GIFT School of Engineering and Applied Sciences

Fall 2022

CS-244: Database Systems-Lab

Lab-10 Manual

Displaying Data from Multiple Tables- SQL Joins

Introduction to Lab

This lab introduces students to selecting data from multiple tables. A *join* is used to view information from multiple tables. Hence, you can *join* tables together to view information from more than one table. Various types and joins are explained and examined with the use of examples. The main topics of this lab include:

1. Creating Joins with the ON Clause
2. Self-Joins Using the ON Clause
3. Applying Additional Conditions to a Join
4. Creating Three-Way Joins with the ON Clause
5. Non-Equijoins
6. Retrieving Records with Non-Equijoins
7. Outer Joins
8. INNER Versus OUTER Joins
9. LEFT OUTER JOIN
10. RIGHT OUTER JOIN
11. Creating Cross Joins
12. SQL Subqueries
13. Practice SQL Statements

Objectives of this Lab

At the end of this lab, students should be able to:

1. Write SELECT statements to access data from more than one table using SQL:1999 joins
2. Join a table to itself by using a self-join
3. View data that does not meet the join condition by using outer joins
4. Generate a Cartesian product of all rows from two or more tables

1. Self-Joins Using the ON Clause

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the `EMPLOYEES` table to itself, or perform a self-join. For example, to find the name of Lorentz's manager, you need to:

- Find Lorentz in the `EMPLOYEES` table by looking at the `LAST_NAME` column.
- Find the manager number for Lorentz by looking at the `MANAGER_ID` column. `Lorentz's` manager number is `103`.
- Find the name of the manager with `EMPLOYEE_ID` 103 by looking at the `LAST_NAME` column. Hunold's employee number is 103, so Hunold is Lorentz's manager.

In this process, you look in the table twice. The first time you look in the table to find Lorentz in the LAST_NAME column and MANAGER_ID value of 103. The second time you look in the EMPLOYEE_ID column to find 103 and the LAST_NAME column to find Hunold.

The below example is a self-join of the EMPLOYEES table, based on the EMPLOYEE_ID and MANAGER_ID columns.

```
SELECT e.last_name emp, m.last_name mgr FROM employees e JOIN
employees m ON (e.manager_id = m.employee_id);
```

The ON clause can also be used to join columns that have different names, within the same table or in a different table.

2. Applying Additional Conditions to a Join

You can apply additional conditions to the join.

This example performs a join on the EMPLOYEES and DEPARTMENTS tables and, in addition, displays only employees who have a manager ID of 149. To add additional conditions to the ON clause, you can add AND clauses. Alternatively, you can use a WHERE clause to apply additional conditions:

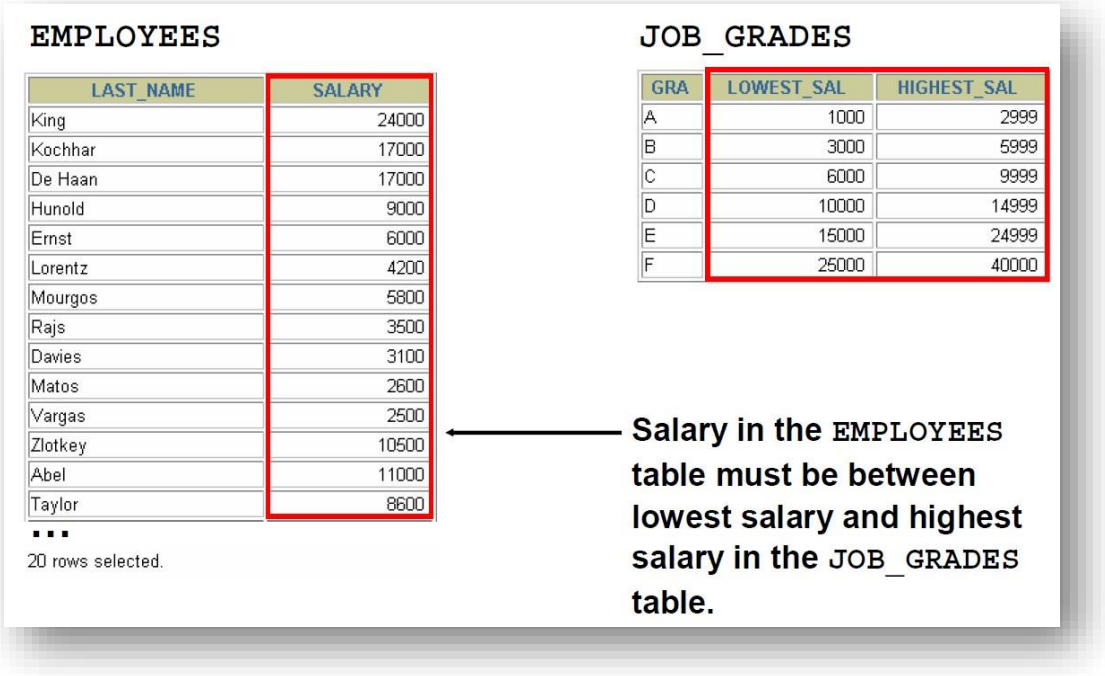
```
SELECT e.employee_id, e.last_name, e.department_id, d.department_id,
d.location_id FROM employees e JOIN departments d ON (e.department_id
= d.department_id) WHERE e.manager_id = 149;
```

3. Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name FROM employees e JOIN
departments d ON d.department_id = e.department_id JOIN locations l
ON d.location_id = l.location_id;
```

A three-way join is a join of three tables. In SQL:1999–compliant syntax, joins are performed from left to right. So, the first join to be performed is EMPLOYEES JOIN DEPARTMENTS. The first join condition can reference columns in EMPLOYEES and DEPARTMENTS but cannot reference columns in LOCATIONS. The second join condition can reference columns from all three tables.

4. Non-Equijoins



A non-equijoin is a join condition containing something other than an equality operator.

The relationship between the EMPLOYEES table and the JOB_GRADES table is an example of a non-equijoin. A relationship between the two tables is that the SALARY column in the

EMPLOYEES table must be between the values in the LOWEST_SALARY and

HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equality (=).

5. Retrieving Records with Non-Equijoins

```
SELECT e.last_name, e.salary, j.grade_level FROM employees e JOIN
job_grades j ON e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

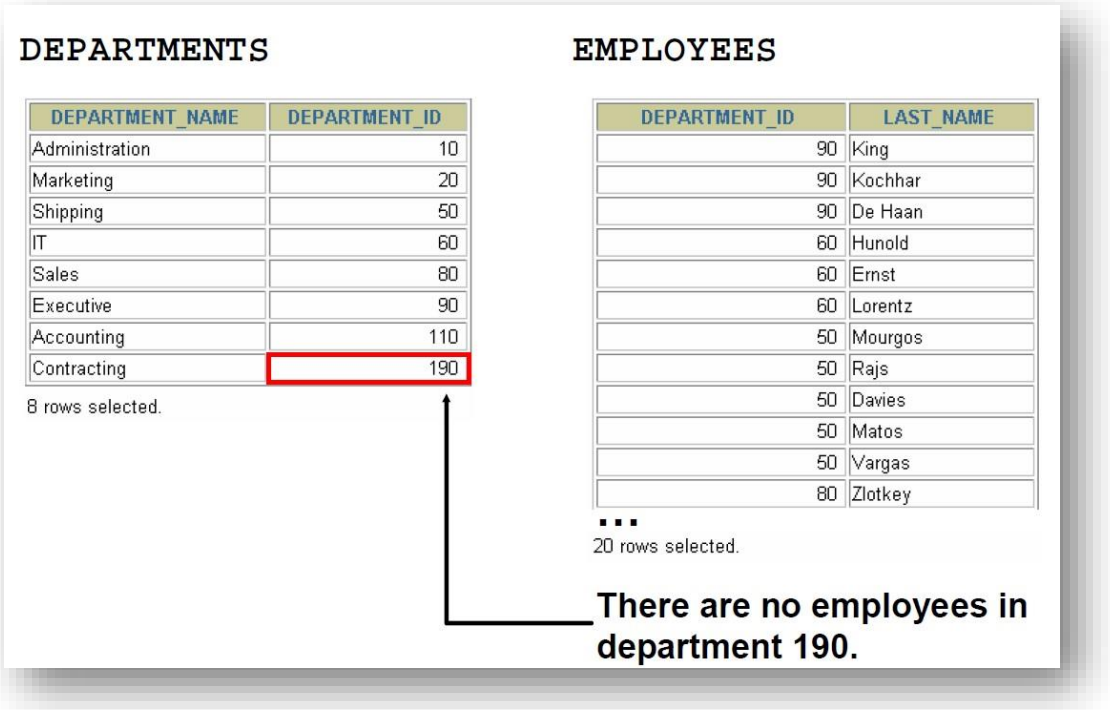
The above example creates a non-equijoin to evaluate an employee's salary grade. The salary must be *between* any pair of the low and high salary ranges.

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:

- None of the rows in the job grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits that are provided by the job grade table. That is, no employee earns less than the lowest value contained in the LOWEST_SAL column or more than the highest value contained in the HIGHEST_SAL column.

Note: Other conditions (such as <= and >=) can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using BETWEEN. Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

6. Outer Joins



If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of `EMPLOYEES` and `DEPARTMENTS` tables, department ID 190 does not

appear because there are no employees with that department ID recorded in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

To return the department record that does not have any employees, you can use an outer join.

7. INNER Versus OUTER Joins

Joining tables with the NATURAL JOIN, USING, or ON clauses results in an inner join.

Any unmatched rows are not displayed in the output. To return the unmatched rows, you can use an outer join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition.

There are three types of outer joins:

- LEFT OUTER
- RIGHT OUTER
- FULL OUTER

8. LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name FROM employees
e LEFT OUTER JOIN departments d ON (e.department_id = d.department_id)
;
```

This query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no match in the DEPARTMENTS table.

9. RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name FROM employees
e RIGHT OUTER JOIN departments d ON (e.department_id =
d.department_id) ;
```

This query retrieves all rows in the DEPARTMENTS table, which is the right table even if there is no match in the EMPLOYEES table.

10. Creating Cross Joins

```
SELECT last_name, department_name FROM employees CROSS JOIN
departments ;
```

The above example produces a Cartesian product of the `EMPLOYEES` and `DEPARTMENTS` tables.

11. SQL Subqueries

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow –

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows –

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name ]
   FROM table1 [, table2 ]
   [WHERE])
```

Example:

Write a query to display the name (first name and last name) for those employees who gets more salary than the employee whose ID is 112

```
SELECT FIRST_NAME, LAST_NAME FROM employees WHERE SALARY >
(SELECT SALARY FROM employees WHERE EMPLOYEE_ID = 112);
```

13. Practice SQL Statements

1. Write a query to display the name, department number, and department name for all employees.
2. Create a unique listing of all jobs that are in department 30.
3. Write a query to display the employee name, department name, location ID, and city of all employees who earn a commission.
4. Display the employee name and department name for all employees who have an *A* in their last names.
5. Write a query to display the name, department number, and department name for all employees who work in Seattle.
6. Display the employee name and employee number along with their manager's name and manager number. Label the columns **Employee, Emp#, Manager,** and **Mgr#,** respectively.
7. Modify query # 6 to display all employees including King, who has no manager. Order the results by the employee number.
8. Create a query that displays employee name, department numbers, and all the employees who work in the same department as a given employee. Label the columns **Department, Employee, Colleague.**
9. Create a query that displays the name, job title for all employees
10. Create a query to display the name, department_name and hire date of all employees
11. Write a query to display the name (first name and last name), salary, department id, job id for those employees who works in the same designation as the employee works whose id is 169

The End