Joining Data from Multiple Tables

Overview

The related tables of a large database are linked through the use of foreign and primary keys or what are often referred to as common columns. In this lesson, you will learn how to join data contained in various tables using conditions specified in the WHERE clause and using the JOIN keyword in the FROM clause. The WHERE clause represents the traditional join approach, whereas the JOIN keyword represents the ANSI join method.

Learning objectives

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

- Identify a Cartesian join
- Create an equality join using the WHERE clause
- Create an equality join using the JOIN keyword
- Create a non-equality join using the WHERE clause
- Create a non-equality join using the JOIN...ON approach
- Create a self-join using the WHERE clause
- Create a self-join using the JOIN keyword
- Distinguish an inner join from an outer join
- Create an outer join using the WHERE clause
- Create an outer join using the OUTER keyword

Contents

1. Combining Tables (JOINS)

Most databases will contain information in multiple tables. SQL provides a variety of methods for combining information across tables. In order to combine two or more tables for a single query, we have to use a concept called "**JOIN**" to achieve the desired result. SQL joins are used to combine rows from two or more tables. The default type of join in a joined table is called an **INNER JOIN** (**JOIN**)**,** where a tuple is included in the result only if a matching tuple exists in the other relation.

1.1. Cartesian Product (CROSS JOIN)

CROSS JOIN replicates each row from the first table with every row from the second table. CROSS JOIN creates a join between tables by displaying every possible record combination Cross join can be created by two methods:

- Using the JOIN method with the CROSS JOIN keywords (explicit)
- Not including a joining condition in a WHERE clause (implicit)

The **explicit** syntax for a cross join

```
SELECT columnlist
FROM table1 CROSS JOIN table2
```

The **implicit** syntax for a cross join

```
SELECT columnlist
FROM table1, table2
```

Example 1: Execute the following query:

```
mysql > SELECT *
        FROM department
        CROSS JOIN project;
```

The Cartesian product by itself combines tuples from department and project that are unrelated to each other. Each tuple in department is combined with every tuple in project, even those that refer to a different project. The result can be an extremely large relation, and it rarely makes sense to create such a Cartesian product.

When two tables are joined, SQL creates a Cartesian product of the tables. A Cartesian product consists of all possible combinations of the rows from each of the tables. For example, performs a cross join of the department and project tables. That CROSS JOIN query generates 99 rows. (There were 9 department rows and 11 project rows, thus giving 9× 11 = 99 rows.)

JOINS and the SELECT Clause (SELECT Statements That Use More than Two Tables)

Joins are used in queries to connect tables. The following formula shows the number of joins required in a query's WHERE clause: *Number of joins = number of tables in the query – 1*

1.2. Join Conditions and Join Types

There are three different types of joins:

- **Inner joins** return a row only when the columns in the join contain values that satisfy the join condition. This means that if a row has a null value in one of the columns in the join condition, then that row isn't returned.
- **Self joins** return rows joined on the same table.
- **Outer joins** return a row even when one of the columns in the join condition contains a null value.

There are two types of join conditions, which are based on the operator used in the join:

- **Equijoins** use the equality operator =.
- **Non-equijoins** use an operator other than the equality operator.
  For example <, >, BETWEEN, and so on.

1.3. JOIN - Relational Operators

The JOIN operators shown in Table 1 determine the basis by which columns are matched and are called relational operators.

Operator	Meaning
=	equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to
!=	not equal to
<>	not equal to

Table 1

1.3.1. Qualifying References to Columns from Joined Tables

If a column name appears in multiple tables, references to the column must be qualified with a table identifier using **table_name.col_name** syntax to specify which table you mean.
Suppose that a table **mytable1** contains columns a and b, and a table **mytable2** contains columns b and c.

References to columns a or c are unambiguous, but references to b must be qualified as either mytbl1.b or mytbl2.b:

```
SELECT a, mytable1.b, mytable2.b, c
FROM mytable1 INNER JOIN mytable2 ... ;
```

Sometimes a table name qualifier is not sufficient to resolve a column reference. For example, if you're

performing a self-join (that is, joining a table to itself), you're using the table multiple times within the query, and it doesn't help to qualify a column name with the table name. In this case, table aliases are useful for communicating your intent. You can assign an alias to any instance of the table and refer to columns from that instance as alias_name.col_name. The following query joins a table to itself, but assigns an alias to one instance of the table to enable column references to be specified unambiguously:

```
SELECT mytable1.col1, m.col2
FROM mytable1 INNER JOIN mytable1 AS m
ON (mytable1.col1 = m.col1);
```

An **ambiguous reference** means that you have a column identifier in your query and the same identifier is used in more than one of the tables used in the query. Therefore, the system does not know which column is being referenced.

1.3.2. Using Table Aliases

Aliasing is the process whereby we can assign a short name to the tables being joined and use those alias names for preceding the column names. For example, the aliases e and d can be used to label the employee and dependent tables as shown in the query below. Any legal table name may be used as an alias. If you assign an alias to a table, you must use that alias to refer to the table throughout your query. You can't use the original table name.

1.4. INNER JOINS

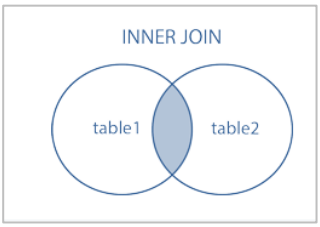
An INNER join is the type of join most often needed. It requires a value in the row of one table to match a value in a row contained in the other table. An inner join creates a join by using a commonly named and defined column. Can be created by two methods:

1. Using the WHERE clause (SQL/86 standard).
2. Using the JOIN method with the NATURAL JOIN, JOIN . . .ON, or JOIN . . . USING keywords. (SQL/92 introduced the INNER JOIN and ON clauses for performing an inner join).

Logically this syntax does a Cartesian product and adds a filter for the records that match on the joining condition. There will always be one less joining condition than there are tables to be joined. Count the number of tables and number of join conditions as a quick check of a completed query containing a join.

Tables are typically joined on the relationship between the primary key in one table and a foreign key in the other table.

If the two columns in a join condition have the same name, you must qualify them with the table name so MySQL can distinguish between them.



JOINS and the FROM Clause

Any SELECT statement that has two or more table names listed in a FROM clause is a JOIN query. By definition, a JOIN operation retrieves rows from two or more tables.

You can always use the FROM clause to list the tables from which columns are to be retrieved by a JOIN query. The order of table name listings is irrelevant to the production of the result table with the one exception noted above; that is if you use an asterisk (*) in the SELECT clause, then the column order in the result table reflects the order in which tables are listed in the FROM clause.

As explained earlier, you can specify the join conditions in a FROM clause by explicitly using the JOIN clause within the FROM clause. You can also specify the type of JOIN such as INNER, LEFT OUTER, RIGHT OUTER, or FULL OUTER. If you do not specify the type of join then, by default, the join operation is always an INNER join.

JOINS and the WHERE Clause

As you learned earlier, the WHERE clause can be used to specify the relationship between tables listed in the FROM clause along with specifying row selection criteria for display in the result table.

1.4.1.Using the WHERE clause

The traditional way to include join conditions and avoid a Cartesian result is to use the WHERE clause to instruct MySQL how to join tables correctly. The WHERE clause can perform two different activities: joining tables and providing conditions to limit or filter the rows that are affected.

The implicit syntax for an inner join

```
SELECT column_name(s)
FROM table_1, table_2 [, table_3]...
WHERE table_1.column_name operator table_2.column_name
 [AND table_2.column_name operator table_3.column_name]...
```

The explicit syntax for an inner join

```
SELECT column_name(s)
FROM table_1 [INNER] JOIN table_2
ON join_condition1 [AND join_condition 2..]
```

Note: The INNER keyword is optional.

Example 2: Display the name of each employee who has a dependent with the same first name as the employee.

```
mysql> SELECT e.fname, e.lname
       FROM employee e, dependent d
       WHERE (e.ssn = d.essn)
       AND e.lname = d.dependent_name;
Empty set (0.03 sec)
```

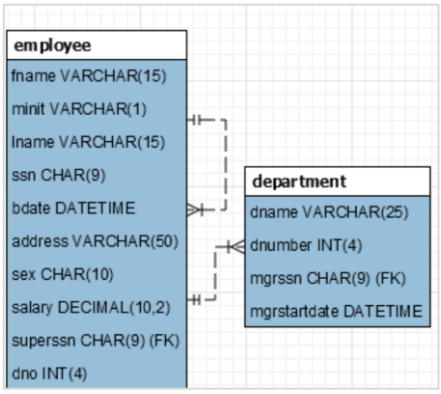
When you use the implicit syntax for an inner join, you code the tables in the from clause separated by commas. Then, you code the join condition in the WHERE clause.

The select statement joins data from the employee and department tables. The statement joins these tables on an equal comparison between the department number in the two tables. AND operator is used to combine the join condition and the search condition.

This syntax for coding joins is referred to as the implicit syntax. It was used prior to the SQL-92 standards, which introduced the explicit syntax.

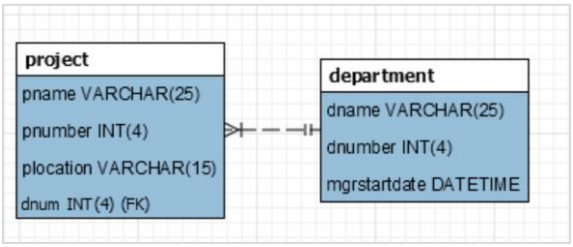
Example 3: Display the department number, the first name, and lase name of all employees who work for the 'Software' department.

```
mysql> SELECT fname,lname, e.dno
       FROM employee e, department d
       WHERE e.dno = d.dnumber
       AND dname = 'Software';
+---------+--------+------+
| fname   | lname  | dno  |
+---------+--------+------+
| Jared   | James  |    6 |
| Jon     | Jones  |    6 |
| Justin  | Mark   |    6 |
| Brad    | Knight |    6 |
| Kim     | Grace  |    6 |
| Jeff    | Chase  |    6 |
| John    | James  |    6 |
| Nandita | Ball   |    6 |
+---------+--------+------+
8 rows in set (0.00 sec)
```



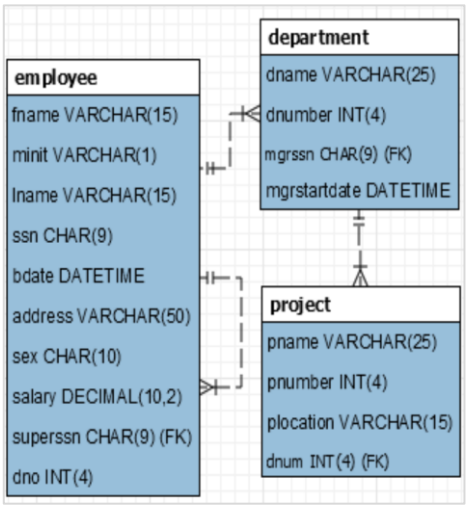
Example 4: Execute the following query:

```
mysql> SELECT d.dnumber, dname
       FROM department d
       JOIN project p
       ON (d.dnumber = p.dnum)
       ORDER BY d.dnumber;
+---------+--------------------+
| dnumber | dname              |
+---------+--------------------+
|       1 | Headquarters       |
|       4 | Administration     |
|       4 | Administration     |
|       5 | Information System |
|       5 | Information System |
|       5 | Information System |
|       6 | Software           |
|       6 | Software           |
|       6 | Software           |
|       6 | Software           |
|       7 | Hardware           |
|       7 | Hardware           |
+---------+--------------------+
12 rows in set (0.00 sec)
```



Example 5: For every project, list the project number, the controlling department number, and the department manager's last name, and birthdate.

```
mysql> SELECT pnumber, d.dnumber, e.lname, e.bdate
       FROM project p
       JOIN department d ON (p.dnum = d.dnumber)
       JOIN employee e ON (d.mgrssn = e.ssn);
+---------+---------+---------+---------------------+
| pnumber | dnumber | lname   | bdate               |
+---------+---------+---------+---------------------+
|      61 |       6 | James   | 1966-10-10 00:00:00 |
|      62 |       6 | James   | 1966-10-10 00:00:00 |
|      63 |       6 | James   | 1966-10-10 00:00:00 |
|     100 |       6 | James   | 1966-10-10 00:00:00 |
|       1 |       5 | Wong    | 1945-12-08 00:00:00 |
|       2 |       5 | Wong    | 1945-12-08 00:00:00 |
|       3 |       5 | Wong    | 1945-12-08 00:00:00 |
|      91 |       7 | Freed   | 1950-10-09 00:00:00 |
|      92 |       7 | Freed   | 1950-10-09 00:00:00 |
|      20 |       1 | Borg    | 1927-11-10 00:00:00 |
|      10 |       4 | Wallace | 1931-06-20 00:00:00 |
|      30 |       4 | Wallace | 1931-06-20 00:00:00 |
+---------+---------+---------+---------------------+
12 rows in set (0.00 sec)
```

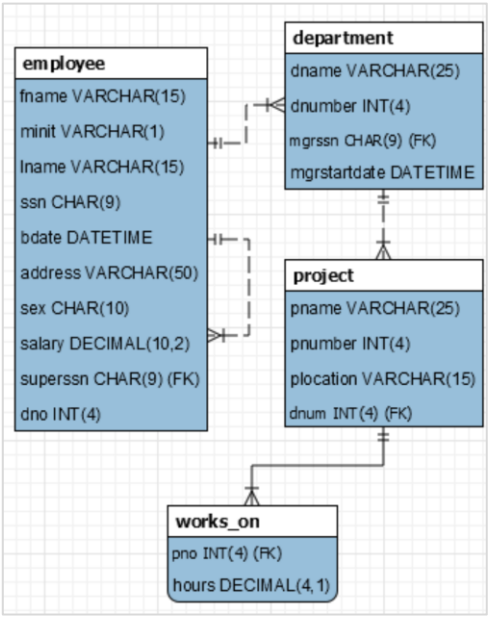


It is common for programmers to need to join data from more than two tables. You can think of multi-table join as a series of two-table joins proceeding from **left to right**.

Example 6: Display a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name and first name.

```
Mysql> SELECT d.dname, e.lname,e.fname,p.pname
       FROM department d
       JOIN project p ON d.dnumber = p.dnum
       JOIN Works_on a ON (p.pnumber = a.pno)
       JOIN employee e ON (a.essn = e.ssn)
      ORDER by dname,lname,fname;
```

dname	lname	fname	pname
Administration	Jabbar	Ahmad	Computerization
Administration	Jabbar	Ahmad	Newbenefits
Administration	Wallace	Jennifer	Newbenefits
Administration	Wong	Franklin	Computerization
Administration	Zelaya	Alicia	Computerization
Administration	Zelaya	Alicia	Newbenefits
Hardware	Ball	Nandita	LaserPrinters
Hardware	Bays	Bonnie	InkjetPrinters
Hardware	Best	Alec	InkjetPrinters
Hardware	Freed	Alex	InkjetPrinters
Hardware	James	John	LaserPrinters
Hardware	Jarvis	Jill	InkjetPrinters
Hardware	King	Billie	InkjetPrinters
Hardware	King	Ray	InkjetPrinters
Hardware	Kramer	Jon	LaserPrinters
Hardware	Leslie	Lyle	InkjetPrinters
Hardware	Snedden	Sam	InkjetPrinters
Headquarters	Borg	James	Reorganization
Headquarters	Wallace	Jennifer	Reorganization
Headquarters	Wong	Franklin	Reorganization
Information System	English	Joyce	ProductX
Information System	English	Joyce	ProductY
Information System	Narayan	Ramesh	ProductZ
Information System	Smith	John	ProductY
Information System	Smith	John	ProductX
Information System	Wong	Franklin	ProductZ
Information System	Wong	Franklin	ProductY
Software	Bacher	Red	DatabaseSystems
Software	Bacher	Red	OperatingSystems
Software	Brand	Tom	DatabaseSystems
Software	Carter	Chris	DatabaseSystems
Software	Chase	Jeff	Middleware
Software	Drew	Naveen	OperatingSystems
Software	Grace	Kim	Middleware
Software	Hall	Sammy	OperatingSystems
Software	Head	Arnold	DatabaseSystems
Software	James	Jared	OperatingSystems
Software	Jones	Jon	OperatingSystems
Software	Knight	Brad	OperatingSystems
Software	Mark	Justin	OperatingSystems
Software	Pataki	Helga	Middleware
Software	Reedy	Carl	OperatingSystems
Software	Small	Gerald	OperatingSystems
Software	Vile	Andy	DatabaseSystems
Software	Vos	Jenny	DatabaseSystems
Software	Wallis	Evan	DatabaseSystems
Software	Zell	Josh	DatabaseSystems



As you examine the previous query, note the following points:

- The **FROM** clause indicates which tables are to be joined. If three or more tables are included, the join operation takes place two tables at a time, starting from left to right. For example, if you are joining tables Table1, Table2, and Table3, first table Table1 is joined to Table2; the results of that join are then joined to table Table3.
- The **JOIN** condition in the WHERE clause tells the SELECT statement which rows will be returned.
- The number of join conditions is always equal to the number of tables being joined minus one. For example, if you join three tables (Table1, Table2, and Table3), you will have two join conditions (j1 and j2). All join conditions are connected through an AND logical operator.
- The first join condition (j1) defines the join criteria for Table1 and Table2. The second join condition (j2) defines the join criteria for the output of the first join and table Table3.

1.4.2. The syntax for a join that uses the USING keyword

A second way to express a join is through the USING keyword. That query returns only the rows with matching values in the column indicated in the USING clause- and that column must exist in both tables.

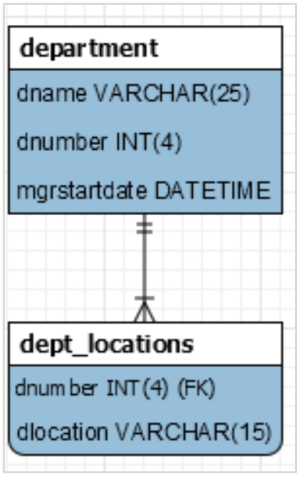
The syntax is:

```
SELECT columnlist
FROM table1
JOIN table2 USING (common-column)
```

You can use the USING keyword to simplify the syntax for joining tables. The tables must be joined by a column that has the same name in both tables.

Example 7: Write a query that performs a join of the department, and dept_locations tables by writing:

```
mysql> SELECT dname, dlocation
       FROM department
       JOIN dept_locations
       USING(dnumber);
+--------------------+-------------+
| dname              | dlocation   |
+--------------------+-------------+
| Administration     | Stafford    |
| Hardware           | Milwaukee   |
| Headquarters       | Houston     |
| Information System | Bellaire    |
| Information System | Houston     |
| Information System | Sugarland   |
| Sales              | Chicago     |
| Sales              | Dallas      |
| Sales              | Miami       |
| Sales              | Philadephia |
| Sales              | Seattle     |
| Software           | Atlanta     |
| Software           | Sacramento  |
+--------------------+-------------+
13 rows in set (0.01 sec)
```



Note: there is a key difference in the output produced by a join created with the comma (,) operator and a join created with the USING clause: In the latter case, MySQL will automatically remove redundant join fields, such that these fields appear only once in the result set.

With this syntax (USING), the joining column must have the same name in both tables. We do not need to qualify the column names in common. We do need to put parentheses around the common column name in the USING clause.

The syntax for a join that uses the ON keyword

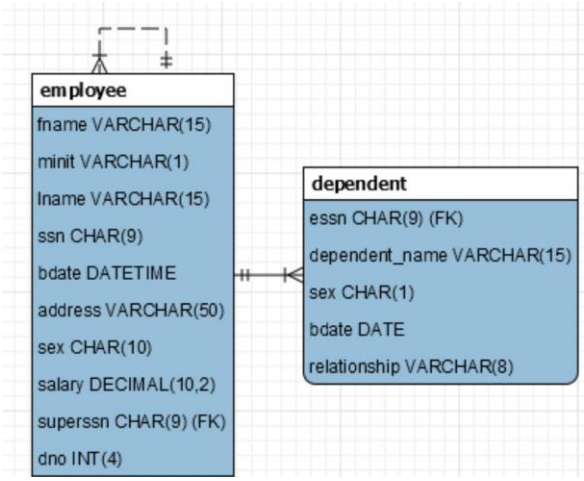
The previous two join styles used common attribute names in the joining tables. Another way to express a join when the tables have no common attribute names is to use the **JOIN ON** operand. That query will return only the rows that meet the indicated join condition. The join condition will typically include an equality comparison expression of two columns. (The columns may or may not share the same name but, obviously, must have comparable data types.)

The syntax is:

```
SELECT column-list
FROM table1
JOIN table2
ON join-condition
```

Example 8: For each employee, display the employee's name and the name of his or her dependents.

```
mysql> SELECT e.fname
       , e.lname
       , d.dependent_name
       FROM employee e
       JOIN dependent d
       ON (e.ssn = d.essn);
+----------+---------+----------------+
| fname    | lname   | dependent_name |
+----------+---------+----------------+
| John     | Smith   | Alice          |
| John     | Smith   | Elizabeth      |
| John     | Smith   | Michael        |
| Franklin | Wong    | Alice          |
| Franklin | Wong    | Joy            |
| Franklin | Wong    | Theodore       |
| Alex     | Freed   | Johnny         |
| Alex     | Freed   | Tommy          |
| Bonnie   | Bays    | Chris          |
| Alec     | Best    | Sam            |
| Jennifer | Wallace | Abner          |
+----------+---------+----------------+
11 rows in set (0.00 sec)
```



Note that unlike the JOIN USING operands, the JOIN ON clause requires a table qualifier for the common attributes. If you do not specify the table qualifier, you will get a "column ambiguously defined" error message.

1.5. SELF-JOINS

A self-join is a join made on the same table. To perform a self-join, you must use a different table alias to identify each reference to the table in the query. Two columns within the same table have a relationship.
Can be created by two methods:

- Using the WHERE clause
- Using the JOIN method with the JOIN . . . ON keywords

When you code a self-join, you must use aliases for the tables, and you must qualify each column name with the alias.

Example 9: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
mysql> SELECT e.fname, e.lname, s.fname, s.lname
       FROM employee e
       JOIN employee s ON (e.superssn = s.ssn);
```

fname	lname	fname	lname
Jon	Jones	Jared	James
Justin	Mark	Jared	James
Brad	Knight	Jared	James
John	Smith	Franklin	Wong
Josh	Zell	Evan	Wallis
Andy	Vile	Evan	Wallis
Tom	Brand	Evan	Wallis
Jenny	Vos	Josh	Zell
Chris	Carter	Josh	Zell
Jeff	Chase	Kim	Grace
Franklin	Wong	James	Borg
Bonnie	Bays	Alex	Freed
Alec	Best	Alex	Freed
Sam	Snedden	Alex	Freed
Joyce	English	Franklin	Wong
Nandita	Ball	John	James
Jill	Jarvis	Bob	Bender
Kate	King	Bob	Bender
Lyle	Leslie	Jill	Jarvis
Billie	King	Lyle	Leslie
Jon	Kramer	Lyle	Leslie
Ray	King	Billie	King
Gerald	Small	Kate	King
Arnold	Head	Kate	King
Helga	Pataki	Kate	King
Naveen	Drew	Gerald	Small
Carl	Reedy	Naveen	Drew
Sammy	Hall	Carl	Reedy
Red	Bacher	Sammy	Hall
Ramesh	Narayan	Franklin	Wong
Jennifer	Wallace	James	Borg
Ahmad	Jabbar	Jennifer	Wallace
Alicia	Zelaya	Jennifer	Wallace

33 rows in set (0.00 sec)

1.6. NATURAL JOIN

This join can be used if the columns in common have the same name. This will join on any and all columns having the same identifier in the two tables. Therefore, it creates maintenance problems if attributes are renamed or additional columns are added to the tables. When the natural join keyword is used in the FROM clause, table aliases cannot be assigned.

The syntax is:

```
SELECT column-list
FROM table1
NATURAL JOIN table2
[NATURAL JOIN table 3]..
```

The natural join is often not used, as same named columns may not always represent common data columns to be used in a join. For example, two tables may have a column named TYPE; however, the column may represent two entirely different data items. Also, table column modifications could lead to unexpected join operations if close attention to related table column names is not considered.

It is appropriate to use the natural join keywords when the tables being joined share common columns with the same names and definitions. In addition, a natural join often yields unexpected results for complex queries. As a result, it's more common to use the ON or USING clause to join tables.

1.7. OUTER JOINS

By default, the JOIN keyword creates an inner join. To use it to create an outer join, you include the keyword LEFT, RIGHT, or FULL to identify the join type. You can also include the OUTER keyword; however, using it isn't necessary to specify an outer join.

Left and Right Outer Joins

Outer joins can be split into two types: Left outer joins and Right outer joins

NOTE: As you'll see, if table1 and table2 both contain rows with null values, you get different results depending on whether you use a left or right outer join.

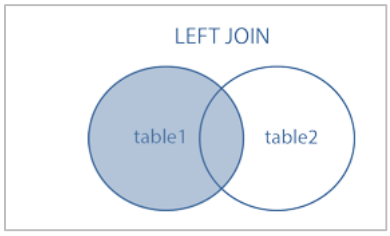
What outer joins do?

Joins of this type	**Retrieve unmatched rows from**
Left outer join	The first (left) table
Right outer join	The second (right) table

1.7.1. LEFT OUTER JOIN

The LEFT OUTER JOIN returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also the rows in the left side table with unmatched values in the right side table.

When you use a left outer join, the result set includes all the rows from the first, or left, table.



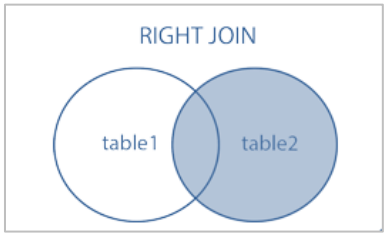
The syntax is:

```
SELECT column-list
FROM table1
LEFT [OUTER] JOIN table2
ON join-condition
```

- table1 and table2 are the tables to join.
- LEFT specifies a left outer join.

1.7.2.RIGHT OUTER JOIN

The RIGHT OUTER JOIN returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also the rows in the right side table with unmatched values in the left side table. When you use a right outer join, the result set includes all the rows from the second, or right, table.



he syntax is:

```
SELECT column-list
FROM table1
RIGHT [OUTER] JOIN table2
ON join-condition
```

- table1 and table2 are the tables to join.
- RIGHT specifies a right outer join

Example 10: This example performs right join of the department and project tables and returns selected attributes: (departments with and without projects) .

```
mysql> SELECT dname, d.dnumber, pname, p.dnum
       FROM project p
       RIGHT JOIN department d
       ON d.dnumber = p.dnum;
```

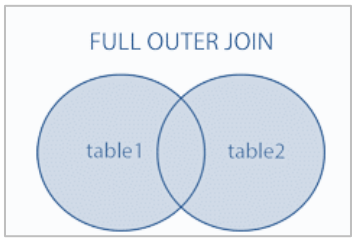
```
+----------------+---------+------------------+------+
| dname          | dnumber | pname            | dnum |
+----------------+---------+------------------+------+
| Administration |       4 | Computerization  |    4 |
| Administration |       4 | Newbenefits      |    4 |
| Hardware       |       7 | InkjetPrinters   |    7 |
| Hardware       |       7 | LaserPrinters    |    7 |
| Headquarters   |       1 | Reorganization   |    1 |
| Research       |       5 | ProductX         |    5 |
| Research       |       5 | ProductY         |    5 |
| Research       |       5 | ProductZ         |    5 |
| Sales          |       8 | NULL             | NULL |
| Software       |       6 | OperatingSystems |    6 |
| Software       |       6 | DatabaseSystems  |    6 |
| Software       |       6 | Middleware       |    6 |
+----------------+---------+------------------+------+
12 rows in set (0.00 sec)
```

Note: The terms “left join” and “right join” are interchangeable, depending on where you’re standing. A left join can be turned into a right join (and vice versa) simply by altering the order of the tables in the join. To illustrate, consider the following two queries, which are equivalent:

```
SELECT * FROM c LEFT JOIN a USING (id);
SELECT * FROM a RIGHT JOIN c USING (id)
```

1.7.3.FULL OUTER JOIN

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2). The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.



The syntax is:

```
SELECT column-list
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

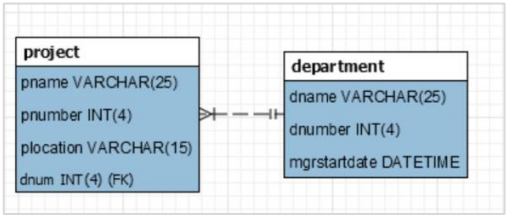
Where

- table1 and table2 are the tables to join.
- FULL specifies a full outer join. A full outer join uses all rows in table1 and table2, including those that have null values in the columns used in the join.

Note: MySQL doesn't support FULL OUTER JOIN.

Example 11: The following example performs left join of the department and project tables and returns selected attributes: (departments with and without projects) .

```
mysql> SELECT dname, d.dnumber, pname ,p.dnum
       FROM department d
       LEFT JOIN project p
       ON d.dnumber = p.dnum;
+----------------+---------+------------------+------+
| dname          | dnumber | pname            | dnum |
+----------------+---------+------------------+------+
| Administration |       4 | Computerization  |    4 |
| Administration |       4 | Newbenefits      |    4 |
| Hardware       |       7 | InkjetPrinters   |    7 |
| Hardware       |       7 | LaserPrinters    |    7 |
| Headquarters   |       1 | Reorganization   |    1 |
| Research       |       5 | ProductX         |    5 |
| Research       |       5 | ProductY         |    5 |
| Research       |       5 | ProductZ         |    5 |
| Sales          |       8 | NULL             | NULL |
| Software       |       6 | OperatingSystems |    6 |
| Software       |       6 | DatabaseSystems  |    6 |
| Software       |       6 | Middleware       |    6 |
+----------------+---------+------------------+------+
12 rows in set (0.00 sec)
```



1.8. Non-equijoins

A non-equijoin uses an operator other than the equality operator = in the join. These operators are not equal <>, less than <, greater than >, less than or equal to <=, greater than or equal to >=, LIKE, IN, and BETWEEN.

Example 12: Display the name of the supervisor for employee's SSN "111111100".

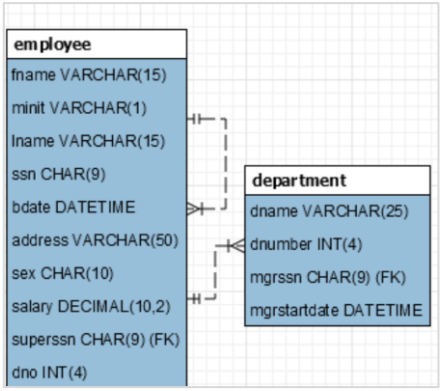
```
mysql> SELECT e.lname "employee", e.fname, s.lname "supervisor"
       FROM employee e
       JOIN employee s ON e.ssn = s.superssn
       AND e.ssn = 111111100;
+----------+-------+------------+
| employee | fname | supervisor |
+----------+-------+------------+
| James    | Jared | Jones      |
| James    | Jared | Mark       |
| James    | Jared | Knight     |
+----------+-------+------------+
3 rows in set (0.00 sec)
```

Example 13*:* Display the department name, the name, and address of all employees who don't work for the 'Research' department.

```
mysql> SELECT dname, fname,lname
       FROM employee e
       JOIN department d ON(e.dno= d.dnumber)
       WHERE (dname<>'Research');
+----------------+----------+---------+
| dname          | fname    | lname   |
+----------------+----------+---------+
| Software       | Jared    | James   |
| Software       | Jon      | Jones   |
| Software       | Justin   | Mark    |
| Software       | Brad     | Knight  |
| Hardware       | Evan     | Wallis  |
| Hardware       | Josh     | Zell    |
| Hardware       | Andy     | Vile    |
| Hardware       | Tom      | Brand   |
| Hardware       | Jenny    | Vos     |
| Hardware       | Chris    | Carter  |
| Software       | Kim      | Grace   |
| Software       | Jeff     | Chase   |
| Hardware       | Alex     | Freed   |
| Hardware       | Bonnie   | Bays    |
| Hardware       | Alec     | Best    |
| Hardware       | Sam      | Snedden |
| Software       | John     | James   |
| Software       | Nandita  | Ball    |
| Sales          | Bob      | Bender  |
| Sales          | Jill     | Jarvis  |
| Sales          | Kate     | King    |
| Sales          | Lyle     | Leslie  |
| Sales          | Billie   | King    |
| Sales          | Jon      | Kramer  |
| Sales          | Ray      | King    |
| Sales          | Gerald   | Small   |
| Sales          | Arnold   | Head    |
| Sales          | Helga    | Pataki  |
| Sales          | Naveen   | Drew    |
| Sales          | Carl     | Reedy   |
| Sales          | Sammy    | Hall    |
| Sales          | Red      | Bacher  |
| Headquarters   | James    | Borg    |
| Administration | Jennifer | Wallace |
| Administration | Ahmad    | Jabbar  |
| Administration | Alicia   | Zelaya  |
+----------------+----------+---------+
36 rows in set (0.00 sec)
```

Example 14: Display all employee names and their department names.

```
mysql> SELECT lname, fname, dname
       FROM employee e
       JOIN department d ON e.dno = d.dnumber;
+---------+----------+----------------+
| lname   | fname    | dname          |
+---------+----------+----------------+
| James   | Jared    | Software       |
| Jones   | Jon      | Software       |
| Mark    | Justin   | Software       |
| Knight  | Brad     | Software       |
| Smith   | John     | Research       |
| Wallis  | Evan     | Hardware       |
| Zell    | Josh     | Hardware       |
| Vile    | Andy     | Hardware       |
| Brand   | Tom      | Hardware       |
| Vos     | Jenny    | Hardware       |
| Carter  | Chris    | Hardware       |
| Grace   | Kim      | Software       |
| Chase   | Jeff     | Software       |
| Wong    | Franklin | Research       |
| Freed   | Alex     | Hardware       |
| Bays    | Bonnie   | Hardware       |
| Best    | Alec     | Hardware       |
| Snedden | Sam      | Hardware       |
| English | Joyce    | Research       |
| James   | John     | Software       |
| Ball    | Nandita  | Software       |
| Bender  | Bob      | Sales          |
| Jarvis  | Jill     | Sales          |
| King    | Kate     | Sales          |
| Leslie  | Lyle     | Sales          |
| King    | Billie   | Sales          |
| Kramer  | Jon      | Sales          |
| King    | Ray      | Sales          |
| Small   | Gerald   | Sales          |
| Head    | Arnold   | Sales          |
| Pataki  | Helga    | Sales          |
| Drew    | Naveen   | Sales          |
| Reedy   | Carl     | Sales          |
| Hall    | Sammy    | Sales          |
| Bacher  | Red      | Sales          |
| Narayan | Ramesh   | Research       |
| Borg    | James    | Headquarters   |
| Wallace | Jennifer | Administration |
| Jabbar  | Ahmad    | Administration |
| Zelaya  | Alicia   | Administration |
+---------+----------+----------------+
40 rows in set (0.00 sec)
```

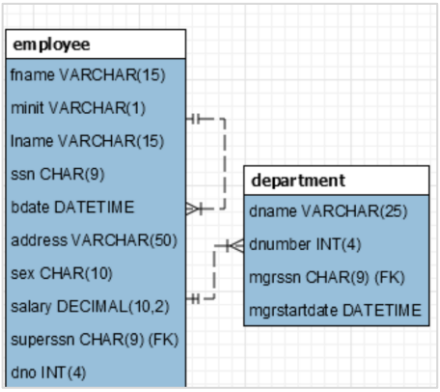


Example 15*:* Find all employees' full names (lastname, firstname format) with salary, and their supervisor's name with salary.

```
mysql> SELECT concat(e.lname, " ", e.fname) AS name
       , e.salary ,s.lname, s.fname, s.salary
       FROM employee e
       JOIN employee s ON  e.superssn = s.ssn;
+-----------------+----------+---------+----------+----------+
| name            | salary   | lname   | fname    | salary   |
+-----------------+----------+---------+----------+----------+
| Jones Jon       | 45000.00 | James   | Jared    | 85000.00 |
| Mark Justin     | 40000.00 | James   | Jared    | 85000.00 |
| Knight Brad     | 44000.00 | James   | Jared    | 85000.00 |
| Smith John      | 30000.00 | Wong    | Franklin | 40000.00 |
| Zell Josh       | 56000.00 | Wallis  | Evan     | 92000.00 |
| Vile Andy       | 53000.00 | Wallis  | Evan     | 92000.00 |
| Brand Tom       | 62500.00 | Wallis  | Evan     | 92000.00 |
| Vos Jenny       | 61000.00 | Zell    | Josh     | 56000.00 |
| Carter Chris    | 43000.00 | Zell    | Josh     | 56000.00 |
| Chase Jeff      | 44000.00 | Grace   | Kim      | 79000.00 |
| Wong Franklin   | 40000.00 | Borg    | James    | 55000.00 |
| Bays Bonnie     | 70000.00 | Freed   | Alex     | 89000.00 |
| Best Alec       | 60000.00 | Freed   | Alex     | 89000.00 |
| Snedden Sam     | 48000.00 | Freed   | Alex     | 89000.00 |
| English Joyce   | 25000.00 | Wong    | Franklin | 40000.00 |
| Ball Nandita    | 62000.00 | James   | John     | 81000.00 |
| Jarvis Jill     | 36000.00 | Bender  | Bob      | 96000.00 |
| King Kate       | 44000.00 | Bender  | Bob      | 96000.00 |
| Leslie Lyle     | 41000.00 | Jarvis  | Jill     | 36000.00 |
| King Billie     | 38000.00 | Leslie  | Lyle     | 41000.00 |
| Kramer Jon      | 41500.00 | Leslie  | Lyle     | 41000.00 |
| King Ray        | 44500.00 | King    | Billie   | 38000.00 |
| Small Gerald    | 29000.00 | King    | Kate     | 44000.00 |
| Head Arnold     | 33000.00 | King    | Kate     | 44000.00 |
| Pataki Helga    | 32000.00 | King    | Kate     | 44000.00 |
| Drew Naveen     | 34000.00 | Small   | Gerald   | 29000.00 |
| Reedy Carl      | 32000.00 | Drew    | Naveen   | 34000.00 |
| Hall Sammy      | 37000.00 | Reedy   | Carl     | 32000.00 |
| Bacher Red      | 33500.00 | Hall    | Sammy    | 37000.00 |
| Narayan Ramesh  | 38000.00 | Wong    | Franklin | 40000.00 |
| Wallace Jennifer| 43000.00 | Borg    | James    | 55000.00 |
| Jabbar Ahmad    | 25000.00 | Wallace | Jennifer | 43000.00 |
| Zelaya Alicia   | 25000.00 | Wallace | Jennifer | 43000.00 |
+-----------------+----------+---------+----------+----------+
33 rows in set (0.00 sec)
```

Example 16: Find all employees in the 'software' department.

```
mysql> SELECT lname, fname, dname
       FROM employee e JOIN department d
       ON (e.dno = d.dnumber)
       WHERE d.dname = 'software';
+--------+---------+----------+
| lname  | fname   | dname    |
+--------+---------+----------+
| James  | Jared   | Software |
| Jones  | Jon     | Software |
| Mark   | Justin  | Software |
| Knight | Brad    | Software |
| Grace  | Kim     | Software |
| Chase  | Jeff    | Software |
| James  | John    | Software |
| Ball   | Nandita | Software |
+--------+---------+----------+
8 rows in set (0.00 sec)
```



Example 17: Display the employee last name and department name for all employees who have an 'a' (lowercase) in their last names.

```
mysql> SELECT lname, dname
       FROM employee e JOIN department d
       ON (e.dno = d.dnumber)
       WHERE lname LIKE '%a%';
+---------+----------------+
| lname   | dname          |
+---------+----------------+
| James   | Software       |
| Mark    | Software       |
| Wallis  | Hardware       |
| Brand   | Hardware       |
| Carter  | Hardware       |
| Grace   | Software       |
| Chase   | Software       |
| Bays    | Hardware       |
| James   | Software       |
| Ball    | Software       |
| Jarvis  | Sales          |
| Kramer  | Sales          |
| Small   | Sales          |
| Head    | Sales          |
| Pataki  | Sales          |
| Hall    | Sales          |
| Bacher  | Sales          |
| Narayan | Research       |
| Wallace | Administration |
| Jabbar  | Administration |
| Zelaya  | Administration |
+---------+----------------+
21 rows in set (0.00 sec)
```

Example 18: Who are the employees assigned for each project? Display project number, project name, and department number. Show Employee's last name and first name.

```
mysql> SELECT P.pnumber, pname, P.dnum, e.ssn, lname,fname
       FROM employee AS e
       JOIN works_on AS a ON (a.essn = e.ssn)
       JOIN project AS p ON (a.pno = p.pnumber);
```

pnumber	pname	dnum	ssn	lname	fname
1	ProductX	5	123456789	Smith	John
1	ProductX	5	453453453	English	Joyce
2	ProductY	5	123456789	Smith	John
2	ProductY	5	333445555	Wong	Franklin
2	ProductY	5	453453453	English	Joyce
3	ProductZ	5	333445555	Wong	Franklin
3	ProductZ	5	666884444	Narayan	Ramesh
10	Computerization	4	333445555	Wong	Franklin
10	Computerization	4	987987987	Jabbar	Ahmad
10	Computerization	4	999887777	Zelaya	Alicia
20	Reorganization	1	333445555	Wong	Franklin
20	Reorganization	1	888665555	Borg	James
20	Reorganization	1	987654321	Wallace	Jennifer
30	Newbenefits	4	987654321	Wallace	Jennifer
30	Newbenefits	4	987987987	Jabbar	Ahmad
30	Newbenefits	4	999887777	Zelaya	Alicia
61	OperatingSystems	6	111111100	James	Jared
61	OperatingSystems	6	111111101	Jones	Jon
61	OperatingSystems	6	111111102	Mark	Justin
61	OperatingSystems	6	111111103	Knight	Brad
61	OperatingSystems	6	666666607	Small	Gerald
61	OperatingSystems	6	666666610	Drew	Naveen
61	OperatingSystems	6	666666611	Reedy	Carl
61	OperatingSystems	6	666666612	Hall	Sammy
61	OperatingSystems	6	666666613	Bacher	Red
62	DatabaseSystems	6	222222200	Wallis	Evan
62	DatabaseSystems	6	222222201	Zell	Josh
62	DatabaseSystems	6	222222202	Vile	Andy
62	DatabaseSystems	6	222222203	Brand	Tom
62	DatabaseSystems	6	222222204	Vos	Jenny
62	DatabaseSystems	6	222222205	Carter	Chris
62	DatabaseSystems	6	666666608	Head	Arnold
62	DatabaseSystems	6	666666613	Bacher	Red
63	Middleware	6	333333300	Grace	Kim
63	Middleware	6	333333301	Chase	Jeff
63	Middleware	6	666666609	Pataki	Helga
63	Middleware	6	666666613	Bacher	Red
91	InkjetPrinters	7	444444400	Freed	Alex
91	InkjetPrinters	7	444444401	Bays	Bonnie
91	InkjetPrinters	7	444444402	Best	Alec
91	InkjetPrinters	7	444444403	Snedden	Sam
91	InkjetPrinters	7	666666601	Jarvis	Jill
91	InkjetPrinters	7	666666603	Leslie	Lyle
91	InkjetPrinters	7	666666604	King	Billie
91	InkjetPrinters	7	666666606	King	Ray
92	LaserPrinters	7	555555500	James	John
92	LaserPrinters	7	555555501	Ball	Nandita
92	LaserPrinters	7	666666605	Kramer	Jon

```
48 rows in set (0.00 sec)
```

Example 19: Who are the employees assigned for each project? Display project number, project name, department number, and department name. Display Employee's last name and first name. Sort by project number in ascending order.

```
mysql> SELECT p.pnumber, pname, d.dname, e.ssn, e.lname, e.fname
       FROM employee AS e
       JOIN works_on AS a ON (a.essn = e.ssn)
       JOIN project AS p ON (a.pno = p.pnumber)
       JOIN department AS d ON (p.dnum = d.dnumber)
       ORDER BY p.pnumber;
```

pnumber	pname	dnum	ssn	lname	fname
1	ProductX	5	123456789	Smith	John
1	ProductX	5	453453453	English	Joyce
2	ProductY	5	123456789	Smith	John
2	ProductY	5	333445555	Wong	Franklin
2	ProductY	5	453453453	English	Joyce
3	ProductZ	5	333445555	Wong	Franklin
3	ProductZ	5	666884444	Narayan	Ramesh
10	Computerization	4	333445555	Wong	Franklin
10	Computerization	4	987987987	Jabbar	Ahmad
10	Computerization	4	999887777	Zelaya	Alicia
20	Reorganization	1	333445555	Wong	Franklin
20	Reorganization	1	888665555	Borg	James
20	Reorganization	1	987654321	Wallace	Jennifer
30	Newbenefits	4	987654321	Wallace	Jennifer
30	Newbenefits	4	987987987	Jabbar	Ahmad
30	Newbenefits	4	999887777	Zelaya	Alicia
61	OperatingSystems	6	111111100	James	Jared
61	OperatingSystems	6	111111101	Jones	Jon
61	OperatingSystems	6	111111102	Mark	Justin
61	OperatingSystems	6	111111103	Knight	Brad
61	OperatingSystems	6	666666607	Small	Gerald
61	OperatingSystems	6	666666610	Drew	Naveen
61	OperatingSystems	6	666666611	Reedy	Carl
61	OperatingSystems	6	666666612	Hall	Sammy
61	OperatingSystems	6	666666613	Bacher	Red
62	DatabaseSystems	6	222222200	Wallis	Evan
62	DatabaseSystems	6	222222201	Zell	Josh
62	DatabaseSystems	6	222222202	Vile	Andy
62	DatabaseSystems	6	222222203	Brand	Tom
62	DatabaseSystems	6	222222204	Vos	Jenny
62	DatabaseSystems	6	222222205	Carter	Chris
62	DatabaseSystems	6	666666608	Head	Arnold
62	DatabaseSystems	6	666666613	Bacher	Red
63	Middleware	6	333333300	Grace	Kim
63	Middleware	6	333333301	Chase	Jeff
63	Middleware	6	666666609	Pataki	Helga
63	Middleware	6	666666613	Bacher	Red
91	InkjetPrinters	7	444444400	Freed	Alex
91	InkjetPrinters	7	444444401	Bays	Bonnie
91	InkjetPrinters	7	444444402	Best	Alec
91	InkjetPrinters	7	444444403	Snedden	Sam
91	InkjetPrinters	7	666666601	Jarvis	Jill
91	InkjetPrinters	7	666666603	Leslie	Lyle
91	InkjetPrinters	7	666666604	King	Billie
91	InkjetPrinters	7	666666606	King	Ray
92	LaserPrinters	7	555555500	James	John
92	LaserPrinters	7	555555501	Ball	Nandita
92	LaserPrinters	7	666666605	Kramer	Jon

```
48 rows in set (0.00 sec)
```

Example 20: Who are the employees assigned for the projects run by the *Research* department? Display project number, project name, department number, and department name. Display employee's last name and first name. Sort by project number in ascending order.

```
mysql> SELECT P.pnumber, pname, D.dname,  E.ssn, lname, fname
       FROM employee  AS e
       JOIN works_on AS a ON a.essn = e.ssn
       JOIN project AS p ON a.pno = p.pnumber
       JOIN department AS d ON p.dnum= d.dnumber
       WHERE dname = 'hardware'
       ORDER BY p.pnumber;
```

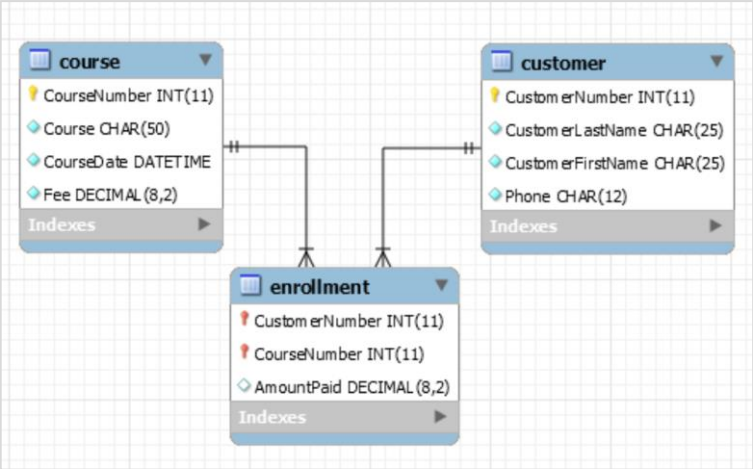
pnumber	pname	dname	ssn	lname	fname
91	InkjetPrinters	Hardware	444444400	Freed	Alex
91	InkjetPrinters	Hardware	444444401	Bays	Bonnie
91	InkjetPrinters	Hardware	444444402	Best	Alec
91	InkjetPrinters	Hardware	444444403	Snedden	Sam
91	InkjetPrinters	Hardware	666666601	Jarvis	Jill
91	InkjetPrinters	Hardware	666666603	Leslie	Lyle
91	InkjetPrinters	Hardware	666666604	King	Billie
91	InkjetPrinters	Hardware	666666606	King	Ray
92	LaserPrinters	Hardware	555555500	James	John
92	LaserPrinters	Hardware	555555501	Ball	Nandita
92	LaserPrinters	Hardware	666666605	Kramer	Jon

11 rows in set (0.00 sec)

Art Course Database

The following is a set of tables for the Art Course database.

CUSTOMER (CustomerNumber, CustomerLastName, CustomerFirstName, Phone)
COURSE (CourseNumber, Course, CourseDate, Fee)
ENROLLMENT (CustomerNumber, CourseNumber, AmountPaid)



Where:

- CustomerNumber in ENROLLMENT must exist in CustomerNumber in CUSTOMER
- CourseNumber in ENROLLMENT must exist in CourseNumber in COURSE
- CustomerNumber and CourseNumber are surrogate keys. Therefore, these numbers will never be modified, and there is no need for cascading updates. No customer data are ever deleted so there is no need to cascade deletions. Courses can be deleted. If there are enrollment entries for a deleted class, they should also be deleted.

We will use the Art Course database for the following examples:

Example 21: Write an SQL statementthat showss CourseNumber, Course, CourseDate, CustomerNumber, CustomerLastName, CustomerFirstName, and Phone.

```
mysql> SELECT co.CourseNumber, Course, CourseDate, cu.CustomerNumber
      ,CustomerLastName, CustomerFirstName, Phone
       FROM course AS co
       JOIN enrollment AS e ON CO.CourseNumber = e.CourseNumber
       JOIN customer CU ON e.CustomerNumber = cu.CustomerNumber;
+--------------+------------+---------------------+----------------+------------------+-------------------+--------------+
| CourseNumber | Course     | CourseDate          | CustomerNumber | CustomerLastName | CustomerFirstName | Phone        |
+--------------+------------+---------------------+----------------+------------------+-------------------+--------------+
|            1 | Adv Pastels | 2015-10-01 00:00:00 |              1 | Johnson          | Ariel             | 206-567-1234 |
|            1 | Adv Pastels | 2015-10-01 00:00:00 |              3 | Jackson          | Charles           | 306-789-3456 |
|            1 | Adv Pastels | 2015-10-01 00:00:00 |              4 | Pearson          | Jeffery           | 206-567-2345 |
|            2 | Beg Oils    | 2015-09-15 00:00:00 |              2 | Green            | Robin             | 425-678-8765 |
|            2 | Beg Oils    | 2015-09-15 00:00:00 |              5 | Sears            | Miguel            | 360-789-4567 |
|            3 | Int Pastels | 2015-03-15 00:00:00 |              1 | Johnson          | Ariel             | 206-567-1234 |
|            4 | Beg Oils    | 2015-10-15 00:00:00 |              7 | Myers            | Lynda             | 360-789-5678 |
|            5 | Adv Pastels | 2015-11-15 00:00:00 |              6 | Kyle             | Leah              | 425-678-7654 |
+--------------+------------+---------------------+----------------+------------------+-------------------+--------------+
8 rows in set (0.00 sec)
```

Example 22: Write an SQL statement that shows CourseNumber, Course, CustomerNumber, CustomerLastName, Phone, Fee, AmountPaid, and the calculated column (Fee—AmountPaid) renamed as AmountOwed.

```
mysql> SELECT co.CourseNumber, Course, cu.CustomerNumber
      ,CustomerLastName,Phone, Fee
       ,AmountPaid, (Fee - AmountPaid) AS AmountOwed
       FROM course AS co
      JOIN enrollment AS e ON co.CourseNumber = e.CourseNumber
      JOIN customer AS cu ON e.CustomerNumber = cu.CustomerNumber;
+--------------+------------+----------------+------------------+--------------+--------+------------+------------+
| CourseNumber | Course     | CustomerNumber | CustomerLastName | Phone        | Fee    | AmountPaid | AmountOwed |
+--------------+------------+----------------+------------------+--------------+--------+------------+------------+
|            1 | Adv Pastels |              1 | Johnson          | 206-567-1234 | 500.00 |     250.00 |     250.00 |
|            1 | Adv Pastels |              3 | Jackson          | 306-789-3456 | 500.00 |     500.00 |       0.00 |
|            1 | Adv Pastels |              4 | Pearson          | 206-567-2345 | 500.00 |     500.00 |       0.00 |
|            2 | Beg Oils    |              2 | Green            | 425-678-8765 | 350.00 |     350.00 |       0.00 |
|            2 | Beg Oils    |              5 | Sears            | 360-789-4567 | 350.00 |     350.00 |       0.00 |
|            3 | Int Pastels |              1 | Johnson          | 206-567-1234 | 350.00 |     350.00 |       0.00 |
|            4 | Beg Oils    |              7 | Myers            | 360-789-5678 | 350.00 |       0.00 |     350.00 |
|            5 | Adv Pastels |              6 | Kyle             | 425-678-7654 | 500.00 |     250.00 |     250.00 |
+--------------+------------+----------------+------------------+--------------+--------+------------+------------+
8 rows in set (0.00 sec)
```

Example 23: Write and run an SQL query to list all students and courses they are registered for. Include, in this order, CustomerNumber, CustomerLastName, CustomerFirstName, Phone, CourseNumber, and AmountPaid.

```
mysql> SELECT cu.CustomerNumber, CustomerLastName, CustomerFirstName
       ,Phone, co.CourseNumber, AmountPaid
       FROM customer AS cu
       JOIN enrollment AS en ON cu.CustomerNumber = en.CustomerNumber
       JOIN course AS co ON en.CourseNumber = co.CourseNumber;
+----------------+-----------------+------------------+--------------+--------------+------------+
| CustomerNumber | CustomerLastName | CustomerFirstName | Phone        | CourseNumber | AmountPaid |
+----------------+-----------------+------------------+--------------+--------------+------------+
|              1 | Johnson          | Ariel            | 206-567-1234 |            1 |     250.00 |
|              3 | Jackson          | Charles          | 306-789-3456 |            1 |     500.00 |
|              4 | Pearson          | Jeffery          | 206-567-2345 |            1 |     500.00 |
|              2 | Green            | Robin            | 425-678-8765 |            2 |     350.00 |
|              5 | Sears            | Miguel           | 360-789-4567 |            2 |     350.00 |
|              1 | Johnson          | Ariel            | 206-567-1234 |            3 |     350.00 |
|              7 | Myers            | Lynda            | 360-789-5678 |            4 |       0.00 |
|              6 | Kyle             | Leah             | 425-678-7654 |            5 |     250.00 |
+----------------+-----------------+------------------+--------------+--------------+------------+
8 rows in set (0.00 sec)
```

Example 24: Write and run an SQL query to list all students registered in Adv. Pastels starting on October 1, 2015. Include in this order, Course, CourseDate, CustomerLastName, CustomerFirstName, Phone, Fee, and AmountPaid. Use a join.

```
mysql> SELECT Course, CourseDate, CustomerLastName, CustomerFirstName
       ,Phone, Fee, AmountPaid
       FROM customer AS cu
       JOIN enrollment AS en ON cu.CustomerNumber = en.CustomerNumber
       JOIN course AS co ON En.CourseNumber = co.CourseNumber
       WHERE Course = 'Adv Pastels' AND CourseDate = '2015-10-01';
+-------------+---------------------+-----------------+-------------------+--------------+--------+------------+
| Course      | CourseDate          | CustomerLastName | CustomerFirstName | Phone        | Fee    | AmountPaid |
+-------------+---------------------+-----------------+-------------------+--------------+--------+------------+
| Adv Pastels | 2015-10-01 00:00:00 | Johnson          | Ariel             | 206-567-1234 | 500.00 |     250.00 |
| Adv Pastels | 2015-10-01 00:00:00 | Jackson          | Charles           | 306-789-3456 | 500.00 |     500.00 |
| Adv Pastels | 2015-10-01 00:00:00 | Pearson          | Jeffery           | 206-567-2345 | 500.00 |     500.00 |
+-------------+---------------------+-----------------+-------------------+--------------+--------+------------+
3 rows in set (0.00 sec)
```

Key Terms

1. **Alias:** An alternate name for a table
2. **Cartesian product:** The combination of all rows in the first table and all rows in the second table
3. **Equality joins** - Links table data in two (or more) tables having equivalent data stored in a common column. These joins might also be called *equijoins*, *inner joins*, or *simple joins*.
4. **Full outer join:** A join in which all rows from both tables will be included, regardless of whether they match rows from the other table
5. **Inner join:** A join that compares the tables in the FROM clause and lists only those rows that satisfy the condition in the WHERE clause
6. **Join:** The process of combining two or more tables by finding rows in the tables that have identical values in matching fields
7. **Left outer join:** A join in which all rows from the table on the left will be included, regardless of whether they match rows from the table on the right
8. **Non-equality join** - Links data in two tables that do not have equivalent rows of data.
9. **Outer join:** A join in which all rows from one table in a join is listed, regardless of whether they match any rows in the other table
10. **Product:** The combination of all rows in the first table and all rows in the second table
11. **Right outer join:** A join in which all rows from the table on the right will be included, regardless of whether they match rows from the table on the left
12. **Self-join:** The process of joining a table to itself
13. **Table alias**: A temporary name for a table, given in the FROM clause.