GIFT School of Engineering and Applied Sciences

Fall 2022

CS-244: Database Systems-Lab

Lab-07 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:

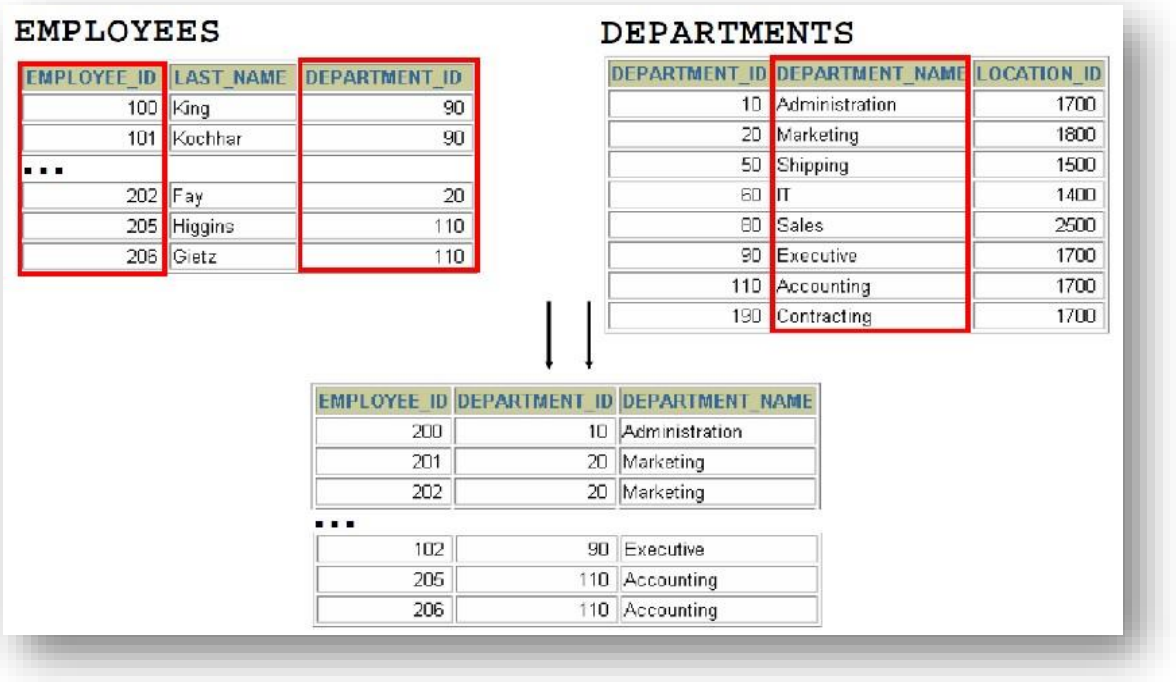
1. Obtaining Data from Multiple Tables
2. Cartesian Products
3. Types of Joins
4. Joining Tables Syntax
5. Creating Natural Joins
6. Creating Joins with the USING Clause
7. Joining Column Names
8. Qualifying Ambiguous Column Names
9. Using Table Aliases
10. Creating Joins with the ON Clause

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. Obtaining data from Multiple Tables



Sometimes you need to select data from more than one table. For example, to produce a report of Employee IDs, Department IDs, and Department Names, we need to select data from the EMPLOYEES (Employee ID) and DEPARTMENTS tables (Department ID, Department Name).
To produce this report, we need to link the EMPLOYEES and DEPARTMENTS tables together and access data from both of them.

2. Cartesian Products

When a join condition is invalid or omitted completely, the result is a *Cartesian product*, in which all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table.

To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

To see an example of a Cartesian product, try executing the following query:

```
SELECT empno, ename, dname, loc FROM emp, dept;
```

The result will be a combination of all rows from the EMP table with all rows from the DEPT table, i.e., 14 x 4 = 56 rows!

3. Types of Joins

To join tables, you can use join syntax that is compliant with the SQL:1999 standard.

1. Cross joins
2. Natural joins
3. USING clause
4. Full (or two-sided) outer joins
5. Arbitrary join conditions for outer joins

4. Joining Tables Syntax

Use the following syntax to join two or more tables:

```
SELECT table1.column, table2.column
FROM table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
ON (table1.column_name = table2.column_name)]| [LEFT|RIGHT|FULL OUTER
JOIN table2
ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

Write the join condition in the WHERE clause. Always prefix the column name with the table name if the same column name appears in more than one joined table.

In the syntax:

- ***table1.column*** denotes the table and column from which data is retrieved
- **NATURAL JOIN** joins two tables based on the same column name
- **JOIN *table* USING *column_name*** performs an equijoin based on the column name
- **JOIN *table ON table1.column_name*** performs an equijoin based on the condition in the `ON clause, = table2.column_name LEFT/RIGHT/FULL OUTER` is used to perform outer joins
- **CROSS JOIN** returns a Cartesian product from the two tables

Important Note: To join *n* tables together, you will need a minimum of *n-1* join conditions. For example, to join four tables together, a minimum of three joins is required.

5. Creating Natural Joins

You can join tables automatically based on columns in the two tables that have matching data types and names. You do this by using the keywords `NATURAL JOIN`.

Note: The join can happen on only those columns that have the same names and data types in both tables. If the columns have the same name but different data types, then the `NATURAL JOIN` syntax causes an error.

Example:

```
SELECT department_id, department_name, location_id, city FROM
departments NATURAL JOIN locations;
```

In the above example, the `LOCATIONS` table is joined to the `DEPARTMENT` table by the `LOCATION_ID` column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

Natural Joins with a WHERE Clause

Additional restrictions on a natural join are implemented by using a **`WHERE`** clause. The following example limits the rows of output to those with a department ID equal to 20 or 50:

```
SELECT department_id, department_name, location_id, city FROM
departments NATURAL JOIN locations WHERE department_id IN (20, 50);
```

6. Creating Joins with a USING Clause

Natural joins use all columns with matching names and data types to join the tables. The `USING` clause can be used to specify only those columns that should be used for an equijoin.

The columns that are referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement.

For example, the following statement is valid:

```
SELECT l.city, d.department_name FROM locations l JOIN departments d
USING (location_id) WHERE location_id = 1400;
```

The following statement is invalid because the `LOCATION_ID` is qualified in the `WHERE` clause:

```
SELECT l.city, d.department_name FROM locations l JOIN departments d
USING (location_id) WHERE d.location_id = 1400;
```

The same restriction also applies to `NATURAL` joins. Therefore, columns that have the same name in both tables must be used without any qualifiers.

Note: The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

7. Joining Column names

To determine an employee's department name, you compare the value in the `DEPARTMENT_ID` column in the `EMPLOYEES` table with the `DEPARTMENT_ID` values in the `DEPARTMENTS` table. The relationship between the `EMPLOYEES` and `DEPARTMENTS`

tables is an *equijoin;* that is, values in the DEPARTMENT_ID column in both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

Note: Equijoins are also called *simple joins* or *inner joins*.

This example joins the DEPARTMENT_ID column in the EMPLOYEES and DEPARTMENTS tables, and thus shows the location where an employee works.

```
SELECT employees.employee_id, employees.last_name,
departments.location_id, department_id FROM employees JOIN
departments USING (department_id);
```

8. Qualifying Ambiguous Column Names

You need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT_ID column in the SELECT list could be from either the DEPARTMENTS table or the EMPLOYEES table. It is necessary to add the table prefix to execute your query:

```
SELECT employees.employee_id, employees.last_name,
departments.department_id, departments.location_id FROM employees
JOIN departments ON employees.department_id =
departments.department_id;
```

If there are no common column names between the two tables, there is no need to qualify the columns. However, using the table prefix improves performance, because you tell the MySQL server exactly where to find the columns.

Note: When joining with the USING clause, you cannot qualify a column that is used in the USING clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it.

9. Using Table Aliases

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. You can use *table aliases* instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, therefore using less memory.

```
SELECT e.employee_id, e.last_name, d.location_id, department_id
FROM employees e JOIN departments d USING (department_id);
```

Notice how table aliases are identified in the FROM clause in the example. The table name is specified in full, followed by a space and then the table alias. The EMPLOYEES table has been given an alias of e, and the DEPARTMENTS table has an alias of d.

Guidelines

1. Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
2. If a table alias is used for a particular table name in the `FROM` clause, then that table alias must be substituted for the table name throughout the `SELECT` statement. Table aliases should be meaningful.
3. The table alias is valid for only the current `SELECT` statement.

10. Creating Joins with the ON Clause

Use the ON clause to specify a join condition. This lets you specify join conditions separate from any search or filter conditions in the `WHERE` clause.

```
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);
```

In this example, the `DEPARTMENT_ID` columns in the `EMPLOYEES` and `DEPARTMENTS` table are joined using the `ON` clause. Wherever a department ID in the `EMPLOYEES` table equals a department ID in the `DEPARTMENTS` table, the row is returned.

You can also use the `ON` clause to join columns that have different names.

The End