Joining Tables	

Objectives

After completing this lesson, you will be able to:

- Explain the concept of a join
- Use the JOIN keyword to query multiple tables
- Execute outer and inner joins

Combining Multiple Tables

- Some queries need data from multiple tables.
- You use a join operation to combine data from different tables.
- A join creates a temporary resultset that contains combined data.
- To join tables, there must be a relationship between certain columns in these tables.

12 - 3

The SELECT queries shown so far in this course retrieve data from a single table. You cannot answer all questions in this way.

If you want to find the details of records referenced in a foreign key, you combine data from two or more tables with a table join.

Table Joins

Category	Туре	Description
Unqualified Join	Cross Join	Combines all the rows from one table with all the rows from another table
Qualified Join	Inner Join	Identifies combinations of matching rows from two tables
Qualified Join	Outer Join	Identifies combinations of matching and mismatching rows from two tables

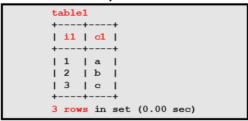
- Unqualified join: Includes all row pairs
- Qualified join: Includes specific row pairs only, according to a particular "join condition"
 Example:
 - A city (City table) is a capital of a country (Country table), and a country (Country table) has cities (City table).

12 - 4

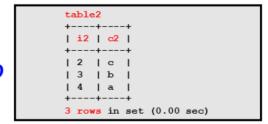
Most joins are qualified and are used to combine related tables. The join condition expresses a meaningful relationship between the data sets.

Cross Joins

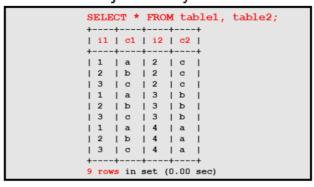
Two separate tables:



AND



Tables joined by SELECT:



- This join results in a Cartesian product.
- The result includes all combinations of rows from table1 and table2.

12 - 5

When you cross-join two tables, you combine each row from one table with each row from the other table. This yields all possible combinations of rows and is known as the Cartesian product. Joining tables this way can produce a very large number of rows because the row count is the product of the number of rows in each table.

A cross-join between two tables each containing 1000 rows returns $1000 \times 1000 = 1$ million rows. That is a lot of rows, even though the individual tables are small. This type of join is also called an "unqualified join".

Multiple Tables in the FROM Clause

Example of two separate queries that can be joined:

```
SELECT Code, Name

FROM Country

WHERE Continent = 'Africa';

SELECT CountryCode, Language

FROM CountryLanguage;
```

Joined tables:

12 - 6

You can join two or more tables by listing them in the FROM clause of the SELECT statement. Separate multiple table names by commas. Use the WHERE clause to indicate the relationship between the tables.

Imagine that you want to list all the languages spoken in every African country by using the world_innodb database. You cannot get this information from just one table. You have to retrieve the country names from the Country table and the languages from the CountryLanguage table.

The join creates a new "virtual" table, which exists only for the duration of the statement. This table contains the Code and country Name from the Country table, and the corresponding Language from the CountryLanguage table.

This type of join can use any of the constructs allowed in a single-table SELECT statement.

INNER JOIN Keyword

- Alternative to listing multiple tables in the FROM clause
- Use with the on or using clause

```
Examples:
mysql> SELECT Country.Name, City.CountryCode column
    -> FROM Country INNER JOIN City
    -> ON Country.Name = City.Name;
 +----+
          | CountryCode |
                                SELECT Country. Name,
                      OR City.CountryCode
 +----+
 | Djibouti | DJI
 | Mexico | PHL
                                FROM Country INNER JOIN City
 | Gibraltar | GIB
                                 USING (Name);
 | Armenia | COL
 | Kuwait | KWT
| Macao | MAC
 | San Marino | SMR
 | Singapore | SGP
8 rows in set (0.19 sec)
```

12 - 8

You can use the INNER JOIN keyword instead of listing join tables in the FROM clause. You then specify how rows are matched between the tables in the FROM clause, not in the WHERE clause. The ON or USING clause conditions instruct MySQL how to perform the join.

Recall that the purpose of the previous example was to join the <code>City</code> and <code>Country</code> tables on matching city names. The first example in the slide uses <code>INNER JOIN</code> and <code>FROM...ON</code> to get the same result. If the name of the joined column is the same in both tables, you can use <code>USING()</code> instead of <code>ON</code> as per the second example in the slide, and list the names of the join columns within its parentheses.

For example, in the join query in the slide, you match the two tables by columns with the same name (Name). Therefore, you can rewrite this statement with USING().

Note: ON and USING are not functionally identical. USING treats the columns from the two tables as the same. ON treats them as two different columns.

JOIN Keyword

- JOIN is equivalent to INNER JOIN
- Use with the on and where clauses
- Example:

12 - 9

The JOIN keyword is equivalent to the INNER JOIN keyword and you can use either to achieve the same result.

The query in the example counts the cities in South America. The City table does not contain continent data. This information is in the Country table, so you need to join the City and Country tables.

The expression following the ON clause must be a condition. In this example, the ON clause states that a City row matches a Country row only when the CountryCode column value in the City table equals the value of the Code column in the Country table.

You can filter the resultset further with the WHERE clause. In this example, the join applies only when the continent is South America.

Use the ON clause for conditions that specify how to join tables, and use the WHERE clause to restrict which rows you want in the result set.

Outer Joins

Identify combinations of matching and mismatching rows from two tables.

LEFT JOIN: Returns all rows from the left table, even if there are no matches in the right table

RIGHT JOIN: Returns all rows from the right table, even if there are no matches in the left table

```
mysql> SELECT column_name(s)
-> FROM left_table
-> LEFT JOIN right_table
-> ON left_table.column_name =
    right_table.column_name;
```

```
mysql> SELECT column_name(s)
-> FROM left_table
-> RIGHT JOIN right_table
-> ON left_table.column_name =
    right_table.column_name;
```

12 - 10

There are two types of outer join: LEFT JOIN and RIGHT JOIN. They answer the same kinds of questions, but differ slightly in syntax. A LEFT JOIN can always be rewritten as an equivalent RIGHT JOIN.

For example, an inner join can match country names in the Country table with the languages spoken in those countries through a join with the CountryLanguage table, based on country codes. But it cannot tell you which countries do not have an associated language in the CountryLanguage table.

To answer this question, you need to identify which country codes in the Country table are not present in the CountryLanguage table. An outer join gives you this information. In the following example, the LEFT JOIN shows nulls for countries that do not have an associated language:

Finding Mismatches with LEFT JOIN

- LEFT JOIN uses the WHERE clause to find mismatches.
- Example:

```
mysql> SELECT Name, Language
    -> FROM Country
    -> LEFT JOIN CountryLanguage
    -> ON Code = CountryCode
    -> WHERE CountryCode IS NULL;
                                       NULL
| Antarctica
| French Southern territories
                                       NULL
| Bouvet Island
                                       NULL
| Heard Island and McDonald Islands
                                      NULL
| British Indian Ocean Territory
                                       NULL
| South Georgia and the South Sandwich Islands | NULL
6 rows in set (0.00 sec)
```

12 - 12

With a LEFT JOIN, the comparison that takes place between the join tables is based on the first (or left-most) table referenced in the SELECT statement.

If you are interested in only mismatches, use an appropriate WHERE clause to check which rows in the left table do not have a matching row in the right table.

In the slide example, countries with no entry in the CountryLanguage table have a null value for CountryCode.

Finding Mismatches with RIGHT JOIN

- RIGHT JOIN uses the WHERE clause to find mismatches.
- Roles of tables are reversed from LEFT JOIN.
- Example:

```
mysql> SELECT Name, Language
    -> FROM Country
    -> RIGHT JOIN CountryLanguage
    -> ON Code = CountryCode
    -> WHERE CountryCode IS NULL;
Empty set (0.00 sec)
```

- The join is based on the CountryCode table.
- Every row in CountryCode has a matching row in the Country table, so the resultset is empty.

12 - 13

In a RIGHT JOIN the roles of the tables are reversed. A RIGHT JOIN produces a result for each row in the right table, irrespective of whether it has any matches in the left table.

The previous example demonstrated a LEFT JOIN between the Country and CountryLanguage tables. This example uses a RIGHT JOIN to join the same two tables. Because all rows in the CountryLanguage table have matching rows in the Country table, the query produces an empty resultset.

Outer Joins: USING and NULL

- USING: Easier way to reference columns that have the same name in both tables. Use instead of on.
 - For example, when tables a and b both contain columns c1, c2, and c3, the following join compares the corresponding columns from each table:

```
... a LEFT JOIN b USING (c1,c2,c3) ...
```

- NULL: Identifies rows with no counterpart in the other table
 - If there are no matching rows, the join condition returns null.
 - Example:

```
SELECT table1.id * FROM table1

LEFT JOIN table2

ON table1.id=table2.id

WHERE table2.id IS NULL
```

12 - 14

 ${\tt USING}$ () is a succinct way of expressing the join condition when column names are the same in both tables. The equivalent statement using ${\tt ON}$ is:

```
mysql> SELECT table1.id * FROM table1
    -> LEFT JOIN table2
    -> ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
    -> WHERE table2.id IS NULL;
```

If there is no matching row for the right table in the ON or USING part of a LEFT JOIN, the right table returns a row with all columns set to NULL.

The example in the slide lists all rows in table1 with an id value that is not present in table2 (that is, all rows in table1 with no corresponding row in table2). This assumes that the table2.id column is declared NOT NULL.

For more information about join syntax, see the MySQL Reference Manual: http://dev.mysql.com/doc/refman/5.6/en/join.html.

Table Name Aliases

- Table references can be aliased:
 - tbl_name AS alias_name
 - tbl name alias name
 - The As keyword aids clarity but is optional.
- Examples:

```
mysql> SELECT tl.Name, t2.CountryCode
    -> FROM Country AS tl, City AS t2
    -> WHERE tl.Name = t2.Name;
    OR

mysql> SELECT tl.Name, t2.CountryCode
    -> FROM Country tl, City t2
    -> WHERE tl.Name = t2.Name;
```

12 - 15

The example in the slide joins the Country and City tables on city names. The Country table is given an alias of t1, and the City table is given an alias of t2 although, typically, aliasing is used to abbreviate table names rather than giving them meaningless names. The query results in a list of country names and codes:

Practice 12-1 Overview: Performing Inner and Outer Joins

In this practice you execute:

- Inner joins by using the INNER JOIN keyword
- Outer joins by using the LEFT JOIN and RIGHT JOIN keywords

Practice 12-2 Overview: Creating Queries Requiring Joins

In this practice, you create join queries to answer specific questions.

Practice 12-3 Overview: Additional Optional Practice

In this optional practice, you are asked more in-depth questions about your answers to Practice 12-2.