

- **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.**
- **Scalar subqueries have been enhanced in Oracle9i.**

ORACLE

18-29

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

You can use multiple-column subqueries to combine multiple `WHERE` conditions into a single `WHERE` clause. Column comparisons in a multiple-column subquery can be pairwise comparisons or non-pairwise comparisons.

You can use a subquery to define a table to be operated on by a containing query.

Oracle 9i enhances the the uses of scalar subqueries. Scalar subqueries can now be used in:

- Condition and expression part of `DECODE` and `CASE`
- All clauses of `SELECT` except `GROUP BY`
- `SET` clause and `WHERE` clause of `UPDATE` 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**

ORACLE

Summary (continued)

The Oracle Server performs a correlated subquery when the subquery references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a `SELECT`, `UPDATE`, or `DELETE` statement. Using the `WITH` clause, you can reuse the same query when it is costly to reevaluate the query block and it occurs more than once within a complex query.

Practice 18 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**

ORACLE

18-31

Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 18 Overview

In this practice, you write multiple-column subqueries, correlated and scalar subqueries. You also solve problems by writing the WITH clause.

Instructor Note

You might want to recap the ALL and ANY operators before the students start the practice. This is required for the questions.

ALL: Compares a value to every value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, >=. Evaluates to TRUE if the query returns no rows.

```
SELECT * FROM employees
WHERE salary > = ALL ( 1400, 3000);
```

ANY: Compares a value to each value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, >=. Evaluates to FALSE if the query returns no rows.

```
SELECT * FROM employees
WHERE salary = ANY
  (SELECT salary FROM employees
   WHERE department_id = 30);
```

Practice 18

1. Write a query to display the last name, department number, and salary of any employee whose department number and salary both match the department number and salary of any employee who earns a commission.

LAST_NAME	DEPARTMENT_ID	SALARY
Taylor	80	8600
Zlotkey	80	10500
Abel	80	11000

2. Display the last name, department name, and salary of any employee whose salary and commission match the salary and commission of any employee located in location ID 1700.

LAST_NAME	DEPARTMENT_NAME	SALARY
Whalen	Administration	4400
Gietz	Accounting	8300
Higgins	Accounting	12000
Kochhar	Executive	17000
De Haan	Executive	17000
King	Executive	24000

6 rows selected.

3. Create a query to display the last name, hire date, and salary for all employees who have the same salary and commission as Kochhar.

Note: Do not display Kochhar in the result set.

LAST_NAME	HIRE_DATE	SALARY
De Haan	13-JAN-93	17000

4. Create a query to display the employees who earn a salary that is higher than the salary of all of the sales managers (`JOB_ID = 'SA_MAN'`). Sort the results on salary from highest to lowest.

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Kochhar	AD_VP	17000
De Haan	AD_VP	17000
Hartstein	MK_MAN	13000
Higgins	AC_MGR	12000
Abel	SA_REP	11000

6 rows selected.

Practice 18 (continued)

5. Display the details of the employee ID, last name, and department ID of those employees who live in cities whose name begins with *T*.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
201	Hartstein	20
202	Fay	20

6. Write a query to find all employees who earn more than the average salary in their departments. Display last name, salary, department ID, and the average salary for the department. Sort by average salary. Use aliases for the columns retrieved by the query as shown in the sample output.

ENAME	SALARY	DEPTNO	DEPT_AVG
Mourgos	5800	50	3500
Hunold	9000	60	6400
Hartstein	13000	20	9500
Abel	11000	80	10033.3333
Zlotkey	10500	80	10033.3333
Higgins	12000	110	10150
King	24000	90	19333.3333

7 rows selected.

7. Find all employees who are not supervisors.
- a. First do this using the `NOT EXISTS` operator.

LAST_NAME
Ernst
Lorentz
Rajs
Davies
Matos
Vargas
Abel
Taylor
Grant
Whalen
Fay
Gietz

12 rows selected.

- b. Can this be done by using the `NOT IN` operator? How, or why not?

Practice 18 (continued)

8. Write a query to display the last names of the employees who earn less than the average salary in their departments.

LAST_NAME
Kochhar
De Haan
Ernst
Lorentz
Davies
Matos
Vargas
Taylor
Fay
Gietz

10 rows selected.

9. Write a query to display the last names of the employees who have one or more coworkers in their departments with later hire dates but higher salaries.

LAST_NAME
Rajs
Davies
Matos
Vargas
Taylor

Practice 18 (continued)

10. Write a query to display the employee ID, last names, and department names of all employees.

Note: Use a scalar subquery to retrieve the department name in the SELECT statement.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT
205	Higgins	Accounting
206	Gietz	Accounting
200	Whalen	Administration
100	King	Executive
101	Kochhar	Executive
102	De Haan	Executive
103	Hunold	IT
104	Ernst	IT
107	Lorentz	IT
201	Hartstein	Marketing
202	Fay	Marketing
149	Zlotkey	Sales
176	Taylor	Sales
174	Abel	Sales
EMPLOYEE_ID	LAST_NAME	DEPARTMENT
124	Mourgos	Shipping
141	Rajs	Shipping
142	Davies	Shipping
143	Matos	Shipping
144	Vargas	Shipping
178	Grant	

20 rows selected.

11. Write a query to display the department names of those departments whose total salary cost is above one eighth (1/8) of the total salary cost of the whole company. Use the WITH clause to write this query. Name the query SUMMARY.

DEPARTMENT_NAME	DEPT_TOTAL
Executive	58000
Sales	30100

